

## ABSTRACT

Title of Dissertation: **EXPERT-IN-THE-LOOP  
FOR SEQUENTIAL DECISIONS  
AND PREDICTIONS**

**Kiante Brantley**  
Doctor of Philosophy, 2022

Dissertation Directed by: **Professor Hal Daumé III**  
Department of Computer Science

Sequential decisions and predictions are common problems in natural language processing, robotics, and video games. Essentially, an agent interacts with an environment to learn how to solve a particular problem. Research in sequential decisions and predictions has increased due in part to the success of reinforcement learning. However, this success has come at the cost of algorithms being very data inefficient, making learning in the real world difficult.

Our primary goal is to make these algorithms more data-efficient using an expert in the loop (e.g., imitation learning). Imitation learning is a technique leveraging an expert in sequential decision and prediction problems. Naive imitation learning has a covariate shift problem (i.e., training distribution differs from test distribution). We propose methods and ideas to address this issue and address other issues that arise in different styles of imitation learning. In particular, we study three broad areas of using an expert in the loop for sequential decisions and predictions.

First, we study the most popular category of imitation learning, interactive imitation learning. Although interactive imitation learning addresses issues around the covariate shift problem in naive imitation, it does this with a trade-off. Interactive imitation learning assumes

access to an online interactive expert, which is unrealistic. Instead, we propose a setting where this assumption is realistic and attempt to reduce the number of queries needed for interactive imitation learning.

We further study modern imitation learning algorithms. Unlike interactive imitation learning, these algorithms only address the covariate shift using demonstration data instead of querying an online interactive expert. These algorithms assume access to an underlying reinforcement learning algorithm to optimize a reward function learned from demonstration data. We benchmark all algorithms in this category and relate them to modern structured prediction NLP problems.

Beyond modern and interactive imitation learning, some problems cannot be naturally expressed and solved using these two categories of algorithms. For example, solving a particular problem while satisfying safety constraints. We introduce expert-in-the-loop techniques that extend beyond traditional imitation learning paradigms, where an expert provides demonstration features or constraints instead of state-action pairs.

# Expert-In-The-Loop for Sequential Decisions and Predictions

by

Kiante Brantley

Dissertation submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
2022

## Advisory Committee:

Professor Hal Daumé III Chair/Advisor

Professor Tom Goldstein Department Representative

Professor John Baras Dean's Representative

Professor Philip Resnik

Professor Geoff Gordon

Professor Kyunghyun Cho

© Copyright by  
Kiante Brantley  
2022

## Acknowledgments

The journey to completing a dissertation is very tough and would not have been possible without the support of so many people around me. An African-American proverb that I often heard when growing up is, “It takes a village to raise a child,” this seems to extend to completing a dissertation.

First and foremost, I would like to thank my advisor, Dr. Hal Daumé III. Hal assisted me in getting into the University of Maryland College Park (UMD). He introduced me to the area of sequential decision-making and provided me with compelling and exciting problems to work on throughout my six years at UMD. Hal always gave me insight into problems that I was working on, whether I was collaborating with him or not. He also introduced me to tons of his research friends, who later became my friends as well. For these things and many more, I am forever indebted to him.

Thank you to my thesis committee members Tom Goldstein, John Baras, Philip Resnik, Geoff Gordon, and Kyunghyun Cho for providing valuable feedback for improving my thesis. Especially Thank you to John Baras, Philip Resnik, and Geoff Gordon for providing me with new and exciting connections to my work.

I want to thank the UMD CLIP faculty: Philip Resnik, Wei Ai, Jordan Boyd-Graber, Marine Carpuat, Naomi Feldman, Vanessa Frias-Martinez, Douglas W. Oard, and Louiqa Raschid. Though I did not directly learn from each CLIP faculty member, I indirectly interacted with all of them and learned a lot. I especially learned a lot from Philip Resnik over the years

because of his unique expertise and natural excitement about research.

I am fortunate to have spent a significant amount of time at New York University (NYU), which would not have been possible without Kyunghyun Cho; words can not explain my gratitude for this opportunity you provided me. Kyunghyun Cho provided me with a research home in New York. Not only did he introduce me to a community of researchers at NYU, but he also made himself available for whatever I needed, whether it was research questions or talking about life issues.

I want to thank the NYU CILVR faculty: Sam Bowman, Joan Bruna, Kyunghyun Cho, Rob Fergus, He He, Yann LeCun, Lerrel Pinto, and Rajesh Ranganath. During my four years at NYU, I spent time hanging out with each of the CILVR faculty members, learning about their group's research. I especially would like to thank the  $ML^2$  faculty: He He, Sam Bowman, and Kyunghyun Cho, where I spent most of my time. The  $ML^2$  faculty made me feel like I was an NYU student by allowing me to contribute to  $ML^2$  group events and extra circular activities. Though my research mainly focuses on machine learning, I was able to keep up with NLP-focused research due mainly to the  $ML^2$  reading group.

Without the UMIACS staff, especially the director Derek Yarnell, I would not have completed any of this work. According to slurm sreport: on opensub I am ranked 5th with the usage of 8490650, on vulcanusub I am ranked 11th with the usage of 12836573, and on clipsub I am ranked 1st with the usage of 59623422. Most students do not have access to all three clusters, so I could be ranked 1st in UMIACS student users if I combine my usage across all clusters. This is a very long-winded way of saying thank you so much for keeping the GPUs and CPUs running smoothly and helping to resolve my issues promptly. I would also like to thank Tom Hurst, Jennifer Story, Jodie Gray, Arlene Schenk, and Janice Perrone at UMD for making all the

administrative tasks easy.

I enjoyed being officemates with all of the UMD students because UMD CLIP students share a large open space with everyone. At NYU, I enjoyed being officemates with Elman Mansimov and Dieterich Lawson.

I am very grateful that I had an opportunity to interact and spend time with many of my brilliant colleagues at; **UMD**: Sweta Agrawal, Wei Ai, Joe Barrow, Jordan Boyd-Graber, Eleftheria Briakou, Yang Cao, Marine Carpuat, Hal Daumé III, Allyson Ettinger, Naomi Feldman, Shi Feng, Vanessa Frias-Martinez, Ahmed Ghoneim, Pranav Goel, Emily Gong, Fenfei Guo, He He, Alexander Hoyle, Mohit, Iyyer, Khanh Nguyen, Xing Niu, Douglas Oard, Denis Peskov, Sudha Rao, Louiqa Raschid, Philip Resnik, Pedro Rodriguez, Amr Sharaf, Han-Chin Shing, Jo Shoemaker, Anna Sotnikova, Craig Thorburn, Yogarshi Vyas, Weijia Xu, Michelle Yuan, Mozhi Zhang, Chen Zhao, Furong Huang, Jiahao Su, Jingling Li, Xuchen You, Yogesh Balaji, Alex Hanson, Moustafa Meshry, Candice Schumann and Rohan Chandra; and at **NYU**: Sam Bowman, David Brandfonbrener, Joan Bruna, Iacer Calixto, Alfredo Canziani, Angelica Chen, Kyunghyun Cho, Rob Fergus, Mark Goldstein, He He, Phu Htut, Aishwarya Kamath, Ilya Kostrikov, Ilya Kulikov, Yann LeCun, Jason Lee, Elman Mansimov, Nikita Nangia, Richard Pang, Ethan Perez, Jason Phang, Lerrel Pinto, Aahlad Puli, Roberta Raileanu, Rajesh Ranganath, Cinjon Resnick, Min Song, Mukund Sudarshan, Clara Vania, Alex Wang, Sean Welleck, Denis Yarats, Aaron Zweig.

During my Ph.D., I had an opportunity to do research internships at Microsoft Research New York, Microsoft Research Montreal, and Microsoft Research Redmond. I want to thank my mentors, collaborators, and friends for such a great learning experience: Philip Bachman, Chris Brockett, Hal Daumé III, Bill Dolan, Miro Dudík, Susan Dumais, Michel Galley, Kevin

Gao, Dan Goldstein, Geoff Gordon, Mikael Henaff, Jake Hofman, Akshay Krishnamurthy, John Langford, Soroush Mehri, Sobhan Miryoosefi, Dipendra Misra, Sudha Rao, Robert Schapire, Siddhartha Sen, Alex Slivkins, Wen Sun, Adam Trischler, Hanna Wallach, Jennifer Wortman Vaughan, and Yizhe Zhang.

Mentors were an essential part of my research journey, especially Miro Dudík. Miro has a natural enthusiasm for research and an extremely high standard for his work. Whether it was theoretical rigor or practical implementation details, research paper plotting style, or research presentation aesthetics, his approach to all aspects of research is a model I aspire to achieve. He always made himself available whether I had technical or non-technical problems, no matter how vague or precise my concerns were. I will forever be grateful for what you have done for me; thank you so much, Miro.

I want to thank the two mentors I had before starting my Ph.D. The first mentor was Randy Green, who mentored me and taught me how to play various sports such as basketball, football, and boxing throughout my child. Not only do I continue to use those skills to play sports today, but he also taught me many life lessons through sports that I carry with me in my everyday life. For instance, one of the essential elements of my work is camaraderie, which is a concept I learned from playing sports. Without my collaborators, I would not have been able to complete any of the work in this dissertation.

The second mentor is Dr. Greg Clark, who mentored me after high school and before starting my Ph.D. We spent a lot of time-solving abstract mathematical problems. Most of the skills and knowledge that I learned from Greg made it possible to complete the body of work in this dissertation. Dr. Clark built my mathematical skills, intuition, and logical reasoning skills. He also taught me that balance, ownership, and fun are essential elements in all aspects of life.



Most importantly, I want to thank Dr. Clark for believing in me. You have been an excellent friend, teacher, mentor, and a great inspiration.

Without mentioning those who provided emotional support during the Ph.D. journey, no acknowledgment section would be complete. I want to thank my friends: Ali Aftab, Vesh Bhatt, Nehemiah Emaikwu, Shi Feng, Ashwinkumar Ganesan, Alex Hanson, Rhea Horton, Chidi Iddianozie, Correy Jones, Andrew Lauziere, Moses Namara, Akshay Peshave, Maniell Workman, Sobhan Miryoosefi, Amr Sharaf, and Tim Vieira. I would also like to thank my gym family: Rob Cuaresma, Rudy (Ruderick Lopez), Spencer Stephens, Chris Robinson, Mark Ervin, and others. Without emotional support and gym family, completing this work would not have been possible. None of us got to where we are alone. These individuals' support may have been obvious or subtle to them, but it was a big part of maintaining my peace of mind and will to continue pushing through tough times.

To my brothers from another mother: Roy Anderson, Dezmond Carter, and Craig Wilson. Thank you for always being yourself, supporting me, and telling me the truth when I needed it the most.

Finally, I must thank my family. I want to thank my parents, Kiesha Byers and Ravon Brantley, for all the love and support throughout this process. Growing up, my parents directly or indirectly allowed me to form and express my opinion in good and bad times. Because of them, I establish critical thinking skills and a knack for thinking outside the box. I want to thank my brothers Beast (Kiandre Murphy), Nemo (Nehemiah Byers), and Reggie Adger, for their help in consistently providing me with a distraction from work. Thank you to my stepparents, aunts, and uncles: Tia Wright, Tom Blagogie, Shelly Norman, Micheal Byers Sr., Tommy Brantley, Nette Patrick, James Patrick, and Shawn Williams, for their love and support as well. To my

grandparents MaMa (Diane Brantley) and Grandma Na (Sharon Byers), thank you for routinely checking in on me to ensure I was ok. Without the love and support of my family, I would not have made it this far in life.

## Table of Contents

<b>Acknowledgements</b>	<b>ii</b>
<b>List of Tables</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xv</b>
<b>Chapter 1: Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Research questions . . . . .	4
1.3 Contribution . . . . .	4
1.4 Organization . . . . .	6
1.4.1 Interactive Imitation Learning . . . . .	6
1.5 Modern Imitation Learning . . . . .	8
1.6 Beyond Traditional Imitation Learning . . . . .	9
<b>Chapter 2: Background</b>	<b>11</b>
2.1 Markov Decision Process . . . . .	11
2.2 Partially Observable Markov Decision Process . . . . .	12
2.2.1 PSRs . . . . .	13
2.3 Successor Features . . . . .	14
2.4 Imitation Learning . . . . .	15
2.4.1 Behavior Cloning . . . . .	16
2.4.2 DAgger . . . . .	17
2.4.3 Roll-In Policies . . . . .	18
2.4.4 Learning to Search . . . . .	18
<b>Part 1 Interactive Imitation Learning</b>	<b>20</b>
<b>Chapter 3: Reducing Interactive Feedback</b>	<b>21</b>
3.1 Introduction . . . . .	21
3.2 Background . . . . .	23
3.2.1 Active Learning . . . . .	24
3.2.2 Active Imitation & Structured Prediction . . . . .	25
3.3 Our Approach: Learning to Query for Imitation . . . . .	25
3.3.1 Learning to Query for Imitation . . . . .	26

3.3.2	Apple Tasting for One-Sided Learning . . . . .	28
3.3.3	Measuring Policy Certainty . . . . .	29
3.3.4	Analysis . . . . .	30
3.4	Experiments . . . . .	32
3.4.1	Algorithms and Baselines . . . . .	32
3.4.2	Data and Representation . . . . .	33
3.4.3	Expert Policy and Heuristics . . . . .	34
3.4.4	Experimental Setup . . . . .	36
3.4.5	Experimental Results . . . . .	37
3.5	Conclusion . . . . .	38
3.6	Extend Details . . . . .	40
3.6.1	Wiktionary to Universal Dependencies . . . . .	40
3.6.2	Hyperparameters . . . . .	40
3.6.3	Ablation Study Difference Classifier Learning Rate . . . . .	41
3.6.4	Ablation Study Confidence Parameter: $b$ . . . . .	42
<b>Chapter 4:</b>	<b>Interactive Text Generation</b>	<b>43</b>
4.1	Introduction . . . . .	44
4.2	Related Work . . . . .	45
4.3	Background . . . . .	46
4.4	Our Approach: Non-Monotonic Sequence Generation . . . . .	47
4.4.1	Cost Measurement . . . . .	49
4.4.2	Oracle Policies . . . . .	50
4.5	Experiments . . . . .	53
4.5.1	Language Modeling . . . . .	53
4.5.2	Sentence Completion . . . . .	57
4.5.3	Word Reordering . . . . .	58
4.5.4	Machine Translation . . . . .	60
4.6	Conclusion . . . . .	61
4.7	Extend Details . . . . .	63
4.7.1	Neural Net Policy Structure . . . . .	63
4.7.2	Word Reordering . . . . .	65
4.7.3	Unconditional Generation . . . . .	65
4.7.4	Machine Translation . . . . .	67
<b>Part 2</b>	<b>Modern Imitation Learning</b>	<b>69</b>
<b>Chapter 5:</b>	<b>Uncertainty-Based Learning</b>	<b>70</b>
5.1	Introduction . . . . .	71
5.2	Related Work . . . . .	72
5.3	Our Approach: Disagreement-Regularized Imitation Learning . . . . .	74
5.4	Analysis . . . . .	76
5.4.1	Coverage Coefficient . . . . .	77
5.4.2	Regret Bound . . . . .	78
5.5	Experiments . . . . .	80

5.5.1	Tabular MDPs . . . . .	80
5.5.2	Atari Environments . . . . .	81
5.5.3	Continuous Control . . . . .	84
5.6	Conclusion . . . . .	85
5.7	Extend Details . . . . .	86
5.7.1	Proofs . . . . .	86
5.7.2	Importance of Behavior Cloning Cost . . . . .	92
5.7.3	Experimental Details . . . . .	95
5.7.4	Ablation Experiments . . . . .	96
<b>Chapter 6:</b>	<b>Emperical Study of Imtation Learning</b>	<b>100</b>
6.1	Introduction . . . . .	100
6.2	Background . . . . .	104
6.3	Algorithms . . . . .	105
6.3.1	Baseline Algorithms . . . . .	106
6.3.2	Existing IL Algorithms . . . . .	108
6.4	Related Work . . . . .	110
6.5	Experiments . . . . .	112
6.6	Results . . . . .	113
6.6.1	Continuous Control Results . . . . .	114
6.6.2	Pixel Results . . . . .	115
6.6.3	Structured Prediction Results . . . . .	116
6.7	Discussion . . . . .	116
6.8	Conclusion . . . . .	118
6.9	Extend Details . . . . .	120
6.9.1	Reinforcement Learning Background . . . . .	120
6.9.2	PPO Hyperparameters . . . . .	121
6.9.3	Neural network architecture . . . . .	122
6.9.4	Data Normalization . . . . .	122
6.9.5	Default Reinforcement Learning Hyperparameters . . . . .	124
6.9.6	Extended Related Work . . . . .	125
6.9.7	Existing Imitation Learning Algorithms Details . . . . .	126
6.9.8	K-Nearest Neighbor(KNN): . . . . .	127
6.9.9	Locally Weight Learning (LWL): . . . . .	129
6.9.10	RL with constant reward: (CR) . . . . .	132
6.9.11	Behavior Cloning (BC): . . . . .	134
6.9.12	Generative Adversarial Imitation Learning (GAIL): . . . . .	136
6.9.13	Adversarial Inverse Reinforcement Learning (AIRL): . . . . .	139
6.9.14	Random Expert Distillation (RED): . . . . .	142
6.9.15	Behavior cloning -regularized Generative Adversarial Imitation Learning (BC-GAIL): . . . . .	145
6.9.16	Behavior cloning -regularized Adversarial Inverse Reinforcement Learning (BC-AIRL): . . . . .	151
6.9.17	Behavior cloning -regularized Random Expert Distillation (BC-RED): . . . . .	154
6.9.18	Behavior cloning -regularized RL with constant reward: (BC-CR) . . . . .	157
6.9.19	Extended Experiment Details . . . . .	160

6.9.20	Continuous Control Tasks	160
6.9.21	Pixel Tasks	162
6.9.22	Structured Prediction Tasks	163
6.9.23	Extended Experiment Results	164
6.9.24	Continuous Control Tasks	164
6.9.25	Structured Prediction Tasks	173
6.9.26	Pixel-base Tasks	174

## **Part 3 Beyond Traditional Imitation Learning 175**

### **Chapter 7: Exact Imitation 176**

7.1	Introduction	177
7.2	Related Work	178
7.3	Background	179
7.4	Imitation by Feature Matching	180
7.5	Extension to POMDPs and PSRs	181
7.6	Successor Feature Sets	184
7.7	Special Cases	185
7.8	Bellman Equations	186
7.9	Feature Matching and Optimal Planning	188
7.10	Implementation	189
7.11	More on Special Cases	191
7.12	Experiments: Dynamic Programming	192
7.13	Conclusion	193
7.14	Extended Details	194
7.14.1	Feature matching	194
7.14.2	Convergence of dynamic programming	196
7.14.3	Background on PSRs	203
7.14.4	Background on policies	208
7.14.5	Successor feature matrices	209
7.14.6	Implementation and experimental setup	211
7.14.7	Convergence of point-based approximations	214

### **Chapter 8: Constraint Feedback 217**

8.1	Introduction	218
8.2	Background	220
8.3	Our Approach: APPROPO	222
8.3.1	Solving zero-sum games using online learning	224
8.3.2	Algorithm and main result	225
8.3.3	Removing the cone assumption	230
8.3.4	Practical implementation of the positive response and estimation oracles	231
8.4	Experiments	234
8.5	Conclusion	236
8.6	Extended Details	238
8.6.1	Online gradient descent (OGD)	238

8.6.2	Proof of Theorem 8.3.1 . . . . .	238
8.6.3	Proof of Theorem 8.3.2 . . . . .	239
8.6.4	APPROPO for feasibility . . . . .	242
8.6.5	Proof of Lemma 6 . . . . .	243
8.6.6	Additional experimental details . . . . .	245
<b>Chapter 9:</b>	<b>Conclusion</b>	<b>246</b>
9.1	Summary of the Thesis . . . . .	246
9.2	Future Work . . . . .	248
9.3	Imitation learning for Structured-Prediction . . . . .	248
9.4	Active Imitation Learning . . . . .	249
9.5	Imitation learning with Successor Features . . . . .	249
<b>Bibliography</b>		<b>250</b>

## List of Tables

3.1	An overview of the three tasks considered in experiments. . . . .	31
3.2	Conversion between Greek, Modern (el) Wiktionary POS tags and Universal Dependencies POS tags. . . . .	40
3.3	Hyperparameters . . . . .	41
4.1	Statistics computed over 10,000 sampled sentences (in-order traversals of sampled trees with $\langle end \rangle$ tokens removed) for policies trained on Persona-Chat. A sample is novel when it is not in the training set. Percent unique is the cardinality of the set of sampled sentences divided by the number of sampled sentences. . . . .	54
4.2	Samples from unconditional generation policies trained on Persona-Chat for each training oracle. The first sample’s underlying tree is shown. Section 4.7.1 for more samples. . . . .	55
4.3	Sentence completion samples from a policy trained on Persona-Chat with the uniform oracle. The left column shows the initial seed tree. In the sampled sentences, seed words are bold. . . . .	58
4.4	Word Reordering results on Persona-Chat, reported according to BLEU score, F1 score, and percent exact match. . . . .	59
4.5	Results of machine translation experiments for different training oracles across four different evaluation metrics. . . . .	60
4.6	Unconditional generation BLEU for various top- $k$ samplers and policies trained with the specified oracle. . . . .	66
4.7	LSTM Policy results for machine translation experiments. . . . .	66
5.1	Hyperparameters (our method) . . . . .	95
5.2	Hyperparameters (GAIL) . . . . .	96
5.3	Hyperparameters (our method) . . . . .	96
6.1	Summary of IL algorithms considered in this work. The table indicates whether a particular algorithm performs BC updates interleaved with RL updates, as well as the reward used to train the policy. The RL component of the objective specifies what reward function each algorithm is optimizing, i.e. $\mathbb{E}_{(s,a) \sim \mu^\pi} [r(s, a)]$ where $r(s, a)$ various for each algorithm’s objective. . . . .	102
6.2	An overview of the three tasks categories considered in experiments. Each task category state space is different. . . . .	104
6.3	Default settings used in RL experiments. . . . .	124
6.4	Default settings used in k-nearest neighbor. . . . .	128
6.5	Final settings used in locally weighted learning experiments. . . . .	131



6.6	Final settings used in reinforcement learning with constant reward experiments.	133
6.7	Default settings used in behavior cloning experiments. . . . .	135
6.8	Final settings used in GAIL experiments. . . . .	138
6.9	Final settings used in AIRL experiments. . . . .	141
6.10	Default settings used in random expert distillation experiments. . . . .	144
6.11	Final settings used in BC-GAIL experiments. . . . .	150
6.12	Final settings used in BC-GAIL experiments. . . . .	153
6.13	Default settings used in BC-RED experiments. . . . .	156
6.14	Final settings used in BC-CR experiments. . . . .	159

## List of Figures

3.1	A named entity recognition example (from the Wikipedia page for <b>Clarence Ellis</b> ). $x$ is the input sentence and $y$ is the (unobserved) ground truth. The predictor $\pi$ operates left-to-right and, in this example, is currently at state $s_{10}$ to tag the 10th word; the state $s_{10}$ (highlighted in purple) combines $x$ with $\hat{y}_{1:9}$ . The heuristic makes two errors at $t = 4$ and $t = 6$ . The heuristic label at $t = 10$ is $y_{10}^h = \text{ORG}$ . Under Hamming loss, the cost at $t = 10$ is minimized for $a = \text{ORG}$ , which is therefore the expert action (if it were queried). The label that would be provided for $s_{10}$ to the difference classifier is 0 because the two policies agree. .	22
3.2	Empirical evaluation on three tasks: (left) named entity recognition, (middle) keyphrase extraction and (right) part of speech tagging. The top rows shows performance (f-score or accuracy) with respect to the number of queries to the expert. The bottom row shows the number of queries as a function of the number of words seen. . . . .	35
3.3	Ablation results on (left) named entity recognition, (middle) keyphrase extraction and (right) part of speech tagging. In addition to LEAQI and DAgger (copied from Figure 3.2), these graphs also show LEAQI+NOAT (apple tasting disabled), and LEAQI+NOISYHEUR. (a heuristic that produces labels uniformly at random).	36
3.4	(top-row) English keyphrase extraction and (bottom-row) low-resource language part of speech tagging on Greek, Modern (el). We show the performance of using different learning for the difference classifier $h$ . These plots indicate that their is small difference in performance depending on the difference classifier learning rate. . . . .	41
3.5	(top-row) English keyphrase extraction and (bottom-row) low-resource language part of speech tagging on Greek, Modern (el). We show the performance of using difference confidence parameters $b$ . These plots indicate that our model is robust to difference confidence parameters. . . . .	42
4.1	A sequence, “how are you ?”, generated by the proposed approach trained on utterances from a dialogue dataset. The model first generated the word “are” and then recursively generated left and right subtrees (“how” and “you ?”, respectively) of this word. At each production step, the model may either generate a token, or an $\langle \text{end} \rangle$ token, which indicates that this subtree is complete. The full generation is performed in a level-order traversal, and the output is read off from an in-order traversal. The numbers in green squares denote generation order (level-order); those in rounded blue squares denote location in the final sequence (in-order). . . . .	44

4.2	A sampled tree for the sentence “a b c d” with an action space $\tilde{V} = (a,b,c,d,e,\langle\text{end}\rangle)$ , showing an oracle’s distribution $\pi^*$ and consecutive subsequences (“valid actions”) $Y_t$ for $t \in \{0, 1, 2, 3, 6\}$ . Each oracle distribution is depicted as 6 boxes showing $\pi^*(a_{t+1} s_t)$ (lighter = higher probability). After b is sampled at the root, two empty left and right child nodes are created, associated with valid actions (a) and (c, d), respectively. Here, $\pi^*$ only assigns positive probability to tokens in $Y_t$ .	48
4.3	POS tag counts by tree-depth, computed by tagging 10,000 sampled sentences. Counts are normalized across each row (depth), then the marginal tag probabilities are subtracted. A light value means the probability of the tag occurring at that depth is higher than the prior probability of the tag occurring.	56
4.4	Normalized entropy of $\pi(\cdot s)$ as a function of tree depth for policies trained with each of the oracles. The anneal-trained policy, unlike the others, makes low entropy decisions early.	60
5.1	Example of a problem where behavioral cloning incurs quadratic regret.	79
5.2	Results on tabular MDP from [231]. Shaded region represents range between 5 <sup>th</sup> and 95 <sup>th</sup> quantiles, computed across 500 trials. Behavior cloning exhibits poor worst-case regret, whereas DRIL has low regret across all trials.	81
5.3	Results on Atari environments. a) Final policy performance for different numbers of expert trajectories. b) Evolution of policy reward and uncertainty cost during training with $N = 5$ trajectories.	82
5.4	Results on continuous control tasks.	84
5.5	Cost ablation experiments on Atari environments.	98
5.6	Comparison of ensembling and MC dropout for posterior estimation on Atari environments.	99
6.1	Aggregate results for continuous control tasks <b>without sub-sampled trajectories</b> . Scores are normalized between a random and expert agent performance. Bars indicate 95% confidence intervals computed using stratified bootstrapping. Red colors perform BC and RL updates, blue colors are baselines and green only perform RL updates. Arrows indicate higher ( $\uparrow$ ) or lower ( $\downarrow$ ) is better.	108
6.2	Aggregate results for continuous control tasks <b>with sub-sampled trajectories</b> . Scores are normalized between a random and expert agent performance. Bars indicate 95% confidence intervals computed using stratified bootstrapping. Red colors perform BC and RL updates, blue colors are baselines and green only perform RL updates. Arrows indicate higher ( $\uparrow$ ) or lower ( $\downarrow$ ) is better.	111
6.3	Aggregate results for pixel tasks. Scores are normalized between a random and expert agent performance. Bars indicate 95% confidence intervals computed using stratified bootstrapping. Red colors perform BC and RL updates, blue colors are baselines and green colors only perform RL updates. Arrows indicate higher ( $\uparrow$ ) or lower ( $\downarrow$ ) is better.	112
6.4	Uncertainty cost MDP seen in chapter 5	117
6.5	Results that are not normalized for continuous control PyBullet/Box2D environments <b>without sub-sampled trajectories</b> . Scores are normalized between 0 (random performance) and 1 (expert performance). Bars indicate 95% confidence intervals computed using stratified bootstrapping. IQM denotes interquartile mean.	165

6.6	Results that are not normalized for continuous control PyBullet/Box2D environments <b>with sub-sampled trajectories</b> . Scores are normalized between 0 (random performance) and 1 (expert performance). Bars indicate 95% confidence intervals computed using stratified bootstrapping. IQM denotes interquartile mean. . . .	166
6.7	Results that are not normalized for continuous control PyBullet/Box2D environments <b>without sub-sampled trajectories</b> . . . . .	167
6.8	Results that are not normalized for continuous control PyBullet/Box2D environments <b>with sub-sampled trajectories</b> . . . . .	168
6.9	Results that are not normalized for continuous control MuJoCo environments <b>without sub-sampled trajectories</b> . Scores are normalized between 0 (random performance) and 1 (expert performance). Bars indicate 95% confidence intervals computed using stratified bootstrapping. IQM denotes interquartile mean. . . .	169
6.10	Results that are not normalized for continuous control MuJoCo environments <b>with sub-sampled trajectories</b> . Scores are normalized between 0 (random performance) and 1 (expert performance). Bars indicate 95% confidence intervals computed using stratified bootstrapping. IQM denotes interquartile mean. . . .	170
6.11	Results that are not normalized for continuous control MuJoCo environments <b>without sub-sampled trajectories</b> . . . . .	171
6.12	Results that are not normalized for continuous control MuJoCo environments <b>with sub-sampled trajectories</b> . . . . .	172
6.13	Results that are not normalized for NLP Gym environments. . . . .	173
6.14	Results that are not normalized for DMC/Box2D environments. . . . .	174
7.1	An example of a policy tree with actions $\uparrow, \downarrow$ and observations $R, G, B$ . . . . .	180
7.2	Maze environment example. . . . .	182
7.3	Visualization of the successor feature set $\Phi$ for a $3 \times 3$ gridworld MDP with 2d features. Start state is in yellow. Gray insets show one-step feature vectors, which depend only on the state, not the action. Each subplot shows one projection $\Phi e_j$ (scale is arbitrary, so no axes are necessary). The red sets illustrate a Bellman backup at the bottom-left state, and the black arrows illustrate the feature-matching policy there. See text for details. . . . .	186
7.4	Bellman error v. iteration for three simple test domains, varying the amount of computation per iteration. We show error separately in directions we have optimized over and in new random directions. Average of 25 random seeds of the direction $m_i$ with the highest bellman error per seed; all error bars are smaller than the line widths. The center panel shows the effect on Bellman error when we have higher- dimensional feature vectors. The rightmost panel shows the effect on Bellman error when the agent has less information about the exact state. In both cases the convergence rate stays similar, but we need more directions $m_i$ to adequately sample the boundary of $\Phi$ (i.e., to lower the asymptotic error on new directions). . . . .	191
7.5	Example of the behavior of point-based approximation. Two convex sets (top row) are very similar. If we retain the maximal points (blue circles) in the indicated directions (arrows, top right), the convex hulls of the two sets of retained points are very different (bottom row). . . . .	214

8.1	<i>Left:</i> The Mars rover environment. The agent starts in top-left and needs to reach the goal in bottom-right while avoiding rocks. <i>Middle, Right:</i> Visitation probabilities of APPROPO (middle) and APPROPO with a diversity constraints (right) at 12k samples. Both plots based on a single run. . . . .	233
8.2	<i>Left:</i> The performance of the algorithms as a function of the number of samples (steps in the environment); showing average and standard deviation over 25 runs. The vertical axes correspond to the three constraints, with thresholds shown as a dashed line; for reward (middle) this is a lower bound; for the others it is an upper bound. <i>Right:</i> Each point in the scatter plot represents the reward and the probability of failure obtained by the policy learnt by the algorithm at the specified number of samples. The grey region is the target set. Different points represent different random runs. . . . .	235

# Chapter 1: Introduction

As the impact of machine learning on all aspects of our lives continues to grow, having systems that learn through interaction with users and the world becomes increasingly pressing. The success of machine learning methods in a wide range of tasks of practical relevance over the past decade can largely be attributed to scalable data-driven learning methods. For example, natural language processing (NLP) and computer vision have seen tremendous success using enormous pre-trained models. In these settings, training on more data leads to higher accuracy and better-learned features [146]. However, these advances have primarily been limited to traditional supervised classification tasks. In interactive sequential decision-making and predictions, on the other hand, a learner’s earlier decision influences later choices leading to complex feedback loops [14, 253]. It is enticing to consider can the same kind of data-driven learning be applied in the context of sequential- decision-making and predictions; e.g., in areas such as robotics [216], virtual assistants (such as Alexa, Cortona, Siri) [241], video games [193], autonomous driving [238], and personalization systems [87].

This thesis addresses the practical and theoretical issues that arise when problem-solving sequential decision-making and predictions when an expert is available (i.e., expert-in-the-loop). We mainly use the framework of imitation learning to study these problems. First, we build on the interactive imitation learning techniques, which assume access to an online interactive expert. Then, we introduce two frameworks, one that motivates the practical use of interactive imitation learning algorithms and another to address the pitfalls of interactive imitation learning

algorithms.

This thesis further addresses issues of interactive imitation learning by studying modern imitation learning algorithms. Unlike interactive imitation learning, these algorithms do not assume access to an online interactive expert but instead assume access to expert demonstration data. These algorithms learn a reward function from data and optimize that reward using any off-the-shelf reinforcement learning algorithm. We further introduced frameworks that incorporate an expert-in-the-loop but extend beyond traditional imitation learning, e.g., incorporating expert constraints instead of expert demonstration.

To establish our contributions in the literature, we next briefly motivate our work and highlight prior work that acts as critical practical and theoretical motivation for this thesis

To establish our contributions in the literature, we next briefly motivate our work and highlight prior work that acts as critical practical and theoretical motivation for this thesis (see section 1.1). After stating our motivation (see section 1.1), we formally define the central research question of this thesis (see section 1.2). We provide a concise statement of this thesis's contributions (see section 1.3) and elaborate on this dissertation's organization (see section 1.4)

## 1.1 Motivation

Supervised learning is the most common learning paradigm of machine learning. In this paradigm, a learner is given input-output pairs and learns a function mapping from each input to output based on the tuple pairs. From the learning perspective, sequential decision-making and predictions problems differ because the decisions made now can have both immediate and long-term effects (where the best choice early on depends on the future). Decisions early on affect later states, so the states seen are not independent and identically distributed, which is assumed in

supervised learning. Sequential decision-making and predictions research has focused on casting the learning paradigm under reinforcement learning. Instead of making predictions in supervised learning, reinforcement learning algorithms take actions in a system where the objective is to maximize a reward function of the state-action (i.e., input-output) pairs.

Unfortunately, today’s sequential decision-making approaches—typically driven by reinforcement learning algorithms—require a considerable number of interactions, which is feasible only for systems we can *fully* simulate or for very narrow applications like ad placement or news recommendations. However, simulators do not already exist for many real-world systems, and building one can be very difficult or impossible. This means we need to develop algorithms that can interact with real systems and have very low sample complexity to learn in a reasonable amount of time.

Imitation learning uses an expert (e.g., human demonstrations) to reduce sample complexity (i.e., system interactions) while solving sequential decision-making and predictions problems. However, although imitation learning algorithms dramatically reduce the sample complexity in comparison to traditional reinforcement learning algorithms, several issues arise in varied domains: in natural language processing, text generation models lead to degeneration—output text generated that is confusing, inconsistent, and repetitive [128, 225, 247]; and in end-to-end learning for autonomous driving, models make errors when visiting areas not seen in the demonstration data [20, 67, 134, 160, 211]. In theory, it has been shown that no matter how much data you have, naively learning a model from demonstration data suffers from issues related to covariate shift between training data features and the state features the learner encounters when executing its own decisions (also called "exposure bias" in NLP) [212, 225, 232, 235].



## 1.2 Research questions

Alvinn: (An autonomous land vehicle in a neural network) [212] was the first attempt to develop an algorithm to interact with real systems using an expert-in-the-loop that occurred 3 decades ago. Instead of training using a reward function (i.e., with reinforcement learning), Alvinn was trained using imitation learning with pre-collected data from an expert to reduce sample complexity. However, the Alvinn experiment introduced a fundamental issue in imitation learning called the ‘covariate shift’ [231]. Essentially, suppose the set of states (i.e. roads maps in the Alvinn example) from the expert demonstrations are different from the set of states the agent sees on the road. In that case, the agent will perform poorly because the agent was not trained on these states. There have been several proposed ideas to address this issue, but they all come with their set of trade-offs. Given the increased emergence of systems that can potentially learn through interaction (e.g., Amazon’s Alexa, Google Assistant, Online Video Games, Amazon Warehouse Robots), addressing these issues has become very important. In this thesis, we address the problem of how do you incorporate *any* kind of expert feedback into sequential decision-making and prediction problems.

## 1.3 Contribution

The primary contribution of this thesis is outlined below:

**Experimental:** We empirically study the merits of recent imitation learning algorithms compared to naive imitation learning algorithms. Prior work has empirically compared current algorithms against naive imitation learning algorithms. However, we fill the gap in the literature, showing the pitfalls of specific categories of imitation learning algorithms (see chapters 3 and

5). We also extensively compare missing domains and tricks that prior did not include (see chapter 6). We further present a new set of problems requiring a different expert-in-the-loop type than demonstration data or online expert (see chapters 7 and 8).

**Algorithmic-Design:** We introduce new algorithms for learning with an expert-in-the-loop motivated by the issues of current imitation learning (see chapters 3, 5, 7, and 8). Naive imitation learning algorithms encounter problems when the expert’s states are different from the states the agent sees. One algorithm, in particular, attempts to guide the agent back to states similar to the expert when interacting in a system (see chapter 5). We also address issues in interactive imitation learning where the expert state-visitation is not taken into account (see chapter 7).

**Theoretical:** We introduce a new imitation learning category that attempts to address the covariate without assuming access to an online expert that can be queried (see chapter 6). We study one algorithm in-depth in this category. We show that this algorithm can overcome the covariate shift issues in naive imitation learning in theory (see chapter 5). We further justify the theoretical guarantees of most techniques introduced in this thesis to understand their respective behavior compared to prior work.

**Direction:** There has been a discount between theoretical and empirical issues in imitation learning. In theory, the covariate shift problem is a big issue, but it is not an issue in practice. We close this gap by connecting some of the most successful modern imitation learning algorithms for structured prediction natural language processing to a new category of imitation learning algorithms. This connection relates some modern natural language processing issues to the covariate shift issues (i.e. exposure bias issues) (see section 6.7).

## 1.4 Organization

The primary contribution of this thesis is outlined in three parts consisting of six content chapters: chapters 3, 4 explore interactive imitation learning, chapters 5, 6 explore modern imitation learning algorithms, and chapters 7, 8 explore techniques beyond traditional imitation learning. Before the three parts is a review of the most commonly used terminology and methods in the imitation learning community (see chapter 2). In particular, we focus on stating the performance guarantees for each algorithm. This is important because it provides a standard to compare all novel algorithms introduced in this thesis.

### 1.4.1 Interactive Imitation Learning

Part 1 covers the most popular category of imitation learning algorithms, interactive imitation learning to address the covariate shift issue. It consists of two chapters, chapter 3 and chapter 4. These algorithms assume you query an online expert at any time instead of learning from pre-collected demonstrations. Algorithms in this category have been applied to a wide range of applications, from quadcopter flight [236] to natural language [75] to games [235].

Chapter 3 explores an interactive setting in which an expert provides labels for pieces of the input rather than the complete input (e.g., labeling at the level of words, not sentences). We introduce an algorithm called Learning to Query for Imitation (LEAQI) to reduce online expert queries in interactive imitation learning. LEAQI assumes access to a noisy heuristic labeling function that can provide low-quality labels instead of querying the expert. There have been various ideas to minimize the number of queries to an expert [140, 149, 163, 318]. Unlike past ideas, LEAQI attempts to address this issue by combining active learning and one-sided feedback

learning idease

Chapter 4 covers a setting where interactive imitation learning algorithms work well. It is possible to compute (near-)optimal behavior at the cost of expanding computation in many contexts. This is the case in structured prediction or game settings (where one can, in principle, expand out a search- or game-tree [76, 260]) or other settings in which simulation is possible. In structured prediction problems, environments are generally known and fully deterministic. For example, consider the problem of training an agent to perform machine translation. Typically, computing the optimal next action (i.e., next word in the output translation) is possible, but the computational cost of that optimal next action will be expensive. Although interactive imitation learning is not query-efficient when the expert is human, they often perform well when the expert is a computational oracle.

We address the research questions around the sequential-generation order of natural language processing models with this computational oracle. Almost all language generation models in the NLP community operate entirely left-to-right over the output string precisely because we know how to compute the next word at training time (by copying it from the training data). This chapter asked whether we could design a computational oracle that could do more computation, thus allowing for non-monotone, non-left-to-right generation. We framed the learning problem as an interactive imitation learning problem, in which we aimed to learn a generation order without having to specify an order in advance. Because the optimal order is unknown, the oracle cannot know the correct actions and has to compute them based on the learned agent decisions.

## 1.5 Modern Imitation Learning

**Part 2** covers modern imitation learning algorithms, consisting of two chapters, **chapter 5** and **chapter 6**. These algorithms, unlike interactive IL, learn a reward function that estimates the support of the expert occupancy measure from demonstration data and optimize it with reinforcement learning (RL). Unlike interactive imitation learning, these algorithms do not assume access to an online interactive expert that can be queried. The most popular algorithm amongst these algorithms is inspired by generative adversarial networks (GAN) [107], called general adversarial imitation learning (GAIL).

**Chapter 5** covers an uncertainty-based learning modern imitation learning algorithm called Disagreement-Regularized Imitation Learning (DRIL). DRIL addresses the problem in imitation learning, where the execution of an agent in a system causes it to move to a set of states different than it was trained on (the covariate shift problem). Intuitively, DRIL is based on the idea that the agent should avoid states with little information. To achieve this, DRIL operates by training an ensemble of policies on the demonstration data and using the disagreement in their predictions as a cost. Thus, the policies in the ensemble will tend to agree on the set of states covered by the expert, leading to low cost, but are more likely to disagree on states not covered by the expert, leading to high cost. We optimize this cost function using an off-the-shelf reinforcement learning method together with a supervised cost to encourage fitting the demonstration data.

**Chapter 6** is an empirical study of modern imitation learning algorithms. The state of modern imitation learning is unclear due to conflicting results in the literature around the quality of evaluation tasks and the performance of baselines. To this end, our study evaluates all IL algorithms using continuous control, pixel, and structured prediction tasks. We find

that interleaving behavioral cloning updates with RL updates is a crucial design choice when designing IL algorithms that learn a reward function from demonstration to optimize with RL. Furthermore, we notice that most modern imitation learning algorithms take advantage of environment interactions to improve baseline algorithms. However, most baseline algorithms do not take advantage of environment interactions, making the comparison unfair, so we introduce new baselines that use environment interactions with a biased reward function and notice that they are competitive. We connect IL algorithms that we study to the most influential reinforcement learning for natural language processing algorithms.

## 1.6 Beyond Traditional Imitation Learning

**Part 3** introduces algorithms that go beyond traditional forms of imitation learning in chapters **7** and **8**. Although imitation learning algorithms work well in practice, some problems are easily solvable through other means of expert-in-the-loop. We explore two non-standard expert-in-the-loop types, constraints, and embeddings.

Chapter **7** covers the first non-standard expert-in-the-loop algorithm, which does exact imitation learning. Unlike imitation learning algorithms discussed in previous chapters, this algorithm learns a rich embedding space that encapsulates all possible rewards, policies, and states. Similar to ideas in NLP, we pretrain this embedding space with demonstration data provided by an expert. Of course, learning it is hard because this embedding space is so rich. We can find and imitate any policy in the embedding space in constant time with this embedding space.

Chapter **8** assumes that an expert can provide constraints for specific problems instead of demonstration data. For example, when training an assembly line machine robot, having safety

constraints is much easier than inferring the safety constraints from demonstration data. Or, when training an autonomous car to go around a track, providing intermediate checkpoints for the agent to monitor progress is easier for the expert than the expert providing demonstration or, worse, being queried online. In comparison to interactions, constraints allow an expert to provide a single declarative statement that generalizes across all behaviors rather than providing a potentially large number of demonstrations to achieve the same goal.

## Chapter 2: Background

This chapter introduces the fundamental Markov Decision Process (MDP), a model that allows agents proposed in this thesis to learn through interaction. We present an extension to the Markov Decision process called a Partially Observable Markov Decision Process (POMDP) and a Predictive state representation (PSR) which generalizes POMDPs. We also cover two styles of learning in an MDP: Successor Features and Imitation Learning.

### 2.1 Markov Decision Process

An MDP is a framework for modeling agent interactions in an environment. At each time step  $t$ , the agent sees an environment state  $s_t$  and takes action  $a_t$  to perform in the environment. The action updates the environment, which updates the state using the transition function  $P(\cdot|s_t, a_t)$  and determines the agent reward received. Formally, a Markov Decision process models the process of agent-environment interaction:

**Definition 1.** A Markov Decision Process can be defined by a tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \rho_0)$  in which:  $\mathcal{S}$  is the state space observed,  $\mathcal{A}$  is the action space,  $\mathcal{P}(\cdot|s, a)$  is the state transition probability  $s \in \mathcal{S}$  to any state  $s' \in \mathcal{S}$  taking action  $a \in \mathcal{A}$ ;  $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the reward function that determines the immediate reward received; and  $\rho_0 : \mathcal{S} \rightarrow [0, 1]$  is the distribution of the initial state.

We can think of a finite MDP (i.e., an MDP with a finite number of states and actions) as being represented as a set of matrices, allowing for more compressed matrix notation. Assume



there are  $k$  possible discrete states, numbered  $1 \dots k$ . The environment starts in one of these states,  $s_k$ . For each possible action  $a \in \{1 \dots \mathcal{A}\}$ , the *transition matrix*  $T_a \in \mathbb{R}^{k \times k}$  tells us how our state changes if we execute action  $a$ :  $[T_a]_{ij}$  is the probability that the next state is  $s_{t+1} = i$  if the current state is  $s_t = j$ . More compactly, we can associate each state  $1, 2, \dots, k$  with a corresponding standard basis vector  $e_1, e_2, \dots, e_k$ , and write  $q_t$  for the vector at time  $t$ . (So, if  $s_t = i$  then  $q_t = e_i$ .) Then,  $T_a q_t$  is the probability distribution over the next states:

$$P(s_{t+1} \mid q_t, \text{do } a) = \mathbb{E}(q_{t+1} \mid q_t, \text{do } a) = T_a q_t$$

Here we have written  $\text{do } a$  to indicate that choosing an action is an *intervention*.

## 2.2 Partially Observable Markov Decision Process

In an MDP, the agent knows the exact state at each time step:  $q_t$  is always a standard basis vector. By contrast, in a POMDP, the agent only receives partial information about the underlying state: at each time step, after choosing our action  $a_t$ , we see an observation  $o_t \in \{1 \dots \mathcal{O}\}$  according to a distribution that depends on the next state  $s_{t+1}$ . The *observation matrix*  $D \in \mathbb{R}^{\mathcal{O} \times k}$  tells us the probabilities:  $D_{ij}$  is the probability of receiving observation  $o_t = i$  if the next state is  $s_{t+1} = j$ . To represent this partial state information, we can let the state vector  $q_t$  range over the probability simplex instead of just the standard basis vectors:  $[q_t]_i$  tells us the probability that the state is  $s_t = i$ , given all actions and observations so far, up to and including  $a_{t-1}$  and  $o_{t-1}$ . The vector  $q_t$  is called our *belief state*; we start in belief state  $q_1$ .

As in an MDP, we have  $\mathbb{E}(q_{t+1} \mid q_t, \text{do } a) = T_a q_t$ . But now, instead of immediately resolving  $q_{t+1}$  to one of the corners of the simplex, we can only take into account partial state

information: if  $o_t = o$ , then by Bayes rule

$$\begin{aligned}
[q_{t+1}]_i &= P(s_{t+1} = i \mid q_t, \text{do } a, o) \\
&= \frac{P(o \mid s_{t+1} = i) P(s_{t+1} = i \mid q_t, \text{do } a)}{P(o \mid q_t, \text{do } a)} \\
&= D_{oi}[T_a q_t]_i / \sum_{o'} D_{o'i}[T_a q_t]_i
\end{aligned}$$

More compactly, if  $u \in \mathbb{R}^k$  is the vector of all 1s, and

$$T_{ao} = \text{diag}(D_{o,\cdot})T_a$$

where  $\text{diag}(\cdot)$  constructs a diagonal matrix from a vector, then our next belief state is

$$q_{t+1} = T_{ao}q_t / u^T T_{ao}q_t$$

### 2.2.1 PSRs

A predictive state representation (PSR) [178] further generalizes a POMDP: we can think of a PSR as dropping the interpretation of  $q_t$  as a belief state and keeping only the mathematical form of the state update. That is, we no longer require our model parameters to have any interpretation in terms of probabilities of partially observable states; we only need them to produce valid observation probability estimates. In a POMDP, the belief state is a latent variable that tracks the agent's current understanding of the environment state. In contrast, a PSR uses the agent's sequence of actions and observations to make predictions about future observations. PSRs representation can potentially be more compact than POMDPs, which is important for

planning in large domains [262].

In more detail, we are given a starting state vector  $q_1$ , matrices  $T_{ao} \in \mathbb{R}^{k \times k}$ , and a normalization vector  $u \in \mathbb{R}^k$ . We define our state vector by the recursion

$$q_{t+1} = T_{a_t o_t} q_t / u^T T_{a_t o_t} q_t$$

and our observation probabilities as

$$P(o_t = o \mid q_t, \text{do } a) = u^T T_{ao} q_t$$

The only requirement on the parameters is that the observation probabilities  $u^T T_{ao} q_t$  should always be nonnegative and sum to 1: under any sequence of actions and observations, if  $q_t$  is the resulting sequence of states,

$$(\forall a, o, t) u^T T_{ao} q_t \geq 0 \quad (\forall a, t) \sum_o u^T T_{ao} q_t = 1$$

## 2.3 Successor Features

Given an MDP, define a vector of features of the current state and action,  $f(s, a) \in \mathbb{R}^d$ ; we call this the *one-step* or *immediate* feature vector. We can define the *successor feature representation* [22, 80] as:

$$\phi^\pi(s) = \mathbb{E}_\pi \left[ \sum_{t=1}^{\infty} \gamma^{t-1} f(s_t, a_t) \mid \text{do } s_1 = s \right]$$

Where now the reward function function  $\mathcal{R}$  is linear in the *one-step* feature vector (i.e.,

$\mathcal{R}(s_t, a_t) = w^T \cdot f(s_t, a_t)$ , for some weight matrix  $w \in R^d$ ). Successor features allow us to learn a representation that decouples the reward from the transition matrix.

## 2.4 Imitation Learning

An MDP is a framework modeling sequential decisions of an agent in an environment. Most structured prediction problems can be framed in terms of the MDP framework [? ]. The “learning to search” approach to structured prediction casts the joint prediction problem of producing complex output as a sequence of smaller predictions [69, 78, 226], which means that MDPs can be viewed as a framework for modeling sequential decisions and predictions of agents in an environment. For sequential structured prediction problems, the actions are the learners’ predictions, and the states are a concatenation of all the predictions made so far. The cost is the loss of taking a particular action in a given state (i.e., cost is the opposite of reward). Formally, a Finite-Horizon Markov Decision process models the process agent-environment interaction:

**Definition 2.** A Finite-Horizon Markov Decision Process can be defined by  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C}, \rho_0, \mathcal{T})$  in which:  $\mathcal{S}$  is the state space observed,  $\mathcal{A}$  is the action space,  $\mathcal{P}(\cdot|s, a)$  is the state transition probability  $s \in \mathcal{S}$  to any state  $s' \in \mathcal{S}$  taking action  $a \in \mathcal{A}$ ;  $\mathcal{C} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the cost function determines the immediate cost received denoted as  $\mathcal{C}(s, a)$ ;  $\mathcal{T}$  is the finite horizon (max number steps) of the MDP and  $\rho_0 : \mathcal{S} \rightarrow [0, 1]$  is the distribution of the initial state.

We consider episodic finite horizon MDP in this work. Let  $\Pi$  the class of policies the learner is considering and  $\pi^*$  the expert policy whose behavior the learner is trying to mimic. For any policy  $\pi$ , let  $d_\pi$  denote the distribution over states induced by following  $\pi$ . We assume the  $\mathcal{C}(s, a)$  is bounded in  $[0, 1]$ . In the imitation learning setting, we do not necessarily know the actual costs  $\mathcal{C}(s, a)$ , and instead, we observe expert demonstrations. Our goal is to find a policy

$\pi$  which minimizes an observed surrogate loss  $\ell$  between its actions and the actions of the expert under its induced distribution of states, i.e.,

$$\hat{\pi} = \arg \min \mathbb{E}_{s \sim d_{\pi}} [\ell(\pi(s), \pi^*(s))] \quad (2.1)$$

Our goal is thus to minimize the following quantity, which represents the distance between the actions taken by our policy  $\pi$  and the expert policy  $\pi^*$ :

$$J_{\text{exp}}(\pi) = \mathbb{E}_{s \sim d_{\pi}} [||\pi(\cdot|s) - \pi^*(\cdot|s)||] \quad (2.2)$$

### 2.4.1 Behavior Cloning

Unfortunately, it is often impossible to optimize  $J_{\text{exp}}$  directly since it requires evaluating the expert policy on the states induced by following the *current* policy. The supervised behavioral cloning cost  $J_{\text{BC}}$ , which is computed on states induced by the expert, is often used instead:

$$J_{\text{BC}}(\pi) = \mathbb{E}_{s \sim d_{\pi^*}} [||\pi^*(\cdot|s) - \pi(\cdot|s)||] \quad (2.3)$$

Minimizing this loss within  $\epsilon$  yields a quadratic regret bound on regret:

**Theorem 2.4.1.** *[231] Let  $J_{\text{BC}}(\pi) = \epsilon$ , then  $J_{\text{C}}(\pi) \leq J_{\text{C}}(\pi^*) + T^2\epsilon$ .*

The drawback of the supervised learning approach (i.e., behavior cloning) is that it does not consider that the learner  $\pi$  and expert  $\pi^*$  state-distribution are different, leading to the covariate shift problem. Essentially, there is a mismatch between the distribution of states trained and tested on. In particular, during training, the learner  $\pi$  is trained on state-action pairs from the

expert policy  $\pi^*$ . But during test time, the learner is executed on state-action pairs induced by its actions, which means the learner may visit states never seen in the expert data.

### 2.4.2 DAgger

To alleviate this problem, we need to train the learner on its own state distribution to resolve the train and test time mismatch problem. The key idea of DAgger is to iteratively learn a policy by teaching the learner the optimal action to take from its own state-distribution. The tradeoff is assuming we have an interactive expert that we can query for every state. Formally, the DAgger algorithm is outlined in Algorithm 1. DAgger allows the learner policy  $\pi$  to learn from mistakes on its state-distribution because it knows what the expert would do, unlike in behavior cloning.

---

**Algorithm 1** DAgger( $\Pi, N, \langle \beta_i \rangle_{i=0}^N, \pi^*$ )

---

- 1: initialize dataset  $D = \{\}$
  - 2: initialize policy  $\hat{\pi}_1$  to any policy in  $\Pi$
  - 3: **for**  $i = 1 \dots N$  **do**
  - 4:    $\triangleright$  *stochastic mixture policy*
  - 5:   Let  $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$
  - 6:   Generate a  $T$ -step trajectory using  $\pi_i$
  - 7:   Accumulate data  $D \leftarrow D \cup \{(s, \pi^*(s))\}$  for all  $s$  in those trajectories
  - 8:   Train classifier  $\hat{\pi}_{i+1} \in \Pi$  on  $D$
  - 9: **end for**
  - 10: **return** best (or random)  $\hat{\pi}_i$
- 

Denote  $Q_t^\pi(s, a)$  as the standard Q function of policy  $\pi$ , which is defined as  $Q_t^\pi(s, a) = \mathcal{C}(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s, a)} [\mathcal{V}_{t-1}^\pi(s')]$  where  $\mathcal{V}_t^\pi(s)$  represented the cost of executing  $\pi$ . The following result for DAgger shows that if  $\ell$  is an upper bound on the 0 – 1 loss and  $C$  satisfies certain smoothness conditions, then minimizing this loss within  $\epsilon$  translates into an  $\mathcal{O}(\epsilon T)$  regret bound on the true task cost  $J_C(\pi) = \mathbb{E}_{s, a \sim d_\pi} [C(s, a)]$ :

**Theorem 2.4.2.** [236] *Let  $\pi$  be such that  $J_{\text{exp}}(\pi) = \epsilon$ , and  $Q_{T-t+1}^{\pi^*}(s, a) - Q_{T-t+1}^{\pi^*}(s, \pi^*) \leq u$  for all  $a \in \mathcal{A}, t \in \{1, 2, \dots, T\}, d_{\pi}^t(s) > 0$ . Then  $J_C(\pi) \leq J_C(\pi^*) + uT\epsilon$ .*

The theorem means if the supervised learning loss is  $\epsilon$ , then loss with respect to the expert is  $\mathcal{O}(uT\epsilon)$ , given the expert recoverability cost is bounded by  $u$ . The recoverability cost is the difference between the expert action and any action under the optimal cost of executing an action.

### 2.4.3 Roll-In Policies

The *roll-in* policy determines the state distribution over which the learned policy  $\pi$  is to be trained. In most formal analyses, the roll-in policy is a stochastic mixture of the learned policy  $\pi$  and the oracle policy  $\pi^*$ , ensuring that  $\pi$  is eventually trained on its own state distribution [54, 79, 233, 235]. Despite this, *experimentally*, it has often been found that simply using the oracle’s state distribution is optimal [167, 225]. This is likely because the noise incurred early on in learning by using  $\pi$ ’s state distribution is not overcome by the benefit of matching state distributions, especially when the policy class is sufficiently high capacity to be nearly realizable on the training data [167]. In preliminary experiments, we observed the same is true in our setting: simply rolling in according to the oracle policy (subsection 4.4.2) yielded the best results experimentally. Therefore, even though this can lead to inconsistency in the learned model [54], all experiments are with oracle roll-ins

### 2.4.4 Learning to Search

In learning-to-search-style algorithms, we aim to learn a policy  $\pi$  that mimics an oracle (or “reference”) policy  $\pi^*$ . To do so, we define a *roll-in* policy  $\pi^{\text{in}}$  and *roll-out* policy  $\pi^{\text{out}}$ . We then repeatedly draw states  $s$  according to the state distribution induced by  $\pi^{\text{in}}$ , and compute

cost-to-go under  $\pi^{\text{out}}$ , for all possible actions  $a$  at that state. The learned policy  $\pi$  is then trained to choose actions to minimize this cost-to-go estimate. Formally, denote the uniform distribution over  $\{1, \dots, T\}$  as  $U[T]$  and denote by  $d_\pi^t$  the distribution of states induced by running  $\pi$  for  $t$ -many steps. Denote by  $\mathcal{C}(\pi; \pi^{\text{out}}, s)$  a scalar cost measuring the loss incurred by  $\pi$  against the cost-to-go estimates under  $\pi^{\text{out}}$  (for instance,  $\mathcal{C}$  may measure the squared error between the vector  $\pi(\cdot|s)$  and the cost-to-go estimates). Then, the quantity being optimized is:

$$\mathbb{E}_{Y \sim D} \mathbb{E}_{t \sim U[2|Y|+1]} \mathbb{E}_{s_t \sim d_{\pi^{\text{in}}}^t} [\mathcal{C}(\pi; \pi^{\text{out}}, s_t)] \quad (2.4)$$

Here,  $\pi^{\text{in}}$  and  $\pi^{\text{out}}$  can use information unavailable at test-time (e.g., the ground-truth  $Y$ ). Learning consists of finding a policy which only has access to states  $s_t$  but performs as well or better than  $\pi^*$ . By varying the choice of  $\pi^{\text{in}}$ ,  $\pi^{\text{out}}$ , and  $\mathcal{C}$ , one obtains different variants of learning-to-search algorithms, such as DAgger [235], AggreVaTe [233] or LOLS [54].

In (interactive) imitation learning, we aim to imitate the behavior of the expert policy,  $\pi^*$ , which provides the true labels. The learning to search view allows us to cast structured prediction as a (degenerate) imitation learning task, where states are (input, prefix) pairs, actions are operations on the output, and the horizon  $T$  is the length of the sequence. States are denoted  $s \in \mathcal{S}$ , actions are denoted  $a \in [K]$ , where  $[K] = \{1, \dots, K\}$ , and the policy class is denoted  $\Pi \subseteq [K]^{\mathcal{S}}$ . The goal in learning is to find a policy  $\pi \in \Pi$  with small loss on the distribution of states that it, itself, visits.



Part Part 1

Interactive Imitation Learning

## Chapter 3: Reducing Interactive Feedback

<sup>1</sup> The primary goal of this chapter (within the broader context of the thesis) is to address issues in interactive imitation learning (see section 2.4.2). Although, interactive imitation learning algorithms provide state-of-the-art results on many structured prediction tasks [27, 78, 168, 234]. They assume that we can query an expert at every state (which is not sample efficient to an expert) and not realistic in many settings. To combat this query complexity, we consider an active learning setting in which the learning algorithm has additional access to a much cheaper *noisy heuristic* that provides noisy guidance. Our algorithm, LEAQI, learns a *difference classifier* that predicts when the expert is likely to disagree with the heuristic and queries the expert only when necessary. We apply LEAQI to three sequence labeling tasks, demonstrating significantly fewer queries to the expert and comparable (or better) accuracies over a passive approach.

### 3.1 Introduction

Structured prediction methods learn models to map inputs to complex outputs with internal dependencies, typically requiring a substantial amount of expert-labeled data. To minimize annotation cost, we focus on a setting in which an expert provides labels for *pieces* of the input, rather than the complete input (e.g., labeling at the level of words, not sentences). A natural starting point for this is imitation learning-based “learning to search” approaches to structured prediction [27, 78, 168, 234]. In imitation learning, training proceeds by incrementally producing

---

<sup>1</sup>A previous version of this work was presented in [42]

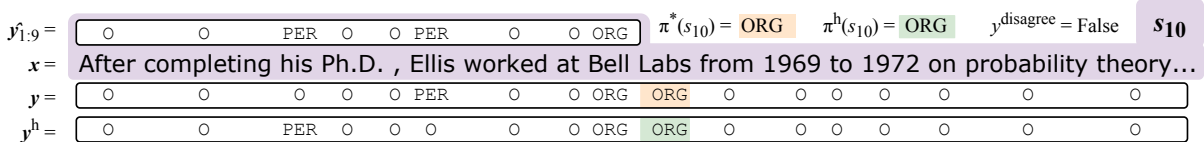


Figure 3.1: A named entity recognition example (from the Wikipedia page for [Clarence Ellis](#)).  $x$  is the input sentence and  $y$  is the (unobserved) ground truth. The predictor  $\pi$  operates left-to-right and, in this example, is currently at state  $s_{10}$  to tag the 10th word; the state  $s_{10}$  (highlighted in purple) combines  $x$  with  $\hat{y}_{1:9}$ . The heuristic makes two errors at  $t = 4$  and  $t = 6$ . The heuristic label at  $t = 10$  is  $y_{10}^h = \text{ORG}$ . Under Hamming loss, the cost at  $t = 10$  is minimized for  $a = \text{ORG}$ , which is therefore the expert action (if it were queried). The label that would be provided for  $s_{10}$  to the difference classifier is 0 because the two policies agree.

structured outputs on piece at a time and, at every step, asking the expert “what would you do here?” and learning to mimic that choice. This interactive model comes at a substantial cost: the expert demonstrator must be continuously available and must be able to answer a potentially large number of queries.

We reduce this annotation cost by only asking an expert for labels that are truly needed; our algorithm, Learning to Query for Imitation (LEAQI, /'li:tʃi:/)<sup>2</sup> achieves this by capitalizing on two factors. First, as is typical in active learning, LEAQI only asks the expert for a label when it is uncertain. Second, LEAQI assumes access to a *noisy heuristic* labeling function (for instance, a rule-based model, dictionary, or inexpert annotator) that can provide low-quality labels. LEAQI operates by always asking this heuristic for a label, and only querying the expert when it thinks the expert is likely to disagree with this label. It trains, simultaneously, a *difference classifier* [317] that predicts disagreements between the expert and the heuristic (see [Figure 3.1](#)).

The challenge in learning the difference classifier is that it must learn based on one-sided feedback: if it predicts that the expert is likely to agree with the heuristic, the expert is not queried and the classifier cannot learn that it was wrong. We address this one-sided feedback problem using the Apple Tasting framework [120], in which errors (in predicting which apples are tasty)

<sup>2</sup>Code is available at: <https://github.com/xkianteb/leaqi>

are only observed when a query is made (an apple is tasted). Learning in this way particularly important in the general case where the heuristic is likely not just to have high variance with respect to the expert, but is also statistically biased.

Experimentally (subsection 3.4.5), we consider three structured prediction settings, each using a different type of heuristic feedback. We apply LEAQI to: English named entity recognition where the heuristic is a rule-based recognizer using gazetteers [148]; English scientific keyphrase extraction, where the heuristic is an unsupervised method [90]; and Greek part-of-speech tagging, where the heuristic is a small dictionary compiled from the training data [113, 316]. In all three settings, the expert is a simulated human annotator. We train LEAQI on all three tasks using fixed BERT [82] features, training only the final layer (because we are in the regime of small labeled data). We do not consider studying improving word representations [298] or model architecture [312] which most SOTA (state-of-the-art) techniques study. Instead we leave these two areas as future to consider how does improving these two areas, improve our algorithm performance. The goal in all three settings is to minimize the number of words the expert annotator must label. In all settings, we’re able to establish the efficacy of LEAQI, showing that it can indeed provide significant label savings over using the expert alone and over several baselines and ablations that establish the importance of both the difference classifier and the Apple Tasting paradigm.

## 3.2 Background

We review online active learning, and then applications of active learning to structured prediction and imitation learning problems.

### 3.2.1 Active Learning

Active learning has been considered since at least the 1980s often under the name “selective sampling” [12, 228]. In agnostic online active learning for classification, a learner operates in rounds [e.g. 19, 31, 32]. At each round, the learning algorithm is presented an example  $x$  and must predict a label; the learner must decide whether to query the true label. An effective margin-based approach for online active learning is provided by Cesa-Bianchi et al. [52] for linear models. Their algorithm defines a sampling probability  $\rho = b/(b + z)$ , where  $z$  is the margin on the current example, and  $b > 0$  is a hyperparameter that controls the aggressiveness of sampling. With probability  $\rho$ , the algorithm requests the label and performs a perceptron-style update.

Our approach is inspired by Zhang and Chaudhuri [317] setting, where two labelers are available: a free weak labeler and an expensive strong labeler. Their algorithm minimizes queries to the strong labeler, by learning a difference classifier that predicts, for each example, whether the weak and strong labelers are likely to disagree. Their algorithm trains this difference classifier using an example-weighting strategy to ensure that its Type II error is kept small, establishing statistical consistency, and bounding its sample complexity.

This type of learning from one-sided feedback falls in the general framework of *partial-monitoring games*, a framework for sequential decision making with imperfect feedback. Apple Tasting is a type of partial-monitoring game [177], where, at each round, a learner is presented with an example  $x$  and must predict a label  $\hat{y} \in \{-1, +1\}$ . After this prediction, the true label is revealed *only* if the learner predicts  $+1$ . This framework has been applied in several settings, such as spam filtering and document classification with minority class distributions [254]. Sculley

[254] also conducts a thorough comparison of two methods that can be used to address the one-side feedback problem: label-efficient online learning [52] and margin-based learning [288].

### 3.2.2 Active Imitation & Structured Prediction

In the context of structured prediction for natural language processing, active learning has been considered both for requesting full structured outputs [e.g. 72, 110, 281] and for requesting only pieces of outputs [e.g. 37, 230]. For sequence labeling tasks, Haertel et al. [111] found that labeling effort depends both on the number of words labeled (which we model), plus a fixed cost for reading (which we do not).

In the context of imitation learning, active approaches have also been considered for at least three decades, often called “learning with an external critic” and “learning by watching” [303]. More recently, Judah et al. [141] describe *RAIL*, an active learning-for-imitation-learning algorithm akin to our *ACTIVEDAGGER* baseline, but which in principle would operate with any underlying i.i.d. active learning algorithm (not just our specific choice of uncertainty sampling).

## 3.3 Our Approach: Learning to Query for Imitation

As a concrete example, return to **Figure 3.1**: at  $s_{10}$ ,  $\pi$  must predict the label of the tenth word. If  $\pi$  is confident in its own prediction, LEAQI can avoid any query, similar to traditional active learning. If  $\pi$  is not confident, then LEAQI considers the label suggested by a noisy heuristic (here: `ORG`). LEAQI predicts whether the true expert label is likely to disagree with the noisy heuristic. Here, it predicts no disagreement and avoids querying the expert.

---

**Algorithm 2** LEAQI( $\Pi, \mathcal{H}, N, \pi^*, \pi^h, b$ )

---

```
1: initialize dataset  $D = \{\}$ 
2: initialize policy  $\pi_1$  to any policy in  $\Pi$ 
3: initialize difference dataset  $S = \{\}$ 
4: initialize difference classifier  $h_1(s) = 1 (\forall s)$ 
5: for  $i = 1 \dots N$  do
6:   Receive input sentence  $\mathbf{x}$ 
7:    $\triangleright$  generate a  $T$ -step trajectory using  $\pi_i$ 
8:   Generate output  $\hat{\mathbf{y}}$  using  $\pi_i$ 
9:   for each  $s$  in  $\hat{\mathbf{y}}$  do
10:     $\triangleright$  draw bernouilli random variable
11:     $Z_i \sim \text{Bern}\left(\frac{b}{b + \text{certainty}(\pi_i, s)}\right)$ ; see subsection 3.3.3
12:    if  $Z_i = 1$  then
13:       $\triangleright$  set difference classifier prediction
14:       $\hat{d}_i = h_i(s)$ 
15:      if AppleTaste( $s, \pi^h(s), \hat{d}_i$ ) then
16:         $\triangleright$  predict agree query heuristic
17:         $D \leftarrow D \cup \{ (s, \pi^h(s)) \}$ 
18:      else
19:         $\triangleright$  predict disagree query expert
20:         $D \leftarrow D \cup \{ (s, \pi^*(s)) \}$ 
21:         $d_i = \mathbb{1}[\pi^*(s) = \pi^h(s)]$ 
22:         $S \leftarrow S \cup \{ (s, \pi^h(s), \hat{d}_i, d_i) \}$ 
23:      end if
24:    end if
25:  end for
26:  Train policy  $\pi_{i+1} \in \Pi$  on  $D$ 
27:  Train difference classifier  $h_{i+1} \in \mathcal{H}$  on  $S$  to minimize Type II errors (see subsection 3.3.2)
28: end for
29: return best (or random)  $\pi_i$ 
```

---

### 3.3.1 Learning to Query for Imitation

Our algorithm, LEAQI, is specified in Alg 2. As input, LEAQI takes a policy class  $\Pi$ , a hypothesis class  $\mathcal{H}$  for the difference classifier (assumed to be symmetric and to contain the “constant one” function), a number of episodes  $N$ , an expert policy  $\pi^*$ , a heuristic policy  $\pi^h$ , and a confidence parameter  $b > 0$ . The general structure of LEAQI follows that of DAgger, but with three key differences:

- (a) roll-in (line 7) is according to the learned policy (not mixed with the expert, as that would require additional expert queries),
- (b) actions are queried only if the current policy is uncertain at  $s$  (line 12), and
- (c) the expert  $\pi^*$  is only queried if it is predicted to disagree with the heuristic  $\pi^h$  at  $s$  by the difference classifier, or if apple tasting method switches the difference classifier label (line 15; see subsection 3.3.2).

In particular, at each state visited by  $\pi_i$ , LEAQI estimates  $z$ , the certainty of  $\pi_i$ 's prediction at that state (see subsection 3.3.3). A sampling probability  $\rho$  is set to  $b/(b+z)$  where  $z$  is the certainty, and so if the model is very uncertain then  $\rho$  tends to zero, following [52]. With probability  $\rho$ , LEAQI will collect *some* label.

When a label is collected (line 12), the difference classifier  $h_i$  is queried on state  $s$  to predict if  $\pi^*$  and  $\pi^h$  are likely to disagree on the correct action. (Recall that  $h_1$  always predicts disagreement per line 4.) The difference classifier's prediction,  $\hat{d}_i$ , is passed to an *apple tasting* method in line 15. Intuitively, most apple tasting procedures (including the one we use, STAP; see subsection 3.3.2) return  $\hat{d}_i$ , unless the difference classifier is making many Type II errors, in which case it may return  $\neg\hat{d}_i$ .

A target action is set to  $\pi^h(s)$  if the apple tasting algorithm returns “agree” (line 17), and the expert  $\pi^*$  is only queried if disagreement is predicted (line 20). The state and target action (either heuristic or expert) are then added to the training data. Finally, if the expert was queried, then a new item is added to the difference dataset, consisting of the state, the heuristic action on that state, the difference classifier's prediction, and the ground truth for the difference classifier whose input is  $s$  and whose label is whether the expert and heuristic *actually* disagree. Finally,  $\pi_{i+1}$  is trained on the accumulated action data, and  $h_{i+1}$  is trained on the difference dataset



---

**Algorithm 3** AppleTaste\_STAP( $S, a_i^h, \hat{d}_i$ )

---

```
1:  $\triangleright$  count examples that are action  $a_i^h$ 
2: let  $t = \sum_{(\_, a, \_, \_) \in S} \mathbb{1}[a_i^h = a]$ 
3:  $\triangleright$  count mistakes made on action  $a_i^h$ 
4: let  $m = \sum_{(\_, a, \hat{d}, d) \in S} \mathbb{1}[\hat{d} \neq d \wedge a_i^h = a]$ 
5:  $w = \frac{t}{|S|}$   $\triangleright$  percentage of time  $a_i^h$  was seen
6: if  $w < 1$  then
7:    $\triangleright$  skew distribution
8:   draw  $r \sim \text{Beta}(1 - w, 1)$ 
9: else
10:  draw  $r \sim \text{Uniform}(0, 1)$ 
11: end if
12: return  $(d = 1) \wedge (r \leq \sqrt{(m + 1)/t})$ 
```

---

(details in [subsection 3.3.3](#)).

There are several things to note about LEAQI:

- ◇ If the current policy is already very certain, an expert annotator is *never* queried.
- ◇ If a label is queried, the expert is queried only if the difference classifier predicts disagreement with the heuristic, or the apple tasting procedure flips the difference classifier prediction.
- ◇ Due to apple tasting, most errors the difference classifier makes will cause it to query the expert unnecessarily; this is the “safe” type of error (increasing sample complexity but not harming accuracy), versus a Type II error (which leads to biased labels).
- ◇ The difference classifier is only trained on states where the policy is uncertain, which is exactly the distribution on which it is run.

### 3.3.2 Apple Tasting for One-Sided Learning

The difference classifier  $h \in \mathcal{H}$  must be trained (line 27) based on one-sided feedback (it only observes errors when it predicts “disagree”) to minimize Type II errors (it should only very rarely predict “agree” when the truth is “disagree”). This helps keep the labeled data for

the learned policies unbiased. The main challenge here is that the feedback to the difference classifier is *one-sided*: that is, if it predicts “disagree” then it gets to see the truth, but if it predicts “agree” it never finds out if it was wrong. We use one of [120]’s algorithms, STAP (see Alg 3), which works by random sampling from apples that are predicted to not be tasted and tasting them anyway (line 12). Formally, STAP tastes apples that are predicted to be bad with probability  $\sqrt{(m+1)/t}$ , where  $m$  is the number of mistakes, and  $t$  is the number of apples tasted so far.

We adapt Apple Tasting algorithm STAP to our setting for controlling the number of Type II errors made by the difference classifier as follows. First, because some heuristic actions are much more common than others, we run a separate apple tasting scheme *per* heuristic action (in the sense that we count the number of error *on this heuristic action* rather than globally). Second, when there is significant action imbalance<sup>3</sup> we find it necessary to skew the distribution from STAP more in favor of querying. We achieve this by sampling from a *Beta* distribution (generalizing the uniform), whose mean is shifted toward zero for more frequent heuristic actions. This increases the chance that Apple Tasting will have on finding bad apples error for each action (thereby keeping the false positive rate low for predicting disagreement).

### 3.3.3 Measuring Policy Certainty

In step 11, LEAQI must estimate the certainty of  $\pi_i$  on  $s$ . Following Cesa-Bianchi et al. [52], we implement this using a margin-based criteria. To achieve this, we consider  $\pi$  as a function that maps actions to scores and then chooses the action with largest score. The certainty

---

<sup>3</sup>For instance, in named entity recognition, both the heuristic and expert policies label the majority of words as  $\bigcirc$  (not an entity). As a result, when the heuristic says  $\bigcirc$ , it is very likely that the expert will agree. However, if we aim to optimize for something other than accuracy—like F1—it is precisely these disagreements that we need to find.

measure is then the difference in scores between the highest and second highest scoring actions:

$$\text{certainty}(\pi, s) = \max_a \pi(s, a) - \max_{a' \neq a} \pi(s, a')$$

### 3.3.4 Analysis

Theoretically, the main result for LEAQI is an interpretation of the main DAgger result(s). Formally, let  $d_\pi$  denote the distribution of states visited by  $\pi$ ,  $C(s, a) \in [0, 1]$  be the immediate cost of performing action  $a$  in state  $s$ ,  $C_\pi(s) = \mathbb{E}_{a \sim \pi(s)} C(s, a)$ , and the total expected cost of  $\pi$  to be  $J(\pi) = T \mathbb{E}_{s \sim d_\pi} C_\pi(s)$ , where  $T$  is the length of trajectories.  $C$  is not available to a learner in an imitation setting; instead the algorithm observes an expert and minimizes a surrogate loss  $\ell(s, \pi)$  (e.g.,  $\ell$  may be zero/one loss between  $\pi$  and  $\pi^*$ ). We assume  $\ell$  is strongly convex and bounded in  $[0, 1]$  over  $\Pi$ .

Given this setup assumptions, let  $\epsilon_{\text{pol-approx}} = \min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^N \mathbb{E}_{s \sim d_{\pi_i}} \ell(s, \pi)$  be the true loss of the best policy in hindsight, let  $\epsilon_{\text{dc-approx}} = \min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \mathbb{E}_{s \sim d_{\pi_i}} \text{err}(s, h, \pi^*(s) \neq \pi^h(s))$  be the true error of the best difference classifier in hindsight, and assuming that the regret of the policy learner is bounded by  $\text{reg}_{\text{pol}}(N)$  after  $N$  steps, Ross et al. [234] shows the following<sup>4</sup>:

**Theorem 3.3.1** (Thm 4.3 of Ross et al. [234]). *After  $N$  episodes each of length  $T$ , under the assumptions above, with probability at least  $1 - \delta$  there exists a policy  $\pi \in \pi_{1:N}$  such that:*

$$\mathbb{E}_{s \sim d_\pi} \ell(s, \pi) \leq \epsilon_{\text{pol-approx}} + \text{reg}_{\text{pol}}(N) + \sqrt{(2/N) \log(1/\delta)}$$

This holds regardless of how  $\pi_{1:N}$  are trained (line 26). The question of how well LEAQI

---

<sup>4</sup>Proving a stronger result is challenging: analyzing the sample complexity of an active learning algorithm that uses a difference classifier—even in the non-sequential setting—is quite involved [317].

Task	Named Entity Recognition	Keyphrase Extraction	Part of Speech Tagging
<b>Language</b>	English (en)	English (en)	Modern Greek (el)
<b>Dataset</b>	CoNLL'03 [282]	SemEval 2017 Task 10 [13]	Universal Dependencies [200]
<b># Ex</b>	14, 987	2, 809	1, 662
<b>Avg. Len</b>	14.5	26.3	25.5
<b># Actions</b>	5	2	17
<b>Metric</b>	Entity F-score	Keyphrase F-score	Per-tag accuracy
<b>Features</b>	English BERT [82]	SciBERT [25]	M-BERT [82]
<b>Heuristic</b>	String matching against an offline gazeteer of entities from Khashabi et al. [148]	Output from an unsupervised keyphrase extraction model Florescu and Caragea [90]	Dictionary from Wiktionary, similar to Zesch et al. [316] and Haghighi and Klein [113]
<b>Heur Quality</b>	P 88%, R 27%, F 41%	P 20%, R 44%, F 27%	10% coverage, 67% acc

Table 3.1: An overview of the three tasks considered in experiments.

performs becomes a question of how well the combination of uncertainty-based sampling and the difference classifier learn. So long as those do a good job on their individual *classification* tasks, DAgger guarantees that the *policy* will do a good job. This is formalized below, where  $Q^*(s, a)$  is the best possible cumulative cost (measured by  $C$ ) starting in state  $s$  and taking action  $a$ :

**Theorem 3.3.2** (Theorem 2.2 of Ross et al. [234]). *Let  $u$  be such that  $Q^*(s, a) - Q^*(s, \pi^*(s)) \leq u$  for all  $a$  and all  $s$  with  $d_\pi(s) > 0$ ; then for some  $\pi \in \pi_{1:N}$ , as  $N \rightarrow \infty$ :*

$$J(\pi) \leq J(\pi^*) + uT\epsilon_{\text{pol-approx}}$$

Here,  $u$  captures the most long-term impact a single decision can have; for example, for average Hamming loss, it is straightforward to see that  $u = \frac{1}{T}$  because any single mistake can increase the number of mistakes by at most 1. For precision, recall and F-score,  $u$  can be as large as one in the (rare) case that a single decision switches from one true positive to no true positives.

## 3.4 Experiments

The primary research questions we aim to answer experimentally are:

- Q1 Does uncertainty-based active learning achieve lower query complexity than passive learning in the learning to search settings?
- Q2 Does learning a difference classifier improve query efficiency over active learning alone?
- Q3 Does Apple Tasting successfully handle the problem of learning from one-sided feedback?
- Q4 Is the approach robust to cases where the noisy heuristic is uncorrelated with the expert?
- Q5 Is casting the heuristic as a policy more effective than using its output as features?

To answer these questions, we conduct experiments on three tasks (see [Table 6.2](#)): English named entity recognition, English scientific keyphrase extraction, and low-resource part of speech tagging on Modern Greek (el), selected as a low-resource setting.

### 3.4.1 Algorithms and Baselines

In order to address the research questions above, we compare LEAQI to several baselines.

The baselines below compare our approach to previous methods:

DAGGER. Passive DAgger ([Alg 1](#))

ACTIVEDAGGER. An active variant of DAgger that asks for labels only when uncertain. (This is equivalent to LEAQI, but with neither the difference classifier nor apple tasting.)

DAGGER+FEAT. DAGGER with the heuristic policy’s output appended as an input feature.

ACTIVEDAGGER+FEAT. ACTIVEDAGGER with the heuristic policy as a feature.

The next set of comparisons are explicit ablations:

LEAQI+NOAT LEAQI with no apple tasting.

LEAQI+NOISYHEUR. LEAQI, but where the heuristic returns a label uniformly at random.

The baselines and LEAQI share a linear relationship. DAGGER is the baseline algorithm used by all algorithms described above but it is very query inefficient with respect to an expert annotator. ACTIVEDAGGER introduces active learning to make DAGGER more query efficient; the delta to the previous addresses [Q1](#). LEAQI+NOAT introduces the difference classifier; the delta addresses [Q2](#). LEAQI adds apple tasting to deal with one-sided learning; the delta addresses [Q3](#). Finally, LEAQI+NOISYHEUR. (vs LEAQI) addresses [Q4](#) and the +FEAT variants address [Q5](#).

### 3.4.2 Data and Representation

For *named entity recognition*, we use training, validation, and test data from CoNLL’03 [\[282\]](#), consisting of IO tags instead of BIO tags (the “B” tag is almost never used in this dataset, so we never attempt to predict it) over four entity types: Person, Organization, Location, and Miscellaneous. For *part of speech tagging*, we use training and test data from modern Greek

portion of the Universal Dependencies (UD) treebanks [200], consisting of 17 universal tags<sup>5</sup>. For *keyphrase extraction*, we use training, validation, and test data from SemEval 2017 Task 10 [13], consisting of IO tags (we use one “I” tag for all three keyphrase types).

In all tasks, we implement both the policy and difference classifier by fine-tuning the last layer of a BERT embedding representation [82]. More specifically, for a sentence of length  $T$ ,  $w_1, \dots, w_T$ , we first compute BERT embeddings for each word,  $\mathbf{x}_1, \dots, \mathbf{x}_T$  using the appropriate BERT model: English BERT and M-BERT<sup>6</sup> for named entity and part-of-speech, respectively, and SciBERT [25] for keyphrase extraction. We then represent the state at position  $t$  by concatenating the word embedding at that position with a one-hot representation of the previous action:  $s_t = [\mathbf{w}_t; \text{onehot}(a_{t-1})]$ . This feature representation is used both for learning the labeling policy and also learning the difference classifier.

### 3.4.3 Expert Policy and Heuristics

In all experiments, the expert  $\pi^*$  is a simulated human annotator who annotates one word at a time. The expert returns the optimal action for the relevant evaluation metric (F-score for named entity recognition and keyphrase extraction, and accuracy for part-of-speech tagging). We take the annotation cost to be the total number of words labeled.

The heuristic we implement for named entity recognition is a high-precision gazeteer-based string matching approach. We construct this by taking a gazeteer from Wikipedia using the CogComp framework [148], and use FlashText [263] to label the dataset. This heuristic achieves a precision of 0.88, recall of 0.27 and F-score of 0.41 on the training data.

The keyphrase extraction heuristic is the output of an “unsupervised keyphrase extraction”

---

<sup>5</sup>ADJ, ADP, ADV, AUX, CCONJ, DET, INTJ, NOUN, NUM, PART, PRON, PROPN, PUNCT, SCONJ, SYM, VERB, X.

<sup>6</sup>Multilingual BERT [82]

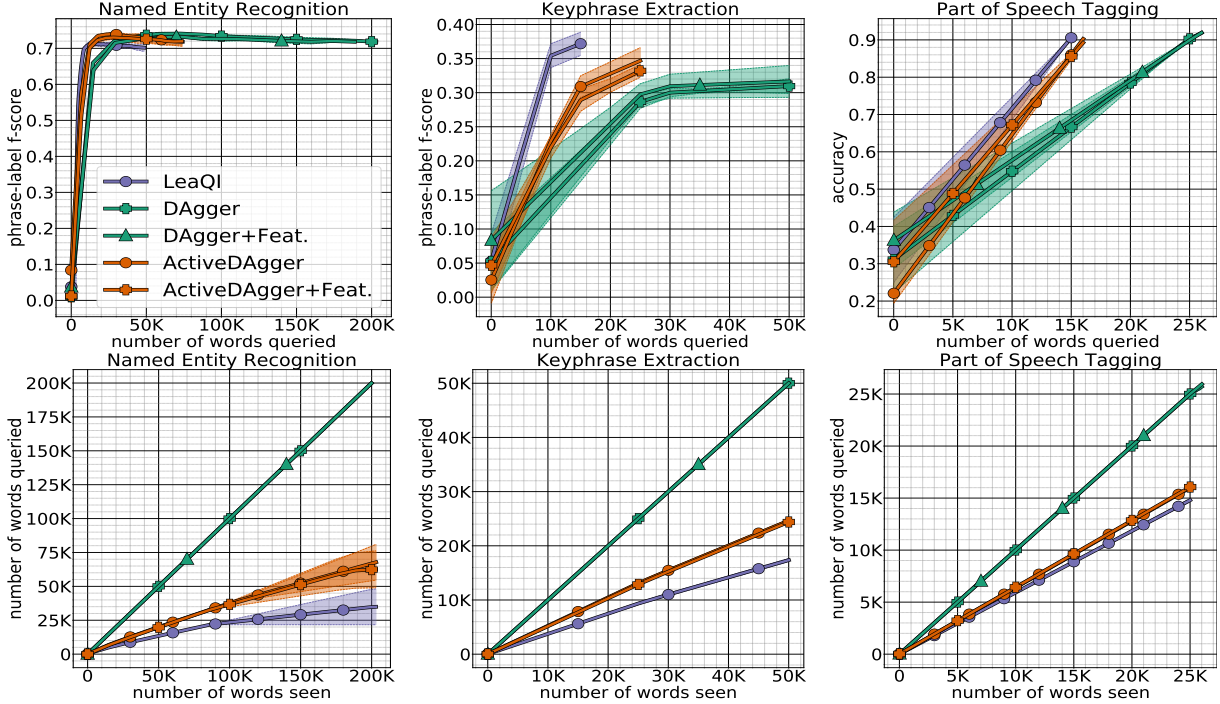


Figure 3.2: Empirical evaluation on three tasks: (left) named entity recognition, (middle) keyphrase extraction and (right) part of speech tagging. The top rows shows performance (f-score or accuracy) with respect to the number of queries to the expert. The bottom row shows the number of queries as a function of the number of words seen.

approach [90]. This system is a graph-based approach that constructs word-level graphs incorporating positions of all word occurrences information; then using PageRank to score the words and phrases. This heuristic achieves a precision of 0.20, recall of 0.44 and F-score of 0.27 on the training data.

The part of speech tagging heuristic is based on a small dictionary compiled from Wiktionary. Following Haghighi and Klein [113] and Zesch et al. [316], we extract this dictionary using Wiktionary as follows: for word  $w$  in our training data, we find the part-of-speech  $y$  by querying Wiktionary. If  $w$  is in Wiktionary, we convert the Wiktionary part of speech tag to a Universal Dependencies tag (see subsection 3.6.1), and if word  $w$  is not in Wiktionary, we use a default label of “X”. Furthermore, if word  $w$  has multiple parts of speech, we select the first part of speech tag in the list. The label “X” is chosen 90% of the time. For the remaining 10%, the



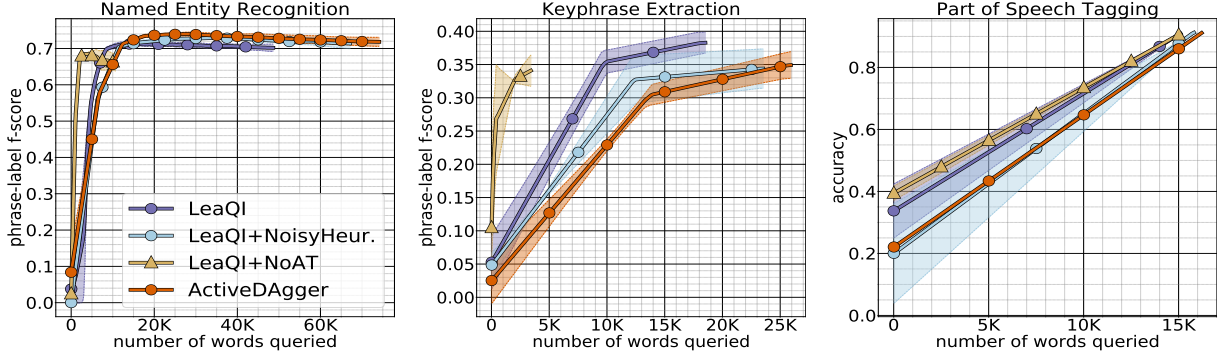


Figure 3.3: Ablation results on (left) named entity recognition, (middle) keyphrase extraction and (right) part of speech tagging. In addition to LEAQI and DAGger (copied from Figure 3.2), these graphs also show LEAQI+NOAT (apple tasting disabled), and LEAQI+NOISYHEUR. (a heuristic that produces labels uniformly at random).

heuristic achieves an accuracy of 0.67 on the training data.

### 3.4.4 Experimental Setup

Our experimental setup is online active learning. We make a single pass over a dataset, and the goal is to achieve an accurate system as quickly as possible. We measure performance (accuracy or F-score) after every 1000 words ( $\approx 50$  sentences) on heldout test data, and produce error bars by averaging across three runs and reporting standard deviations.

Hyperparameters for DAGGER are optimized using grid-search on the named entity recognition training data and evaluated on development data. We then fix DAGGER hyperparameters for all other experiments and models. The difference classifier hyperparameters are subsequently optimized in the same manner. We fix the difference classifier hyperparameters for all other experiments.<sup>7</sup>

<sup>7</sup>We note that this is a somewhat optimistic hyperparameter setting: in the real world, model selection for active learning is extremely challenging. Details on hyperparameter selection and LEAQI’s robustness across a rather wide range of choices are presented in subsection 3.6.2, subsection 3.6.3 and subsection 3.6.4 for keyphrase extraction and part of speech tagging.

### 3.4.5 Experimental Results

The main results are shown in the top two rows of [Figure 3.2](#); ablations of LEAQI are shown in [Figure 3.3](#). In [Figure 3.2](#), the top row shows traditional learning curves (performance vs number of queries), and the bottom row shows the number of queries made to the expert as a function of the total number of words seen.

**Active vs Passive (Q1).** In all cases, we see that the active strategies improve on the passive strategies; this difference is largest in keyphrase extraction, middling for part of speech tagging, and small for NER. While not surprising given previous successes of active learning, this confirms that it is also a useful approach in our setting. As expected, the active algorithms query far less than the passive approaches, and LEAQI queries the least.

**Heuristic as Features vs Policy (Q5).** We see that while adding the heuristic’s output as a feature can be modestly useful, it is not uniformly useful and, at least for keyphrase extraction and part of speech tagging, it is not as effective as LEAQI. For named entity recognition, it is not effective at all, but this is also a case where all algorithms perform essentially the same. Indeed, here, LEAQI learns quickly with few queries, but never quite reaches the performance of ActiveDagger. This is likely due to the difference classifier becoming overly confident too quickly, especially on the “O” label, given the (relatively well known) oddness in mismatch between development data and test data on this dataset.

**Difference Classifier Efficacy (Q2).** Turning to the ablations ([Figure 3.3](#)), we can address Q2 by comparing the ActiveDagger curve to the LeaQI+NoAT curve. Here, we see that on NER and keyphrase extraction, adding the difference classifier without adding apple tasting results

in a far worse model: it learns very quickly but plateaus much lower than the best results. The exception is part of speech tagging, where apple tasting does not seem necessary (but also does not hurt). Overall, this essentially shows that without controlling Type II errors, the difference classifier on it's own does not fulfill its goals.

Apple Tasting Efficacy (Q3). Also considering the ablation study, we can compare LeaQI+NoAT with LeaQI. In the case of part of speech tagging, there is little difference: using apple tasting to combat issues of learning from one sided feedback neither helps nor hurts performance. However, for both named entity recognition and keyphrase extraction, removing apple tasting leads to faster learning, but substantially lower final performance (accuracy or f-score). This is somewhat expected: without apple tasting, the training data that the policy sees is likely to be highly biased, and so it gets stuck in a low accuracy regime.

Robustness to Poor Heuristic (Q4). We compare LeaQI+NoisyHeur to ActiveDagger. Because the heuristic here is useless, the main hope is that it does not *degrade* performance below ActiveDagger. Indeed, that is what we see in all three cases: the difference classifier is able to learn quite quickly to essentially ignore the heuristic and only rely on the expert.

### 3.5 Conclusion

In this paper, we considered the problem of reducing the number of queries to an expert labeler for structured prediction problems. We took an imitation learning approach and developed an algorithm, LEAQI, which leverages a source that has low-quality labels: a heuristic policy that is suboptimal but free. To use this heuristic as a policy, we learn a difference classifier that effectively tells LEAQI when it is safe to treat the heuristic's action as if it were optimal.

We showed empirically—across Named Entity Recognition, Keyphrase Extraction and Part of Speech Tagging tasks—that the active learning approach improves significantly on passive learning, and that leveraging a difference classifier improves on that.

1. In some settings, learning a difference classifier may be as hard or harder than learning the structured predictor; for instance if the task is binary sequence labeling (e.g., word segmentation), minimizing its usefulness.
2. The true labeling cost is likely more complicated than simply the number of individual actions queried to the expert.

Despite these limitations, we hope that LEAQI provides a useful (and relatively simple) bridge that can enable using rule-based systems, heuristics, and unsupervised models as building blocks for more complex supervised learning systems. This is particularly attractive in settings where we have very strong rule-based systems, ones which often outperform the best statistical systems, like coreference resolution [170], information extraction [229], and morphological segmentation and analysis [265].

## 3.6 Extend Details

### 3.6.1 Wiktionary to Universal Dependencies

POS Tag Source	Greek, Modern (el) Wiktionary	Universal Dependencies
	adjective	ADJ
	adposition	ADP
	preposition	ADP
	adverb	ADV
	auxiliary	AU
	coordinating conjunction	CCONJ
	determiner	DET
	interjection	INTJ
	noun	NOUN
	numeral	NUM
	particle	PART
	pronoun	PRON
	proper noun	pROPN
	punctuation	PUNCT
	subordinating conjunction	SCONJ
	symbol	SYM
	verb	VERB
	other	X
	article	DET
	conjunction	PART

Table 3.2: Conversion between Greek, Modern (el) Wiktionary POS tags and Universal Dependencies POS tags.

### 3.6.2 Hyperparameters

Here we provide a table of all of hyperparameters we considered for LEAQI and baselines models. (see section 3.4.4)

Table 3.3: Hyperparameters

Hyperparameter	Values Considered	Final Value
Policy Learning rate	$10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 5.5 \cdot 10^{-6}, 10^{-6}$	$10^{-6}$
Difference Classifier Learning rate $h$	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$	$10^{-2}$
Confidence parameter (b)	$5.0 \cdot 10^{-1}, 10 \cdot 10^{-1}, 15 \cdot 10^{-1}$	$5.0 \cdot 10^{-1}$

### 3.6.3 Ablation Study Difference Classifier Learning Rate

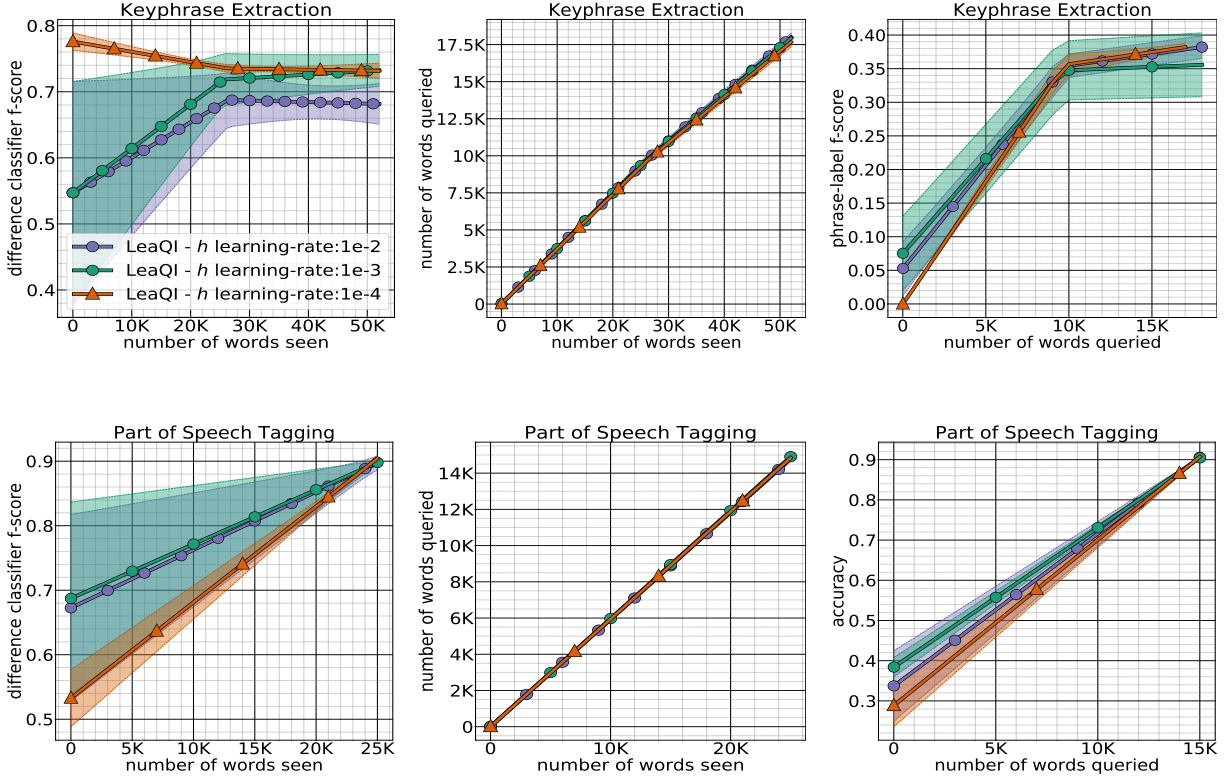


Figure 3.4: (top-row) English keyphrase extraction and (bottom-row) low-resource language part of speech tagging on Greek, Modern (el). We show the performance of using different learning rates for the difference classifier  $h$ . These plots indicate that there is a small difference in performance depending on the difference classifier learning rate.

### 3.6.4 Ablation Study Confidence Parameter: $b$

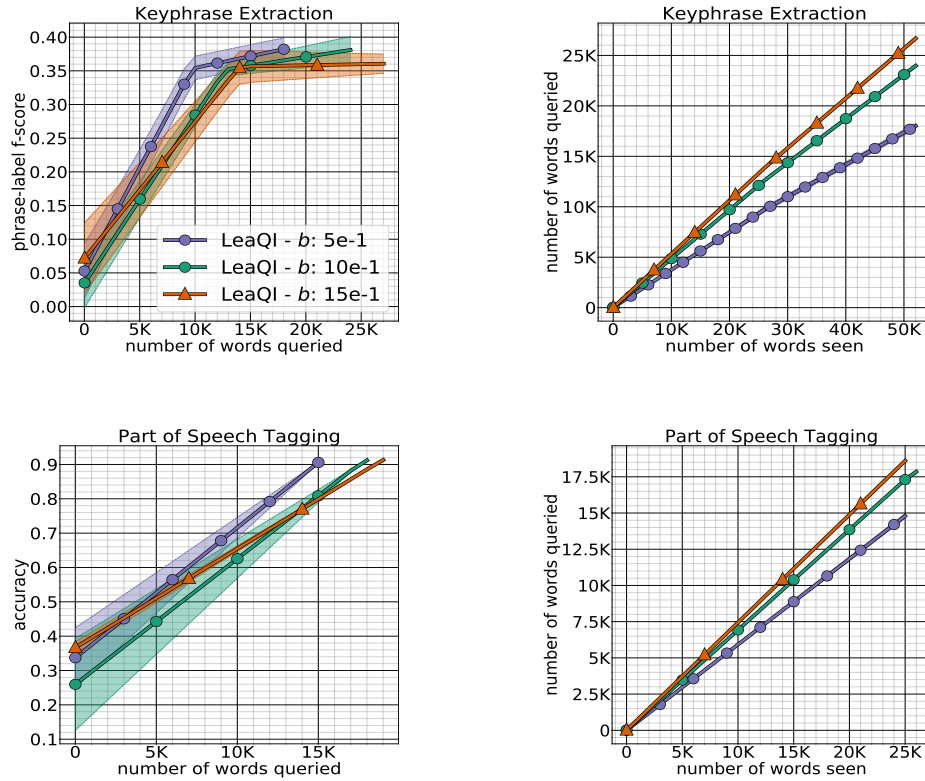


Figure 3.5: (top-row) English keyphrase extraction and (bottom-row) low-resource language part of speech tagging on Greek, Modern (el). We show the performance of using difference confidence parameters  $b$ . These plots indicate that our model is robust to difference confidence parameters.

## Chapter 4: Interactive Text Generation

Although interactive imitation learning algorithms such as DAgger (see section 2.4.2) assumes that we query an online expert for every state that an agent visit, there are settings where this assumption holds. This assumption is unrealistic in most settings, as discussed in the previous chapter (see chapter 3), except when the expert is a computational oracle. The computational oracle can learn the near-(optimal) decision at the expense of e.g., expanding a game-tree or search-tree. We use a computational oracle to study a fundamental issue in modern sequence generation models.

<sup>1</sup> In this chapter, we study the problem that standard sequential generation methods assume a pre-specified generation order, such as text generation methods that generate words from left to right. We propose a framework for training models of text generation that operate in non-monotonic orders; the model directly learns good orders without additional annotation. Our framework works by generating a word at an arbitrary position and then recursively generating words to its left and then words to its right, yielding a binary tree. The learning problem is framed as an interactive imitation learning problem using DAgger [236] (see section 2.4.2). We include computational oracles such as a coaching method that moves from imitating an oracle to reinforcing the policy’s preferences. Experimental results demonstrate that it is possible to learn policies that generate text without pre-specifying a generation order using the proposed method while achieving competitive performance with conventional left-to-right generation.

---

<sup>1</sup> A previous version of this work was presented in [299]



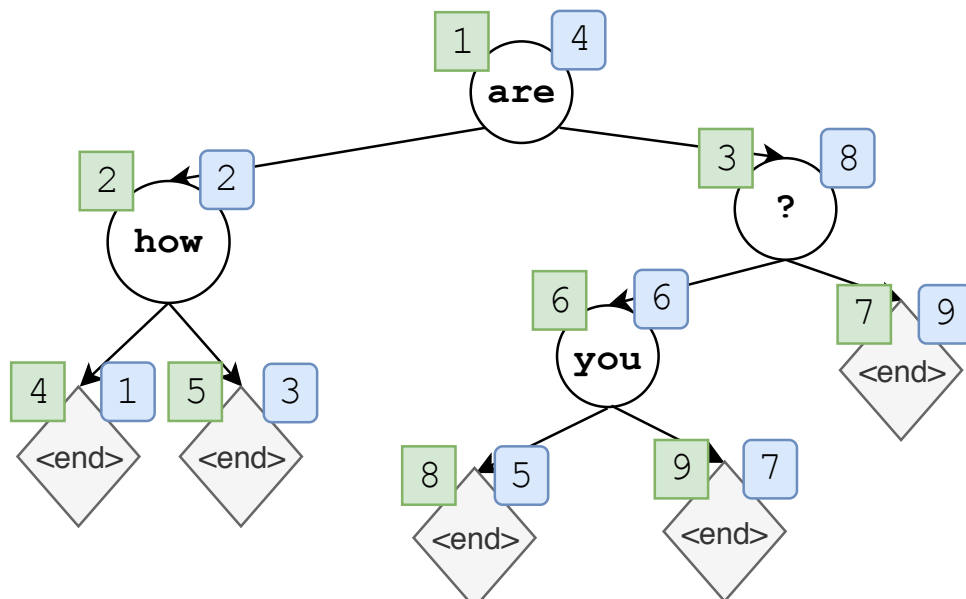


Figure 4.1: A sequence, “how are you ?”, generated by the proposed approach trained on utterances from a dialogue dataset. The model first generated the word “are” and then recursively generated left and right subtrees (“how” and “you ?”, respectively) of this word. At each production step, the model may either generate a token, or an `<end>` token, which indicates that this subtree is complete. The full generation is performed in a level-order traversal, and the output is read off from an in-order traversal. The numbers in green squares denote generation order (level-order); those in rounded blue squares denote location in the final sequence (in-order).

## 4.1 Introduction

Most sequence-generation models, from n-grams [18] to neural language models [29] generate sequences in a purely left-to-right, monotonic order. This raises the question of whether alternative, non-monotonic orders are worth considering [92], especially given the success of “easy first” techniques in natural language tagging [286], parsing [105], and coreference [271], which allow a model to effectively learn their own ordering. In investigating this question, we are solely interested in considering non-monotonic generation that does not rely on external supervision, such as parse trees [6, 86].

In this paper, we propose a framework for training sequential text generation models which learn a generation order without having to specifying an order in advance. An example generation from our model is shown in Figure 4.1. We frame the learning problem as an *imitation*

*learning* problem, in which we aim to learn a generation policy that mimics the actions of an oracle generation policy. Because the tree structure is unknown, the oracle policy cannot know the exact correct actions to take; to remedy this we propose a method called *annealed coaching* which can yield a policy with learned generation orders, by gradually moving from imitating a maximum entropy oracle to reinforcing the policy’s own preferences. Experimental results demonstrate that using the proposed framework, it is possible to learn policies which generate text without pre-specifying a generation order, achieving easy first-style behavior. The policies achieve performance metrics that are competitive with or superior to conventional left-to-right generation in language modeling, word reordering, and machine translation.<sup>2</sup>

## 4.2 Related Work

Arguably one of the most successful approaches for generating discrete sequences, or sentences, is neural autoregressive modeling [273, 284]. It has become *de facto* standard in machine translation [62, 274] and is widely studied for dialogue response generation [292] as well as speech recognition [63]. On the other hand, recent works have shown that it is possible to generate a sequence of discrete tokens in parallel by capturing strong dependencies among the tokens in a non-autoregressive way [109, 171, 201]. Stern et al. [270] and Wang et al. [295] proposed to mix in these two paradigms and build a semi-autoregressive sequence generator, while largely sticking to left-to-right generation. Our proposal radically departs from these conventional approaches by building an algorithm that automatically captures a distinct generation order.

In (neural) language modeling, there is a long tradition of modeling the probability of a

---

<sup>2</sup>Code and trained models available at [https://github.com/wellecks/nonmonotonic\\_text](https://github.com/wellecks/nonmonotonic_text).

sequence as a tree or directed graph. For example, Emami and Jelinek [85] proposed to factorize the probability over a sentence following its syntactic structure and train a neural network to model conditional distributions, which was followed more recently by Zhang et al. [324] and by Dyer et al. [84]. This approach was applied to neural machine translation by Eriguchi et al. [86] and Aharoni and Goldberg [6]. In all cases, these approaches require the availability of the ground-truth parse of a sentence or access to an external parser during training or inference time. This is unlike the proposed approach which does not require any such extra annotation or tool and learns to sequentially generate a sequence in an automatically determined non-monotonic order.

### 4.3 Background

Formally, we consider the problem of sequentially generating a sequence of discrete tokens  $Y = (w_1, \dots, w_N)$ , such as a natural language sentence, where  $w_i \in V$ , a finite vocabulary. Let  $\tilde{V} = V \cup \{\langle \text{end} \rangle\}$ .

Unlike conventional approaches with a fixed generation order, often left-to-right (or right-to-left), our goal is to build a sequence generator that generates these tokens in an order automatically determined by the sequence generator, without any extra annotation nor supervision of what might be a good order. We propose a method which does so by generating a word at an arbitrary position, then recursively generating words to its left and words to its right, yielding a binary tree like that shown in [Figure 4.1](#).

We view the generation process as deterministically navigating a state space  $\mathcal{S} = \tilde{V}^*$  where a state  $s \in \mathcal{S}$  corresponds to a sequence of tokens from  $\tilde{V}$ . We interpret this sequence of tokens as a top-down traversal of a binary tree, where  $\langle \text{end} \rangle$  terminates a subtree. The initial

state  $s_0$  is the empty sequence. For example, in [Figure 4.1](#),  $s_1 = \langle \text{are} \rangle$ ,  $s_2 = \langle \text{are, how} \rangle$ ,  $\dots$ ,  $s_4 = \langle \text{are, how, ?, } \langle \text{end} \rangle \rangle$ . An action  $a$  is an element of  $\tilde{V}$  which is deterministically appended to the state. Terminal states are those for which all subtrees have been  $\langle \text{end} \rangle$ 'ed. If a terminal state  $s_T$  is reached, we have that  $T = 2N + 1$ , where  $N$  is the number of words (non- $\langle \text{end} \rangle$  tokens) in the tree. We use  $\tau(t)$  to denote the level-order traversal index of the  $t$ -th node in an in-order traversal of a tree, so that  $\langle a_{\tau(1)}, \dots, a_{\tau(T)} \rangle$  corresponds to the sequence of discrete tokens generated. The final sequence returned is this, postprocessed by removing all  $\langle \text{end} \rangle$ 's. In [Figure 4.1](#),  $\tau$  maps from the numbers in the blue squares to those in the green squares.

A policy  $\pi$  is a (possibly) stochastic mapping from states to actions, and we denote the probability of an action  $a \in \tilde{V}$  given a state  $s$  as  $\pi(a|s)$ . A policy  $\pi$ 's behavior decides which and whether words appear before and after the token of the parent node. Typically there are many unique binary trees with an in-order traversal equal to a sequence  $Y$ . Each of these trees has a different level-order traversal, thus the policy is capable of choosing from many different generation orders for  $Y$ , rather than a single predefined order. Note that left-to-right generation can be recovered if  $\pi(\langle \text{end} \rangle | s_t) = 1$  if and only if  $t$  is odd (or non-zero and even for right-to-left generation).

## 4.4 Our Approach: Non-Monotonic Sequence Generation

Learning in our non-monotonic sequence generation model amounts to inferring a policy  $\pi$  from data. We first consider the *unconditional* generation problem (akin to language modeling) in which the data consists simply of sequences  $Y$  to be generated. Subsequently ([paragraph 4.7.1](#)) we consider the conditional case in which we wish to learn a mapping from inputs  $X$  to output sequences  $Y$ .

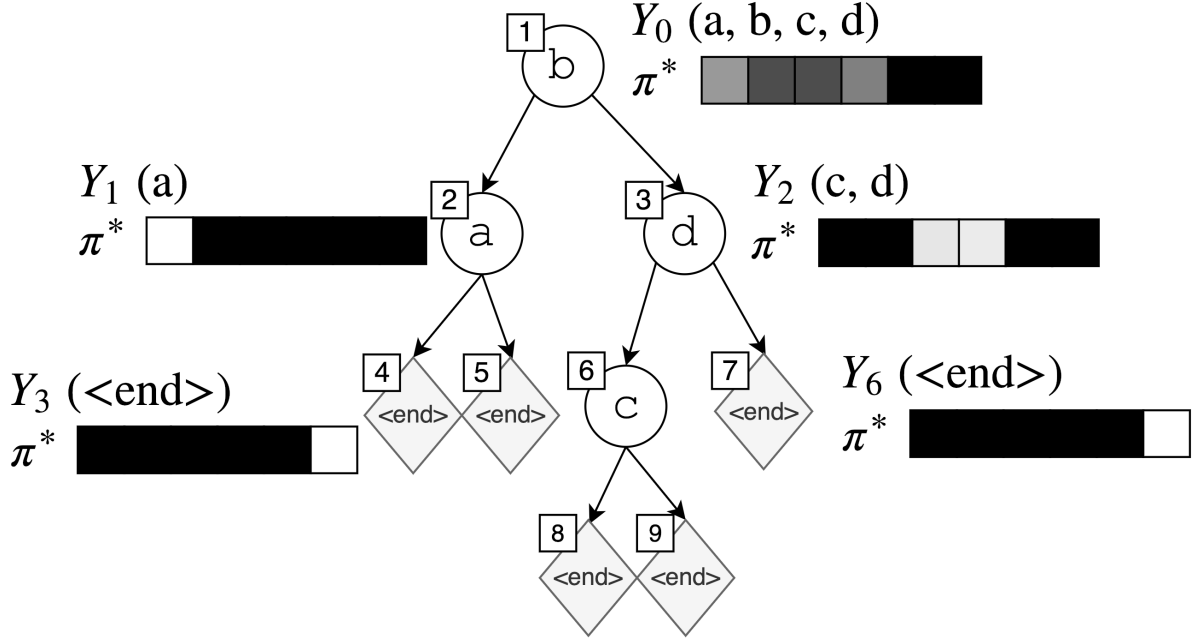


Figure 4.2: A sampled tree for the sentence “a b c d” with an action space  $\tilde{V} = (a, b, c, d, \langle \text{end} \rangle)$ , showing an oracle’s distribution  $\pi^*$  and consecutive subsequences (“valid actions”)  $Y_t$  for  $t \in \{0, 1, 2, 3, 6\}$ . Each oracle distribution is depicted as 6 boxes showing  $\pi^*(a_{t+1}|s_t)$  (lighter = higher probability). After  $b$  is sampled at the root, two empty left and right child nodes are created, associated with valid actions (a) and (c, d), respectively. Here,  $\pi^*$  only assigns positive probability to tokens in  $Y_t$ .

This learning problem is challenging because the sequences  $Y$  alone only tell us what the final output sequences of words should be, but not what tree(s) should be used to get there. In left-to-right generation, the observed sequence  $Y$  fully determines the sequence of actions to take. In our case, however, the tree structure is effectively a latent variable, which will be determined by the policy itself. This prevents us from using conventional supervised learning for training the parameterized policy. On the other hand, at training time, we do know *which words* should eventually appear, and their order; this substantially constrains the search space that needs to be explored, suggesting learning-to-search [79] and imitation learning [233, 235] as a learning strategy.<sup>3</sup>

<sup>3</sup>One could consider applying reinforcement learning to this problem. This would ignore the fact that at training time we *know* which words will appear, reducing the size of the feasible search space from  $O(|V|^T)$  to  $O(|X|^T)$ , a huge savings. Furthermore, even with a fixed generation order, RL has proven to be difficult without partially relying on supervised learning [15, 17, 225].

**The key idea** in our imitation learning framework is that at the first step, an oracle policy’s action is to produce *any* word  $w$  that appears anywhere in  $Y$ . Once picked, in a quicksort-esque manner, all words to the left of  $w$  in  $Y$  are generated recursively on the left (following the same procedure), and all words to the right of  $w$  in  $Y$  are generated recursively on the right. (See [Figure 4.2](#) for an example.) Because the oracle is *non-deterministic* (many “correct” actions are available at any given time), we inform this oracle policy with the current learned policy, encouraging it to favor actions that are preferred by the current policy, inspired by work in direct loss minimization [\[116\]](#) and related techniques [\[57, 117\]](#).

We describe the cost function we use, a set of oracle policies and a set of roll-in policies (see background section), both of which are specifically designed for the proposed problem of non-monotonic sequential generation of a sequence. These sets of policies are empirically evaluated later in the experiments.

#### 4.4.1 Cost Measurement

There are many ways to measure the prediction cost  $\mathcal{C}(\pi; \pi^{\text{out}}, s)$ ; arguably the most common is squared error between cost-predictions by  $\pi$  and observed costs obtained by  $\pi^{\text{out}}$  at the state  $s$ . However, recent work has found that, especially when dealing with recurrent neural network policies (which we will use), using a cost function more analogous to a cross-entropy loss can be preferred [\[56, 167, 302\]](#). In particular, we use a KL-divergence type loss, measuring the difference between the action distribution produced by  $\pi$  and the action distribution preferred by  $\pi^{\text{out}}$ .

$$\mathcal{C}(\pi; \pi^{\text{out}}, s) = D_{\text{KL}}(\pi^{\text{out}}(\cdot|s) \parallel \pi(\cdot|s)) \quad (4.1)$$

$$= \sum_{a \in \tilde{V}} \pi^{\text{out}}(a|s) \log \pi(a|s) + \text{const.}$$

Our approach estimates the loss in Eq. (2.4) by first sampling one training sequence, running the roll-in policy for  $t$  steps, and computing the KL divergence Eq. (4.1) at that state using  $\pi^*$  as  $\pi^{\text{out}}$ . Learning corresponds to minimizing this KL divergence iteratively with respect to the parameters of  $\pi$ .

#### 4.4.2 Oracle Policies

In this section we formalize the oracle policies that we consider. To simplify the discussion (we assume that the roll-in distribution is the oracle), we only need to define an oracle policy that takes actions on states it, itself, visits. All the oracles we consider have access to the ground truth output  $Y$ , and the current state  $s$ . We interpret the state  $s$  as a partial binary tree and a “current node” in that binary tree where the next prediction will go. It is easiest to consider the behavior of the oracle as a top-down, level-order traversal of the tree, where in each state it maintains a sequence of “possible tokens” at that state. An oracle policy  $\pi^*(\cdot|s_t)$  is defined with respect to  $Y_t$ , a consecutive subsequence of  $Y$ . At  $s_0 = \langle \rangle$ ,  $\pi^*$  uses the full  $Y_0 = Y$ . This is subdivided as the tree is descended. At each state  $s_t$ ,  $Y_t$  contains “valid actions”; labeling the current node with *any* token from  $Y_t$  keeps the generation leading to  $Y$ . For instance, in Figure 4.2, after sampling  $b$  for the root, the valid actions  $(a, b, c, d)$  are split into  $(a)$  for the left child and  $(c, d)$  for the right child.

Given the consecutive subsequence  $Y_t = (w'_1, \dots, w'_{N'})$ , an oracle policy is defined as:

$$\pi^*(a|s_t) = \begin{cases} 1 & \text{if } a = \langle \text{end} \rangle \text{ and } Y_t = \langle \rangle \\ p_a & \text{if } a \in Y_t \\ 0 & \text{otherwise} \end{cases} \quad (4.2)$$

where the  $p_a$ s are arbitrary such that  $\sum_{a \in Y} p_a = 1$ . An oracle policy places positive probability only on valid actions, and forces an  $\langle \text{end} \rangle$  output if there are no more words to produce. This is guaranteed to *always* generate  $Y$ , regardless of how the random coin flips come up.

When an action  $a$  is chosen, at  $s_t$ , this “splits” the sub-sequence  $Y_t = (w'_1, \dots, w'_{N'})$  into left and right sub-sequences,  $\overleftarrow{Y}_t = (w'_1, \dots, w'_{i-1})$  and  $\overrightarrow{Y}_t = (w'_{i+1}, \dots, w'_N)$ , where  $i$  is the index of  $a$  in  $Y_t$ . (This split may not be unique due to duplicated words in  $Y_t$ , in which case we choose a valid split arbitrarily.) These are “passed” to the left and right child nodes, respectively.

There are many possible oracle policies, and each of them is characterized by how  $p_a$  in Eq. Eq. (4.2) is defined. Specifically, we propose three variants.

**Uniform Oracle.** Motivated by Welleck et al. [302] who applied learning-to-search to the problem of multiset prediction, we design a uniform oracle  $\pi^*_{\text{uniform}}$ . This oracle treats all possible generation orders that lead to the target sequence  $Y$  as equally likely, without preferring any specific set of orders. Formally,  $\pi^*_{\text{uniform}}$  gives uniform probabilities  $p_a = 1/n$  for all words in  $Y_t$  where  $n$  is the number of unique words in  $Y_t$ . (Daumé [77] used a similar oracle for unsupervised structured prediction, which has a similar non-deterministic oracle complication.)



Coaching Oracle. An issue with the uniform oracle is that it does not prefer any specific set of generation orders, making it difficult for a parameterized policy to imitate. This gap has been noticed as a factor behind the difficulty in learning-to-search by He et al. [117], who propose the idea of coaching. In coaching, the oracle takes into account the preference of a parameterized policy in order to facilitate its learning. Motivated by this, we design a coaching oracle as the product of the uniform oracle and current policy  $\pi$ :

$$\pi_{\text{coaching}}^*(a|s) \propto \pi_{\text{uniform}}^*(a|s) \pi(a|s) \quad (4.3)$$

This coaching oracle ensures that no invalid action is assigned any probability, while preferring actions that are preferred by the current parameterized policy, reinforcing the selection by the current policy if it is valid.

Annealed Coaching Oracle. The multiplicative nature of the coaching oracle gives rise to an issue, especially in the early stage of learning, as it does not encourage learning to explore a diverse set of generation orders. We thus design a mixture of the uniform and coaching policies, which we refer to as an annealed coaching oracle:

$$\pi_{\text{annealed}}^*(a|s) = \beta \pi_{\text{uniform}}^*(a|s) + (1 - \beta) \pi_{\text{coaching}}^*(a|s) \quad (4.4)$$

We anneal  $\beta$  from 1 to 0 over learning, on a linear schedule.

Deterministic Left-to-Right Oracle. In addition to the proposed oracle policies above, we also experiment with a deterministic oracle that corresponds to generating the target sequence from left to right:  $\pi_{\text{left-right}}^*$  always selects the first un-produced word as the correct action, with

probability 1. When both roll-in and oracle policies are set to the left-to-right oracle  $\pi_{\text{left-right}}^*$ , the proposed approach recovers to maximum likelihood learning of an autoregressive sequence model, which is *de facto* standard in neural sequence modeling. In other words, supervised learning of an autoregressive sequence model is a special case of the proposed approach.

## 4.5 Experiments

In this section we experiment with our non-monotone sequence generation model across four tasks. The first two are *unconditional* generation tasks: language modeling (subsection 4.5.1) and out-of-order sentence completion (subsection 4.5.2). Our analysis in these tasks is primarily qualitative: we seek to understand what the non-monotone policy is learning and how it compares to a left-to-right model. The second two tasks are *conditional* generation tasks, which generate output sequences based on some given input sequence: word reordering (subsection 4.5.3) and machine translation (subsection 4.5.4).

### 4.5.1 Language Modeling

We begin by considering generating samples from our model, trained as a language model. Our goal in this section is to qualitatively understand what our model has learned. It would be natural also to evaluate our model according to a score like perplexity. Unfortunately, unlike conventional autoregressive language models, it is intractable to compute the probability of a given sequence in the non-monotonic generation setting, as it requires us to marginalize out all possible binary trees that lead to the sequence.

Oracle	%Novel	%Unique	Avg. Tokens	Avg. Span	BLEU
left-right	17.8	97.0	11.9	1.0	47.0
uniform	98.3	99.9	13.0	1.43	40.0
annealed	93.1	98.2	10.6	1.31	56.2
Validation	97.0	100	12.1	-	-

Table 4.1: Statistics computed over 10,000 sampled sentences (in-order traversals of sampled trees with  $\langle end \rangle$  tokens removed) for policies trained on Persona-Chat. A sample is novel when it is not in the training set. Percent unique is the cardinality of the set of sampled sentences divided by the number of sampled sentences.

**Dataset.** We use a dataset derived from the Persona-Chat [321] dialogue dataset, which consists of multi-turn dialogues between two agents. Our dataset here consists of all unique persona sentences and utterances in Persona-Chat. We derive the examples from the same train, validation, and test splits as Persona-Chat, resulting in 133,176 train, 16,181 validation, and 15,608 test examples. Sentences are tokenized by splitting on spaces and punctuation. The training set has a vocabulary size of 20,090 and an average of 12.0 tokens per example.

**Model.** We use a uni-directional LSTM that has 2 layers of 1024 LSTM units. See section 4.7.1 for more details.

**Basic Statistics.** We draw 10,000 samples from each trained policy (by varying the oracle) and analyze the results using the following metrics: percentage of novel sentences, percentage of unique, average number of tokens, average span size<sup>4</sup> and BLEU (Table 4.1). We use BLEU to quantify the sample quality by computing the BLEU score of the samples using the validation set as reference, following Yu et al. [314] and Zhu et al. [325]. In section 4.7 we report additional scores. We see that the non-monotonically trained policies generate many more novel sentences, and build trees that are bushy (span  $\sim 1.3$ ), but not complete binary trees. The policy

<sup>4</sup>The average span is the average number of children for non-leaf nodes excluding the special token  $\langle end \rangle$ , ranging from 1.0 (chain, as induced by the left-right oracle) to 2.0 (full binary tree).

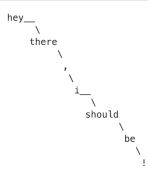
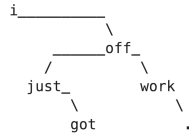
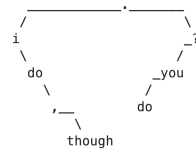
$\pi^*$ Samples	
left-right	<ul style="list-style-type: none"> <li>○ hey there , i should be !</li> <li>○ not much fun . what are you doing ?</li> <li>○ not . not sure if you .</li> <li>○ i love to always get my nails done .</li> <li>○ sure , i can see your eye underwater while riding a footwork .</li> </ul> 
uniform	<ul style="list-style-type: none"> <li>○ i just got off work .</li> <li>○ yes but believe any karma , it is .</li> <li>○ i bet you are . i read most of good tvs on that horror out . cool .</li> <li>○ sometimes , for only time i practice professional baseball .</li> <li>○ i am rich , but i am a policeman .</li> </ul> 
annealed	<ul style="list-style-type: none"> <li>○ i do , though . do you ?</li> <li>○ i like iguanas . i have a snake . i wish i could win . you ?</li> <li>○ i am a homebody .</li> <li>○ i care sometimes . i also snowboard .</li> <li>○ i am doing okay . just relaxing , and you ?</li> </ul> 

Table 4.2: Samples from unconditional generation policies trained on Persona-Chat for each training oracle. The first sample’s underlying tree is shown. Section 4.7.1 for more samples.

trained with the annealed oracle is most similar to the validation data according to BLEU.

**Content Analysis.** We investigate the content of the models in Table 4.2, which shows samples from policies trained with different oracles. Each of the displayed samples are not a part of the training set. We provide additional samples organized by length in section section 4.7 and show the underlying trees that generated them in section section 4.7. We additionally examined word frequencies and part-of-speech tag frequencies, finding that the samples from each policy typically follow the validation set’s word and tag frequencies.

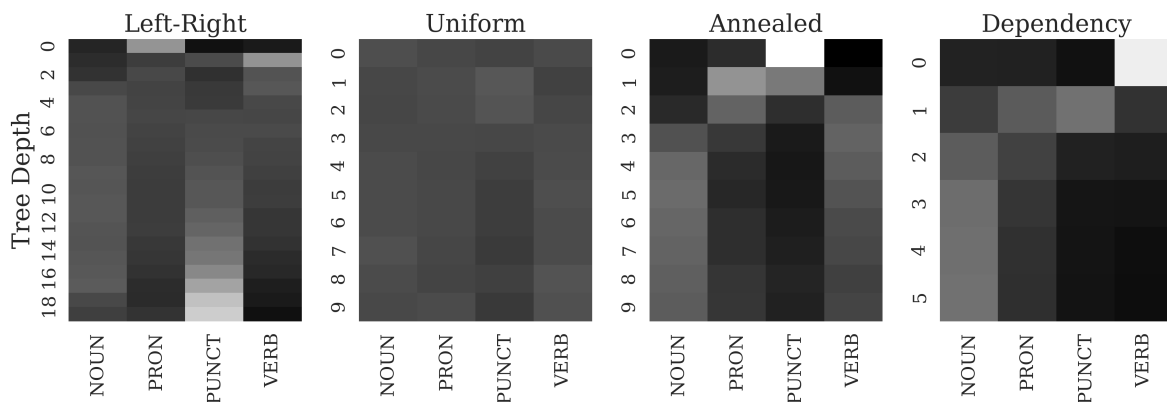


Figure 4.3: POS tag counts by tree-depth, computed by tagging 10,000 sampled sentences. Counts are normalized across each row (depth), then the marginal tag probabilities are subtracted. A light value means the probability of the tag occurring at that depth is higher than the prior probability of the tag occurring.

**Generation Order.** We analyze the generation order of our various models by inspecting the part-of-speech (POS) tags each model tends to put at different tree depths (i.e. number of edges from node to root). [Figure 4.3](#) shows POS counts by tree depth, normalized by the sum of counts at each depth (we only show the four most frequent POS categories). We also show POS counts for the validation set’s dependency trees, obtained with an off-the-shelf parser. Not surprisingly, policies trained with the uniform oracle tend to generate words with a variety of POS tags at each level. Policies trained with the annealed oracle on the other hand, learned to frequently generate punctuation at the root node, often either the sentence-final period or a comma, in an “easy first” style, since most sentences contain a period. Furthermore, we see that the policy trained with the annealed oracle tends to generate a pronoun before a noun or a verb (tree depth 1), which is a pattern that policies trained with the left-right oracle also learn. Nouns typically appear in the middle of the policy trained with the annealed oracle’s trees. Aside from verbs, the annealed policy’s trees, which place punctuation and pronouns near the root and nouns deeper, follow a similar structure as the dependency trees.

### 4.5.2 Sentence Completion

A major weakness of the conventional autoregressive model, especially with unbounded context, is that it cannot be easily used to fill in missing parts of a sentence except at the end. This is especially true when the number of tokens per missing segment is not given in advance. Achieving this requires significant changes to both model architecture, learning and inference [30].

Our proposed approach, on the other hand, can naturally fill in missing segments in a sentence. Using models trained as language models from the previous section (subsection 4.5.1), we can achieve this by initializing a binary tree with observed tokens in a way that they respect their relative positions. For instance, the first example shown in Table 4.3 can be seen as the template “\_\_\_ favorite \_\_\_ food \_\_\_ ! \_\_\_” with variable-length missing segments. Generally, an initial tree with nodes  $(w_i, \dots, w_k)$  ensures that each  $w_j$  appears in the completed sentence, and that  $w_i$  appears at *some* position to the left of  $w_j$  in the completed sentence when  $w_i$  is a left-descendant of  $w_j$  (analogously for right-descendants).

To quantify the completion quality, we first create a collection of initial trees by randomly sampling three words  $(w_i, w_j, w_k)$  from each sentence  $Y = (w_1, \dots, w_T)$  from the Persona-Chat validation set of subsection 4.5.1. We then sample one completion for each initial tree and measure the BLEU of each sample using the validation set as reference as in subsection 4.5.1. According to BLEU, the policy trained with the annealed oracle sampled completions that were more similar to the validation data (BLEU 44.7) than completions from the policies trained with the uniform (BLEU 38.9) or left-to-right (BLEU 14.3) oracles.

In Table 4.3, we present some sample completions using the policy trained with the uniform



Oracle	Validation			Test		
	BLEU	F1	EM	BLEU	F1	EM
left-right	46.6	0.910	0.230	46.3	0.903	0.208
uniform	44.7	0.968	0.209	44.3	0.960	0.197
annealed	46.8	0.960	0.230	46.0	0.950	0.212

Table 4.4: Word Reordering results on Persona-Chat, reported according to BLEU score, F1 score, and percent exact match.

**Model.** For encoding each unordered input  $x = \{w_1, \dots, w_N\}$ , we use a simple bag-of-words encoder:  $f^{\text{enc}}(\{w_1, \dots, w_N\}) = \frac{1}{T} \sum_{i=1}^N \text{emb}(w_i)$ . We implement  $\text{emb}(w_i)$  using an embedding layer followed by a linear transformation. The embedding layer is initialized with GloVe [208] vectors and updated during training. As the policy (decoder) we use a flat LSTM with 2 layers of 1024 LSTM units. The decoder hidden state is initialized with a linear transformation of  $f^{\text{enc}}(\{w_1, \dots, w_T\})$ .

**Results.** Table 4.4 shows BLEU, F1 score, and exact match for policies trained with each oracle. The uniform and annealed policies outperform the left-right policy in F1 score (0.96 and 0.95 vs. 0.903). The policy trained using the annealed oracle also matches the left-right policy’s performance in terms of BLEU score (46.0 vs. 46.3) and exact match (0.212 vs. 0.208). The model trained with the uniform policy does not fare as well on BLEU or exact match. See section 4.7.1 for example predictions.

**Easy-First Analysis.** Figure 4.4 shows the entropy of each model as a function of depth in the tree (normalized to fall in  $[0, 1]$ ). The left-right-trained policy has high entropy on the first word and then drops dramatically as additional conditioning from prior context kicks in. The uniform-trained policy exhibits similar behavior. The annealed-trained policy, however, makes its highest confidence (“easiest”) predictions at the beginning (consistent with Figure 4.3) and defers harder decisions until later.



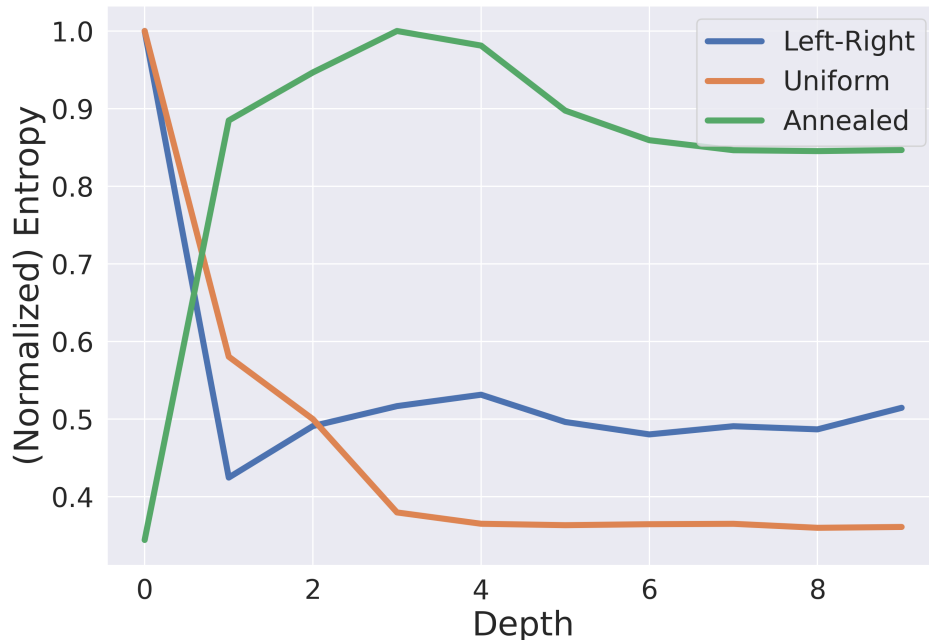


Figure 4.4: Normalized entropy of  $\pi(\cdot|s)$  as a function of tree depth for policies trained with each of the oracles. The anneal-trained policy, unlike the others, makes low entropy decisions early.

Oracle	BLEU (BP)	Validation			BLEU (BP)	Test		
		Meteor	YiSi	Ribes		Meteor	YiSi	Ribes
left-right	32.30 (0.95)	31.96	69.41	84.80	28.00 (1.00)	30.10	65.22	82.29
uniform	24.50 (0.84)	27.98	66.40	82.66	21.40 (0.86)	26.40	62.41	80.00
annealed	26.80 (0.88)	29.67	67.88	83.61	23.30 (0.91)	27.96	63.38	80.91
+tree-encoding	28.00 (0.86)	30.15	68.43	84.36	24.30 (0.91)	28.59	63.87	81.64
+⟨end⟩-tuning	29.10 (0.99)	31.00	68.81	83.51	24.60 (1.00)	29.30	64.18	80.53

Table 4.5: Results of machine translation experiments for different training oracles across four different evaluation metrics.

#### 4.5.4 Machine Translation

**Dataset.** We evaluate the proposed models on IWSLT’16 German  $\rightarrow$  English (196k pairs) translation task. The data sets consist of TED talks. We use TED tst2013 as a validation dataset and tst-2014 as test.

**Model & Training.** We use a Transformer policy, following the architecture of [289]. We use auxiliary  $\langle \text{end} \rangle$  prediction by introducing an additional output head, after observing a low

brevity penalty in preliminary experiments. For the  $\langle \text{end} \rangle$  prediction threshold  $\tau$  we use 0.5, and also report a variant (+ $\langle \text{end} \rangle$  tuning) in which  $\tau$  is tuned based on validation BLEU ( $\tau = 0.67$ ). Finally, we report a variant which embeds each token by additionally encoding its path from the root (+tree-encoding) based on [257]. See section 4.7 for additional details and results with a Bi-LSTM encoder-decoder architecture.

**Results.** Results on validation and test data are in Table 4.7 according to four (very) different evaluation measures: BLEU, Meteor [164], YiSi [181], and Ribes [133]. First focusing on the non-monotonic models, we see that the annealed policy outperforms the uniform policy on all metrics, with tree-encoding yielding further gains. Adding  $\langle \text{end} \rangle$  tuning to the tree-encoding model decreases the Ribes score but improves the other metrics, notably increasing the BLEU brevity penalty.

Compared to the best non-monotonic model, the left-to-right model has superior performance according to BLEU. As previously observed [50, 304], BLEU tends to strongly prefer models with left-to-right language models because it focuses on getting a large number of 4-grams correct. The other three measures of translation quality are significantly less sensitive to exact word order and focus more on whether the “semantics” is preserved (for varying definitions of “semantics”). For those, we see that the best annealed model is more competitive, typically within one percentage point of left-to-right.

## 4.6 Conclusion

We described an approach to generating text in non-monotonic orders that fall out naturally as the result of learning. We explored several different oracle models for imitation, and found that

an annealed “coaching” oracle performed best, and learned a “best-first” strategy for language modeling, where it appears to significantly outperform alternatives. On a word re-ordering task, we found that this approach essentially ties left-to-right decoding, a rather promising finding given the decades of work on left-to-right models. In a machine translation setting, we found that the model learns to translate in a way that tends to preserve meaning but not n-grams.

There are several potentially interesting avenues for future work. One is to solve the “learning to stop” problem directly, rather than through an after-the-fact tuning step. Another is to better understand how to construct an oracle that generalizes well after mistakes have been made, in order to train off of the gold path(s).

Moreover, the proposed formulation of sequence generation by tree generation is limited to binary trees. It is possible to extend the proposed approach to  $n$ -ary trees by designing a policy to output up to  $n + 1$  decisions at each node, leading to up to  $n$  child nodes. This would bring a set of generation orders, that could be captured by the proposed approach, which includes all projective dependency parses. A new oracle must be designed for  $n$ -ary trees, and we leave this as a follow-up work.

Finally, although the proposed approach indeed learns to sequentially generate a sequence in a non-monotonic order, it cannot consider all possible orders. It is due to the constraint that there cannot be any crossing of two edges when the nodes are arranged on a line following the inorder traversal, which we refer to as projective generation. Extending the proposed approach to non-projective generation, which we leave as future work, would expand the number of generation orders considered during learning.

## 4.7 Extend Details

### 4.7.1 Neural Net Policy Structure

We use a neural network to implement the proposed binary tree generating policy, as it has been shown to encode a variable-sized input and predict a structured output effectively [24, 46, 61, 65, 91, 274, 278]. This neural network takes as input a partial binary tree, or equivalently a sequence of nodes in this partial tree by level-order traversal, and outputs a distribution over the action set  $\tilde{V}$ .

**LSTM Policy.** The first policy we consider is implemented as a recurrent network with long short-term memory (LSTM) units [127] by considering the partial binary tree as a flat sequence of nodes in a level-order traversal  $(a_1, \dots, a_t)$ . The recurrent network encodes the sequence into a vector  $h_t$  and computes a categorical distribution over the action set:

$$\pi(a|s_t) \propto \exp(u_a^\top h_t + b_a) \quad (4.5)$$

where  $u_a$  and  $b_a$  are weights and bias associated with  $a$ .

This LSTM structure relies entirely on the linearization of a partial binary tree, and minimally takes advantage of the actual tree structure *or* the surface order. It may be possible to exploit the tree structure more thoroughly using a recurrent architecture that is designed to encode a tree [8, 39, 83, 324], which we leave for future investigation. We did experiment with additionally conditioning  $\pi$ 's action distribution on the *parent* of the current node in the tree, but preliminary experiments did not show gains.

Transformer Policy. We additionally implement a policy using a Transformer [289]. The level-order sequence  $a_1, \dots, a_t$  is again summarized by a vector  $h_t$ , here computed using a multi-head attention mechanism. As in the LSTM policy, the vector  $h_t$  is used to compute a categorical distribution over the action set Eq. (4.5).

Auxiliary  $\langle \text{end} \rangle$  Prediction. We also consider separating the action prediction into token ( $a_i \in \mathcal{V}$ ) prediction and  $\langle \text{end} \rangle$  prediction. The policy under this view consists of a categorical distribution over tokens Eq. (4.5) as well as an  $\langle \text{end} \rangle$  predictor which parameterizes a Bernoulli distribution,  $\pi_{\text{end}}(\langle \text{end} \rangle | s_t) \propto \sigma(u_e^\top h_t + b_e)$ , where  $\pi_{\text{end}}(\langle \text{end} \rangle = 1 | s_t)$  means  $a_t$  is  $\langle \text{end} \rangle$ , and  $a_t$  is determined by  $\pi$  according to Eq. (4.5) otherwise. At test time, we threshold the predicted  $\langle \text{end} \rangle$  probability at a threshold  $\tau$ . In our experiments, we only use this approach with the Transformer policy (subsection 4.5.4).

Conditional Sentence Generation An advantage of using a neural network to implement the proposed policy is that it can be easily conditioned on an extra context. It allows us to build a conditional non-monotonic sequence generator that can for instance be used for machine translation, image caption generation, speech recognition and generally multimedia description generation [59]. To do so, we assume that a conditioning input (e.g. an image or sentence)  $X$  can be represented as a set of  $d_{\text{enc}}$ -dimensional context vectors, obtained with a learned *encoder* function  $f^{\text{enc}}(X)$  whose parameters are learned jointly with the policy’s.

For word-reordering experiments subsection 4.5.3, the encoder outputs a single vector which is used to initialize the LSTM policy’s state  $h_0$ . In the machine translation experiments subsection 4.5.4, the Transformer encoder outputs  $|X|$  vectors,  $H \in \mathbb{R}^{|X| \times d_{\text{enc}}}$ , which are used as input to a decoder (i.e. policy) attention function; see [289] for further details.

### 4.7.2 Word Reordering

**Model** The decoder is a 2-layer LSTM with 1024 hidden units, dropout of 0.0, based on a preliminary grid search of  $n_{\text{layers}} \in \{1, 2\}$ ,  $n_{\text{hidden}} \in \{512, 1024, 2048\}$ ,  $\text{dropout} \in \{0.0, 0.2, 0.5\}$ . Word embeddings are initialized with GloVe vectors and updated during training. All presented Word Reordering results use greedy decoding.

**Training** Each model was trained on a single GPU using a maximum of 500 epochs, batch size of 32, Adam ] optimizer, gradient clipping with maximum  $\ell_2$ -norm of 1.0, and a learning rate starting at 0.001 and multiplied by a factor of 0.5 every 20 epochs. For evaluation we select the model state which had the highest validation BLEU score, which is evaluated after each training epoch.

**Oracle** For  $\pi_{\text{annealed}}^*$ ,  $\beta$  is linearly annealed from 1.0 to 0.0 at a rate of 0.05 each epoch, after a burn-in period of 20 epochs in which  $\beta$  is not decreased. We use greedy decoding when  $\pi_{\text{coaching}}^*$  is selected at a roll-in step; we did not observe significant performance variations with stochastically sampling from  $\pi_{\text{coaching}}^*$ . These settings are based on a grid search of  $\beta_{\text{rate}} \in \{0.01, 0.05\}$ ,  $\beta_{\text{burn-in}} \in \{0, 20\}$ ,  $\text{coaching-rollin} \in \{\text{greedy}, \text{stochastic}\}$  using the model selected in the **Model** section above.

### 4.7.3 Unconditional Generation

We use the same settings as the Word Reordering experiments, except we always use stochastic sampling from  $\pi_{\text{coaching}}^*$  during roll-in. For evaluation we select the model state at the end of training.

Table 4.6: Unconditional generation BLEU for various top- $k$  samplers and policies trained with the specified oracle.

Oracle	k	BLEU-2	BLEU-3	BLEU-4
$\pi_{\text{left-right}}^*$	10	0.905	0.778	0.624
	100	0.874	0.705	0.514
	1000	0.853	0.665	0.466
	all	0.853	0.668	0.477
$\pi_{\text{uniform}}^*$	10	0.966	0.906	0.788
	100	0.916	0.751	0.544
	1000	0.864	0.651	0.435
	all	0.831	0.609	0.395
$\pi_{\text{annealed}}^*$	10	0.966	0.895	0.770
	100	0.931	0.804	0.628
	1000	0.907	0.765	0.585
	all	0.894	0.740	0.549

Oracle	Validation				Test			
	BLEU (BP)	Meteor	YiSi	Ribes	BLEU (BP)	Meteor	YiSi	Ribes
left-right	29.47 (0.97)	29.66	52.03	82.55	26.23 (1.00)	27.87	47.58	79.85
uniform	14.97 (0.63)	21.76	41.62	77.70	13.17 (0.64)	19.87	36.48	75.36
+⟨end⟩-tuning	18.79 (0.89)	25.30	46.23	78.49	17.68 (0.96)	24.53	42.46	74.12
annealed	19.50 (0.71)	26.57	48.00	81.48	16.94 (0.72)	23.15	42.39	78.99
+⟨end⟩-tuning	21.95 (0.90)	26.74	49.01	81.77	19.19 (0.91)	25.24	43.98	79.24

Table 4.7: LSTM Policy results for machine translation experiments.

## Unconditional Samples

**Additional BLEU Scores** Since absolute BLEU scores can vary by using a softmax temperature [49] or top- $k$  sampler, we report additional scores for  $k \in \{10, 100, 1000\}$  and BLEU- $\{2, 3, 4\}$  in Table 4.6. Generally the policy trained with the annealed oracle achieves the highest metrics.

#### 4.7.4 Machine Translation

**Data and Preprocessing** We use the default Moses tokenizer script [154] and segment each word into a subword using BPE [255] creating 40k tokens for both source and target. Similar to [16], during training we filter sentence pairs that exceed 50 words.

**Transformer Policy** The Transformer policy uses 4 layers, 4 attention heads, hidden dimension 256, feed-forward dimension 1024, and is trained with batch-size 32 and a learning rate  $1e^{-5}$ . For this model and experiment, we define an epoch as 1,000 model updates. The learning rate is divided by a factor of 1.1 every 100 epochs. For  $\pi_{\text{annealed}}^*$ ,  $\beta$  is linearly annealed from 1.0 to 0.0 at a rate of 0.01 each epoch, after a burn-in period of 100 epochs. We compute metrics after each validation epoch, and following training we select the model with the highest validation BLEU.

**Loss with Auxiliary  $\langle \text{end} \rangle$  Predictor** A binary cross-entropy loss is used for the  $\langle \text{end} \rangle$  predictor for all time-steps, so that the total loss is  $\mathcal{L}_{\text{bce}}(\pi^*, \pi_{\text{end}}) + \mathcal{L}_{\text{KL}}(\pi^*, \pi)$  where  $\mathcal{L}_{\text{KL}}$  is the loss from Section 3.2. For time-steps in which  $\langle \text{end} \rangle$  is sampled,  $\mathcal{L}_{\text{KL}}$  is masked, since the policy’s token distribution is not used when  $a_t$  is  $\langle \text{end} \rangle$ .  $\mathcal{L}_{\text{KL}}$  is averaged over time by summing the loss from unmasked time-steps, then dividing by the number of unmasked time-steps.

**Tree Position Encodings** We use an additional *tree position encoding*, based on [257], which may make it easier for the policy to identify and exploit structural relationships in the partially decoded tree. Each node is encoded using its path from the root, namely a sequence of left or right steps from parent to child. Each step is represented as a 2-dimensional binary vector ( $[0, 0]$  for the root,  $[1, 0]$  for left and  $[0, 1]$  for right), so that the path is a vector  $e(a_i) \in \{0, 1\}^{2 \times \text{max-depth}}$  after zero-padding. Finally,  $e(a_i)$  is multiplied element-wise by a geometric series of a learned



parameter  $p$ , that is,  $e(a_i) \cdot [1, p, p, p^2, p^3, \dots]$ . We only use this approach with the Transformer policy.

**Additional LSTM Policy** Results are shown in [Table 4.7](#). We use a bi-directional LSTM encoder-decoder architecture that has a single layer of size 512, with global concat attention [\[183\]](#). The learning rate is initialized to 0.001 and multiplied by a factor of 0.5 on a fixed interval.

## Part Part 2

### Modern Imitation Learning

## Chapter 5: Uncertainty-Based Learning

Having shown the usefulness and addressed issues of interactive imitation learning in the previous chapters (see chapters 3 and 4), we turn to solve the covariate shift problem without an online interactive expert. There have been some preliminary results under certain assumptions, showing that this is possible [266]. However, this work does not propose a practical algorithm; instead is a theoretical observation. In contrast, proposed practical algorithms do not have any theoretical guarantees [126, 296].

<sup>1</sup> This chapter presents a simple and effective algorithm designed to address the covariate shift problem in imitation learning without interacting with an online expert. It operates by training an ensemble of policies on the expert demonstration data and using the variance of their predictions as a cost minimized with reinforcement learning and a supervised behavioral cloning cost. The uncertainty-based algorithm introduced in this chapter does not assume access to a cheap heuristic function, unlike the uncertainty-based technique, LEAQI introduced in chapter 3. We prove a regret bound for the algorithm, which is linear in the time horizon multiplied by a coefficient which we show to be low for specific problems on which behavioral cloning fails. We evaluate our algorithm empirically across multiple pixel-based Atari environments and continuous control tasks and show that it matches or significantly outperforms behavioral cloning and generative adversarial imitation learning.

---

<sup>1</sup>A previous version of this work was presented in [43]

## 5.1 Introduction

Training artificial agents to perform complex tasks is essential for many applications in robotics, video games and dialogue. If success on the task can be accurately described using a reward or cost function, reinforcement learning (RL) methods offer an approach to learning policies which has proven to be successful in a wide variety of applications [123, 175, 190, 192]. However, in other cases the desired behavior may only be roughly specified and it is unclear how to design a reward function to characterize it. For example, training a video game agent to adopt more human-like behavior using RL would require designing a reward function which characterizes behaviors as more or less human-like, which is difficult.

Imitation learning (IL) offers an elegant approach whereby agents are trained to mimic the demonstrations of an expert rather than optimizing a reward function. Its simplest form consists of training a policy to predict the expert’s actions from states in the demonstration data using supervised learning. While appealingly simple, this approach suffers from the fact that the distribution over states observed at execution time can differ from the distribution observed during training. Minor errors which initially produce small deviations become magnified as the policy encounters states further and further from its training distribution. This phenomenon, initially noted in the early work of [213], was formalized in the work of [231] who proved a quadratic  $\mathcal{O}(\epsilon T^2)$  bound on the regret and showed that this bound is tight. The subsequent work of [236] showed that if the policy is allowed to further interact with the environment and make queries to the expert policy, it is possible to obtain a *linear* bound on the regret. However, the ability to query an expert can often be a strong assumption.

In this work, we propose a new and simple algorithm called DRIL (Disagreement-

Regularized Imitation Learning) to address the covariate shift problem in imitation learning, in the setting where the agent is allowed to interact with its environment. Importantly, the algorithm does not require any additional interaction with the expert. It operates by training an ensemble of policies on the demonstration data, and using the disagreement in their predictions as a cost which is optimized through RL together with a supervised behavioral cloning cost. The motivation is that the policies in the ensemble will tend to agree on the set of states covered by the expert, leading to low cost, but are more likely to disagree on states not covered by the expert, leading to high cost. The RL cost thus guides the agent back towards the distribution of the expert, while the supervised cost ensures that it mimics the expert within the expert’s distribution.

Our theoretical results show that, subject to realizability and optimization oracle assumptions<sup>2</sup>, our algorithm obtains a  $\mathcal{O}(\epsilon\kappa T)$  regret bound, where  $\kappa$  is a measure which quantifies a tradeoff between the concentration of the demonstration data and the diversity of the ensemble outside the demonstration data. We evaluate DRIL empirically across multiple pixel-based Atari environments and continuous control tasks, and show that it matches or significantly outperforms behavioral cloning and generative adversarial imitation learning, often recovering expert performance with only a few trajectories.

## 5.2 Related Work

Imitation learning has been used within the context of modern RL to help improve sample efficiency [54, 55, 124, 165, 233, 272] or overcome exploration [198]. These settings assume the reward is known and that the policies can then be fine-tuned with reinforcement learning. In this

---

<sup>2</sup>We assume for the analysis the action space is discrete, but the state space can be exponentially large or infinite.

case, covariate shift is less of an issue since it can be corrected using the reinforcement signal.

The work of [182] also proposed a method to address the covariate shift problem when learning from demonstrations when the reward is known, by conservatively extrapolating the value function outside the training distribution using negative sampling. This addresses a different setting from ours, and requires generating plausible states which are off the manifold of training data, which may be challenging when the states are high dimensional such as images. The work of [227] proposed to treat imitation learning within the Q-learning framework, setting a positive reward for all transitions inside the demonstration data and zero reward for all other transitions in the replay buffer. This rewards the agent for repeating (or returning to) the expert’s transitions. The work of [242] also incorporates a mechanism for reducing covariate shift by fitting a Q-function that classifies whether the demonstration states are reachable from the current state. Random Expert Distillation [296] uses Random Network Distillation (RND) [48] to estimate the support of the expert’s distribution in state-action space, and minimizes an RL cost designed to guide the agent towards the expert’s support. This is related to our method, but differs in that it minimizes the RND prediction error rather than the ensemble variance and does not include a behavior cloning cost. The behavior cloning cost is essential to our theoretical results and avoids certain failure modes, see section 5.7.2 for more discussion.

Generative Adversarial Imitation Learning (GAIL) [126] is a state-of-the-art algorithm which addresses the same setting as ours. It operates by training a discriminator network to distinguish expert states from states generated by the current policy, and the negative output of the discriminator is used as a reward signal to train the policy. The motivation is that states which are outside the training distribution will be assigned a low reward while states which are close to it will be assigned a high reward. This encourages the policy to return to the expert

distribution if it strays away from it. However, the adversarial training procedure means that the reward function is changing over time, which can make the algorithm unstable or difficult to tune. In contrast, our approach uses a simple fixed reward function. We include comparisons to GAIL in our experiments.

Using disagreement between models in an ensemble to represent uncertainty has recently been explored in several contexts. The works of [121, 206, 258] used disagreement between different dynamics models to drive exploration in the context of model-based RL. Conversely, [122] used variance across different dropout masks to prevent policies from exploiting error in dynamics models. Ensembles have also been used to represent uncertainty over Q-values in model-free RL in order to encourage exploration [204]. Within the context of imitation learning, the work of [189] used the variance of the ensemble together with the DAGGER algorithm to decide when to query the expert demonstrator to minimize unsafe situations. Here, we use disagreement between different policies trained on demonstration data to address covariate shift in the context of imitation learning.

### 5.3 Our Approach: Disagreement-Regularized Imitation Learning

Our algorithm is motivated by two criteria: i) the policy should act similarly to the expert within the expert’s data distribution, and ii) the policy should move towards the expert’s data distribution if it is outside of it. These two criteria are addressed by combining two losses: a standard behavior cloning loss, and an additional loss which represents the variance over the outputs of an ensemble  $\Pi_E = \{\pi_1, \dots, \pi_E\}$  of policies trained on the demonstration data  $\mathcal{D}$ . We call this the uncertainty cost, which is defined as:

---

**Algorithm 4** Disagreement-Regularized Imitation Learning (DRIL)

---

- 1: **Input:** Expert demonstration data  $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^N$
  - 2: Initialize policy  $\pi$  and policy ensemble  $E = \{\pi_e\}$
  - 3: **for**  $e = 1, E$  **do**
  - 4:   Sample  $\mathcal{D}_e \sim \mathcal{D}$  with replacement, with  $|\mathcal{D}_e| = |\mathcal{D}|$ .
  - 5:   Train  $\pi_e$  to minimize  $J_{\text{BC}}(\pi_e)$  on  $\mathcal{D}_e$  to convergence.
  - 6: **end for**
  - 7: **for**  $i = 1, \dots$  **do**
  - 8:   Perform one gradient update to minimize  $J_{\text{BC}}(\pi)$  using a minibatch from  $\mathcal{D}$ .
  - 9:   Perform one step of policy gradient to minimize  $\mathbb{E}_{s \sim d_\pi, a \sim \pi(\cdot|s)}[C_{\text{U}}^{\text{clip}}(s, a)]$ .
  - 10: **end for**
- 

$$C_{\text{U}}(s, a) = \text{Var}_{\pi \sim \Pi_{\text{E}}}(\pi(a|s)) = \sum_{i=1}^{|E|} \left( \pi_i(a|s) - \sum_{i=1}^{|E|} \pi_i(a|s) / |E| \right)^2$$

The motivation is that the variance over plausible policies is high outside the expert's distribution, since the data is sparse, but it is low inside the expert's distribution, since the data there is dense. Minimizing this cost encourages the policy to return to regions of dense coverage by the expert. Intuitively, this is what we would expect the expert policy  $\pi^*$  to do as well. The total cost which the algorithm optimizes is given by:

$$J_{\text{alg}}(\pi) = \underbrace{\mathbb{E}_{s \sim d_{\pi^*}}[||\pi^*(\cdot|s) - \pi(\cdot|s)||]}_{J_{\text{BC}}(\pi)} + \underbrace{\mathbb{E}_{s \sim d_\pi, a \sim \pi(\cdot|s)}[C_{\text{U}}(s, a)]}_{J_{\text{U}}(\pi)}$$

The first term is a behavior cloning loss and is computed over states generated by the expert policy, of which the demonstration data  $\mathcal{D}$  is a representative sample. The second term is computed over the distribution of states generated by the current policy and can be optimized using policy gradient.



Note that the demonstration data is fixed, and this ensemble can be trained once offline. We then interleave the supervised behavioral cloning updates and the policy gradient updates which minimize the variance of the ensemble. The full algorithm is shown in Algorithm 7. Dropout [268] has been proposed as an approximate form of ensembling. We also experimented with using dropout (see section 5.7.4.2) and found that it also worked well.

In practice, for the supervised loss we optimize the KL divergence between the actions predicted by the policy and the expert actions, which is an upper bound on the total variation distance due to Pinsker’s inequality. We also found it helpful to use a clipped uncertainty cost:

$$C_U^{\text{clip}}(s, a) = \begin{cases} -1 & \text{if } C_U(s, a) \leq q \\ +1 & \text{else} \end{cases}$$

where the threshold  $q$  is a top quantile of the raw uncertainty costs computed over the demonstration data. The threshold  $q$  defines a normal range of uncertainty based on the demonstration data, and values above this range incur a positive cost (or negative reward).

The RL cost can be optimized using any policy gradient method. In our experiments we used advantage actor-critic (A2C) [190], which estimates the expected cost using rollouts from multiple parallel actors all sharing the same policy (see section 5.7.3 for details). We note that model-based RL methods could in principle be used as well if sample efficiency is a constraint.

## 5.4 Analysis

### 5.4.1 Coverage Coefficient

We now analyze DRIL for MDPs with discrete action spaces and potentially large or infinite state spaces. We will show that, subject to assumptions that the policy class contains an optimal policy and that we are able to optimize costs within  $\epsilon$  of their global minimum, our algorithm obtains a regret bound which is linear in  $\kappa T$ , where  $\kappa$  is a quantity which depends on the environment dynamics,  $d_\pi^\star$  and our learned ensemble. Intuitively,  $\kappa$  represents a tradeoff between how concentrated the demonstration data is and how high the variance of the ensemble is outside the expert distribution.

**Assumption 1.** (*Realizability*)  $\pi^\star \in \Pi$ .

**Assumption 2.** (*Optimization Oracle*) For any given cost function  $J$ , our minimization procedure returns a policy  $\hat{\pi} \in \Pi$  such that  $J(\hat{\pi}) \leq \operatorname{argmin}_{\pi \in \Pi} J(\pi) + \epsilon$ .

The motivation behind our algorithm is that the policies in the ensemble agree inside the expert's distribution and disagree outside of it. This defines a reward function which pushes the learner back towards the expert's distribution if it strays away. However, what constitutes inside and outside the distribution, or sufficient agreement or disagreement, is ambiguous. Below we introduce quantities which makes these ideas precise.

**Definition 3.** For any set  $\mathcal{U} \subseteq \mathcal{S}$ , define the concentrability inside of  $\mathcal{U}$  as

$$\alpha(\mathcal{U}) = \max_{\pi \in \Pi} \sup_{s \in \mathcal{U}} \frac{d_\pi(s)}{d_{\pi^\star}(s)}.$$

The notion of concentrability has been previously used to give bounds on the performance of value iteration [196]. For a set  $\mathcal{U}$ ,  $\alpha(\mathcal{U})$  will be low if the expert distribution has high mass at the states in  $\mathcal{U}$  that are reachable by policies in the policy class.

**Definition 4.** Define the minimum variance of the ensemble outside of  $\mathcal{U}$  as

$$\beta(\mathcal{U}) = \min_{s \notin \mathcal{U}, a \in \mathcal{A}} \text{Var}_{\pi \sim \Pi_E} [\pi(a|s)].$$

We now define the  $\kappa$  coefficient as the minimum ratio of these two quantities over all possible subsets of  $\mathcal{S}$ .

**Definition 5.** We define  $\kappa = \min_{\mathcal{U} \subseteq \mathcal{S}} \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})}$ .

We can view  $\kappa$  as the quantity which minimizes the tradeoff over different subsets  $\mathcal{U}$  between coverage by the expert policy inside of  $\mathcal{U}$ , and variance of the ensemble outside of  $\mathcal{U}$ . Note that by making  $\mathcal{U}$  very small, it may be easy to make  $\alpha(\mathcal{U})$  small, but doing so may also make  $\beta(\mathcal{U})$  small and  $\kappa(\mathcal{U})$  large. Conversely, making  $\mathcal{U}$  large may make  $\beta(\mathcal{U})$  large but may also make  $\alpha(\mathcal{U})$  large as a result.

### 5.4.2 Regret Bound

We now establish a relationship between the  $\kappa$  coefficient just defined, the cost our algorithm optimizes, and  $J_{\text{exp}}$  defined in Equation (2) which we would ideally like to minimize and which translates into a regret bound. All proofs can be found in section 5.7.1.

**Lemma 1.** For any  $\pi \in \Pi$ , we have  $J_{\text{exp}}(\pi) \leq \kappa J_{\text{alg}}(\pi)$ .

This result shows that if  $\kappa$  is not too large, and we are able to make our cost function  $J_{\text{alg}}(\pi)$  small, then we can ensure  $J_{\text{exp}}(\pi)$  is also be small. This result is only useful if our cost function can indeed achieve a small minimum. The next lemma shows that this is the case.

**Lemma 2.**  $\min_{\pi \in \Pi} J_{\text{alg}}(\pi) \leq 4\epsilon$ .

Here  $\epsilon$  is the threshold specified in Assumption 2. Combining these two lemmas with the previous result of [236], we get a regret bound which is linear in  $\kappa T$ .

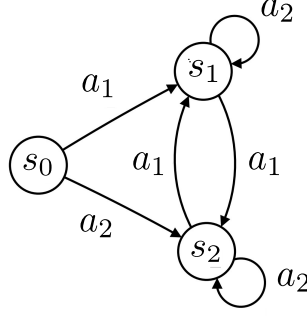


Figure 5.1: Example of a problem where behavioral cloning incurs quadratic regret.

**Theorem 5.4.1.** *Let  $\hat{\pi}$  be the result of minimizing  $J_{\text{alg}}$  using our optimization oracle, and assume that  $Q_{T-t+1}^{\pi^*}(s, a) - Q_{T-t+1}^{\pi^*}(s, \pi^*) \leq u$  for all  $a \in \mathcal{A}, t \in \{1, 2, \dots, T\}, d_{\pi}^t(s) > 0$ . Then  $\hat{\pi}$  satisfies  $J_C(\hat{\pi}) \leq J_C(\pi^*) + 5u\kappa\epsilon T$ .*

Our bound is an improvement over that of behavior cloning if  $\kappa$  is less than  $\mathcal{O}(T)$ . Note that DRIL does not require knowledge of  $\kappa$ . The quantity  $\kappa$  is problem-dependent and depends on the environment dynamics, the expert policy and the class of policies available to the learner. We next compute  $\kappa$  exactly for a problem for which behavior cloning is known to perform poorly, and show that it is independent of  $T$ .

**Example 1.** *Consider the tabular MDP given in [231] as an example of a problem where behavioral cloning incurs quadratic regret, shown in Figure 5.1. There are 3 states  $\mathcal{S} = (s_0, s_1, s_2)$  and two actions  $(a_1, a_2)$ . Each policy  $\pi$  can be represented as a set of probabilities  $\pi(a_1|s)$  for each state  $s \in \mathcal{S}$ <sup>3</sup>. Assume the models in our ensemble are drawn from a posterior  $p(\pi(a_1|s)|\mathcal{D})$  given by a Beta distribution with parameters  $\text{Beta}(n_1 + 1, n_2 + 1)$  where  $n_1, n_2$  are the number of times the pairs  $(s, a_1)$  and  $(s, a_2)$  occur, respectively, in the demonstration data  $\mathcal{D}$ . The agent always starts in  $s_0$  and the expert's policy is given by  $\pi^*(a_1|s_0) = 1, \pi^*(a_1|s_1) = 0, \pi^*(a_1|s_2) = 1$ . Here  $d_{\pi}^* = (\frac{1}{T}, \frac{T-1}{T}, 0)$ . For any  $\pi$ ,  $d_{\pi}(s_0) = \frac{1}{T}$  and  $d_{\pi}(s_1) \leq \frac{T-1}{T}$  due*

<sup>3</sup>Note that  $\pi(a_2|s) = 1 - \pi(a_1|s)$ .

to the dynamics of the MDP, so  $\frac{d_\pi(s)}{d_\pi^*(s)} \leq 1$  for  $s \in \{s_0, s_1\}$ . Writing out  $\alpha(\{s_0, s_1\})$ , we get:

$$\alpha(\{s_0, s_1\}) = \max_{\pi \in \Pi} \sup_{s \in \{s_0, s_1\}} \frac{d_\pi(s)}{d_\pi^*(s)} \leq 1.$$

Furthermore, since  $s_2$  is never visited in the demonstration data, for each policy  $\pi_i$  in the ensemble we have  $\pi_i(a_1|s_2), \pi_i(a_2|s_2) \sim \text{Beta}(1, 1) = \text{Uniform}(0, 1)$ . It follows that  $\text{Var}_{\pi \sim \Pi_E}(\pi(a|s_2))$  is approximately equal <sup>4</sup> to the variance of a uniform distribution over  $[0, 1]$ , i.e.  $\frac{1}{12}$ . Therefore:

$$\kappa = \min_{\mathcal{U} \subseteq \mathcal{S}} \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})} \leq \frac{\alpha(\{s_0, s_1\})}{\beta(\{s_0, s_1\})} \lesssim \frac{1}{\frac{1}{12}} = 12$$

Applying our result from Theorem 5.4.1, we see that our algorithm obtains an  $\mathcal{O}(\epsilon T)$  regret bound on this problem, in contrast to the  $\mathcal{O}(\epsilon T^2)$  regret of behavioral cloning<sup>5</sup>.

## 5.5 Experiments

### 5.5.1 Tabular MDPs

As a first experiment, we applied DRIL to the tabular MDP of [231] shown in Figure 5.1. We computed the posterior over the policy parameters given the demonstration data using a separate Beta distribution for each state  $s$  with parameters determined by the number of times each action was performed in  $s$ . For behavior cloning, we sampled a single policy from this posterior. For DRIL, we sampled an ensemble of 5 policies and used their negative variance to

<sup>4</sup>Via Hoeffding's inequality, with probability  $1 - \delta$  the two differ by at most  $\mathcal{O}(\sqrt{\log(1/\delta)/|\Pi_E|})$ .

<sup>5</sup>Observe that a policy with  $\pi(a_1|s_0) = 1 - \epsilon T$ ,  $\pi(a_2|s_1) = \epsilon T$ ,  $\pi(a_2|s_2) = 1$  has a behavioral cloning cost of  $\epsilon$  but a regret of  $\epsilon T^2$ .

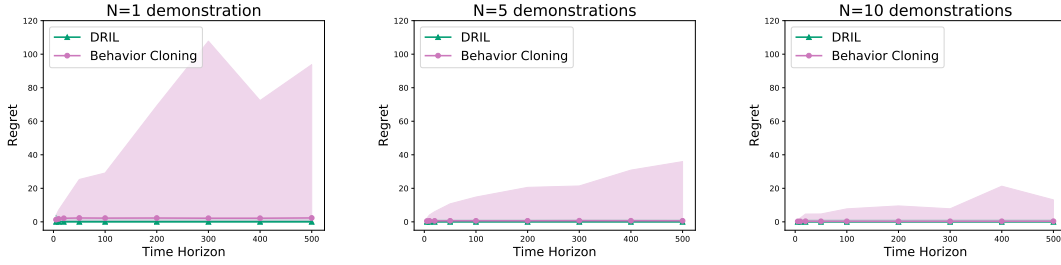


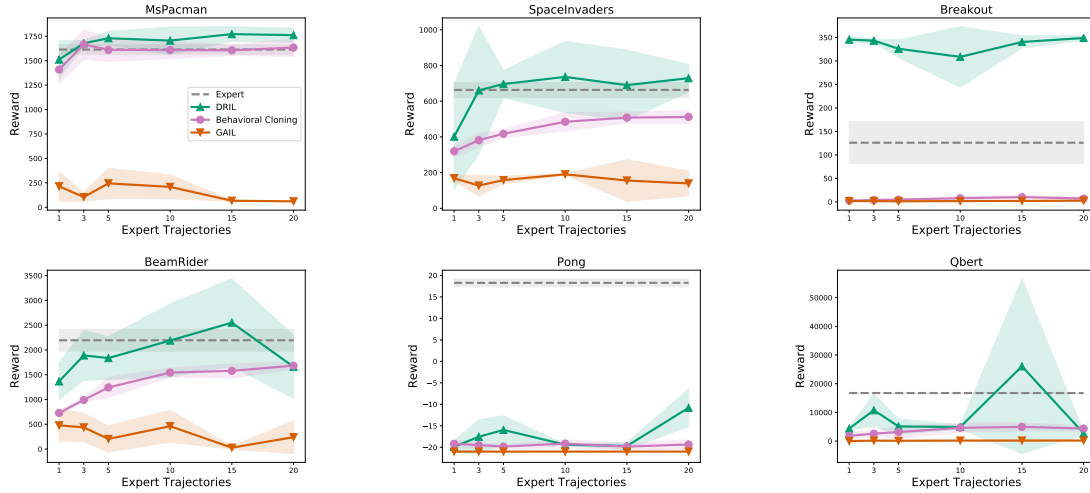
Figure 5.2: Results on tabular MDP from [231]. Shaded region represents range between 5<sup>th</sup> and 95<sup>th</sup> quantiles, computed across 500 trials. Behavior cloning exhibits poor worst-case regret, whereas DRIL has low regret across all trials.

define an additional reward function. We combined this with a reward which was the probability density function of a given state-action pair under the posterior distribution, which corresponds to the supervised learning loss, and used tabular Q-learning to optimize the sum of these two reward functions. This experiment was repeated 500 times for time horizon lengths up to 500 and  $N = 1, 5, 10$  expert demonstration trajectories.

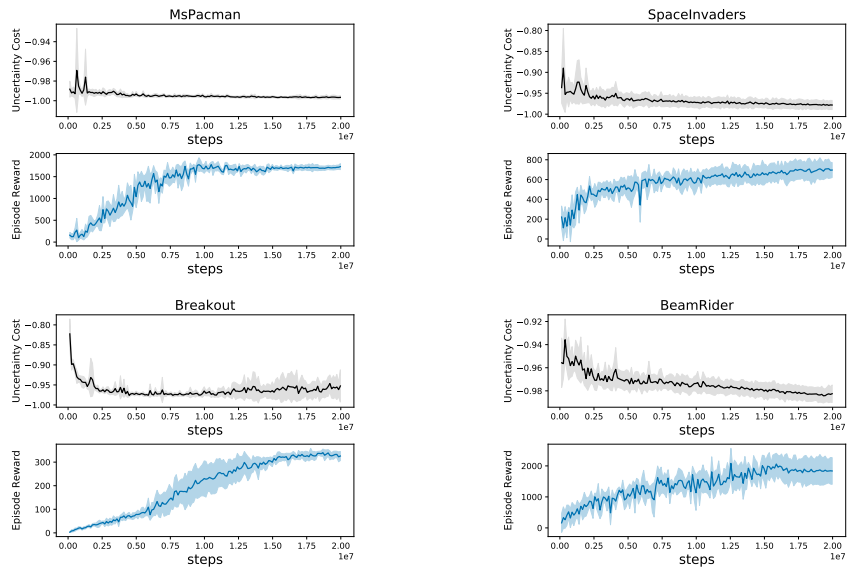
Figure 5.2 shows plots of the regret over the 500 different trials across different time horizons. Although the average performance of BC improves with more expert demonstrations, it exhibits poor worst-case performance with some trials incurring very high regret, especially when using fewer demonstrations. Our method has low regret across all trials, which stays close to constant independantly of the time horizon, even with a single demonstration. This performance is better than that suggested by our analysis, which showed a worst-case linear bound with respect to time horizon.

### 5.5.2 Atari Environments

We next evaluated our approach on six different Atari environments. We used pretrained PPO [251] agents from the stable baselines repository [125] to generate  $N = \{1, 3, 5, 10, 15, 20\}$  expert trajectories. We compared against two other methods: standard behavioral cloning (BC)



a)



b)

Figure 5.3: Results on Atari environments. a) Final policy performance for different numbers of expert trajectories. b) Evolution of policy reward and uncertainty cost during training with  $N = 5$  trajectories.

and Generative Adversarial Imitation Learning (GAIL). Results are shown in Figure 5.3a. DRIL outperforms behavioral cloning across most environments and numbers of demonstrations, often by a substantial margin. In the worst case its performance matches that of behavior cloning. In many cases, our method is able to match the expert’s performance using a small number of trajectories.

Figure 5.3b shows the evolution of the uncertainty cost and the policy reward throughout training. In all cases, the test reward improves while the uncertainty cost decreases. Interestingly, there is correspondence between the change in the uncertainty cost during training and the gap in performance between behavior cloning and DRIL. For example, in MsPacman there is both a small improvement in uncertainty cost over time and a small gap between behavior cloning and our method, whereas in Breakout there is a large improvement in uncertainty cost and a large gap between behavior cloning and our method. This suggests that the gains from our method comes from redirecting the policy back towards the expert manifold, which is manifested as a decrease in the uncertainty cost.

We were not able to obtain meaningful performance for GAIL on these domains, despite performing a hyperparameter search across learning rates for the policy and discriminator, and across different numbers of discriminator updates. We additionally experimented with clipping rewards in an effort to stabilize performance. These results are consistent with those of [227], who also reported negative results when running GAIL on images. While improved performance might be possible with more sophisticated adversarial training techniques, we note that this contrasts with our method which uses a fixed reward function obtained through simple supervised learning.

In section 5.7.4 we provide ablation experiments examining the effects of different cost function choices and the role of the BC loss. We also compare the ensemble approach to a dropout-based approach for posterior approximation and show that DRIL works well in both cases.



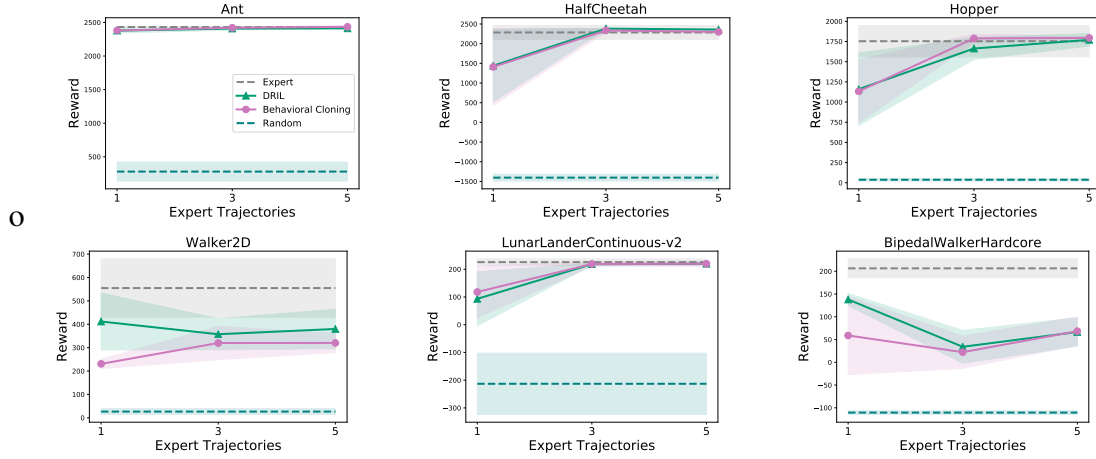


Figure 5.4: Results on continuous control tasks.

### 5.5.3 Continuous Control

We next report results of running our method on a 6 different continuous control tasks from the PyBullet<sup>6</sup> and OpenAI Gym [45] environments. We again used pretrained agents to generate expert demonstrations. Results are shown in Figure 5.4. In these environments we found behavior cloning to be a much stronger baseline than for the Atari environments, and in many tasks it was able to match expert performance using as little as 3 trajectories. Our method exhibits a modest improvement on Walker2D and BipedalWalkerHardcore when a single trajectory is used, and otherwise has similar performance to behavior cloning. The fact that our method does not perform worse than behavior cloning on tasks where covariate shift is likely less of an issue provides evidence of its robustness.

<sup>6</sup>[https://github.com/bulletphysics/bullet3/tree/master/examples/pybullet/gym/pybullet\\_envs/examples](https://github.com/bulletphysics/bullet3/tree/master/examples/pybullet/gym/pybullet_envs/examples)

## 5.6 Conclusion

Addressing covariate shift has been a long-standing challenge in imitation learning. In this work, we have proposed a new method to address this problem by penalizing the disagreement between an ensemble of different policies sampled from the posterior. Importantly, our method requires no additional labeling by an expert. Our experimental results demonstrate that DRIL can often match expert performance while using only a small number of trajectories across a wide array of tasks, ranging from tabular MDPs to pixel-based Atari games and continuous control tasks. On the theoretical side, we have shown that our algorithm can provably obtain a low regret bound for tabular problems in which the  $\kappa$  parameter is low.

There are multiple directions for future work. On the theoretical side, characterizing the  $\kappa$  parameter on a larger array of problems would help to better understand the settings where our method can expect to do well. Empirically, there are many other settings in structured prediction [75] where covariate shift is an issue and where our method could be applied. For example, in dialogue and language modeling it is common for generated text to become progressively less coherent as errors push the model off the manifold it was trained on. Our method could potentially be used to fine-tune language or translation models [60, 300] after training by applying our uncertainty-based cost function to the generated text.

## 5.7 Extend Details

Imitation learning has been used within the context of modern RL to help improve sample efficiency [54, 55, 124, 165, 233, 272] or overcome exploration [198]. These settings assume the reward is known and that the policies can then be fine-tuned with reinforcement learning. In this case, covariate shift is less of an issue since it can be corrected using the reinforcement signal.

### 5.7.1 Proofs

**Lemma 3.** *For any  $\pi \in \Pi$  we have  $J_{\text{exp}}(\pi) \leq \kappa J_{\text{alg}}(\pi)$*

*Proof.* We will first show that for any  $\pi \in \Pi$  and  $\mathcal{U} \subseteq \mathcal{S}$ , we have  $J_{\text{exp}}(\pi) \leq \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})} J_{\text{alg}}(\pi)$ . We can rewrite this as:

$$\begin{aligned} J_{\text{exp}}(\pi) &= \mathbb{E}_{s \sim d_\pi} \left[ \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \right] \\ &= \mathbb{E}_{s \sim d_\pi} \left[ \mathbb{I}(s \in \mathcal{U}) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \right] + \mathbb{E}_{s \sim d_\pi} \left[ \mathbb{I}(s \notin \mathcal{U}) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \right] \end{aligned}$$

We begin by bounding the first term:

$$\begin{aligned} \mathbb{E}_{s \sim d_\pi} \left[ \mathbb{I}(s \in \mathcal{U}) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \right] &= \sum_{s \in \mathcal{U}} d_\pi(s) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \\ &= \sum_{s \in \mathcal{U}} \frac{d_\pi(s)}{d_{\pi^*}(s)} d_{\pi^*}(s) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \end{aligned}$$

$$\begin{aligned}
&\leq \sum_{s \in \mathcal{U}} \underbrace{\left( \max_{\pi' \in \Pi} \sup_{s \in \mathcal{U}} \frac{d_{\pi'}(s)}{d_{\pi^*}(s)} \right)}_{\alpha(\mathcal{U})} d_{\pi^*}(s) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \\
&= \alpha(\mathcal{U}) \sum_{s \in \mathcal{U}} d_{\pi^*}(s) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \\
&\leq \alpha(\mathcal{U}) \sum_{s \in \mathcal{S}} d_{\pi^*}(s) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \\
&= \alpha(\mathcal{U}) \mathbb{E}_{s \sim d_{\pi^*}} \left[ \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \right] \\
&= \alpha(\mathcal{U}) J_{\text{BC}}(\pi)
\end{aligned}$$

We next bound the second term:

$$\begin{aligned}
\mathbb{E}_{s \sim d_\pi} \left[ \mathbb{I}(s \notin \mathcal{U}) \|\pi(\cdot|s) - \pi^*(\cdot|s)\| \right] &\leq \mathbb{E}_{s \sim d_\pi} \left[ \mathbb{I}(s \notin \mathcal{U}) \right] \\
&\leq \mathbb{E}_{s \sim d_\pi} \left[ \mathbb{I}(s \notin \mathcal{U}) \frac{\min_{a \in \mathcal{A}} \text{Var}_{\pi_i \sim \Pi_E} [\pi_i(a|s)]}{\beta(\mathcal{U})} \right] \\
&= \frac{1}{\beta(\mathcal{U})} \mathbb{E}_{s \sim d_\pi} \left[ \mathbb{I}(s \notin \mathcal{U}) \sum_{a \in \mathcal{A}} \pi(a|s) \text{Var}_{\pi_i \sim \Pi_E} [\pi_i(a|s)] \right] \\
&= \frac{1}{\beta(\mathcal{U})} \underbrace{\sum_{s \notin \mathcal{U}} d_\pi(s) \sum_{a \in \mathcal{A}} \pi(a|s) \text{Var}_{\pi_i \sim \Pi_E} [\pi_i(a|s)]}_{A(\pi)}
\end{aligned}$$

Now observe we can decompose the RL cost as follows:

$$\begin{aligned}
J_{\text{U}}(\pi) &= \mathbb{E}_{s \sim d_\pi, a \sim \pi(\cdot|s)} \left[ \text{Var}_{\pi_i \sim \Pi_E} \pi_i(a|s) \right] \\
&= \sum_s d_\pi(s) \sum_a \pi(a|s) \left[ \text{Var}_{\pi_i \sim \Pi_E} \pi_i(a|s) \right]
\end{aligned}$$

$$= \underbrace{\sum_{s \in U} d_\pi(s) \sum_a \pi(a|s) \left[ \text{Var}_{\pi_i \sim \Pi_E} \pi_i(a|s) \right]}_{B(\pi)} + \underbrace{\sum_{s \notin U} d_\pi(s) \sum_a \pi(a|s) \left[ \text{Var}_{\pi_i \sim \Pi_E} \pi_i(a|s) \right]}_{A(\pi)}$$

Putting these together, we get the following:

$$\begin{aligned} J_{\text{exp}}(\pi) &\leq \alpha(\mathcal{U}) J_{\text{BC}}(\pi) + \frac{1}{\beta(\mathcal{U})} A(\pi) \\ &= \frac{\alpha(\mathcal{U})\beta(\mathcal{U})}{\beta(\mathcal{U})} J_{\text{BC}}(\pi) + \frac{\alpha(\mathcal{U})}{\alpha(\mathcal{U})\beta(\mathcal{U})} A(\pi) \\ &\leq \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})} J_{\text{BC}}(\pi) + \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})} A(\pi) \\ &\leq \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})} \left( J_{\text{BC}}(\pi) + A(\pi) \right) \\ &\leq \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})} \left( J_{\text{BC}}(\pi) + J_{\text{U}}(\pi) \right) \\ &= \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})} J_{\text{alg}}(\pi) \end{aligned}$$

Here we have used the fact that  $\beta(\mathcal{U}) \leq 1$  since  $0 \leq \pi(a|s) \leq 1$  and

$\alpha(\mathcal{U}) \geq \sup_{s \in \mathcal{U}} \frac{d_\pi^*(s)}{d_\pi^*(s)} = 1$  hence  $\frac{1}{\alpha(\mathcal{U})} \leq 1$ . Taking the minimum over subsets  $\mathcal{U} \subseteq \mathcal{S}$ , we get

$$J_{\text{exp}}(\pi) \leq \kappa J_{\text{alg}}(\pi).$$

□

**Lemma 4.**  $\min_{\pi \in \Pi} J_{\text{alg}}(\pi) \leq 4\epsilon$

*Proof.* Plugging the optimal policy into  $J_{\text{alg}}$ , we get:

$$\begin{aligned}
J_{\text{alg}}(\pi^\star) &= J_{\text{BC}}(\pi^\star) + J_{\text{U}}(\pi^\star) \\
&= 0 + \mathbb{E}_{s \sim d_{\pi^\star}, a \sim \pi^\star(\cdot|s)} \left[ \text{Var}_{\pi_i \sim \Pi_E} [\pi_i(a|s)] \right] \\
&= \mathbb{E}_{s \sim d_{\pi^\star}, a \sim \pi^\star(\cdot|s)} \left[ \frac{1}{|E|} \sum_i \left( \pi_i(a|s) - \bar{\pi}(a|s) \right)^2 \right] \\
&\leq 2 \mathbb{E}_{s \sim d_{\pi^\star}, a \sim \pi^\star(\cdot|s)} \left[ \frac{1}{|E|} \sum_i \left( \pi_i(a|s) - \pi^\star(a|s) \right)^2 + \left( \bar{\pi}(a|s) - \pi^\star(a|s) \right)^2 \right] \\
&= \underbrace{2 \mathbb{E}_{s \sim d_{\pi^\star}, a \sim \pi^\star(\cdot|s)} \left[ \frac{1}{|E|} \sum_i \left( \pi_i(a|s) - \pi^\star(a|s) \right)^2 \right]}_{\text{Term1}} + \underbrace{2 \mathbb{E}_{s \sim d_{\pi^\star}, a \sim \pi^\star(\cdot|s)} \left[ \left( \bar{\pi}(a|s) - \pi^\star(a|s) \right)^2 \right]}_{\text{Term2}}
\end{aligned}$$

We will first bound Term 1

$$\mathbb{E}_{s \sim d_{\pi^\star}, a \sim \pi^\star(\cdot|s)} \left[ \frac{1}{|E|} \sum_{i=1}^{|E|} \left( \pi_i(a|s) - \pi^\star(a|s) \right)^2 \right]$$

:

$$\begin{aligned}
&= \frac{1}{|E|} \mathbb{E}_{s \sim d_{\pi^\star}} \left[ \sum_{a \in \mathcal{A}} \pi^\star(a|s) \sum_{i=1}^{|E|} \left( \pi_i(a|s) - \pi^\star(a|s) \right)^2 \right] \\
&\leq \frac{1}{|E|} \mathbb{E}_{s \sim d_{\pi^\star}} \left[ \sum_{a \in \mathcal{A}} \pi^\star(a|s) \sum_{i=1}^{|E|} \left| \pi_i(a|s) - \pi^\star(a|s) \right| \right] \\
&\leq \frac{1}{|E|} \mathbb{E}_{s \sim d_{\pi^\star}} \left[ \sum_{i=1}^{|E|} \sum_{a \in \mathcal{A}} \left| \pi_i(a|s) - \pi^\star(a|s) \right| \right] \\
&\leq \frac{1}{|E|} \sum_{i=1}^{|E|} \mathbb{E}_{s \sim d_{\pi^\star}} \left[ \left| \pi_i(\cdot|s) - \pi^\star(\cdot|s) \right| \right]
\end{aligned}$$

$$\begin{aligned}
&\leq \frac{1}{|E|} \sum_{i=1}^{|E|} \epsilon \\
&= \epsilon
\end{aligned}$$

We will next bound Term 2

$$\mathbb{E}_{s \sim d_{\pi^*}, a \sim \pi^*(\cdot|s)} \left[ \left( \bar{\pi}(a|s) - \pi^*(a|s) \right)^2 \right]$$

:

$$\begin{aligned}
&= \mathbb{E}_{s \sim d_{\pi^*}, a \sim \pi^*(\cdot|s)} \left[ \left( \pi^*(a|s) - \frac{1}{|E|} \sum_{i=1}^{|E|} \pi_i(a|s) \right)^2 \right] \\
&= \mathbb{E}_{s \sim d_{\pi^*}, a \sim \pi^*(\cdot|s)} \left[ \left( \frac{1}{|E|} \sum_{i=1}^{|E|} \pi^*(a|s) - \frac{1}{|E|} \sum_{i=1}^{|E|} \pi_i(a|s) \right)^2 \right] \\
&= \mathbb{E}_{s \sim d_{\pi^*}, a \sim \pi^*(\cdot|s)} \left[ \left( \frac{1}{|E|} \sum_{i=1}^{|E|} (\pi^*(a|s) - \pi_i(a|s)) \right)^2 \right] \\
&\leq \mathbb{E}_{s \sim d_{\pi^*}, a \sim \pi^*(\cdot|s)} \left[ \frac{1}{|E|^2} |E| \sum_{i=1}^{|E|} \left( \pi^*(a|s) - \pi_i(a|s) \right)^2 \right] \text{ (Cauchy - Schwarz)} \\
&= \frac{1}{|E|} \sum_{i=1}^{|E|} \mathbb{E}_{s \sim d_{\pi^*}, a \sim \pi^*(\cdot|s)} \left[ \left( \pi^*(a|s) - \pi_i(a|s) \right)^2 \right] \\
&\leq \frac{1}{|E|} \sum_{i=1}^{|E|} \mathbb{E}_{s \sim d_{\pi^*}, a \sim \pi^*(\cdot|s)} \left[ \left| \pi^*(a|s) - \pi_i(a|s) \right| \right] \\
&\leq \frac{1}{|E|} \sum_{i=1}^{|E|} \mathbb{E}_{s \sim d_{\pi^*}} \left[ \left| \pi^*(\cdot|s) - \pi_i(\cdot|s) \right| \right] \\
&= \frac{1}{|E|} \sum_{i=1}^{|E|} J_{\text{BC}}(\pi_i)
\end{aligned}$$

$$\leq \epsilon$$

The last step follows from our optimization oracle assumption:  $0 \leq \min_{\pi \in \Pi} J_{\text{BC}}(\pi) \leq J_{\text{BC}}(\pi^*) = 0$ , hence  $J_{\text{BC}}(\pi_i) \leq 0 + \epsilon = \epsilon$ . Combining the bounds on the two terms, we get  $J_{\text{alg}}(\pi^*) \leq 4\epsilon$ . Since  $\pi^* \in \Pi$ , the result follows.  $\square$

**Theorem 5.7.1.** *Let  $\hat{\pi}$  be the result of minimizing  $J_{\text{alg}}$  using our optimization oracle, and assume that  $Q_{T-t+1}^{\pi^*}(s, a) - Q_{T-t+1}^{\pi^*}(s, \pi^*) \leq u$  for all  $a \in \mathcal{A}, t \in \{1, 2, \dots, T\}, d_{\pi}^t(s) > 0$ . Then  $\hat{\pi}$  satisfies  $J(\hat{\pi}) \leq J(\pi^*) + 3u\kappa\epsilon T$ .*

*Proof.* By our optimization oracle and Lemma 2, we have

$$\begin{aligned} J_{\text{alg}}(\hat{\pi}) &\leq \min_{\pi \in \Pi} J_{\text{alg}}(\pi) + \epsilon \\ &\leq 4\epsilon + \epsilon \\ &= 5\epsilon \end{aligned}$$

Combining with Lemma 1, we get:

$$\begin{aligned} J_{\text{exp}}(\hat{\pi}) &\leq \kappa J_{\text{alg}}(\hat{\pi}) \\ &\leq 5\kappa\epsilon \end{aligned}$$

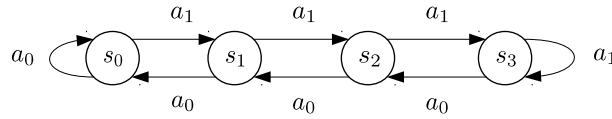
Applying Theorem 1 from [236], we get  $J(\hat{\pi}) \leq J(\pi^*) + 5u\kappa\epsilon T$ .



□

### 5.7.2 Importance of Behavior Cloning Cost

The following example shows how minimizing the uncertainty cost alone without the BC cost can lead to highly sub-optimal policies if the demonstration data is generated by a stochastic policy which is only slightly suboptimal. Consider the following deterministic chain MDP:



The agent always starts in  $s_1$ , and gets a reward of 1 in  $s_3$  and 0 elsewhere. The optimal policy is given by:

$$\pi^*(\cdot|s_0) = (0, 1)$$

$$\pi^*(\cdot|s_1) = (0, 1)$$

$$\pi^*(\cdot|s_2) = (0, 1)$$

$$\pi^*(\cdot|s_3) = (0, 1)$$

Assume the demonstration data is generated by the following policy, which is only slightly suboptimal:

$$\pi_{\text{demo}}(\cdot|s_0) = (0, 1)$$

$$\pi_{\text{demo}}(\cdot|s_1) = (0, 1)$$

$$\pi_{\text{demo}}(\cdot|s_2) = (0.1, 0.9)$$

$$\pi_{\text{demo}}(\cdot|s_3) = (0, 1)$$

Let us assume realizability and perfect optimization for simplicity. If both transitions  $(s_2, a_0)$  and  $(s_2, a_1)$  appear in the demonstration data, then Random Expert Distillation (RED) will assign zero cost to both transitions. If we do not use bootstrapped samples to train the ensemble, then DRIL without the BC cost (we will call this UO-DRIL for Uncertainty-Only DRIL) will also assign zero cost to both transitions since all models in the ensemble would recover the Bayes optimal solution given the demonstration data. If we are using bootstrapped samples, then the Bayes optimal solution for each bootstrapped sample may differ and thus the different policies in the ensemble might disagree in their predictions, although given enough demonstration data we would expect these differences (and thus the uncertainty cost) to be small.

Note also that since no samples at the state  $s_0$  occur in the demonstration data, both RED and UO-DRIL will likely assign high uncertainty costs to state-action pairs at  $(s_0, a_0), (s_0, a_1)$  and thus avoid highly suboptimal policies which get stuck at  $s_0$ .

Now consider policies  $\hat{\pi}_1, \hat{\pi}_2$  given by:

$$\hat{\pi}_1(\cdot|s_0) = (0, 1)$$

$$\hat{\pi}_1(\cdot|s_1) = (0, 1)$$

$$\hat{\pi}_1(\cdot|s_2) = (1, 0)$$

$$\hat{\pi}_1(\cdot|s_3) = (0, 1)$$

and

$$\hat{\pi}_2(\cdot|s_0) = (0, 1)$$

$$\hat{\pi}_2(\cdot|s_1) = (0, 1)$$

$$\hat{\pi}_2(\cdot|s_2) = (0.2, 0.8)$$

$$\hat{\pi}_2(\cdot|s_3) = (0, 1)$$

Both of these policies only visit state-action pairs which are visited by the demonstration policy. In the case described above, both RED and UO-DRIL will assign  $\hat{\pi}_1$  and  $\hat{\pi}_2$  similarly low costs. However,  $\hat{\pi}_1$  will cycle forever between  $s_1$  and  $s_2$ , never collecting reward, while  $\hat{\pi}_2$  will with high probability reach  $s_3$  and stay there, thus achieving high reward. This shows that minimizing the uncertainty cost alone does not necessarily distinguish between good and bad policies. However,  $\hat{\pi}_1$  will incur a higher BC cost than  $\hat{\pi}_2$ , since  $\hat{\pi}_2$  more closely matches the demonstration data at  $s_2$ . This shows that including the BC cost can be important for further disambiguating between policies which all stay within the distribution of the demonstration data, but have different behavior within that distribution.

### 5.7.3 Experimental Details

#### 5.7.3.1 Atari Environments

All behavior cloning and ensemble models were trained to minimize the negative log-likelihood classification loss on the demonstration data for 500 epochs using Adam [152] and a learning rate of  $2.5 \cdot 10^{-4}$ . For our method, we initially performed a hyperparameter search on Space Invaders over the following values:

Table 5.1: Hyperparameters (our method)

Hyperparameter	Values Considered	Final Value
Policy Learning rate	$2.5 \cdot 10^{-2}$ , $2.5 \cdot 10^{-3}$ , $2.5 \cdot 10^{-4}$	$2.5 \cdot 10^{-3}$
Quantile cutoff	0.8, 0.9, 0.95, 0.98	0.98
Number of supervised updates	1, 5	1
Number of policies in ensemble	5	5
Gradient clipping	0.1	0.1
Entropy coefficient	0.01	0.01
Value loss coefficient	0.5	0.5
Number of steps	128	128
Parallel Environments	16	16

We then chose the best values and kept those hyperparameters fixed for all other environments. All other A2C hyperparameters follow the default values in the repo [156]: policy networks consisted of 3-layer convolutional networks with  $8 - 32 - 64$  feature maps followed by a single-layer MLP with 512 hidden units.

For GAIL, we used the implementation in [156] and replaced the MLP discriminator by a CNN discriminator with the same architecture as the policy network. We initially performed a hyperparameter search on Breakout with 10 demonstrations over the values shown in Table 5.2. However, we did not find any hyperparameter configuration which performed better than behavioral cloning.

Table 5.2: Hyperparameters (GAIL)

Hyperparameter	Values Considered	Final Value
Policy Learning rate	$2.5 \cdot 10^{-2}$ , $2.5 \cdot 10^{-3}$ , $2.5 \cdot 10^{-4}$	$2.5 \cdot 10^{-3}$
Discriminator Learning rate	$2.5 \cdot 10^{-2}$ , $2.5 \cdot 10^{-3}$ , $2.5 \cdot 10^{-4}$	$2.5 \cdot 10^{-3}$
Number of discriminator updates	1, 5, 10	5
Gradient clipping	0.1	0.1
Entropy coefficient	0.01	0.01
Value loss coefficient	0.5	0.5
Number of steps	128	128
Parallel Environments	16	16

### 5.7.3.2 Continuous Control

All behavior cloning and ensemble models were trained to minimize the mean-squared error regression loss on the demonstration data for 500 epochs using Adam [152] and a learning rate of  $2.5 \cdot 10^{-4}$ . Policy networks were 2-layer fully-connected MLPs with tanh activations and 64 hidden units.

Table 5.3: Hyperparameters (our method)

Hyperparameter	Values Considered	Final Value
Policy Learning rate	$2.5 \cdot 10^{-3}$ , $2.5 \cdot 10^{-4}$ , $1 \cdot 10^{-4}$ , $5 \cdot 10^{-5}$	$2.5 \cdot 10^{-5}$
Quantile cutoff	0.98	0.98
Number of supervised updates	1	1
Number of policies in ensemble	5	5
Gradient clipping	0.1	0.1
Entropy coefficient	0.01	0.01
Value loss coefficient	0.5	0.5
Number of steps	128	128
Parallel Environments	16	16

### 5.7.4 Ablation Experiments

### 5.7.4.1 Cost Terms

Here we provide ablation experiments showing the effect of different cost function choices.

We compare the following variants:

- **DRIL** This is the regular DRIL agent, which optimizes both the BC cost and the clipped cost:

$$C_U^{\text{clip}}(s, a) = \begin{cases} -1 & \text{if } C_U(s, a) \leq q \\ +1 & \text{else} \end{cases}$$

- **DRIL (clipped cost 0/1)** This is the same as the regular DRIL agent, except that we use the following clipped cost function:

$$C_U^{\text{clip}}(s, a) = \begin{cases} 0 & \text{if } C_U(s, a) \leq q \\ +1 & \text{else} \end{cases}$$

- **DRIL (raw cost)** This is the same as the regular DRIL agent, except that we use the raw cost  $C_U(s, a)$  rather than the clipped cost.
- **DRIL (no BC cost)** This is the same as the regular DRIL agent, except that we remove the BC updates and only optimize  $C_U^{\text{clip}}(s, a)$ .

Results are shown in Figure 5.5. First, switching from the clipped cost in  $\{-1, +1\}$  to the clipped cost in  $\{0, 1\}$  or the raw cost causes a large drop in performance across most environments. One explanation may be that since the costs are always positive for both variants (which corresponds to a reward which is always negative), the agent may learn to terminate the episode early in order to minimize the total cost incurred. Using a cost/reward which has both

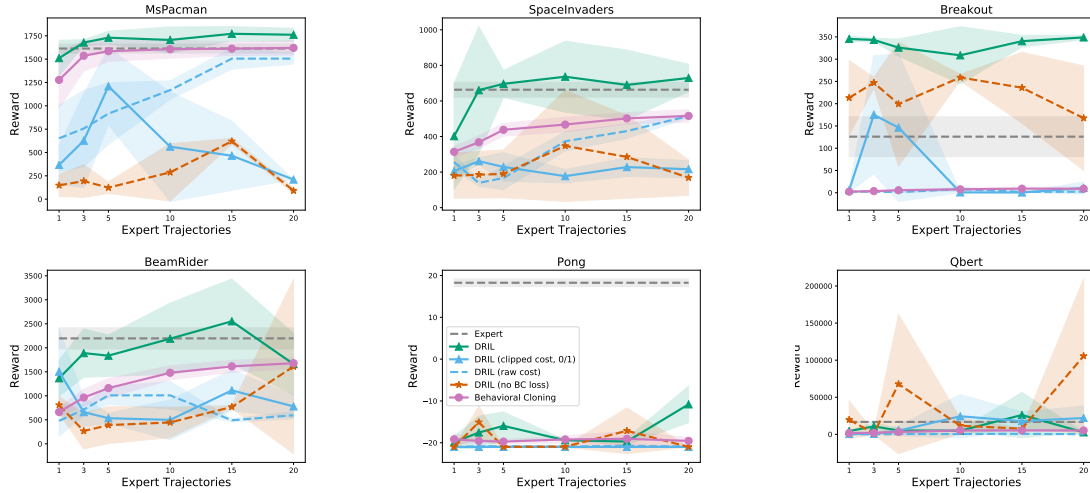


Figure 5.5: Cost ablation experiments on Atari environments.

positive and negative values avoids this behavior.

Second, optimizing the pure BC cost performs better than the pure uncertainty cost for some environments (MsPacman, SpaceInvaders, BeamRider) while optimizing the pure uncertainty cost performs better than BC for others (Breakout, Qbert). DRIL, which optimizes both, has robust performance and performs the best out of the different variants over most environments and numbers of trajectories.

#### 5.7.4.2 Uncertainty Estimation Comparison

We provide additional results comparing the ensembling and MC-dropout [98] approaches to posterior estimation. For MC-Dropout we trained a single policy network with a dropout rate of 0.1 applied to all layers except the last, and estimated the variance for each state-action pair using 10 different dropout masks. Similarly to the ensemble approach, we computed the 98<sup>th</sup> quantile of the variance on the demonstration data and used this value in our clipped cost. Results are shown for three environments in Figure 5.6. MC-dropout performs similarly to the ensembling approach, which shows that our method can be paired with different approaches to

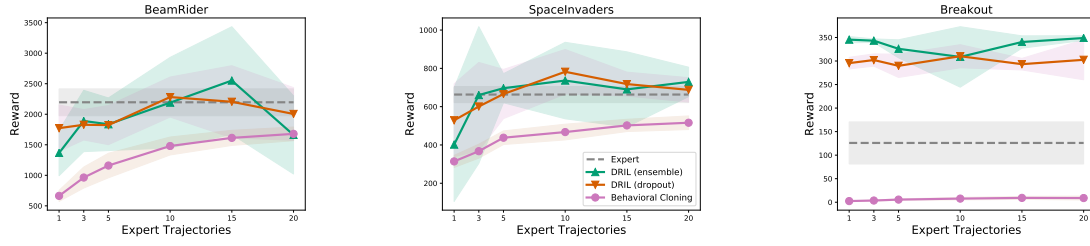


Figure 5.6: Comparison of ensembling and MC dropout for posterior estimation on Atari environments.

posterior estimation.



## Chapter 6:    Emperical Study of Imtation Learning

The previous chapter introduced an algorithm to address the covariate shift issues in naive imitation learning. While we only compare that algorithm introduced to GAIL, other algorithms learn a reward function from demonstration data to optimize using reinforcement learning, introduced in the literature.

We perform a thorough investigation into imitation learning algorithms that optimize a reward function learned from demonstration data in this chapter. These algorithms were never fairly compared because they all use different environments to run experiments. For example, GAIL used Mujoco, and DRIL (see chapter 5) used Pybullet. To this end, we run all the algorithms on all environments considered in recent and past imitation learning literature to get a comprehensive understanding of the relative performance of every algorithm. We expand each algorithm to include all tricks to ensure that differences exist solely in algorithmic design. We further draw a connection between these algorithms and modern success reinforcement/imitation learning algorithms in natural language processing.

### 6.1 Introduction

Learning from demonstration (LFD) is a paradigm that uses expert demonstration data to solve sequential decision-making and prediction problems. Unlike reinforcement learning (RL) algorithms, which assume a reward function exists no matter how complex tasks are, LFD assumes the behavior in the demonstration data describes the tasks. Moreover, using

demonstration data instead of a reward function reduces the time needed to solve tasks [4, 223, 236].

Imitation learning (IL) is a widely studied approach to doing LFD that directly mimics the behavior provided in the expert demonstration data. The most naive IL approach assumes no environment interactions and uses supervised learning to predict the actions of the corresponding states in the expert demonstration data offline [213]. Although this approach requires no environment interactions, it suffers from a feedback-driven covariate shift issue. For example, in end-to-end learning for autonomous driving, models make errors when visiting areas not seen in the demonstration data [20, 67, 134, 160, 213]. Similarly, in natural language processing (NLP), text generation models lead to degeneration, where the output text generated is confusing, inconsistent, and repetitive [58, 173, 225, 247, 301]. The feedback-driven covariate issue occurs because the states seen when training on the expert demonstration data and the states seen while interacting in an environment during evaluation are different [213, 225, 231], resulting in compounding errors.

Interactive IL is another approach to IL that attempts to mitigate the feedback-driven covariate shift issue using environment interactions and an interactive expert. Algorithms under this approach [54, 233, 236], query an interactive expert at arbitrary states seen when interacting in an environment during training. However, in many cases querying an interactive expert in an online setting is not practical for many situations where online human feedback is not possible or computationally expensive with automated experts [141, 149, 162]. Under certain assumptions, new approaches in IL address the feedback-driven covariate shift issue without an interactive expert.

Unlike interactive IL, these new IL approaches only use environment interactions to

Algorithm	BC	RL	Performance Objective Function
KNN	✗	✗	-
LWL	✗	✗	-
BC	✓	✗	$\mathbb{E}_{(s^*, a^*) \sim \mathcal{D}_{\text{exp}}} [\ell(\pi(s^*), \pi^*(s^*))]$
CR	✗	✓	$\mathbb{E}_{(s, a) \sim \mu^\pi} [1]$
GAIL+	✗	✓	$\mathbb{E}_{(s, a) \sim \mu^\pi} [\log D_\theta(s, a)]$
GAIL-	✗	✓	$\mathbb{E}_{(s, a) \sim \mu^\pi} [-\log(1 - D_\theta(s, a))]$
AIRL	✗	✓	$\mathbb{E}_{(s, a) \sim \mu^\pi} [-\log(1 - D_\theta(s, a)) + \log(D_\theta(s, a))]$
RED	✗	✓	$\mathbb{E}_{(s, a) \sim \mu^\pi} [\exp(-\sigma^{-1}(\ f_{\hat{\theta}}(s, a) - f_\theta(s, a)\ _2^2))]$
DRIL	✓	✓	$J_{\text{BC}}(\pi) + \mathbb{E}_{(s, a) \sim \mu^\pi} [-\text{Var}_{\pi_e \sim \Pi_E}[\pi_e(a s, )]]$
BC-GAIL+	✓	✓	$J_{\text{BC}}(\pi) + \mathbb{E}_{(s, a) \sim \mu^\pi} [\log D_\theta(s, a)]$
BC-GAIL-	✓	✓	$J_{\text{BC}}(\pi) + \mathbb{E}_{(s, a) \sim \mu^\pi} [-\log(1 - D_\theta(s, a))]$
BC-AIRL	✓	✓	$J_{\text{BC}}(\pi) + \mathbb{E}_{(s, a) \sim \mu^\pi} [-\log(1 - D_\theta(s, a, s')) + \log(D_\theta(s, a, s'))]$
BC-RED	✓	✓	$J_{\text{BC}}(\pi) + \mathbb{E}_{(s, a) \sim \mu^\pi} [-\exp(-\sigma\ f_\theta(s, a) - \hat{f}_\theta(s, a)\ _2^2)]$
BC-CR	✓	✓	$J_{\text{BC}}(\pi) + \mathbb{E}_{(s, a) \sim \mu^\pi} [1]$

Table 6.1: Summary of IL algorithms considered in this work. The table indicates whether a particular algorithm performs BC updates interleaved with RL updates, as well as the reward used to train the policy. The RL component of the objective specifies what reward function each algorithm is optimizing, i.e.  $\mathbb{E}_{(s, a) \sim \mu^\pi} [r(s, a)]$  where  $r(s, a)$  varies for each algorithm’s objective.

address the feedback-driven covariate shift issue. In particular, these algorithms learn a reward function that estimates the support of the expert occupancy measure from demonstration data and optimize it with RL [44, 126, 267, 296]. These IL algorithms cannot be fully classified as Inverse Reinforcement Learning (IRL) because they do not recover the underlying reward function of the MDP [203]. The focus of this study is to understand these algorithms.

Although these new IL algorithms that address the feedback-driven covariate shift issue using only environment interactions have been successful, there have been concerns regarding around evaluation of these algorithms. In particular, Spencer et al. [266] mentioned that the continuous-control tasks used for evaluation are "too easy". It is not evident that all of these IL algorithms will generalize to tasks beyond continuous-control tasks, such as pixel or structured-prediction tasks. Moreover, there have been conflicting performance results of the Behavioral Cloning (BC) baseline algorithm across various papers [44, 73, 126, 150, 267, 296]. For example,

authors of GAIL [126], sub-sample expert demonstration data to reduce the performance of BC, making their results not comparable with results that do not sub-sample expert demonstration data. Because of these concerns, we perform a thorough empirical evaluation of the performance of baseline algorithms and notice BC performs exceptionally well when sub-sampling is not applied.

Our study investigates crucial pieces for designing and implementing IL algorithms that leverage only environment interactions. Some of these IL algorithms only perform RL updates, whereas others interleave BC updates with RL updates. To this end, we benchmark all these algorithms on 36 environments spanning three broad task categories: continuous-control, pixel, and structure prediction. We introduce two new simple baselines, Constant Reward (CR) and Behavioral Cloning -Regularized Constant Reward (BC-CR) inspired from Jena et al. [136] and Wang et al. [296]. We notice that the BC-CR baseline we introduce is competitive and sometimes better than most IL algorithms in this study across all three task categories. Finally, we relate algorithms in this study to a broader set of algorithms and problems in NLP that attempt to address the feedback-driven covariate shift issue (i.e., exposure-bias [225]) in language models.

To summarize, the contributions of this work are:

- Benchmarking IL algorithms that only use environment interactions across three broad tasks
- Providing insight to important IL algorithm components and introducing new algorithm variants
- Relating IL algorithms in this study to modern NLP algorithms

## 6.2 Background

Task	Continuous Control	Pixel	Structured Prediction
<b>Toolkits</b>	MuJoCo [283], Pybullet [70]	DeepMind Control Suite [279], Box2D [51]	NLPGym [224]
<b>Trajectories</b>	[1,2,3,5,10]	[1,3,5,10]	Full Dataset
<b># Environments</b>	25	6	5
<b># Seeds</b>	5	5	5
<b># Experiments Per Task</b>	$(5 \times 25 \times 5) = 625$	$(4 \times 6 \times 5) = 120$	$(1 \times 5 \times 5) = 25$
<b>Action Space</b>	continuous	continuous	discrete
<b>Observation space</b>	state features	pixels	state word embeddings and context features

Table 6.2: An overview of the three tasks categories considered in experiments. Each task category state space is different.

We consider a standard episodic finite-horizon Markov Decision Process (MDP) discussed in (see chapter 2). For any policy  $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ , let  $d_t^\pi$  represent the state visitation distribution induced by following  $\pi$  for  $t$  time steps. Let  $d^\pi = \frac{1}{T} \sum_{t=1}^T d_t^\pi$  be the average state visitation distribution if we follow policy  $\pi$  for  $T$  steps. Similarly, we define the state-action visitation distribution  $\mu_t^\pi$  of  $\pi$ . Let  $\mu^\pi = \frac{1}{T} \sum_{t=1}^T \mu_t^\pi$  be the average state-action visitation distribution if we follow policy  $\pi$  for  $T$  steps. It is known that  $\mu^\pi(s, a) = d^\pi(s) \cdot \pi(a|s)$  [219]. Given  $a \in \mathcal{A}$ ,  $s \in \mathcal{S}$  and immediate cost  $c(s, a)$  cost of performing action  $a$  in state  $s$ , the total cost of executing policy  $\pi$  for  $T$  steps is  $J(\pi) = \sum_{t=1}^T \mathbb{E}_{(s_t, a_t) \sim \mu_t^\pi} [c(s_t, a_t)] = T \mathbb{E}_{(s, a) \sim \mu^\pi} [c(s, a)]$ .

In IL, we do not have access to the true costs  $c(s, a)$  for a particular task. Instead, we are given a finite amount of demonstration data  $\mathcal{D}_{\text{exp}} = \{(s^*, a^*)\}$  generated by executing an optimal expert policy  $\pi^*$ . The environment dynamics are often either unknown or complex. However, there are many settings, for example, a video game or physics engine simulator,

where we can assume that the learner can interact with an environment denoted as  $\Sigma$  and collect state-action pairs  $\{(s, a)\}_{t=1}^T \sim \mu^\pi$ . Note that we use  $\mathcal{D}_{\text{exp}}$  to correspond to the finite state-action pairs of demonstration provided by an expert, where  $\mu^{\pi^*}$  is the expert's entire state-action visitation distribution. The goal is to train a policy  $\pi$  that minimizes the performance difference  $J(\pi) - J(\pi^*)$ .

The naive approach to IL, Behavioral Cloning (BC) [213] ignores access to the environment  $\Sigma$  and trains a policy  $\hat{\pi}$  using the expert demonstration data  $\mathcal{D}_{\text{exp}}$  by minimizing the following objective:

$$J_{\text{BC}}(\pi) = \mathbb{E}_{(s^*, a^*) \sim \mathcal{D}_{\text{exp}}} [\ell(\pi(s^*), \pi^*(s^*))]$$

where  $\ell$  is a surrogate loss; for theoretical analysis, this surrogate loss can be any convex loss function used for training, for example, hinge loss. In practice, this surrogate loss can be maximum-likelihood [147]. It has been shown that if we drive training error down  $\mathbb{E}_{(s^*, a^*) \sim \mathcal{D}_{\text{exp}}} [\ell(\pi(s^*), \pi^*(s^*))] \leq \epsilon$  and have low holdout error, the learner may still not perform well. The issue is that in sequential decision-making there is a feedback loop between past actions  $a_{t-1}$  and the current input  $s_t$  of at time  $t$ , i.e.  $\hat{\pi}(a_t | s_t, a_{t-1})$ . This feedback loop creates covariate shift between the input the learner sees interacting in an environment and the input trained on from the expert demonstration data  $\mathcal{D}_{\text{exp}}$ . This covariate shift leads to compounding error of  $\mathcal{O}(T^2\epsilon)$ , formalized in Ross and Bagnell [231]

### 6.3 Algorithms

In this section, we summarize all algorithms in this study, shown in table 6.1. For further details see section 6.9.7.

### 6.3.1 Baseline Algorithms

The standard baseline that all IL algorithms commonly compare against is Behavioral Cloning. We additionally consider two classic baselines algorithms  $k$ -nearest neighbor and Locally Weight Learning.

**BC:** Behavioral Cloning (BC) [213] is the simplest IL baseline algorithm to run discussed in section 6.2.

**KNN:** K-Nearest Neighbor (KNN) [215, 261] test whether simple memorization of the expert’s actions is sufficient to solve a task. At each time step we use a distance function  $d$  to find the  $k$  closest states in the expert demonstration data  $\mathcal{D}_{\text{exp}}$  to the current learners state  $s_t$  and execute the corresponding action  $a^*$ .

**LWL:** Locally Weighted Learning (LWL) [11, 131] is a nonparametric instance-based learning algorithm that leverages demonstration similar to KNN. LWL finds the  $k$  closest states in the expert demonstration data  $\mathcal{D}_{\text{exp}}$  and uses the distance of these neighboring points to train a weighted regression function.

**CR:** A few IL algorithms have optimized a constant reward [227, 296]. Kostrikov et al. [157] shared for specific environments, a survival bonus in the form of a per-step positive reward may correlate with task performance. Inspired by ideas, we include a simple baseline to train an RL agent with a constant reward (CR)  $r > 0$ . Formally, the reward is  $r$  if the current state is not terminal and 0 otherwise.

**BC-CR:** Most IL algorithms improve upon baselines by taking advantage of interacting in an environment. Jena et al. [137] proposed a baseline that took advantage of environment interactions by optimizing a random reward. We propose a new and straightforward baseline

Behavior Cloning -Regularized Constant Reward (BC-CR) inspired by Jena et al. [137], which combines BC updates with a constant reward function, thus taking advantage of the expert data  $\mathcal{D}_{\text{exp}}$  and environment interactions.

**KNN:** K-Nearest Neighbor (KNN) [215, 261] test whether simple memorization of the expert’s actions is sufficient to solve a task. At each time step we use a distance function  $d$  to find the  $k$  closest states in the expert demonstration data  $\mathcal{D}_{\text{exp}}$  to the current learners state  $s_t$  and execute the corresponding action  $a^*$ .

**LWL:** Locally Weighted Learning (LWL) [11, 131] is a nonparametric instance-based learning algorithm that leverages demonstration similar to KNN. LWL finds the  $k$  closest states in the expert demonstration data  $\mathcal{D}_{\text{exp}}$  and uses the distance of these neighboring points to train a weighted regression function.

**CR:** A few IL algorithms have optimized a constant reward [227, 296]. Kostrikov et al. [157] shared for specific environments, a survival bonus in the form of a per-step positive reward may correlate with task performance. Inspired by ideas, we include a simple baseline to train an RL agent with a constant reward (CR)  $r > 0$ . Formally, the reward is  $r$  if the current state is not terminal and 0 otherwise.

**BC-CR:** Most IL algorithms improve upon baselines by taking advantage of interacting in an environment. Jena et al. [137] proposed a baseline that took advantage of environment interactions by optimizing a random reward. We propose a new and straightforward baseline Behavior Cloning -Regularized Constant Reward (BC-CR) inspired by Jena et al. [137], which combines BC updates with a constant reward function, thus taking advantage of the expert data  $\mathcal{D}_{\text{exp}}$  and environment interactions.



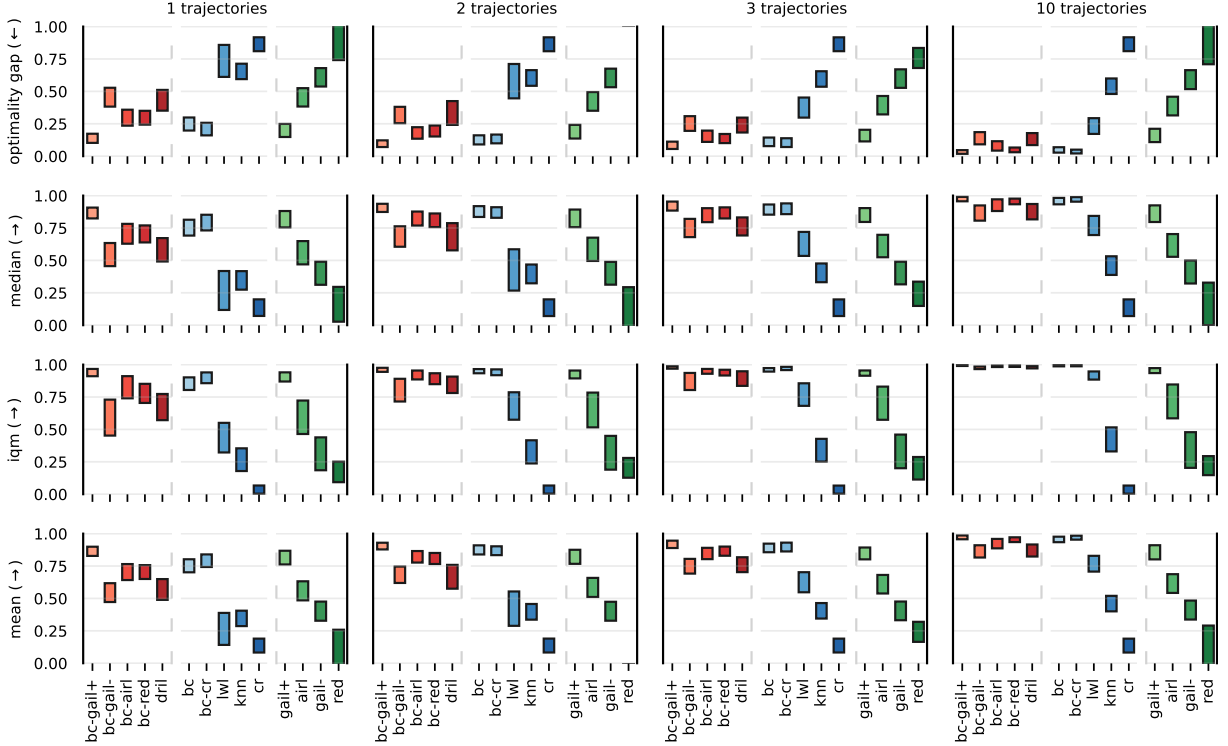


Figure 6.1: Aggregate results for continuous control tasks **without sub-sampled trajectories**. Scores are normalized between a random and expert agent performance. Bars indicate 95% confidence intervals computed using stratified bootstrapping. Red colors perform BC and RL updates, blue colors are baselines and green only perform RL updates. Arrows indicate higher (↑) or lower (↓) is better.

### 6.3.2 Existing IL Algorithms

IL algorithms in this study learn a cost function designed to estimate the support of the expert [296]. Some methods interleave BC updates with RL updates, while others only perform RL updates. We create new algorithms that interleave updates for those that do not interleave BC and RL updates. All algorithm objectives are defined in table 6.1.

**GAIL:** Generative Adversarial Imitation Learning (GAIL) [126] draws connections between GANs [106] and maximum entropy IRL [327]. GAIL trains a discriminator,  $D_\theta : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$  to distinguish between state-action pairs drawn from the expert and the learner policy. While the learner maximizes a reward using RL for confusing the discriminator. The

original GAIL paper proposes to use a particular reward, while in their codebase <sup>1</sup>, they use an alternative reward for specific tasks [157]. We evaluate both, denoted as GAIL- and GAIL+ (see table 6.1).

**AIRL:** Adversarial Inverse Reinforcement Learning (AIRL) [95] builds on the GAIL framework. Although the original intention of the algorithm is to deal with changes in environment dynamics, their proposed reward function can assign both positive and negative rewards [157]. Besides reward, we do not include any of their other proposed modifications in our implementation since the large-scale study of [202] did not report any significant improvement.

**RED:** Inspired by Random Network Distillation [48], Random Expert Distillation [296] (RED) is trained to minimize the mean-squared error distance between outputs of a fixed random neural network  $f_\theta : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^k$  and a predictor network  $f_{\hat{\theta}} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^k$ , using state-action pairs  $(s^*, a^*)$  drawn from demonstration data  $\mathcal{D}_{\text{exp}}$ . The cost function captures a measure of similarity between the learner state-actions and those of the expert. The policy objective function is in table 6.1 where  $\sigma$  is a bandwidth hyperparameter which is tuned.

**DRIL:** Disagreement-Regularized Imitation Learning (DRIL) (see chapter 5) learns a reward function based on variance of an ensemble of BC policies  $\Pi_E = \{\pi_1, \dots, \pi_E\}$  trained on the expert demonstration data  $\mathcal{D}_{\text{exp}}$ . Unlike previous methods, DRIL interleaves BC updates and RL updates, using a reward based on the variance across the ensemble. The BC updates help the learner policy mimic the expert’s state-action distribution. The RL updates guide the learner policy towards the expert’s state-action distribution if it deviates. These BC updates are necessary for this algorithm’s theoretical guarantees.

**BC-GAIL:** Behavior Cloning -Regularized Generative Adversarial Imitation Learning

---

<sup>1</sup>see section 6.9.12 for links

(BC-GAIL) was proposed by [137] and consists of interleaving GAIL RL updates with BC updates. The original work anneals the weight between the two cost terms during training. We do not anneal to be comparable with DRIL.

**BC-RED:** The authors of DRIL [44] presented a counterexample of the importance of interleaving BC updates with RL updates when using an estimation reward such as RED. Behavior Cloning -Regularized Rand Expert Distillation (BC-RED) consists of interleaving the RED RL updates with BC updates.

**BC-AIRL:** Behavior Cloning -Regularized Adversarial Inverse Reinforcement Learning (BC-AIRL) combines AIRL and BC-GAIL, by interleaving the AIRL reward function RL updates with BC updates.

## 6.4 Related Work

We focus on additional IL algorithms that are similar to the algorithms we study in this section and discuss other related work in the extended related work section (see section 6.9.6). There have been more IL algorithms proposed than the ones discussed so far that learn a reward function from demonstration data and optimize it with RL. Generative Moment Matching Imitation Learning (GMMIL) [150] constructs reward functions using maximum mean discrepancy. Primal Wasserstein Imitation Learning (PWIL) [73] uses the optimal transport distance between the current policy and the expert’s policy to construct a reward function. We do not include these algorithms in our study because it is unclear how to apply them to pixel tasks.

Soft-Q imitation learning (SQIL) [227] adapts off-policy Q-learning by assigning reward +1 to expert transitions and reward 0 to transitions from the current policy. Discriminator Actor critic (DAC) uses an off-policy algorithm to make GAIL sample efficient Kostrikov et al. [157].

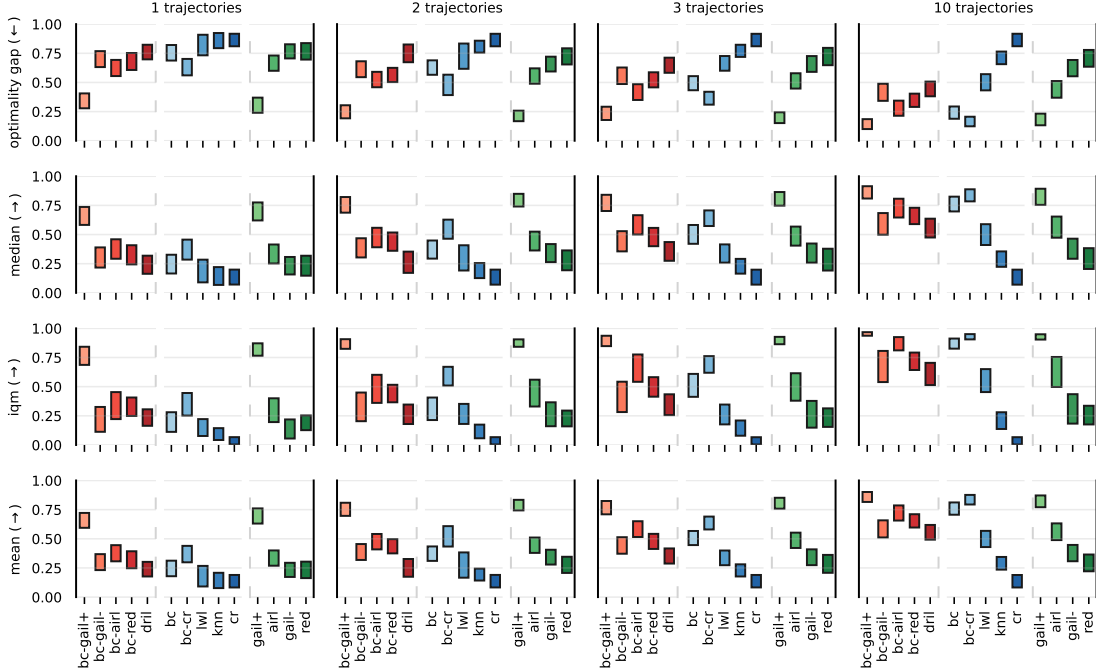


Figure 6.2: Aggregate results for continuous control tasks **with sub-sampled trajectories**. Scores are normalized between a random and expert agent performance. Bars indicate 95% confidence intervals computed using stratified bootstrapping. Red colors perform BC and RL updates, blue colors are baselines and green only perform RL updates. Arrows indicate higher ( $\uparrow$ ) or lower ( $\downarrow$ ) is better.

Sasaki et al. [242] and Kostrikov et al. [158] are two off-policy algorithms that minimize the Jensen-Shannon divergence between the occupancy measures of learner and expert. Sample-efficient Adversarial Mimic (SAM) [34] maintains three functions that approximate a reward function, state-action value function, and policy. Neural Density Imitation (NDI) performs density estimation to estimate the expert occupancy measure, then performs an off-policy RL update with an entropy regularization on the occupancy measure [151]. Swamy et al. [276] shows that many proposed algorithms for imitation learning are doing some form of moment matching with respect to distributions or features and proposes an algorithm that uses an off-policy algorithm. Liu et al. [180] uses energy-based models (EMB) in optimizing imitation learning. We strictly use an on-policy RL optimizer (PPO), so our study does not include off-policy methods.

Adversarial imitation learning (AIL) algorithms solve a min-max game with discriminators and generator networks, similar to (GANs) [107]. Orsini et al. [202]<sup>2</sup> did a thorough study of various AIL algorithms Blondé et al. [35], Ghasemipour et al. [103], Wang et al. [297], Xiao et al. [310], Xu and Denil [311]. We use their results when deciding what AIL algorithms to include.

## 6.5 Experiments

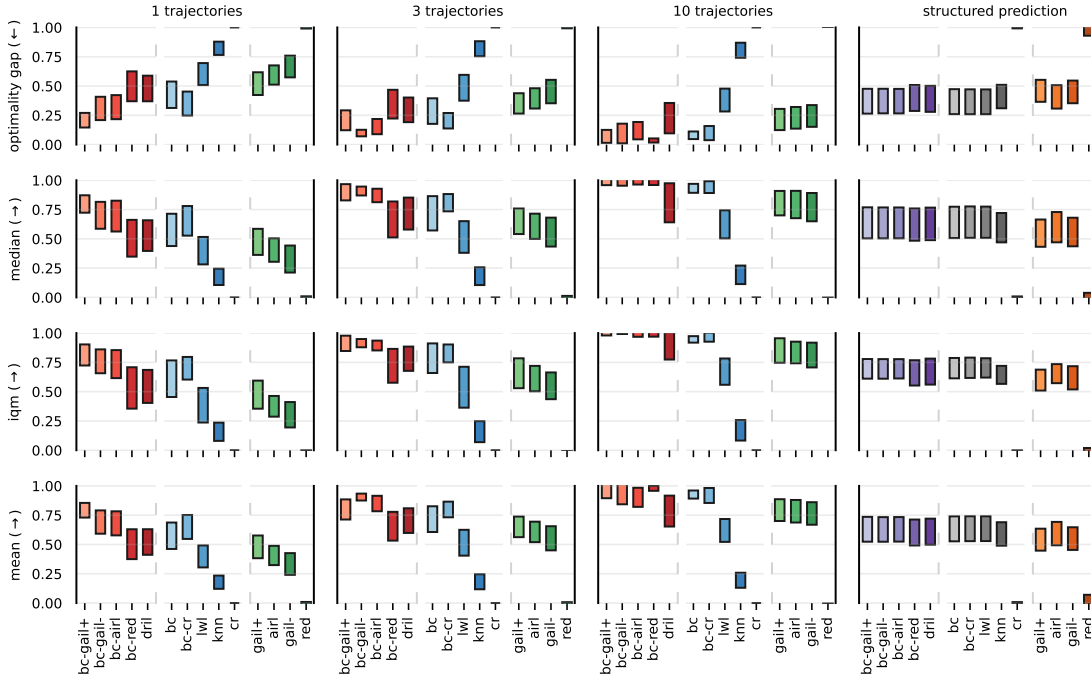


Figure 6.3: Aggregate results for pixel tasks. Scores are normalized between a random and expert agent performance. Bars indicate 95% confidence intervals computed using stratified bootstrapping. Red colors perform BC and RL updates, blue colors are baselines and green colors only perform RL updates. Arrows indicate higher (→) or lower (↕) is better.

We consider three broad categories of tasks: continuous control, pixel, and structured prediction (see table 6.2). We chose Proximal Optimal Policy (PPO) [251] as our RL algorithm because it is an on-policy algorithm that supports both discrete and continuous actions. Maintaining

<sup>2</sup>Benchmarks adversarial imitation learning algorithms

the same underlying RL algorithm across all tasks makes the difference IL algorithms consider in this study performance mainly due to their algorithmic design choices. See the section 6.9.19 for additional experiments details.

**Evaluation Setup:** We normalize the scores between random and expert performance on each task. After normalization, we use performance profiles to compare all algorithms scores[5]. There are four aggregated metrics in performance profiles: mean, median, interquartile mean (IQM), and optimality gap. Mean is the average score of a task across five runs but is often dominated by the performance of outlier tasks. Optimality gap is an alternative to mean, which measures the amount an algorithm fails to meet a minimum score of  $\gamma$ . We set  $\gamma = 1$  because imitation learning algorithms aim to match expert performance. Median is the middle score of an order list of task scores but is a poor indicator of overall performance because it does not include all the scores for calculation (i.e., changes in scores may not affect the median value). IQM is an alternative to the median, discarding the bottom and top 25% runs and using the remaining 50% runs to calculate the mean score.

## 6.6 Results

This section reports results for continuous control, pixel, and structured prediction tasks. We provide additional details and results in section 6.9.23. Furthermore, in all figures, we use three colors to represent three groups of algorithms: red represents algorithms that interleave RL updates with BC updates, blue represents baseline algorithms, and green represents algorithms that only perform RL updates.

### 6.6.1 Continuous Control Results

An important detail not mentioned in the original GAIL paper but seen in the original GAIL codebase is that the expert trajectories are sub-sampled (see section 6.9.19). Because of this we compare all algorithms in two settings, when the expert provides full trajectories (figure 6.1) and sub-sampled trajectories (figure 6.2). Although there have been conflicting results regarding BC, we notice in the full trajectories in figure 6.1, BC performs well even with 1 trajectory, leaving no room for improvement. Whereas, in the sub-sampled trajectories results in figure 6.2, BC performs badly when there are a low number of trajectories provided. This shows that sub-sampling alone is insufficient to reduce BC performance because the number of trajectories is equally important.

We evaluate the performance of all algorithms across both settings. The DRIL authors reported better results than GAIL, but the authors do not report the number of seeds they used in their experiments. Similar to DRIL authors, we notice that DRIL performs competitively to GAIL across both settings. Furthermore, we were unable to reproduce the results in RED, hence the poor performance in figure 6.1 and figure 6.2, which is consistent with Blondé et al. [36] and Jena et al. [136]. In the original RED codebase, the authors use separate hyperparameters for each environment. Instead, we use one set of hyperparameters for all environments similar to Blondé et al. [36]. The original BC-GAIL+ results were compared against BC in the full-trajectory setting using version v2 of MuJoCo. We use version v3 of MuJoCo task <sup>3</sup> instead of version v2. Although BC performs better on version v3 tasks than version v2, we notice that BC-GAIL+ still performs competitively to all other algorithms.

We see some trends when comparing the three algorithm groups using their respective

---

<sup>3</sup><https://github.com/openai/gym/issues/1541>

color schemes. First, the baseline algorithms in blue, simply copying (KNN) or weighting (LWL) the demonstration data, does not perform well. However, the new BC-CR baseline that takes advantage of environment interactions performs similar to and sometimes better than BC. Moreover, in both settings, BC-CR performs similar or better than all IL algorithms except for BC-GAIL+ or GAIL+, showing that a *simple* constant reward function is better than learning the reward function from demonstration data. Furthermore, all algorithms in red that interleave BC updates with RL updates perform better than their counterparts that do not interleave RL updates in green.

### 6.6.2 Pixel Results

A subset of algorithms in this study were evaluated on pixel tasks. The left-side of figure 6.3 shows pixel tasks results. In the original GAIL codebase, the authors did not evaluate on any pixel task. However, despite conflicting results on GAIL working [68, 136, 287] and not working on pixel task [44, 220, 315], we noticed that GAIL performs well when using DrQ data augmentation. The DRIL authors showed that GAIL performed significantly worse than DRIL on pixel Atari task and reported a bug in their official codebase regarding how the expert demonstration data was collected. We notice that DRIL performs slightly better than GAIL when provided a low number of trajectories and is similar to GAIL as the number of trajectories increases. The original RED codebase was not evaluated on pixel tasks. RED performs worse than all algorithms in figure 6.3.

The trends in results for pixel tasks are similar to continuous control tasks. The two simple baselines KNN and LWL perform worse than most IL algorithms. Furthermore, both BC and BC-CR (i.e., the highest blue algorithms in figure 6.3) are very competitive baselines,



performing similar to or better than all most IL algorithms. BC-CR performs better than BC and all algorithms that do not interleave BC updates in green. Moreover, all algorithms that interleave BC updates in red perform better than their green counterparts is consistent with continuous control results. These results show that these IL algorithms can generalize beyond continuous control tasks.

### 6.6.3 Structured Prediction Results

Unlike the previous two tasks, structure-prediction tasks use discrete actions instead of continuous actions. We notice in the rightmost column of figure 6.3 that simple memorization (KNN) and weighting demonstration distance (LWL) perform competitively to all IL algorithms. We also see that all IL algorithms perform well on structured prediction tasks except for RED and CR. A slight improvement exists amongst algorithms that interleave BC updates with RL updates versus their counterparts.

## 6.7 Discussion

**Interleaving BC Updates.** We introduce the Uncertainty cost MDP (figure 6.4) introduced in chapter 5 to understand the importance of interleaving BC updates. The agent always starts in state  $s_1$  (blue state) and only gets a reward in  $s_3$  (green state). Lets assume three things: the expert is slightly suboptimal in-state  $s_2$  (red state), i.e.,  $\pi^*(\cdot|s_2) = (0.1, 0.9)$ , a perfect optimizer, and perfect realizability. If both transitions  $(s_2, a_0)$  and  $(s_2, a_1)$  appear in the demonstration data then it is possible for algorithms in this study to assign zero cost to both transitions. We could learn two policies  $\hat{\pi}_1$  and  $\hat{\pi}_2$ , by optimizing the RL reward function learned from demonstration data, that are suboptimal in state  $s_2$ , i.e.  $\hat{\pi}_1(\cdot|s_2) = (1, 0)$  and  $\hat{\pi}_2(\cdot|s_2) = (0.2, 0.8)$ . Furthermore,

$\hat{\pi}_1$  and  $\hat{\pi}_2$  are assigned similar low demonstration data RL cost, where  $\hat{\pi}_1$  cycles forever between  $s_1$  and  $s_2$ , and  $\hat{\pi}_2$  with high-probability reaches the goal state  $s_3$ . However,  $\hat{\pi}_1$  has higher BC cost than  $\hat{\pi}_2$  because  $\hat{\pi}_2$  matches the demonstration data at  $s_2$ . This shows that the BC cost disambiguates policies that are within distribution of the demonstration data that exhibit different behaviors.

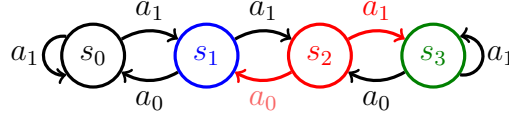


Figure 6.4: Uncertainty cost MDP seen in chapter 5

**Modern Covariate Shift.** IL algorithms have been applied to address the NLP exposure bias problem [79, 225] (i.e., the feedback-driven covariate shift problem of BC [212, 232]). DAGGER (i.e., DAD [290] applied in Schedule Sampling [28])<sup>4</sup> is the most popular interactive IL algorithms amongst, Zhang et al. [323], Goyal et al. [108], MIXER [225], and Beam-Search Optimization [305] that attempt to address the NLP exposure bias issue. But DAGGER suffers from the counterexample mentioned in [235] and Schedule Sampling is considered an improper training objective [130] for NLP generative modeling. Moreover, these interactive IL algorithms assume access to a dynamic-oracle (i.e., interactive expert), but properly defining the oracle actions when the generated sequence no longer exists in the demonstration data is difficult [130]. Besides interactive IL algorithms, other IL algorithms have succeeded on complex NLP problems, using only demonstration data.

The most widely used RL algorithms in NLP optimize a reward such as Bleu [205], ROUGE-2 [176] or some other evaluation metric that is non-differentiable. However, it is possible to increase the model score without increasing the quality and readability of the model output [179]. Most of the highly influential RL in NLP papers: Li et al. [173, 174], Paulus et al.

<sup>4</sup><https://nlpers.blogspot.com/2016/03/a-dagger-by-any-other-name-scheduled.html>

[207], Ranzato et al. [225], Wu et al. [307, 309] are a part of the IL algorithms discussed in this study. These papers optimize RL updates with a fixed reward NLP metric interleaved with BC updates, akin to IL algorithms which interleave BC updates in this study. They interleave BC updates to stabilize training, decrease convergence time, and create more human-like text. These algorithms have succeeded in dialogue generation, neural machine translation, abstract summarization, and Language Modeling. A future research question is how can these IL algorithms solve exposure bias problems in large-scale NLP models, such as text-degeneration issues [58, 301]? Studying exposure bias problems in large-scale NLP models provides a practical problem that can be studied in a low-cost and safe simulator.

**Sub-sampling.** Although sub-sampling has been accepted as the standard for comparing algorithms, we argue that this setting is unnatural. A more natural setting is the *low-resource* setting often studied in NLP. The *low-resource* [118] setting in NLP, assumes a lack task-specific labeled data. Unlike sub-sampling trajectories in demonstration data, this setting assumes the availability of full-trajectories. However, the data labels are noisy [88], scarce [71] or low-quality non-expert labeled data [100].

## 6.8 Conclusion

This paper studies IL algorithms that learn reward functions from demonstration data and optimize them with RL. We found that for these IL algorithms, continuous control tasks are "easy" [266]. The traditional baseline algorithm BC can solve continuous control tasks. We investigate other tasks such as structured prediction and pixel continuous control and notice that BC performs well. In fact, *low-resource* imitation learning (see section 6.7) seems to be the only setting where BC performs badly on tasks. We notice that our new baseline algorithms

BC-CR performs similar to and sometimes better than BC and other IL algorithms on all tasks. Furthermore, interleaving BC updates with these algorithms seems essential (similar to Chang et al. [53], Fujimoto and Gu [97] findings) and relates to highly influential RL NLP algorithms. We hope this study inspires ideas in this research area.

## 6.9 Extend Details

### 6.9.1 Reinforcement Learning Background

In order to compare algorithms, we require the ability to optimize a reinforcement learning algorithm. The reinforcement learning algorithm that we use across all experiments is Proximal Policy Optimization (PPO) Schulman et al. [252]. We follow the works of Zhang [322], Andrychowicz et al. [9], and Kostrikov [155] when setting our parameters for PPO. For completeness, we include a brief description of each component of PPO and share the default hyperparameters used across all tasks. We also provide convenience links that allow navigating between the description of the hyperparameter and the default value inspired by Orsini et al. [202] and [9].

Trust region style policy gradient methods [143, 144, 249, 252] attempt to constrain the local variation of the parameters in policy-space by restricting the distributional distance between successive policies when updating. In particular, PPO Schulman et al. [252] tries to enforce a trust region with a different objective that does not require computing a projection. PPO proposes replacing the KL-constrained objective of TRPO Schulman et al. [249] by clipping the objective function

$$\mathcal{L}_{\text{PPO}}^\epsilon = - \min \left[ \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \hat{A}_t^\pi, \text{clip} \left( \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}, \frac{1}{1+\epsilon}, 1+\epsilon \right) \hat{A}_t^\pi \right]$$

where  $\epsilon \in R$  is a hyperparameter (R1). We adopt the nomenclature from Andrychowicz et al. [9] to describe the various RL hyperparameters which we use. Please see Andrychowicz et al. [9] and Schulman et al. [252] for a more in-depth discussion of PPO. When the action space is continuous we use a global standard deviation (R2, R3) and set the initial standard deviation

to a non-zero value (R4). Below is a description of various PPO hyperparameter that we used in our experiments.

### 6.9.2 PPO Hyperparameters

PPO alternates between a data collection step and a policy and value function optimization step. Additionally, the PPO algorithm involves several choices: how many environments to run in parallel (R5), how long collected rollouts should be (R6), and hyperparameters related to the gradient-based updates. When performing policy and value function optimization, we have to set the minibatch size (R7) and the number of passes over the random minibatch to perform value and policy updates (R8). We also perform per minibatch advantage normalization (R9, R10). This idea was discussed in Andrychowicz et al. [9] and seen in various common codebases such as Raffin et al. [221]. We use the recommended discount factor (R11) and gradient clipping parameter (R12) discussed in Andrychowicz et al. [9]. We use the GAE advantage estimator (R13, R14) when approximating the value function. For computing the value function loss we use MSE (R15).

We use the Adam Kingma and Ba [153] gradient-based optimizer (R16) for all our experiments and set the adam hyperparameters (R17, R18, R74) to recommend values in Andrychowicz et al. [9]. For further understanding of adam please see Kingma and Ba [153] for more details. We use a linear learning rate decay (R20), where the learning rate decays from the initial learning to zero.

### 6.9.3 Neural network architecture

We use multilayer perceptron (MLPs) neural networks for continuous control and structured-prediction tasks and convolutional neural networks for pixel tasks to represent policies and value functions. For continuous control tasks, we use separate networks for the policy and value function R21. Whereas for structured-prediction tasks, we use a shared network for the policy and value function. For continuous control and structured-prediction tasks, both the policy (R22, R23) and value (R24, R25) networks we use the same width, depth, and activation function (R26). For pixel tasks we use a single convolutional (R27) network with two linear heads, one for the policy and one for the value function R28 and the ReLU activation function (R26).

We use orthogonal weight normalization to initialize all the layers in MLP and CNN networks except for the last layer (R29, R30). For the initialization of the last layer in the policy MLP head (R31) and the last layer in the value MLP value head (R32), we choose them based on Andrychowicz et al. [9].

### 6.9.4 Data Normalization

Data normalization is a subtle detail that can significantly impact performance in RL. For continuous control tasks, we perform observation normalization (R33) (i.e., average), which involves normalizing observations based on the observations seen so far. We then normalize observations by subtracting the mean and dividing the standard deviation for each dimension. We clip the normalized observations to the range  $[-o_{\max}, o_{\max}]$  where  $o_{\max}$  is a hyperparameter (R34).

For pixel tasks, we perform image normalization (R33) by dividing each pixel by 255 (i.e.,

scale). We then construct inputs by stacking the 3 consecutive past frames and repeating actions for each environment according to Hafner et al. [112]. We use the Data regularized Q (DrQ) [313] (R35) data augmentation technique. It works by padding each side of the image by 4 pixels and then selecting random  $84 \times 84$  crops, yielding the original image shifted by  $\pm 4$  pixels.

For continuous control and pixel continuous control tasks, we clip the actions (R36) because most environments expect actions to be within a bounded range. We do not perform reward normalization (R37). Lastly, we do not perform any data normalization for a structured-prediction task.



### 6.9.5 Default Reinforcement Learning Hyperparameters

Table 6.3: Default settings used in RL experiments.

	Name	Feature	Pixel	SP
R5	num_envs	1	1	1
R6	iteration_size	2048	2048	2048
R8	num_epochs	10	10	10
R7	batch_size	32	32	64
R9	batch_mode	Recompute	Recompute	Recompute
R11	Discount factor $\gamma$	0.99	.99	0.99
R13	advantage_estimator	GAE	GAE	GAE
R14	GAE $\lambda$	0.95	0.95	0.95
R15	Value function loss	MSE	MSE	MSE
R1	PPO $\epsilon$	0.3	0.3	0.2
R16	Optimizer	Adam	Adam	Adam
R74	RL Adam learning rate	3e-04	3e-04	5e-04
R18	Adam momentum	0.9	0.9	0.9
R17	Adam $\epsilon$	1e-05	1e-05	1e-05
R20	Learning rate decay	Anneal	Anneal	Anneal
R21	Shared MLPs?	Separate	-	Shared
R22	Policy MLP width	64	512	100
R24	Value MLP width	64	512	100
R23	Policy MLP depth	2	1	2
R25	Value MLP depth	2	1	2
R26	Activation	Tanh	-	Relu
R27	Policy CNN layers	-	4	-
R28	Shared CNNs?	-	True	-
R29	MLP Initializer Gain	1.	1.	1.
R30	CNN initializer Gain	-	1.	-
R31	Last policy layer scaling	0.01	0.01	0.01
R32	Last value layer scaling	0.01	.01	1.0
R2	Global standard deviation?	True	True	-
R3	Standard deviation $T_\rho$	Softplus	Softplus	-
R4	Initial standard deviation $i_\rho$	0.5	0.5	-
R36	Action transformation $T_u$	Clip	Clip	-
R33	Input normalization	Average	Scale	-
R34	Input clipping	10.0	10.0	-
R37	Reward normalization	-	-	-
R10	Per minibatch adv. norm.	True	True	True
R12	Gradient clipping	0.5	0.5	0.5
R35	RL Data Augmentation	-	DrQ	-

### 6.9.6 Extended Related Work

This section focuses on the remainder of the related work not mentioned in section 6.4. This section discusses imitation learning benchmarks and algorithms that interleave RL updates with BC updates.

Several recent works have benchmarked different aspects of IL related to our study. [202] performed a thorough investigation of design choices for adversarial IL algorithms in the continuous control domain with sub-sampled trajectories. Orsini et al. [202] most surprising findings are that most discriminator regularization techniques perform the same, and there is a difference in performance depending on whether the demonstration data is synthetic or human. This study focuses on more algorithms than adversarial IL. Arulkumaran and Lillrank [10] compared various IL algorithms cost functions on continuous control tasks, and most algorithms performed the same, except for the kernel-based GMMIL, which showed improvement. Kanervisto et al. [145] focuses on benchmarking BC on video games using demonstration data, showing that there was a gap between the BC performance and human performance. Zhang et al. [320] focuses on BC performance on video games as well. Both Toyer et al. [285] and Freire et al. [93] introduce several new tasks in order to benchmark how algorithms generalize the demonstrator of the demonstration intent to new tasks where different kinds of distributions shifts exist. Memmesheimer et al. [188] creates new environments where humans perform daily activities in the real world and uses them for benchmarking. Our work explicitly investigates the role of sub-sampling in continuous control tasks. Furthermore, we also evaluate all methods on continuous control, pixel, and structured prediction tasks. Moreover, we focus on the importance of interleaving BC updates with RL updates.

### 6.9.7 Existing Imitation Learning Algorithms Details

Some imitation learning algorithms require access to a reinforcement learning optimizer in this study. For those that do, we use the PPO reinforcement learning algorithm discussed in section 6.9.1. Unless otherwise specified, these algorithms use the default PPO values for continuous control, pixel, and structured prediction mentioned in (section 6.9.4). During evaluation for all algorithms, we act greedily, either taking the mean if the action space is continuous and taking the argmax if the action space is discrete (R38).

For continuous control we experiment with clipping the action space when optimizing each IL objective (see R36) (R39). We warmed start the learning process using a trained behavior cloning model (R40) for each algorithm across all task categories. For continuous control tasks, we normalize the learner input using the demonstration data to compute mean and standard deviation denoted as Fixed in the table below, or we do not apply any normalization to the learner input (R41) denoted as None.

We experiment with clipping the action space for pixel tasks when optimizing each IL algorithm objective similar to continuous control tasks. We experiment with applying DrQ data augmentation before using the input images for training each IL algorithm.

This section discusses the hyperparameters we considered while tuning each IL algorithm and the default hyperparameters used in the final evaluation. Furthermore, the environments used in this study for tuning each algorithm can be found in section 6.9.19.

### 6.9.8 K-Nearest Neighbor(KNN):

$k$ -nearest neighbors [214, 261] is a nonparametric instance-based learning algorithm that leverages demonstration data from an expert by strictly comparing the learners current state with instances seen in the demonstration data. The demonstration data contains state-action pairs  $(s_i, a_i) \in \mathcal{D}_{\text{exp}}$ . During evaluation, the learner  $\hat{\pi}$  observes a state  $s_t$  and queries using a distance metric (e.g. Euclidean distance) to find the  $k$  closet states in the expert demonstration data to state  $s_t$ . The resulting  $k$  closest states is denoted by  $\mathcal{D}_{\text{exp}}^k = \{(s_i, a_i)\}_{i=1}^k$  where  $|\mathcal{D}_{\text{exp}}^k| < |\mathcal{D}_{\text{exp}}|$ . We can define the action resulting from  $k$ -nearest neighbors as the following if the actions are continuous:

$$\hat{\pi}(\cdot|s_t) = \frac{1}{k} \sum_{i=0}^k a_k$$

and the following if the actions are discrete:

$$\hat{\pi}(\cdot|s_t) = \underset{\bar{a} \in |\mathcal{A}|}{\operatorname{argmax}} \sum_{i=0}^k \delta(\bar{a}, a_i)$$

where  $\delta$  is the Kronecker Delta function. There has been recent work in the area of reinforcement learning to suggest that nonparametric models work in some environments [187].

We perform a random Gaussian projection from 84x84 pixel space to 100-dimensional vectors for pixel tasks. The Johnson-Lindenstrauss lemma nearly preserves distance when projecting points from a high-dimensional space into a lower-dimensional space [138]. We reduce the dimensionality in this manner because computing  $k$ -nearest neighbors on high-dimensional data is expensive, whereas projecting the data down while preserving distance is much more efficient.

## KNN Hyperparameters:

We use the default hyperparameters in table 6.4 below for our experiments. KNN has two hyperparameters; the first pertains to the number of neighbors used from the demonstration data to compute an action to take (R42). The second hyperparameter is applying either DrQ data augmentation or not applying data augmentation to the pixel input before applying a random Gaussian project to a lower-dimensional space (R43). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous Control Tasks:

IL Input normalization (R41): {None, Fixed}

- Pixel Tasks:

KNN Data Augmentation (R43): {None, DrQ}

Table 6.4: Default settings used in k-nearest neighbor.

Name	Feature				Pixel	SP
	Pybullet		Mujoco		All	All
	Sub	Full	Sub	Full		
R42	Number of nearest neighbor					
R43	KNN Data Augmentation					
R41	IL Input normalization					
	1	1	1	1	1	1
	-	-	-	-	DrQ	-
	None	None	Fixed	Fixed	Scale	-

### 6.9.9 Locally Weight Learning (LWL):

Locally Weighted Learning [11, 131] is a nonparametric instance-Based learning algorithm that uses data demonstration from an expert similar to  $k$ -nearest neighbors. During evaluation the learner  $\hat{\pi}$  see a state  $s_t$  and queries using a distance metric (e.g. Euclidean distance) to find the  $k$  closet states in the expert demonstration data to the state  $s_t$ . The resulting  $k$  closest states is denoted by  $\mathcal{D}_{\text{exp}}^k = \{(s_i, a_i)\}_{i=1}^k$  where  $|\mathcal{D}_{\text{exp}}^k| < |\mathcal{D}_{\text{exp}}|$ . We denote the estimation cost of the learner queried state  $s_t$  as:

$$c(s_t) = \sum_{i=0}^k (\hat{\pi}(\cdot|s_t) - a_i)^2$$

In LWL, we can either weight the data directly (see equation (6.1)) or weight the error criterion (see equation (6.2)) used to choose  $\hat{\pi}(\cdot|s_t)$ . For example, if we want to weight the data directly, we could use kernel function  $K()$  such as the Gaussian Kernel  $K(d) = e^{-d^2}$  where  $d$  is some distance metric (e.g., Euclidean distance) to weight the distance of each point to the queried state  $s_t$ . Then the weights can be used in a weighted average to compute the learner  $\hat{\pi}$  actions:

$$\hat{\pi}_t(\cdot|s_t) = \frac{\sum_{i=0}^k a_i \cdot K(d(s_i, s_t))}{\sum_{i=0}^k K(d(s_i, s_t))} \quad (6.1)$$

If we want to weight the error criterion, we could, for example, do distance weighting error criterion:

$$J_{\text{LWL}}(\pi) = \sum_{t=1}^T \mathbb{E}_{s_t \sim d_t^{\pi_{t-1}}} [c^k(s_t)] \quad \textbf{where} \quad c^k(s_t) = \sum_{i=0}^k (\pi_t(\cdot | s_t) - a_i) \cdot K(d(s_i, s_t))^2 \quad (6.2)$$

Formally, let  $\hat{\pi}_t$  denote the policy learned and executed at time step  $t$  after minimizing the cost function  $c^k$  in equation (6.2) with respect to the  $k$  closest states. At each time step  $t$ , the cost function  $c^k$  can be minimized using weighted regression where the weight corresponds to the kernel function  $k(\cdot)$ . The resulting learner policy  $\hat{\pi}$  is a non-stationary policy defined for  $t = 1, \dots, T$  steps, where  $\hat{\pi}_t$  is trained on the state distribution  $d_t^{\pi_{t-1}}$  of the previous policy  $\pi_{t-1}$  and all the other previous policies remain the same.

Similar to KNN discussed in section 6.9.8, for pixel tasks, we perform a random Gaussian projection from 84x84 pixel space to 100-dimensional space [138]. We do this to make the LWL algorithm more computationally efficient.

LWL Hyperparameters:

We use the default hyperparameters in table 6.5 for our experiments. We perform hyperparameter tuning over the possible kernel  $K$  (R47) and kernel bandwidth (R48) used for each task. The kernels that we consider when running experiments are: Mean kernel, Neural Episodic Control (Nec) kernel [218], Inverse distance (Id) kernel [11, 306], Bell Kernel (Bell) [66], and Gaussian kernel (Gaussian) [11, 294]. We also experiment with applying the DrQ data augmentation to the pixel tasks images before performing a random projection of images to a lower-dimensional state space (R46). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous Control Tasks:

LWL Kernel (R47): {Mean, Nec, Id, Bell, Gaussian}

Number of nearest neighbor (R49): {50, 75, 100, 150, 200, 250, 300, 500, 750}

LWL Kernel Bandwidth (R48): {1e-02, 1e-03, 1e-04, 1e-05, 1e-06, 1e-07}

IL Action transformation  $T_u$  (R39): {None, Clip}

IL Input normalization (R41): {None, Fixed}

- Pixel Tasks:

LWL Kernel (R47): {Mean, Nec, Id, Bell, Gaussian}

Number of nearest neighbor (R49): {50, 75, 100, 150, 200, 250, 300, 500, 750}

LWL Kernel Bandwidth (R48): {1e-03, 1e-04, 1e-05, 1e-06}

LWL Data Augmentation (R46): {None, DrQ}

- Structured-Prediction Tasks

LWL Kernel (R47): {Mean, Nec, Id, Bell, Gaussian}

Number of nearest neighbor (R49): {50, 75, 100, 150, 200, 250, 300, 500, 750}

LWL Kernel Bandwidth (R48): {1e-02, 1e-03, 1e-04, 1e-05}

Table 6.5: Final settings used in locally weighted learning experiments.

Name	Feature				Pixel	SP
	Pybullet		Mujoco		All	All
	Sub	Full	Sub	Full		
R47 LWL Kernel	Id	Id	Id	Id	Id	Id
R48 LWL Kernel Bandwidth	1e-06	1e-03	1e-04	1e-03	1e-06	1e-03
R49 Number of nearest neighbor	150	250	50	250	500	100
R39 IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R46 LWL Data Augmentation	-	-	-	-	DrQ	-
R41 IL Input normalization	None	None	Fixed	Fixed	Scale	-



### 6.9.10 RL with constant reward: (CR)

Imitation learning techniques that optimize reinforcement learning using a constant reward function have been successful in prior work SQIL [227] and RED [296]. The authors of SQIL explicitly optimize a piecewise reward function that is either +1 or 0. Though RED authors do not explicitly state that they use a constant reward, Arulkumaran and Lillrank [10] showed that the original RED implementation used hyperparameters that resulted in the optimized reward function being close to 1.

Finally, DAC [157] showed that having a survival bonus (i.e., per-step positive reward) encourages the agent to survive longer in certain environments. DAC also showed that having a strong prior on the reward function may lead to good performance even without expert demonstration data  $\mathcal{D}_{\text{exp}}$ . The expert does not provide any demonstration data for this baseline but instead provides a biased reward function. Combining ideas from SQIL [227], RED[296], and DAC[157] we introduce a simple imitation learning baseline algorithm that optimizes reinforcement learning using a constant reward function. This Reinforcement Learning with a Constant Reward baseline optimizes the following objective:

$$J_{\text{CR}}(\pi) = \mathbb{E}_{(s,a) \sim \mu^\pi} [r(s, a)]$$

where the immediate reward  $r(s, a) > 0$  and fixed.

CR Hyperparameters:

We use the default hyperparameters in table 6.6 for our experiments. For the reinforcement learning with a constant reward baseline, we experiment with different fixed reward values (R50).

We also experimented with using DrQ data augmentation (R51) on pixel tasks. For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous Control Tasks:

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04, 1e-05}

Constant reward value (R50):{1e0, 1e-01, 1e-02}

- Pixel Tasks:

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

Constant reward value (R50):{1e0, 1e-01, 5e-01}

CR Data Augmentation (R51): {None, DrQ}

- Structured-Prediction Tasks

RL Adam learning rate (R74):{1e-02, 1e-03, 3e-04, 1e-04}

Constant reward value (R50):{1e0, 1e-01, 1e-02, 5e-01}

Table 6.6: Final settings used in reinforcement learning with constant reward experiments.

Name	Feature				Pixel	SP
	Pybullet		Mujoco		All	All
	Sub	Full	Sub	Full		
R74 RL Adam learning rate	1e-03	1e-03	1e-03	1e-03	1e-03	3e-4
R50 Constant reward value	1e0	1e0	1e0	1e0	1e0	0.5
R51 CR Data Augmentation	-	-	-	-	DrQ	-
R41 IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38 evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

### 6.9.11 Behavior Cloning (BC):

Behavior Cloning [213] is the standard imitation learning baseline. It uses demonstration data as supervised learning data to learn a policy  $\hat{\pi}$  that predicts expert actions  $a^*$  given expert states  $s^*$ , under the expert state-action distribution  $\mu^{\pi^*}$ . At each iteration of training, we collect a training pair  $(\pi(s^*), \pi^*(s^*))$ , where  $\pi^*(s^*)$  is the expert action (i.e. class label) for state  $s^*$ . Let  $\ell(\pi(s^*), \pi^*(s^*))$  denote the surrogate loss of executing  $\pi$  in state  $s^*$  with respect to action  $\pi^*(s^*)$ . For theoretical analysis, this surrogate loss can be any convex loss function used for training the classifier, for example, hinge loss or total variation distance. Often in practice this surrogate loss can be maximum-likelihood  $\ell(\pi(s^*), \pi^*(s^*)) = -\log \pi(a^*|s^*)$  [79, 147, 197]. Formally, using any standard supervised learning algorithm, we can learn a policy

$$\hat{\pi} = \underset{\pi \in \Pi}{\operatorname{argmin}} \mathbb{E}_{(s^*, a^*) \sim \mathcal{D}_{\text{exp}}} [\ell(\pi(s^*), \pi^*(s^*))]$$

A drawback of this approach is that it ignores the fact that the state-action distribution of the expert  $\mu^{\pi^*}$  and the learner  $\mu^{\pi}$  are different. This is an issue when the learner is unable to mimic the expert perfectly resulting in classification error  $\epsilon$  occurring, i.e.  $\mathbb{E}_{(s^*, a^*) \sim \mu^{\pi^*}} [\ell(\pi(s^*), \pi^*(s^*))] \leq \epsilon$ . The subtle problem is that, the sequential decision-making tasks contain inherent feedback-loops, where past action  $a_{t-1}$  affect our learner decision-making at the next time step input state  $s_t$ . This phenomenon results in an issue known as feedback-driven covariate shift, where the states the learner experiences when interactive in an environment diverges from the demonstration data [142, 213]. This is formalized below:

**Theorem 6.9.1** (Theorem 2.1 in Ross and Bagnell [232]). .

*Let  $J_{\text{BC}}(\pi) = \mathbb{E}_{(s^*, a^*) \sim \mu^{\pi^*}} [\ell(\pi(s^*), \pi^*(s^*))] \leq \epsilon$  be the bounded on-policy training error, where*

$\ell$  is the 0-1 loss (or an upper bound). We have  $J(\hat{\pi}) \leq J(\pi^*) + T^2\epsilon$

#### BC Hyperparameters:

We use the default hyperparameters in table 6.7 for our experiments. We perform a thorough hyperparameter sweep over the BC learning rate (BC learning rate (R52)) and apply early stopping for all experiments (R53). Furthermore, for pixel tasks we experiment with the DrQ data-augmentation technique [313] (R54). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous Control Tasks:

BC learning rate (R52): {1e-02, 1e-03, 1e-04, 1e-05, 2e-03, 3e-03, 4e-03, 5e-03, 2.5e-02, 2.5e-03, 2.5e-04, 2.4e-05}

IL Action transformation  $T_u$  (R39): {None, Clip}

IL Input normalization (R41): {None, fixed}

- Pixel Tasks:

BC learning rate (R52): {1e-04, 2.5e-04, 3e-04, 4e-04, 5e-04}

BC Data Augmentation (R54): {None, DrQ}

- Structured-Prediction Tasks

BC learning rate (R52): {1e-03, 1e-04, 1e-05, 2e-05, 3e-05, 1e-06}

Table 6.7: Default settings used in behavior cloning experiments.

Name	Feature				Pixel	SP
	Pybullet		Mujoco		All	All
	Sub	Full	Sub	Full		
R52 BC learning rate	3e-03	1e-03	2e-03	1e-03	3e-04	1e-03
R39 IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R53 BC early stopping	True	True	True	True	True	True
R54 BC Data Augmentation	-	-	-	-	DrQ	-
R41 IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38 evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

### 6.9.12 Generative Adversarial Imitation Learning (GAIL):

General adversarial imitation learning (GAIL) [126] draws connections between Generative Adversarial Networks (GANs) [107] and Maximum Entropy Inverse Reinforcement Learning (IRL) [327]. GAIL uses a discriminator denoted by  $D_\theta(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$  to distinguish between the learner and expert state-action pairs. While the learner is rewarded for deceiving the discriminator, which is maximized using some reinforcement learning optimization procedure. The full GAIL objective is the following:

$$\begin{aligned} \hat{\pi} = \operatorname{argmin}_{\pi \in \Pi} \operatorname{argmax}_{\theta} \mathcal{L}_{\text{GAIL}}(\pi, \theta) = & \mathbb{E}_{(s,a) \sim \mu^\pi} [\log D_\theta(s, a)] \\ & + \mathbb{E}_{(s^*, a^*) \sim \mu^{\pi^*}} [\log(1 - D_\theta(s^*, a^*))] - \lambda \mathcal{H}(\pi) \end{aligned}$$

where  $\mathcal{H}(\pi)$  is an entropy regularization term. When learning the discriminator  $D_\theta(s, a)$  binary classifier, positive examples are samples from the expert  $\pi^*$  demonstration data  $\mathcal{D}_{\text{exp}}$  and negative examples are samples from the  $\hat{\pi}$  interactions in an environment. A generator gradient is calculated in the original GAN framework by backpropagating through the discriminator. However,  $\mathcal{L}_{\text{GAIL}}$  is non-differentiable with respect to  $\pi$ , requiring some reinforcement learning optimizer. The original GAIL paper proposes to use a reward of  $r(s, a) = -\log(D_\theta(s, a))$  to train the policy which we denote as GAIL+. In their codebase<sup>5</sup>, they also use an alternate reward of  $r(s, a) = \log(1 - D_\theta(s, a))$  for certain tasks which we denote as GAIL-. Kostrikov et al. [157] formalizes and discusses issues regarding this bias in the reward function. We define the

---

<sup>5</sup>see section 6.9.19 for links

performance objective of executing policy  $\pi$  for both reward functions as:

$$J_{\text{GAIL}^+}(\pi) = \mathbb{E}_{(s,a) \sim \mu^\pi} [\log D_\theta(s, a)]$$

$$J_{\text{GAIL}^-}(\pi) = \mathbb{E}_{(s,a) \sim \mu^\pi} [-\log(1 - D_\theta(s, a))]$$

#### GAIL Hyperparameters: <sup>6</sup>

We use the default hyperparameters in table 6.8 for our experiments. We follow Orsini et al. [202]<sup>7</sup> for tuning hyperparameters for specific GAIL (R58, R56, R57, R58, R59). We also experiment with ratio between performing GAIL discriminator updates and PPO RL updates (R60). Although we apply the DrQ data augmentation (R61) to the discriminator, we follow DrAC Raileanu et al. [222] policy architecture design because the DrAC experiments use PPO, whereas DrQ did not. The main difference between DrQ and DrAC besides the data augmentation scheme, is that DrAC only has a shared encoder that both the actor and critic network can update. This architecture design choice mimics the same architecture design mentioned for PPO in the Neural network architecture section in the reinforcement learning background (see section 6.9.3 (R62, R63, R64, R65, R66, R67)). In our experiments we do not consider the absorbing state issues discussed in Kostrikov et al. [157] because Jena et al. [135] hypothesize that DAC maybe sample inefficient given that it has to match the state occupancy of the absorbing state (R68). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous Control Tasks:

RL Adam learning rate (R74) : {1e-03, 3e-04, 1e-04}

---

<sup>6</sup>Official codebase: <https://github.com/openai/imitation>

<sup>7</sup>Benchmarks adversarial imitation learning algorithms

discr. learning rate (R58):{1e-02, 1e-03, 1e-04}

discr. to RL updates ratio (R60):{5, 10, 15}

IL Action transformation  $T_u$  (R39): {None, Clip}

IL Input normalization (R41): {None, Fixed}

- Pixel Tasks:

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

discr. learning rate (R58):{1e-02, 1e-03, 1e-04}

discr. to RL updates ratio (R60):{5, 10, 15}

disc. data augmentation (R61): {None, DrQ}

- Structured-Prediction Tasks

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

discr. learning rate (R58):{1e-02, 1e-03, 1e-04}

discr. to RL updates ratio (R60):{5, 10, 15}

Table 6.8: Final settings used in GAIL experiments.

Name	Feature				Pixel	SP
	Pybullet		Mujoco		All	All
	Sub	Full	Sub	Full		
R74 RL Adam learning rate	3e-04	3e-04	3e-04	3e-04	1e-04	3e-04
R68 absorbing state	False	False	False	False	False	False
R56 gradient penalty $\lambda$	1.0	1.0	1.0	1.0	1.0	1.0
R57 gradient penalty $k$	0.0	0.0	0.0	0.0	0.0	0.0
R59 mixup $\alpha$	Rand	Rand	Rand	Rand	Rand	Rand
R58 discr. learning rate	3e-04	3e-04	3e-04	2.5e-04	1e-04	1e-03
R60 discr. to RL updates ratio	5	5	5	5	5	10
R62 discr. MLP depth	2	2	2	2	-	2
R63 discr. MLP width	64	64	64	64	-	100
R64 discr. activation	Tanh	Tanh	Tanh	Tanh	Tanh	Tanh
R65 discr. cnn layers	-	-	-	-	4.	-
R66 discr. CNN Initializer Gain	-	-	-	-	1.	-
R67 discr. MLP Initializer Gain	1.	1.	1.	1.	-	1.
R40 BC pretraining	True	True	True	True	True	True
R39 IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R61 disc. data augmentation	-	-	-	-	DrQ	-
R41 IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38 evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

### 6.9.13 Adversarial Inverse Reinforcement Learning (AIRL):

Adversarial inverse reinforcement learning (AIRL) [96] is an inverse reinforcement learning algorithm based on adversarial learning that draws connections between GAIL and Guided Cost Learning (GCL) [89]. The discriminator  $D_\theta(s, a)$  used in AIRL is the same discriminator used in GCL, except that AIRL optimizes this discriminator in an adversarial manner similar to GAIL. Given expert demonstration data  $\mathcal{D}_{\text{exp}}$  and a parameterized cost function  $c_\theta(s, a) = \sum_{t=1}^T c_\theta(s_t, a_t)$  where  $c_\theta(s_t, a_t)$  is the immediate cost function; the expert true distribution  $\mu^{\pi^*}$  is estimated by the following discriminator [89]:

$$D_\theta(s, a) = \frac{\frac{1}{Z} \exp(-c_\theta(s, a))}{\frac{1}{Z} \exp(-c_\theta(s, a)) + \mu^\pi(s, a)}$$

where  $\frac{1}{Z}$  is the partition function  $Z$  of  $\exp(-c_\theta(s, a))$ . The learner  $\hat{\pi}$  is trained to maximize the reward function  $r(s, a) = \log(1 - D_\theta(s, a)) - \log D_\theta(s, a)$ . Using the full trajectory in the cost calculation could result in high variance [96], so instead AIRL optimizes an alternative discriminator based on state-action  $\{(s_t, a_t)\}_{t=1}^T$  pairs:

$$D_\theta(s_t, a_t) = \frac{\exp(f_\theta(s_t, a_t))}{\exp(f_\theta(s_t, a_t)) + \mu^\pi(s_t, a_t)}$$

where  $f_\theta$  is a learned function. Like GAIL, the discriminator  $D_\theta(s, a)$  takes state-action pair as input and learns a binary classifier to distinguish between an expert  $\pi^*$  and the learner  $\hat{\pi}$  state-action pairs. The learner  $\hat{\pi}$  produces state-action pairs  $(s_t, a_t) \sim d^\pi$  and is rewarded for deceiving the discriminator, which is maximized using some reinforcement learning optimizer. The performance objective of executing policy  $\pi$  using the AIRL discriminator is:



$$J_{\text{AIRL}}(\pi) = \mathbb{E}_{(s,a) \sim \mu^\pi} [\log(1 - D_\theta(s, a)) - \log D_\theta(s, a)]$$

We follow Orsini et al. [202]<sup>8</sup> for our implementation. In particular, we do not implement the two modifications to the discriminator proposed by Fu et al. [96]: reward shaping term and the logit shift (see [96, 202]) because Orsini et al. [202] showed these two modifications do not provide any improvement. Essentially, our implementation is similar to GAIL, except we use the AIRL reward function specified above instead of the two GAIL variants.

AIRL Hyperparameters: <sup>9</sup>

We use the default hyperparameters in table 6.9 for our experiments. Most of the AIRL hyperparameters are the same as the GAIL hyperparameters mentioned in section 6.9.12. We follow Orsini et al. [202] for tuning hyperparameters for AIRL. For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous control Tasks:

RL Adam learning rate (R74): {1e-03, 3e-04, 1e-04}

discr. learning rate (R58): {1e-02, 1e-03, 1e-04}

discr. to RL updates ratio (R60): {5, 10, 15}

IL Action transformation  $T_u$  (R39): {None, clip}

IL Input normalization (R41): {None, fixed}

- Pixel Tasks:

---

<sup>8</sup>Benchmarks adversarial imitation learning algorithms

<sup>9</sup>Official codebase: <https://github.com/justinjfu/inverse-rl>

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

discr. learning rate (R58):{1e-02, 1e-03, 1e-04}

discr. to RL updates ratio (R60):{5, 10, 15}

disc. data augmentation (R61): {None, DrQ}

- Structured-Prediction Tasks

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

discr. learning rate (R58):{1e-02, 1e-03, 1e-04}

discr. to RL updates ratio (R60):{5, 10, 15}

Table 6.9: Final settings used in AIRL experiments.

Name	Feature				Pixel	SP
	Pybullet		Mujoco		All	All
	Sub	Full	Sub	Full		
R74 RL Adam learning rate	3e-04	3e-04	3e-04	3e-04	3e-04	1e-04
R68 absorbing state	False	False	False	False	False	False
R56 gradient penalty $\lambda$	1.0	1.0	1.0	1.0	1.0	1.0
R57 gradient penalty k	0.0	0.0	0.0	0.0	0.0	0.0
R59 mixup $\alpha$	Rand	Rand	Rand	Rand	Rand	Rand
R58 discr. learning rate	3e-04	3e-04	3e-04	2.5e-04	1e-04	1e-03
R60 discr. to RL updates ratio	5	5	5	5	5	10
R62 discr. MLP depth	2	2	2	2	-	2
R63 discr. MLP width	64	64	64	64	-	100
R64 discr. activation	Tanh	Tanh	Tanh	Tanh	Tanh	Tanh
R65 discr. cnn layers	-	-	-	-	4	-
R66 discr. CNN Initializer Gain	-	-	-	-	1.	-
R67 discr. MLP Initializer Gain	1.	1.	1.	1.	-	1.
R40 BC pretraining	True	True	True	True	True	True
R39 IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R61 disc. data augmentation	-	-	-	-	DrQ	-
R41 IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38 evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

#### 6.9.14 Random Expert Distillation (RED):

Random expert distillation (RED) [296] combines ideas from Random Network Distillation [48] and kernel-based support estimation [81, 248], to estimate the support of the expert  $\pi^*$  using the expert demonstration data  $\mathcal{D}_{\text{exp}}$ . This support is estimated using a fixed learned reward function parameterized with  $\hat{\theta}$ , minimizing the following objective:

$$\mathcal{L}_{\text{RED}}(s, a) = \min_{\hat{\theta}} \mathbb{E}_{(s,a) \sim \mu^\pi} [\|f_{\hat{\theta}}(s, a) - f_{\theta}(s, a)\|_2^2]$$

where  $f_{\hat{\theta}} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^k$  is the predictor network,  $f_{\theta} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^k$  is a fixed randomly initialized target network and  $\mathbb{R}^k$  the dimension of the embedding of size. The predictor network  $f_{\hat{\theta}}$  is trained to minimize the expected mean square error by gradient descent. The reward function is the predicted value of  $\mathcal{L}_{\text{RED}}(s, a)$  which estimates a state-action pair  $(s_t, a_t) \sim \mu^\pi$  belonging to the support of expert  $\mu^{\pi^*}$ . Wang et al. [296] uses the the following reward function  $r(s, a) = \exp(-\sigma^{-1}(\|f_{\hat{\theta}}(s, a) - f_{\theta}(s, a)\|_2^2))$  in practice resembling a Gaussian kernel where  $\sigma$  is the bandwidth. Furthermore,  $\mathcal{L}_{\text{RED}}(s, a)$  is positive, which means that  $r(s, a) \in [0, 1]$ . We define the performance objective of RED as:

$$J_{\text{RED}}(\pi) = \mathbb{E}_{(s,a) \sim \mu^\pi} [\exp(-\sigma^{-1}(\|f_{\hat{\theta}}(s, a) - f_{\theta}(s, a)\|_2^2))]$$

The authors chose the bandwidth  $\sigma$  such that the  $r(s, a)$  is close to 1. However, Arulkumaran and Lillrank [10] expressed issues with achieving this value for a given dataset if the analytically choice of  $\sigma = 0$  is not used, which results in a constant 1 reward for all state-action. Furthermore, Blondé et al. [35] pointed out that the official implementation of RED used per-environment

hyperparameters instead of environment agnostic hyperparameters. Jena et al. [137] also report negative results when trying to reproduce RED results in the original paper. In particular, the per-environment hyperparameters in RED official implementation span several orders of magnitude. We opt for environment-agnostic hyperparameters for a fair comparison to other algorithms.

RED Hyperparameters: <sup>10</sup>

We use the default hyperparameters in table 6.10 for our experiments. For the RED experiments, we did not experiment with different kernel functions and instead used a Gaussian kernel, the same as the official implementation (R69). Given that the official implementation has different bandwidths for each environment, we performed hyperparameter optimization on the kernel bandwidth (R70). We experiment with the learning rate when minimizing the difference between the fixed and random initialized network (RED learning rate (R75)) and keep the number of training epochs fixed (R72). For pixel tasks, we experiment with applying DrQ data augmentation to images before passing them to both the fixed and random initialized network for minimizing the difference between them (R73). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous Control Tasks:

RL Adam learning rate (R74): {1e-03, 3e-04, 1e-04}

RED learning rate (R75): {1e-03, 3e-04, 1e-04}

RED Kernel Bandwidth (R70): {1e0, 1e-02, 1e-03, 1e-04, 1e-05, 3e-04}

IL Action transformation  $T_u$  (R39): {None, Clip}

IL Input normalization (R41): {None, Fixed}

- Pixel Tasks:

---

<sup>10</sup>Official codebase: <https://github.com/RuohanW/RED>

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

RED learning rate (R75):{1e-03, 3e-04, 1e-04}

RED Kernel Bandwidth (R70):{1e0, 1e-01, 1e-02, 1e-03, 1e-04}

RED Data Augmentation (R73): {None, DrQ}

- Structured-Prediction Tasks

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

RED learning rate (R75):{1e-03, 3e-04, 1e-04}

RED Kernel Bandwidth (R70):{1e0, 1e-01, 1e-02, 1e-03, 1e-04}

Table 6.10: Default settings used in random expert distillation experiments.

Name		Feature				Pixel	SP
		Pybullet		Mujoco		All	All
		Sub	Full	Sub	Full		
R75	RED learning rate	1e-03	1e-03	1e-03	1e-03	3e-04	3e-04
R69	RED Gaussian Kernel	Yes	Yes	Yes	Yes	Yes	Yes
R70	RED Kernel Bandwidth	1e0	1e-04	1e-03	1e-03	1e-04	1e-03
R72	RND training epoch	1000	1000	1000	1000	1000	1000
R40	BC pretraining	True	True	True	True	True	True
R39	IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R73	RED Data Augmentation	-	-	-	-	DrQ	-
R41	IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38	evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

### 6.9.15 Behavior cloning -regularized Generative Adversarial Imitation Learning (BC-GAIL):

Jena et al. [137] proposed Behavior cloning -regularized generative adversarial imitation learning (BC-GAIL). The motivation of the work was to combine the benefits of BC being sample efficient to the environment and the benefit GAIL of being sample efficient to the number of expert demonstration data. When optimizing the BC term alone, the learner  $\hat{\pi}$  can mimic the expert if the state encounters while interacting in an environment are similar to states in the expert demonstration data. However, suppose the learner  $\hat{\pi}$  state-action distribution  $\mu^\pi$  is different from the expert state-action distribution  $\mu^{\pi^*}$ . In this case, the agent will experience feedback-driven covariate shift issues. On the other hand, GAIL learns a cost function using the discriminator that prioritizes the trajectories that are similar to the expert demonstration data, mitigating the covariate shift issues [126]. Combining the two ideas would provide an algorithm that is sample efficient to the number of trajectories needed and environment interactions. Formally, the original GAIL objective is the following:

$$\begin{aligned} \underset{\pi_\phi \in \Pi}{\operatorname{argmin}} \underset{\theta}{\operatorname{argmax}} \mathcal{L}_{\text{GAIL}}(\phi, \theta) &= \mathbb{E}_{(s,a) \sim \mu^\pi} [\log D_\theta(s, a)] \\ &+ \mathbb{E}_{(s^*, a^*) \sim \mu^{\pi^*}} [\log(1 - D_\theta(s^*, a^*))] - \lambda \mathcal{H}(\pi_\phi) \end{aligned}$$

where  $\pi$  is a parameterized policy by  $\phi$ ,  $D_\theta(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$  is a discriminator parameterized by  $\theta$  and  $\mathcal{H}(\pi)$  is the entropy of the policy. The discriminator trains a binary

classifier to distinguish between the learner and expert state-action pairs and rewards the learner for deceiving it. The learner maximizes some reinforcement learning optimizer using this reward from the discriminator. The gradient of  $\mathcal{L}_{\text{GAIL}}(\phi, \theta)$  with respect to each component is defined as:

$$\begin{aligned}\nabla_{\theta} \mathcal{L}_{\text{GAIL}}(\phi, \theta) &= \mathbb{E}_{(s,a) \sim \mu^{\pi}} [\nabla_{\theta} \log D_{\theta}(s, a)] + \mathbb{E}_{(s^*, a^*) \sim \mu^{\pi^*}} [\nabla_{\theta} \log(1 - D_{\theta}(s^*, a^*))] \\ \nabla_{\phi} \mathcal{L}_{\text{GAIL}}(\phi, \theta) &= \mathbb{E}_{(s,a) \sim \mu^{\pi}} [\nabla_{\theta} \log D_{\theta}(s, a)] - \lambda \mathcal{H}(\pi_{\phi}) \\ &= \mathbb{E}_{(s,a) \sim \mu^{\pi}} [\nabla_{\theta} \log \pi_{\phi}(a|s) Q_{\theta}(s, a)] - \lambda \mathcal{H}(\pi_{\phi})\end{aligned}$$

where  $Q_{\theta}(s, a) = \mathbb{E}_{(s,a) \sim \mu^{\pi}} [\log D_{\theta}(s, a)]$  in his original interpretation in Ho and Ermon [126]. In the literature the output of the discriminator has also been interpreted as a reward function  $r(s, a)$  Kostrikov et al. [157]. The authors of Jena et al. [137] uses interpretation to construct an advantage function:

$$A_{\omega, \psi}(s, a) = -\log(1 - D_{\theta}(s, a)) + \gamma \mathbb{E}_{s' \sim P(s'|s, a)} [V_{\psi}(s')] - V_{\psi}(s)$$

where  $P$  is the transition function and  $V_{\psi}$  is the value network parameterized by  $\psi$ . The policy gradient theorem has high variance [275], so often in practice general advantage estimation (GAE) [250] is used to reduce variance, which makes Jena et al. [137] interpretation reasonable. The new gradient of  $\mathcal{L}_{\text{GAIL}}(\phi, \theta)$  with respect to the parameterized policy  $\pi_{\phi}$  is:

$$\nabla_{\phi} \mathcal{L}_{\text{GAIL}}(\phi, \theta) = \mathbb{E}_{(s,a) \sim \mu^{\pi}} [\nabla_{\theta} \log \pi_{\phi}(a|s) A_{\omega, \psi}(s, a)] - \lambda \mathcal{H}(\pi_{\phi})$$

The authors apply importance sampling to the original BC performance objective when

$\ell(\pi(s^*), \pi^*(s^*)) = -\log \pi(a^*|s^*)$  (i.e. maximum-likelihood):

$$\begin{aligned}
J_{\text{BC}}(\pi) &= \mathbb{E}_{(s^*, a^*) \sim \mu^{\pi^*}} [-\log \pi(a^*|s^*)] \\
&= - \sum_{s^*, a^*} \mu^{\pi^*}(s^*, a^*) \cdot \log \pi(a^*|s^*) \\
&= - \sum_{s, a} \mu^{\pi}(s, a) \left[ \frac{\mu^{\pi^*}(s, a)}{\mu^{\pi}(s, a)} \cdot \log \pi(a|s) \right] \text{ (importance-sampling)} \\
&= \mathbb{E}_{(s, a) \sim \mu^{\pi}} \left[ \frac{\mu^{\pi^*}(s, a)}{\mu^{\pi}(s, a)} \cdot \log \pi(a|s) \right]
\end{aligned}$$

where  $\mu^{\pi}(s, a)$  is the state-action visitation distribution for state  $s$  and action  $a$ . Because we only have access to the expert  $\pi^*$  via a finite number of samples, they approximate the expert state-action visitation distribution using a Kronecker delta function:

$$\mu^{\pi^*}(s, a) = \delta_{\mathcal{D}_{\text{exp}}}(s, a) = \begin{cases} 1 & \text{if } (s, a) \in \mathcal{D}_{\text{exp}} \\ 0 & \text{otherwise} \end{cases}$$

Their final objective is the following:

$$\begin{aligned}
J_{\text{BC-GAIL}}(\pi) &= \mathbb{E}_{(s, a) \sim \mu^{\pi}} \left[ \left( \alpha \cdot \frac{\mu^{\pi^*}(s, a)}{\mu^{\pi}(s, a)} + (1 - \alpha) \cdot A_{\omega, \psi}(s, a) \right) \cdot \log \pi(a|s) \right] \\
&= \alpha \cdot \mathbb{E}_{(s, a) \sim \mu^{\pi}} \left[ \frac{\mu^{\pi^*}(s, a)}{\mu^{\pi}(s, a)} \cdot \log \pi(a|s) \right] \\
&\quad + (1 - \alpha) \cdot \mathbb{E}_{(s, a) \sim \mu^{\pi}} [A_{\omega, \psi}(s, a) \cdot \log \pi(a|s)] \\
&= \alpha \cdot J_{\text{BC}}(\pi) + (1 - \alpha) \cdot J_{\text{GAIL}}(\pi). \text{ (using equation (6.9.15))}
\end{aligned} \tag{6.3}$$



where  $\alpha$  creates a weighted sum between the BC and GAIL objective. The intuition of equation (6.3) is that the discriminator base advantage function learns to optimize actions that have higher  $Q$  values. These high  $Q$  actions are the ones that potentially will match the expert state-action pairs, in turn confusing the discriminator even further. However, the issue is that in order to learn and optimize this  $A_{\omega,\psi}$  GAIL requires lots of environmental interactions. The importance sampling provides a similar bonus as used in exploration strategies for exploration in normal reinforcement learning [245, 246]. The bonus is zero if the expert did not perform some action  $a$  in a state  $s$  and  $\frac{1}{\mu^\pi(s,a)}$  otherwise. As the agent starts to mimic the expert better, the advantage  $A_{\omega,\psi}(s, a)$  and the bonus decreases to a summed value close to 1.

In practice, we do not use the weight  $\alpha$  proposed by the authors to compare to other techniques that interleave BC updates fairly. The original GAIL paper proposes to use a reward of  $r(s, a) = -\log(D_\theta(s, a))$  to train the policy which we denote as BC-GAIL+. In their codebase<sup>11</sup>, they also use an alternate reward of  $r(s, a) = \log(1 - D_\theta(s, a))$  for certain tasks which we denote as BC-GAIL-. Kostrikov et al. [157] formalized and discussed issues regarding this bias in the reward function. We define the performance objective of executing policy  $\pi$  for both reward functions as:

$$J_{\text{BC-GAIL}+}(\pi) = J_{\text{BC}}(\pi) + \mathbb{E}_{(s,a) \sim \mu^\pi} [\log D_\theta(s, a)]$$

$$J_{\text{BC-GAIL}-}(\pi) = J_{\text{BC}}(\pi) + \mathbb{E}_{(s,a) \sim \mu^\pi} [-\log(1 - D_\theta(s, a))]$$

BC-GAIL Hyperparameters: <sup>12</sup>

We use the default hyperparameters in table 6.11 for our experiments. For all parameters

---

<sup>11</sup>see section 6.9.19 for links

<sup>12</sup>Official codebase: <https://github.com/rohitrango/BC-regularized-GAIL>

for BC-GAIL we combine the hyperparameters used for BC (see section 6.9.11) and GAIL (see section 6.9.12). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous Control Tasks:

```
RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}
BC learning rate (R52):{1e-03, 1e-04, 2.5e-04}
discr. learning rate (R58):{1e-02, 1e-03, 1e-04}
IL Action transformation  $T_u$  (R39): {None, Clip}
IL Input normalization (R41): {None, Fixed}
```

- Pixel Tasks:

```
RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}
BC learning rate (R52):{1e-03, 1e-04, 2.5e-04}
discr. learning rate (R58):{1e-03, 1e-04}
BC Data Augmentation (R54): {None, DrQ}
disc. data augmentation (R61): {None, DrQ}
```

- Structured-Prediction Tasks

```
RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04, 2.5e-04}
BC learning rate (R52):{1e-03, 1e-04}
discr. learning rate (R58):{1e-03, 1e-04}
```

Table 6.11: Final settings used in BC-GAIL experiments.

Name		Feature				Pixel	SP
		Pybullet		Mujoco		All	All
		Sub	Full	Sub	Full		
R74	RL Adam learning rate	3e-04	3e-04	1e-03	1e-03	3e-04	3e-04
R68	absorbing state	False	False	False	False	False	False
R56	gradient penalty $\lambda$	1.0	1.0	1.0	1.0	1.0	1.0
R57	gradient penalty $\kappa$	0.0	0.0	0.0	0.0	0.0	0.0
R59	mixup $\alpha$	Rand	Rand	Rand	Rand	Rand	Rand
R58	discr. learning rate	1e-3	1e-3	1e-3	1e-04	1e-04	1e-03
R60	discr. to RL updates ratio	5	5	5	5	5	5
R62	discr. MLP depth	2	2	2	2	2	2
R63	discr. MLP width	64	64	64	64	512	100
R64	discr. activation	Tanh	Tanh	Tanh	Tanh	Tanh	Tanh
R65	discr. cnn layers	-	-	-	-	3	-
R66	discr. CNN Initializer Gain	-	-	-	-	1.	-
R67	discr. MLP Initializer Gain	1.	1.	1.	1.	1.	1.
R40	BC pretraining	True	True	True	True	True	True
R52	BC learning rate	2.5e-04	2.5e-04	2.5e-04	2.5e-04	2.5e-04	2.5e-04
R54	BC Data Augmentation	-	-	-	-	DrQ	-
R39	IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R61	disc. data augmentation	-	-	-	-	DrQ	-
R41	IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38	evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

### 6.9.16 Behavior cloning -regularized Adversarial Inverse Reinforcement

#### Learning (BC-AIRL):

Behavior cloning -regularized Adversarial Inverse Reinforcement Learning (BC-AIRL) extends AIRL (see section 6.9.13) to incorporate BC update similar to BC-GAIL (see section 6.9.12). We follow Orsini et al. [202]<sup>13</sup> for our implementation of AIRL. In particular, AIRL proposed three modifications to GAIL: changing the reward function, adding a reward shaping term, and a logit shift (see [96, 202]. Orsini et al. [202] showed that the latter two modifications do not provide any improvement, so we do not include them in our experiments, so the BC-AIRL only changes the reward function of BC-GAIL. The performance objective of BC-AIRL is below:

$$J_{\text{BC-AIRL}}(\pi) = J_{\text{BC}}(\pi) + \mathbb{E}_{(s,a) \sim \mu^\pi} [-\log(1 - D_\theta(s, a, s')) + \log(D_\theta(s, a, s'))]$$

#### BC-AIRL Hyperparameters:

We use the default parameters in table 6.12 for our experiments. For all hyperparameters for BC-AIRL we combine the hyperparameters used for BC (see section 6.9.11) and AIRL (see section 6.9.13). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continous Control Tasks:

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

---

<sup>13</sup>Benchmarks adversarial imitation learning algorithms

BC learning rate (R52):{1e-03, 1e-04, 2.5e-04}  
discr. learning rate (R58):{1e-02, 1e-03, 1e-04}  
IL Action transformation  $T_u$  (R39): {None, Clip}  
IL Input normalization (R41): {None, Fixed}

- Pixel Tasks:

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}  
BC learning rate (R52):{1e-03, 1e-04, 2.5e-04}  
discr. learning rate (R58):{1e-03, 1e-04}  
BC Data Augmentation (R54): {None, DrQ}  
disc. data augmentation (R61): {None, DrQ}

- Structured-Prediction Tasks

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}  
BC learning rate (R52):{1e-03, 1e-04, 2.5e-04}  
discr. learning rate (R58):{1e-03, 1e-04}

Table 6.12: Final settings used in BC-GAIL experiments.

Name		Feature				Pixel	SP
		Pybullet		Mujoco		All	All
		Sub	Full	Sub	Full		
R74	RL Adam learning rate	3e-04	3e-04	1e-03	1e-03	3e-04	3e-04
R68	absorbing state	False	False	False	False	False	False
R56	gradient penalty $\lambda$	1.0	1.0	1.0	1.0	1.0	1.0
R57	gradient penalty $k$	0.0	0.0	0.0	0.0	0.0	0.0
R59	mixup $\alpha$	Rand	Rand	Rand	Rand	Rand	Rand
R58	discr. learning rate	1e-03	1e-03	1e-03	1e-04	1e-04	1e-03
R60	discr. to RL updates ratio	5	5	5	5	5	5
R62	discr. MLP depth	2	2	2	2	2	2
R63	discr. MLP width	64	64	64	64	512	100
R64	discr. activation	Tanh	Tanh	Tanh	Tanh	Tanh	-
R65	discr. cnn layers	-	-	-	-	3	-
R66	discr. CNN Initializer Gain	-	-	-	-	1.	-
R67	discr. MLP Initializer Gain	1.	1.	1.	1.	1.	-
R40	BC pretraining	True	True	True	True	True	True
R52	BC learning rate	2.5e-04	2.5e-04	2.5e-04	2.5e-04	2.5e-04	2.5e-04
R54	BC Data Augmentation	-	-	-	-	DrQ	-
R39	IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R61	disc. data augmentation	-	-	-	-	DrQ	-
R41	IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38	evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

### 6.9.17 Behavior cloning -regularized Random Expert Distillation (BC-RED):

Behavior cloning -regularized Random Expert Distillation (BC-RED) combines ideas from RED (see section 6.9.14) with interleaving BC updates ideas from DRIL (see chapter 5) and BC-GAIL (see section 6.9.15). Originally, RED learns a fixed reward function parameterized by  $\hat{\theta}$  to estimate the support of the expert  $\pi^*$  by minimizing the following objective:

$$\mathcal{L}_{\text{RED}}(s, a) = \min_{\hat{\theta}} \mathbb{E}_{(s,a) \sim \mu^{\pi}} [\|f_{\hat{\theta}}(s, a) - f_{\theta}(s, a)\|_2^2]$$

where the  $\mathcal{L}_{\text{RED}}(s, a)$  can be interpreted as an estimated score of pair  $(s, a)$  belonging to the support of  $\mu^{\pi^*}$ .

Similar to issues pointed out by Jena et al. [137] about GAIL, RED could require lots of environment interactions to learn using this fixed reward function.

Jena et al. [137] points out that RED could perform suboptimally because the reward function is fixed and not updated in an online fashion similar to GAIL. Furthermore, the authors Brantley et al. [44] of DRIL showed a counterexample that emphasized the importance of adding in the BC cost to the objective when optimizing an estimation score. This counterexample is in section 6.7. BC-RED builds on these ideas by minimizing the following objective:

$$J_{\text{BC-RED}}(\pi) = J_{\text{BC}}(\pi) + \mathbb{E}_{(s,a) \sim \mu^{\pi}} [-\exp(-\sigma \|f_{\theta}(s, a) - f_{\hat{\theta}}(s, a)\|_2^2)]$$

BC-RED Hyperparameters:

We use the default parameters in table 6.13 for our experiments. For all hyperparameters for BC-RED we combine the hyperparameters used for BC (see section 6.9.11) and RED (see

section 6.9.14). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous control Tasks:

RL Adam learning rate (R74): {1e-03, 3e-04, 1e-04}

BC learning rate (R52): {1e-03, 1e-04, 2.5e-04}

RED Kernel Bandwidth (R70): {1e0, 1e-02, 1e-03, 1e-04, 1e-05, 3e-04}

IL Action transformation  $T_u$  (R39): {None, Clip}

IL Input normalization (R41): {None, Fixed}

- Pixel Tasks:

RL Adam learning rate (R74): {1e-03, 3e-04, 1e-04}

BC learning rate (R52): {1e-03, 1e-04, 2.5e-04}

RED Kernel Bandwidth (R70): {1e0, 1e-03, 1e-04}

RED Data Augmentation (R73): {None, DrQ}

- Structured-Prediction Tasks

RL Adam learning rate (R74): {1e-03, 3e-04, 1e-04}

BC learning rate (R52): {1e-03, 1e-04, 1e-05}

RED Kernel Bandwidth (R70): {1e0, 1e-01, 1e-02, 1e-03, 1e-04}



Table 6.13: Default settings used in BC-RED experiments.

Name		Feature				Pixel	SP
		Pybullet		Mujoco		All	All
		Sub	Full	Sub	Full		
R74	RL Adam learning rate	3e-04	3e-04	3e-04	3e-04	1e-03	1e-03
R75	RED learning rate	1e-03	1e-03	1e-03	1e-03	1e-03	1e-03
R76	BCRED Gaussian Kernel	Yes	Yes	Yes	Yes	Yes	Yes
R77	BCRED Kernel Bandwidth	1e-03	1e0	1e-03	1e-01	1e-04	1e-03
R78	BCRED training epoch	1000	1000	1000	1000	1000	1000
R40	BC pretraining	True	True	True	True	True	True
R52	BC learning rate	1e-03	1e-03	1e-03	1e-03	1e-04	1e-03
R39	IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R73	RED Data Augmentation	-	-	-	-	DrQ	-
R41	IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38	evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

### 6.9.18 Behavior cloning -regularized RL with constant reward: (BC-CR)

There has been recent work around optimizing imitation learning with a constant reward (see section 6.9.10), and separately work on interleaving BC optimizations with optimizing RL on a reward derived from demonstration data DRIL (see chapter 5) and BC-GAIL (see section 6.9.15). Jena et al. [137] experimented with training BC-GAIL with a fixed randomly initialized discriminator and keeping the  $\alpha$  term (see section 6.9.15) fixed. Annealing  $\alpha$  would eventually ignore BC and only train using random rewards from the untrained discriminator. This untrained discriminator BC-GAIL technique performed reasonably well. Jena et al. [137] state that this is a better baseline than BC because it can learn about state-action pairs not in the expert demonstration using the random discriminator. Motivated by this observation and recent techniques that optimize BC and RL, we derive a new simple baseline algorithm, Behavior cloning -regularized Reinforcement Learning with Constant Reward (BC-CR). This baseline algorithm builds on observations discussed in section 6.9.10 by interleaving BC optimization with RL trained with a constant reward. Most imitation learning algorithms improve upon BC using environment interactions  $\Sigma$ , but there has not been any simple baseline that takes advantage of both the demonstration data and the environment  $\Sigma$ . This baseline algorithm takes advantage of both components and optimizes the following objective:

$$J_{\text{BC-CR}} = J_{\text{BC}}(\pi) + \mathbb{E}_{(s,a) \sim \mu^\pi} [1]$$

BC-CR Hyperparameters:

We use the default parameters in table 6.14 for our experiments. For all hyperparameters for BC-CR we combine the hyperparameters used for BC (see section 6.9.11) and CR (see

section 6.9.10). For each of the three-task, we optimized hyperparameters over the following range of values:

- Continuous Control Tasks:

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04, 1e-05, 1e-06, 1e-07}

BC learning rate (R52):{1e-02, 1e-03, 1e-04, 1e-05, 2e-03, 3e-03, 4e-03, 5e-03}

Constant reward value (R50):{1e0, 1e-01, 1e-02}

IL Action transformation  $T_u$  (R39): {None, Clip}

IL Input normalization (R41): {None, Fixed}

- Pixel Tasks:

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

BC learning rate (R52):{1e-03, 1e-04}

Constant reward value (R50):{.01, .05, 1.0}

BC Data Augmentation (R54): {None, DrQ}

- Structured-Prediction Tasks

RL Adam learning rate (R74):{1e-03, 3e-04, 1e-04}

BC learning rate (R52):{1e-03, 2e-03, 1e-04}

Constant reward value (R50):{.01, .01, 1.0}

Table 6.14: Final settings used in BC-CR experiments.

Name		Feature				Pixel	SP
		Pybullet		Mujoco		All	All
		Sub	Full	Sub	Full		
R79	BC-Constant learning rate	1e-06	3e-04	1e-06	1e-04	1e-03	1e-04
R80	BC-Constant BC learning rate	1e-04	1e-03	1e-03	1e-04	1e-03	1e-04
R50	Constant reward value	1.0	1.0	1.0	1.0	1.0	1.0
R40	BC pretraining	True	True	True	True	True	True
R39	IL Action transformation $T_u$	Clip	Clip	None	None	None	-
R54	BC Data Augmentation	-	-	-	-	DrQ	-
R41	IL Input normalization	None	None	Fixed	Fixed	Scale	-
R38	evaluation behavior policy	Mean	Mean	Mean	Mean	Mean	Argmax

### 6.9.19 Extended Experiment Details

We performed hyperparameter sweeps over the best parameters for PPO recommended by Andrychowicz et al. [9]. We consider three broad categories of task: continuous control [51, 70, 283], pixel tasks [279] and structured-prediction tasks [79]. The goal is to evaluate all algorithms performance agnostics of the underlying category of the task. We chose to benchmark these tasks because they have been the most widely used task in the past and present for comparing imitation learning algorithms.

### 6.9.20 Continuous Control Tasks

We compare all algorithms using 25 tasks for continuous control, both sub-sampling and full trajectories. We follow Andrychowicz et al. [9]<sup>14</sup> for all continuous control hyperparameters tuning. Continuous control tasks have been used widely in reinforcement learning and imitation learning. However, not all the algorithms in this study have been evaluated in the same setting. For example, DRIL [44] used the Pybullet physics engine whereas GAIL [126] used the MuJoCo physics engine. We evaluate all algorithms across three simulators Box2D [51], Pybullet [70] and MuJoCo [283] given that agents do not generalize across physics engines [194]<sup>15</sup>. In particular, we consider 2 Box2D tasks, 11 Pybullet tasks, and 12 MuJoCo tasks. In both physics engines, we consider tasks ranging from simple continuous Cartpole to as complex as humanoid flag runner, where an agent can run and capture various flags as they span onto a map. For all tasks in both physics engines, we used the default observation of the task without any modifications.

---

<sup>14</sup>Benchmarks what matter in on-policy reinforcement learning

<sup>15</sup>Pybullet Robots are heavier exhibiting different behavior than MuJoCo <https://github.com/DLR-RM/rl-baselines3-zoo/issues/111>

### 6.9.20.1 Different GAIL reward functions

The code to switch between different GAIL reward functions can be seen at the links below:

- Strictly positive reward function favoring prolonged episodes (good for tasks which require survival): <https://github.com/openai/imitation/blob/8a2ed905e2ac54bda0f71e5ee364e90568e6d031/policyopt/imitation.py#L141-L145>
- Strictly negative reward function favoring short episodes (good for goal-reaching tasks): <https://github.com/openai/imitation/blob/8a2ed905e2ac54bda0f71e5ee364e90568e6d031/policyopt/imitation.py#L147-L150>

### 6.9.20.2 Discussion of trajectory sub-sampling

The original GAIL paper does not mention sub-sampling, however, the code to perform subs-sampling in the code repository can be seen here: [https://github.com/openai/imitation/blob/8a2ed905e2ac54bda0f71e5ee364e90568e6d031/scripts/imitate\\_mj.py#L30-L40](https://github.com/openai/imitation/blob/8a2ed905e2ac54bda0f71e5ee364e90568e6d031/scripts/imitate_mj.py#L30-L40)

Other works sometimes mention sub-sampling and sometimes do not. The work of [150] does not, but they use the GAIL numbers taken directly from the paper, and hence evaluate the BC baselines using sub-sampled trajectories. The paper [104] explicitly mentions in the introduction that they sub-sample trajectories by a factor of 20. They also mention that the work of [96] sub-samples trajectories, but we could not find details one way or the other in that paper.

The benchmark paper of [202] explicitly mentions that they sub-sample trajectories. The work of Kostrikov et al. [157] mentions sub-sampling for one experiment, but it is unclear whether it was applied for all. We recommend that future IL publications explicitly clarify whether sub-sampling is applied or not.

### 6.9.21 Pixel Tasks

For pixel task environments, we compare all algorithms using 6 pixel tasks spanning two toolkits DMC suite and Box2D CarRacing. We apply Data-regularized Q (DrQ) [313] data augmentation technique on the 84x84 image experience fragments gathered each PPO iteration. Most algorithms discussed in this study have not been evaluated on any pixel setting except for DRIL [44], which was evaluated on pixel Atari task. We choose to evaluate all algorithms on pixel tasks that support GPU acceleration. We do not consider Atari task because Atari does not support GPU acceleration, it has been shown that the simulator suffers in performance [74]. We consider a subset of the pixel locomotion DeepMind Control Suite [279] tasks that have been considered in previous work [159, 220] which are Cartpole swingup, Cheetah run, Finger spin, Ball in cup and Walker walk. The Box2D simulator supports GPU acceleration as well, so we include the Car-Racing task as well.

For pixel-based task environment we follow [313] and [222]. During each iteration of PPO, we typically gather a fixed amount of experience fragments, each consisting of a fixed size. Then we perform mini-batches updates with these fixed experience fragments. We apply the data augmentation technique proposed by DrQ [313] on the experience fragments gathered each iteration. The images that we experiment with are 84 x 84. The DrQ data augmentation works by padding each side of the image by 4 pixels and then selecting random  $84 \times 84$  crops, yielding

the original image shifted by  $\pm 4$  pixels. We follow DrAC[222] architecture design, but apply DrQ data augmentation. The main difference between DrQ[313] and DrAC[222] architecture design is that DrAC only has a shared encoder that both the actor and critic networks can update.

### 6.9.22 Structured Prediction Tasks

IL algorithms such as DAGGER [235], DAGGER [79], SMILe [231], V-Dagger [293] and LOLS [54] were evaluated on structured prediction NLP task [26, 79]. Structured-prediction NLP sequence labeling problems can be cast as an MDP [79, 186] where the sentence is parsed one word at a time in a left-to-right order. The environment state at time step  $t$  consists of the current word and some context consisting of previously predicted labels. We only consider the immediately predicted label as context. We consider 5 NLP sequence labeling task: Named Entity Recognition [240], Part of Speech Tagging [259], Keyphrase Extraction [13], Chunking [239], Super Sense Disambiguation [64] and Semantic Role Labeling [217]. We use the NLPGym toolkit for modeling all of these task [224]. We use the PPO hyperparameters recommended by Ramamurthy et al. [224]. Furthermore, we used fastText embeddings [139] for all tasks.



### 6.9.23 Extended Experiment Results

In this section, we provide full experiment results for each of the three tasks: continuous control, pixel, and structured prediction, as well as aggregate results for different numbers of trajectories that are omitted from the main text due to space constraints.

### 6.9.24 Continuous Control Tasks

#### 6.9.24.1 PyBullet/Box2D

- We test on the following PyBullet/Box2D environments:

```
{AntBulletEnv-v0, HumanoidFlagrunHarderBulletEnv-v0,  
Walker2DBulletEnv-v0, HalfCheetahBulletEnv-v0,  
HopperBulletEnv-v0, CartPoleContinuousBulletEnv-v0,  
ReacherBulletEnv-v0, InvertedDoublePendulumBulletEnv-v0,  
BipedalWalker-v3, InvertedPendulumSwingupBulletEnv-v0,  
LunarLanderContinuous-v2, InvertedPendulumBulletEnv-v0,  
Pendulum-v0 HumanoidFlagrunBulletEnv-v0,  
MountainCarContinuous-v0}
```

- We do hyperparameter tuning on the following PyBullet/Box2D environments:

```
{AntBulletEnv-v0, Walker2DBulletEnv-v0,  
HalfCheetahBulletEnv-v0, HopperBulletEnv-v0}
```

## PyBullet/Box2D Aggregated Results:

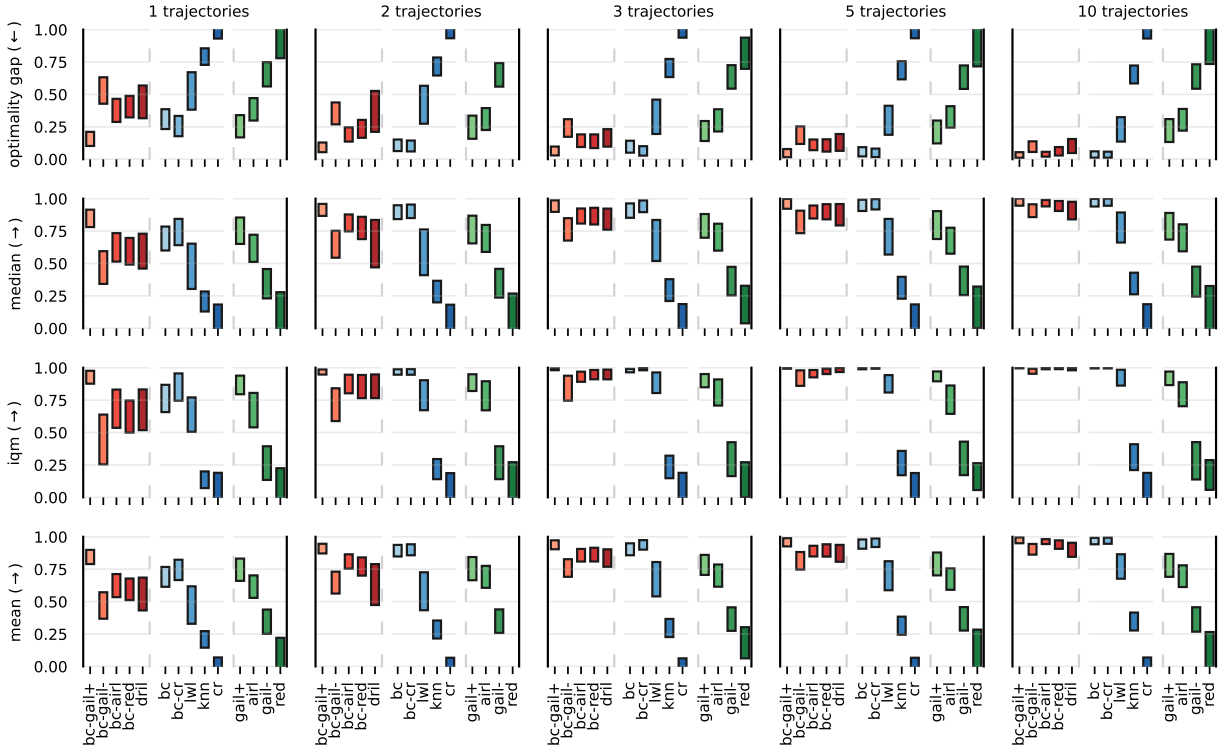


Figure 6.5: Results that are not normalized for continuous control PyBullet/Box2D environments **without sub-sampled trajectories**. Scores are normalized between 0 (random performance) and 1 (expert performance). Bars indicate 95% confidence intervals computed using stratified bootstrapping. IQM denotes interquartile mean.

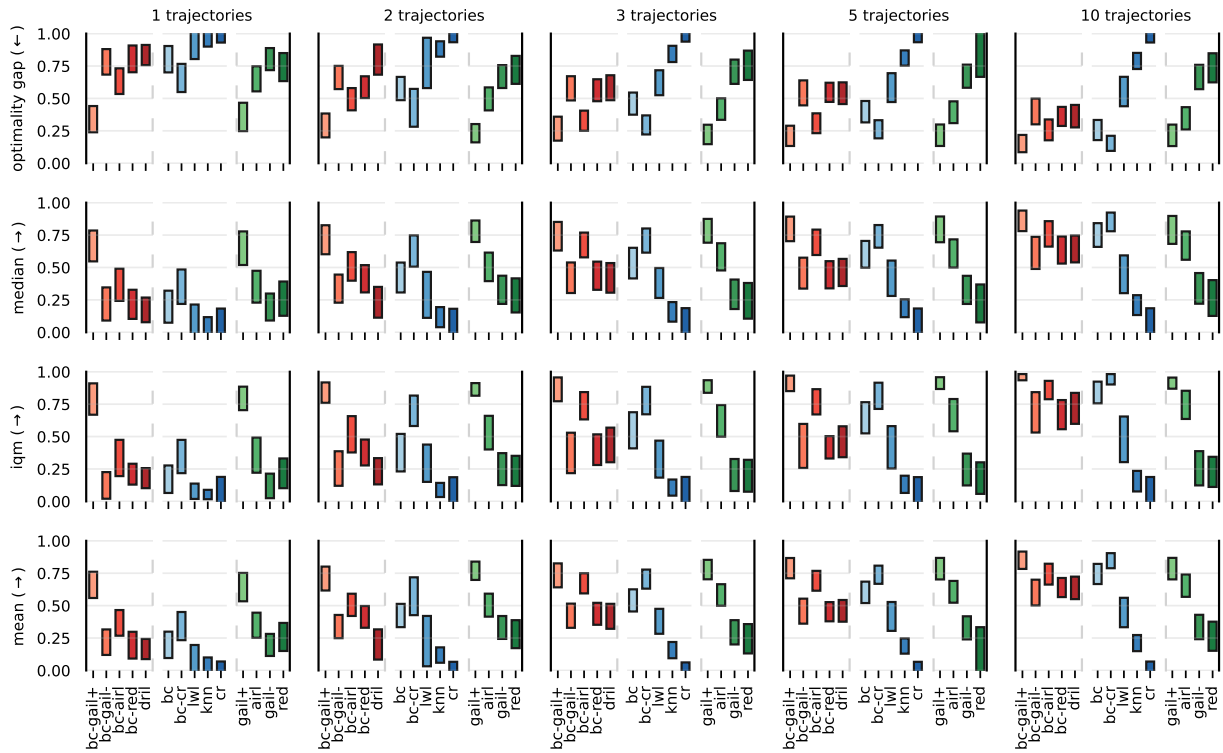


Figure 6.6: Results that are not normalized for continuous control PyBullet/Box2D environments **with sub-sampled trajectories**. Scores are normalized between 0 (random performance) and 1 (expert performance). Bars indicate 95% confidence intervals computed using stratified bootstrapping. IQM denotes interquartile mean.

## PyBullet/Box2D Environment Results:

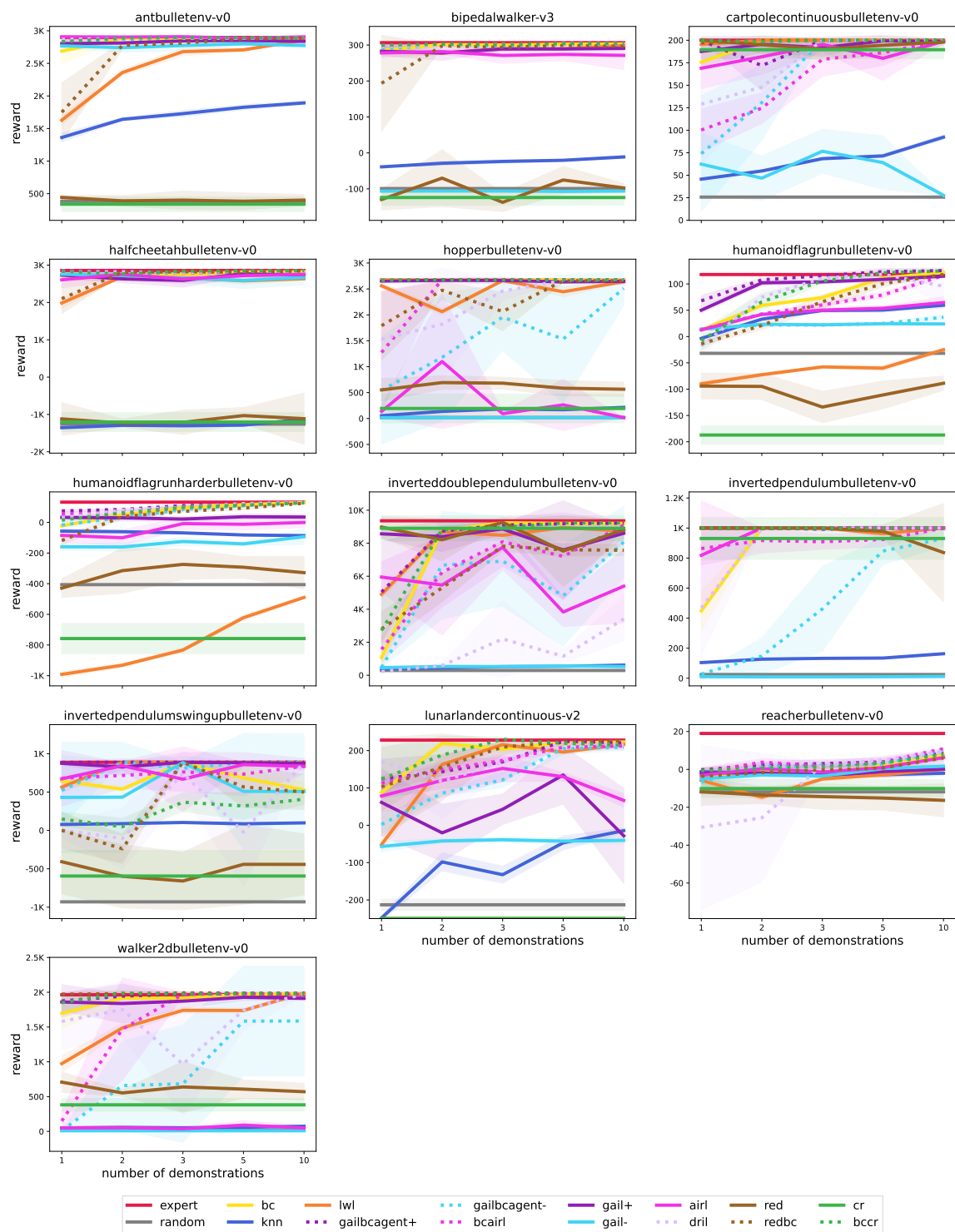


Figure 6.7: Results that are not normalized for continuous control PyBullet/Box2D environments without sub-sampled trajectories.

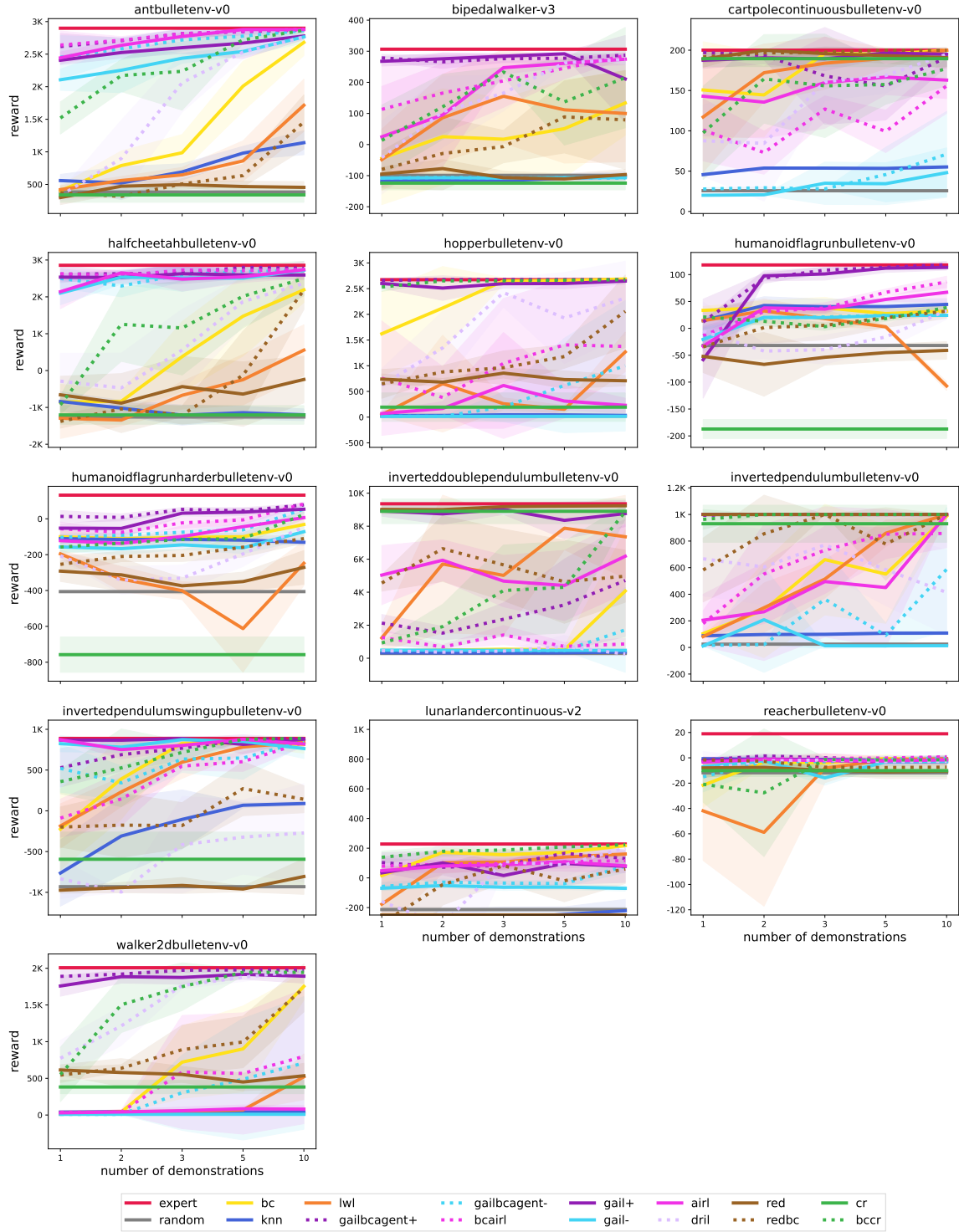


Figure 6.8: Results that are not normalized for continuous control PyBullet/Box2D environments with sub-sampled trajectories.

### 6.9.24.2 MuJoCo

- We test on the following MuJoCo environments:

{Ant-v3, HalfCheetah-v3, Hopper-v3, Humanoid-v3,  
HumanoidStandup-v2, InvertedDoublePendulum-v2,  
InvertedPendulum-v2, Reacher-v2, Swimmer-v3, Walker2d-v3,  
Pusher-v2, Thrower-v2, Striker-v2 }

- We do hyperparameter tuning on the following MuJoCo environments:

{Ant-v3, HalfCheetah-v3, Hopper-v3, Walker2d-v3}

#### MuJoCo Aggregated Results:

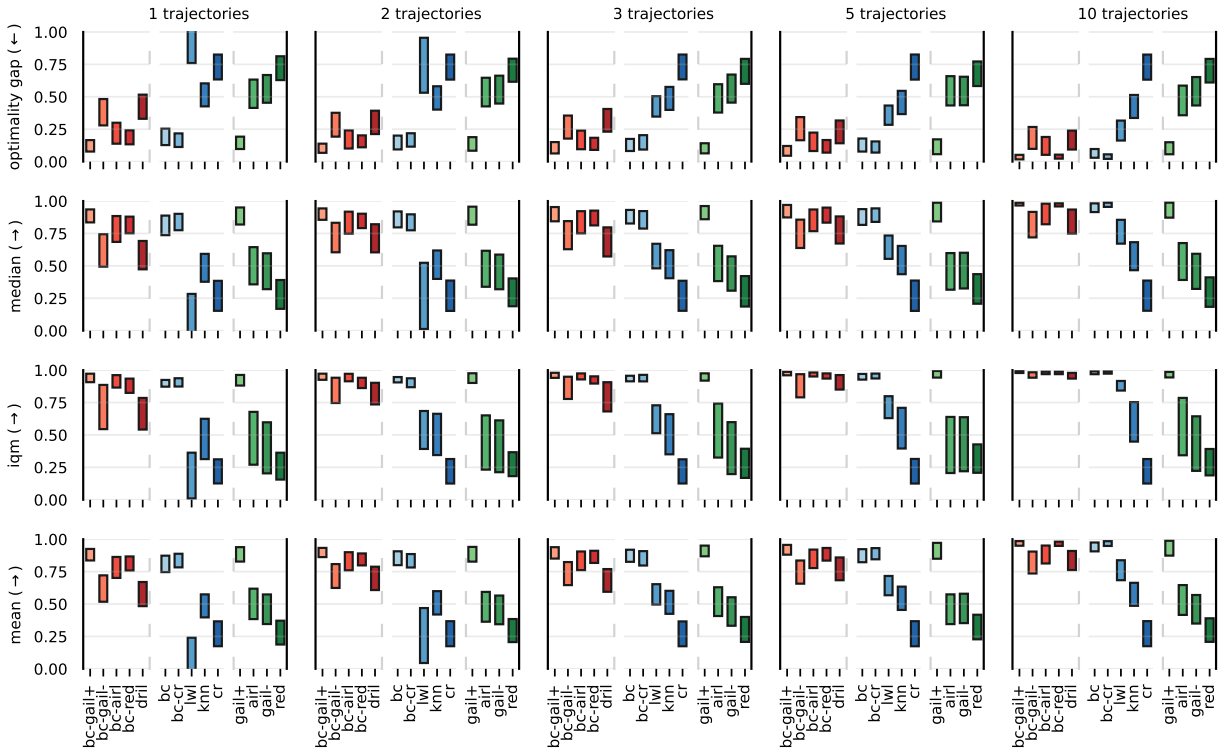


Figure 6.9: Results that are not normalized for continuous control MuJoCo environments **without sub-sampled trajectories**. Scores are normalized between 0 (random performance) and 1 (expert performance). Bars indicate 95% confidence intervals computed using stratified bootstrapping. IQM denotes interquartile mean.

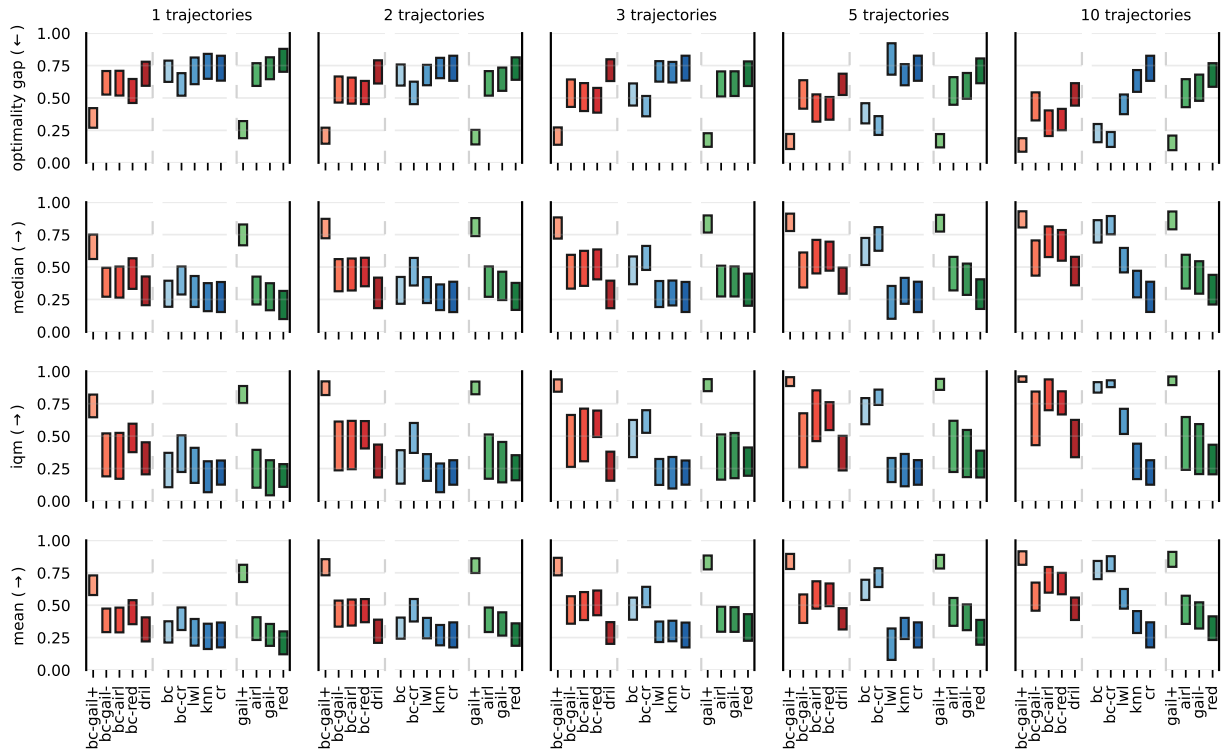


Figure 6.10: Results that are not normalized for continuous control MuJoCo environments **with sub-sampled trajectories**. Scores are normalized between 0 (random performance) and 1 (expert performance). Bars indicate 95% confidence intervals computed using stratified bootstrapping. IQM denotes interquartile mean.

## MuJoCo Environment Results:

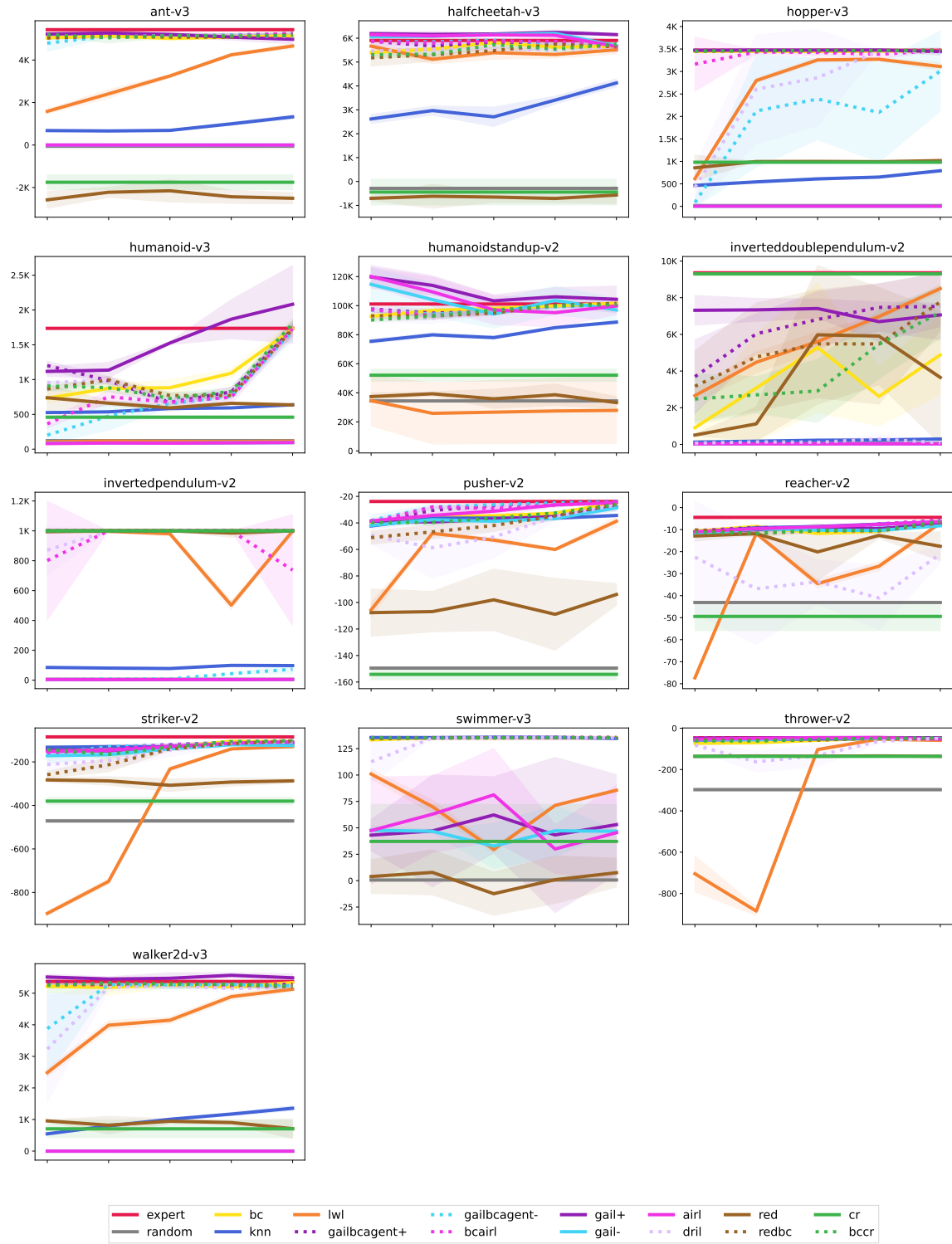


Figure 6.11: Results that are not normalized for continuous control MuJoCo environments without sub-sampled trajectories.



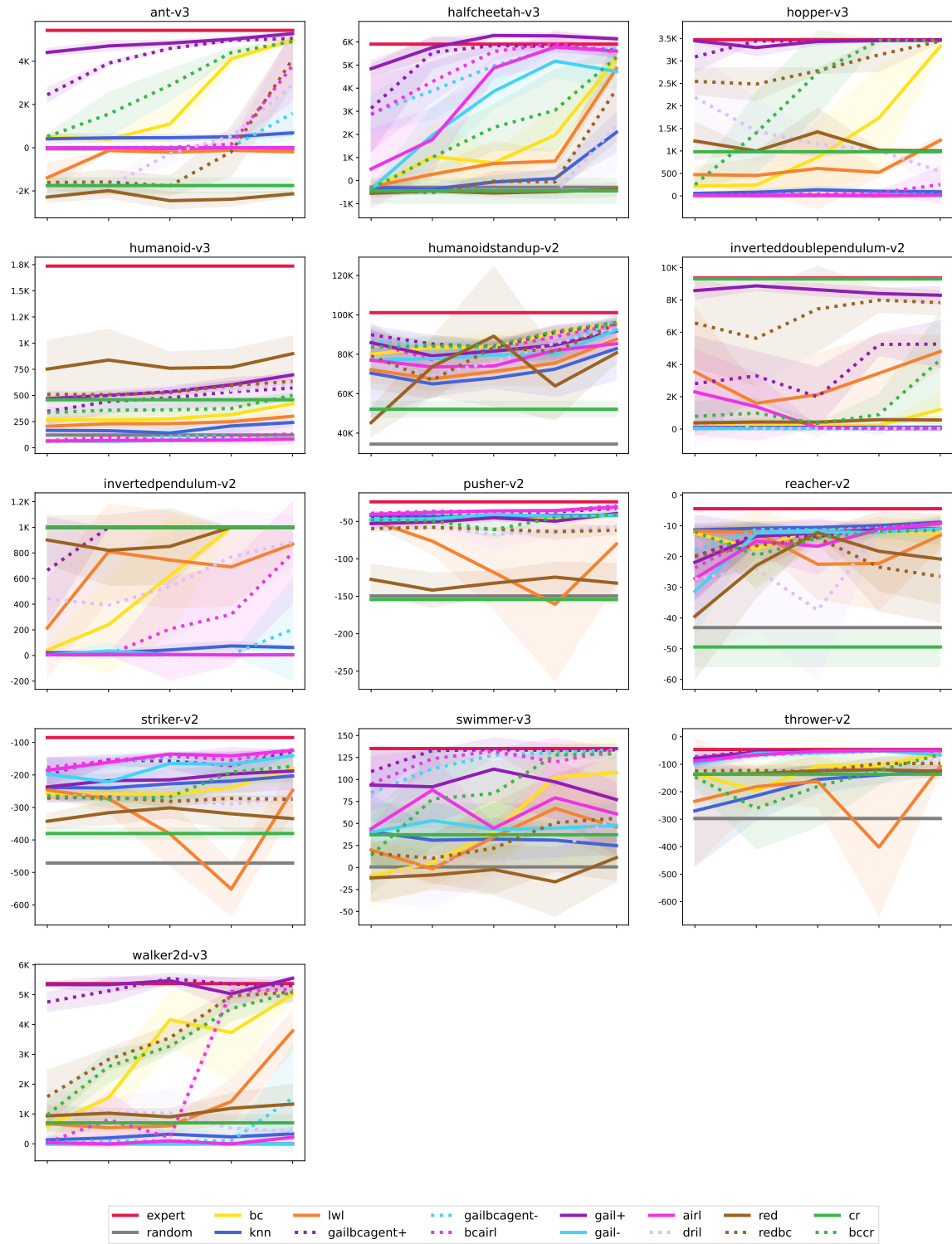


Figure 6.12: Results that are not normalized for continuous control MuJoCo environments **with sub-sampled trajectories**.

## 6.9.25 Structured Prediction Tasks

### 6.9.25.1 NLP Gym

- We test on the following NLP Gym environments:

{Part-of-Speech, Name-Entity-Recognition,

Keyphrase-Extraction, Chunking,

Word-Sense-Disambiguation}

- We do hyperparameter tuning on the following NLP Gym environments:

{Part-of-Speech, Name-Entity-Recognition, Chunking}

#### NLP Gym Environment Results:

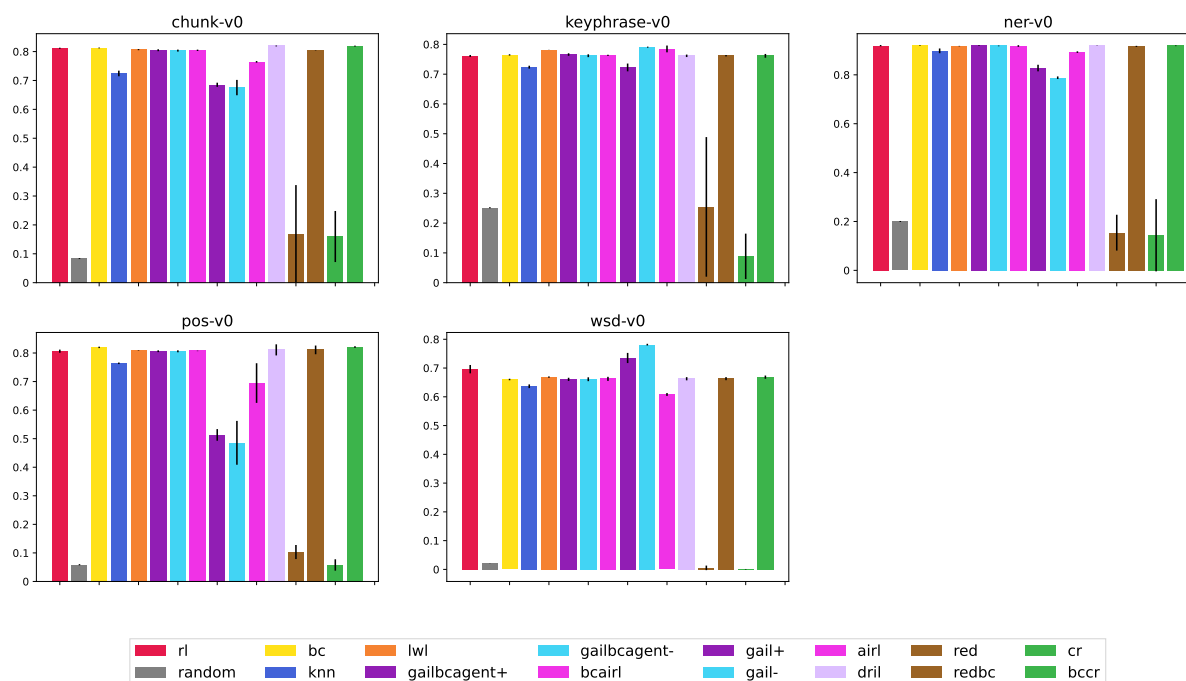


Figure 6.13: Results that are not normalized for NLP Gym environments.

## 6.9.26 Pixel-base Tasks

### 6.9.26.1 DeepMind Control Suite (DMC)/Box2D

- We test on the following DMC/Box2D environments:

{cheetah-run, walker-walk, CarRacing-v0, cartpole-swingup,  
finger-spin ball\_in\_cup-catch, }

- We do hyperparameter tuning on the following DMC/Box2D environments:

{cheetah-run, walker-walk, CarRacing-v0}

#### DMC/Box2D Environment Results:

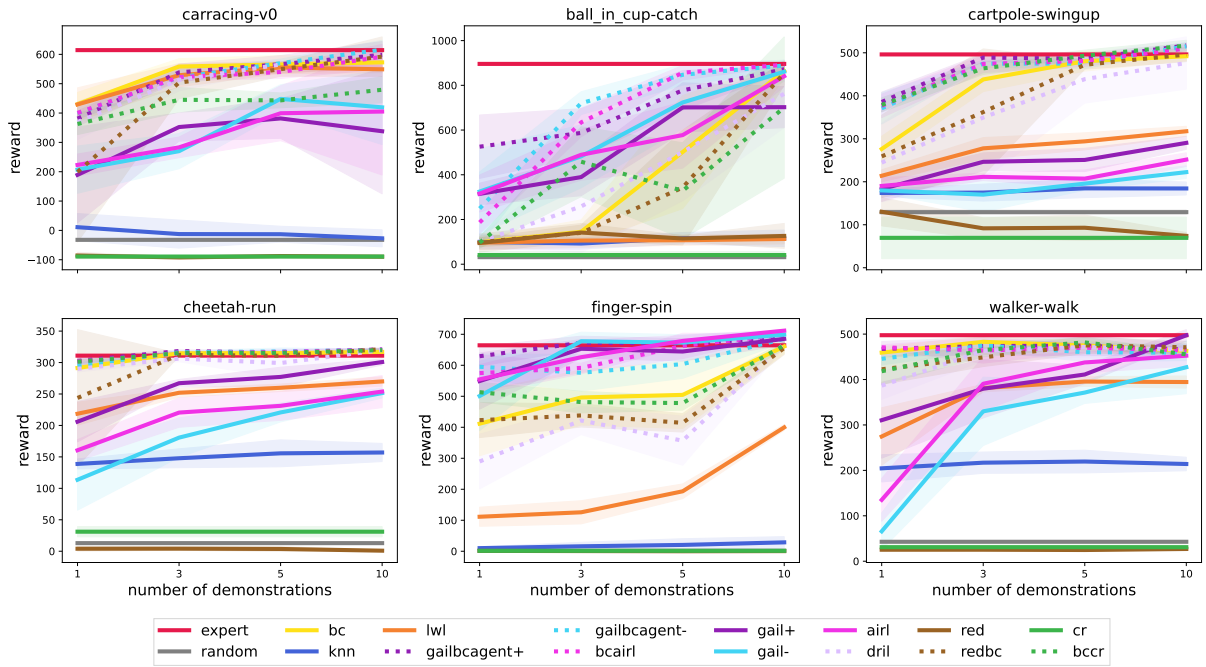


Figure 6.14: Results that are not normalized for DMC/Box2D environments.

### Part Part 3

#### Beyond Traditional Imitation Learning

## Chapter 7: Exact Imitation

The previous chapters highlighted key points regarding imitation learning techniques. In **Part 1** we discussed and addressed issues in interactive imitation learning. Although interactive imitation learning address the covariate shift issue in imitation, these algorithms make a strong assumption around having an online expert. We attempt to remove this assumption in **Part 2** by introducing a category of algorithms that deal with the covariate shift with an online expert. In general, these settings assume expert demonstration data or the ability to query an online expert is not sufficient.

<sup>1</sup> This chapter assumes that we have access to expert demonstration data in the form of expected features, which are the average features seen when an expert interacts with a system. Note that the demonstration data does not include actions but only state features. We describe a new representation for decision-theoretic planning, reinforcement learning, and imitation learning. This representation has many advantages for reinforcement learning. For example, they can help an agent generalize experiences to new goals. They have been proposed to explain behavioral and neural data from human and animal learners. They also form a natural bridge between model-based and model-free reinforcement learning methods: like the former, they make predictions about future experiences, and like the latter, they allow efficient prediction of total discounted rewards. The new representation is highly expressive: for example, it lets us efficiently read off an optimal policy for a new reward function or a policy that imitates a new demonstration. This chapter focuses on the exact computation of the new representation in small,

---

<sup>1</sup>A previous version of this work was presented in [41]

known environments since even this restricted setting offers plenty of interesting questions.

## 7.1 Introduction

We describe a new representation for decision-theoretic planning, reinforcement learning, and imitation learning: the *successor feature set*. This representation generalizes a number of previous ideas in the literature, including successor features and POMDP/PSR value functions. Comparing to these previous representations: successor features assume a fixed policy or list of policies, while our goal is to reason efficiently about many policies at once; value functions assume a fixed reward function, while our goal is to reason efficiently about many reward functions at once.

Roughly, the successor feature set tells us how features of our future observations and actions depend on our current state and our choice of policy. More specifically, the successor feature set is a convex set of matrices; each matrix corresponds to a policy  $\pi$ , and describes how the features we will observe in the future depend on the current state under  $\pi$ .

The successor feature set provides a number of useful capabilities. These include reading off the optimal value function or policy for a new reward function, predicting the range of outcomes that we can achieve starting from a given state, and reading off a policy that imitates a desired state-action visitation distribution.

We describe a convergent dynamic programming algorithm for computing the successor feature set, generalizing the value iteration algorithm for POMDPs or PSRs. We also give algorithms for reading off the above-mentioned optimal policies and feature-matching policies from the successor feature set. Since the exact dynamic programming algorithm can be prohibitively expensive, we also experiment with randomized numerical approximations.

In this paper we focus on model-based reasoning about successor feature sets — that is, we assume access to an accurate world model. We also focus on algorithms that are exact in the limit of increasing computation. Successor-style representations are of course also extremely useful for approximate reasoning about large, unknown environments, and we believe that many of the ideas discussed here can inform that case as well, but we leave that direction for future work.

To summarize, our contributions are: a new successor-style representation that allows information to flow among different states, policies, and reward functions; algorithms for working with this new representation in small, known environments, including a convergent dynamic programming algorithm and ways to read off optimal policies and feature-matching policies; and computational experiments that evaluate the strengths and limitations of our new representation and algorithms.

## 7.2 Related Work

Successor features, a version of which were first introduced by Dayan [80], provide a middle ground between model-free and model-based RL [237]. They have been proposed as neurally plausible explanations of learning [99, 101, 102, 195, 269, 291].

Recently, numerous extensions have been proposed. Most similar to the current work are methods that generalize to a set of policies or tasks. Barreto et al. [22] achieve transfer learning by generalizing across tasks with successor features; Barreto et al. [21] use generalized policy improvement (GPI) over a set of policies. A few methods [38, 184] recently combined universal value function approximators [243] with GPI to perform multi-task learning, generalizing to a set of goals by conditioning on a goal representation. Barreto et al. [23] extend policy improvement

and policy evaluation from single tasks and policies to a list of them, but do not attempt to back up across policies.

Many authors have trained nonlinear models such as neural networks to predict successor-style representations, e.g., Hansen et al. [114], Kulkarni et al. [161], Machado et al. [185], Zhang et al. [319], Zhu et al. [326]. These works are complementary to our goal here, which is to design and analyze new, more general successor-style representations. We hope our generalizations eventually inform training methods for large-scale nonlinear models.

At the intersection of successor features and imitation learning, Zhu et al. [326] address visual semantic planning; Lee et al. [169] address off-policy model-free RL in a batch setting; and Hsu [129] addresses active imitation learning.

As mentioned above, the individual elements of  $\Phi$  are related to the work of Lehnert and Littman [172]. And, we rely on point-based methods [209, 256] to compute  $\Phi$ .

## 7.3 Background

### Policy Trees

We will need to work with policies for MDPs, POMDPs, and PSRs, handling different horizons as well as partial observability. For this reason, we will use a general policy representation: we will view a policy as a mixture of trees, with each tree representing a deterministic, nonstationary policy. A policy tree’s nodes are labeled with actions, and its edges are labeled with observations (Fig. 7.1). To execute a policy tree  $\pi$ , we execute  $\pi$ ’s root action; then, based on the resulting observation  $o$ , we follow the edge labeled  $o$  from the root, leading to a subtree that we will call  $\pi(o)$ . To execute a mixture, we randomize over its elements. If desired we can



randomize lazily, committing to each decision just before it affects our actions. We will work with finite, balanced trees, with depth equal to a horizon  $H$ ; we can reason about infinite-horizon policies by taking a limit as  $H \rightarrow \infty$ .

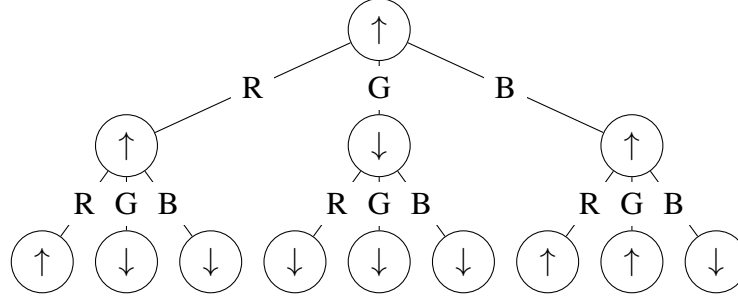


Figure 7.1: An example of a policy tree with actions  $\uparrow, \downarrow$  and observations  $R, G, B$ .

## 7.4 Imitation by Feature Matching

Successor feature sets have many uses, but we will start by motivating them with the goal of imitation. Often we are given demonstrations of some desired behavior in a dynamical system, and we would like to imitate that behavior. There are lots of ways to specify this problem, but one reasonable one is *apprenticeship learning* [1] or *feature matching*. In this method, we define features of states and actions, and ask our learner to match some statistics of the observed features of our demonstrations.

In more detail, given an MDP, define a vector of features of the current state and action,  $f(s, a) \in \mathbb{R}^d$ ; we call this the *one-step* or *immediate* feature vector. We can calculate the observed discounted features of a demonstration: if we visit states and actions  $s_1, a_1, s_2, a_2, s_3, a_3, \dots$ , then the empirical discounted feature vector is

$$f(s_1, a_1) + \gamma f(s_2, a_2) + \gamma^2 f(s_3, a_3) + \dots$$

where  $\gamma \in [0, 1)$  is our *discount factor*. We can average the feature vectors for all of our demonstrations to get a *demonstration or target* feature vector  $\phi^d$ .

Analogously, for a policy  $\pi$ , we can define the *expected* discounted feature vector:

$$\phi^\pi = \mathbb{E}_\pi \left[ \sum_{t=1}^{\infty} \gamma^{t-1} f(s_t, a_t) \right]$$

We can use a finite horizon  $H$  by replacing  $\sum_{t=1}^{\infty}$  with  $\sum_{t=1}^H$  in the definitions of  $\phi^d$  and  $\phi^\pi$ ; in this case we have the option of setting  $\gamma = 1$ .

Given a target feature vector in any of these models, we can ask our learner to design a policy that matches the target feature vector in expectation. That is, we ask the learner to find a policy  $\pi$  with

$$\phi^\pi = \phi^d$$

For example, suppose our world is a simple maze MDP like Fig. 7.2a. Suppose that our one-step feature vector  $f(s, a) \in [0, 1]^3$  is the RGB color of the current state in this figure, and that our discount is  $\gamma = 0.75$ . If our demonstrations spend most of their time toward the left-hand side of the state space, then our target vector will be something like  $\phi^d = [0.5, 3, 0.5]^T$ : the green feature will have the highest expected discounted value. On the other hand, if our demonstrations spend most of their time toward the bottom-right corner, we might see something like  $\phi^d = [2, 1, 1]^T$ , with the blue feature highest.

## 7.5 Extension to POMDPs and PSRs

We can generalize the above definitions to models with partial observability as well. This is not a typical use of successor features: reasoning about partial observability requires a model,

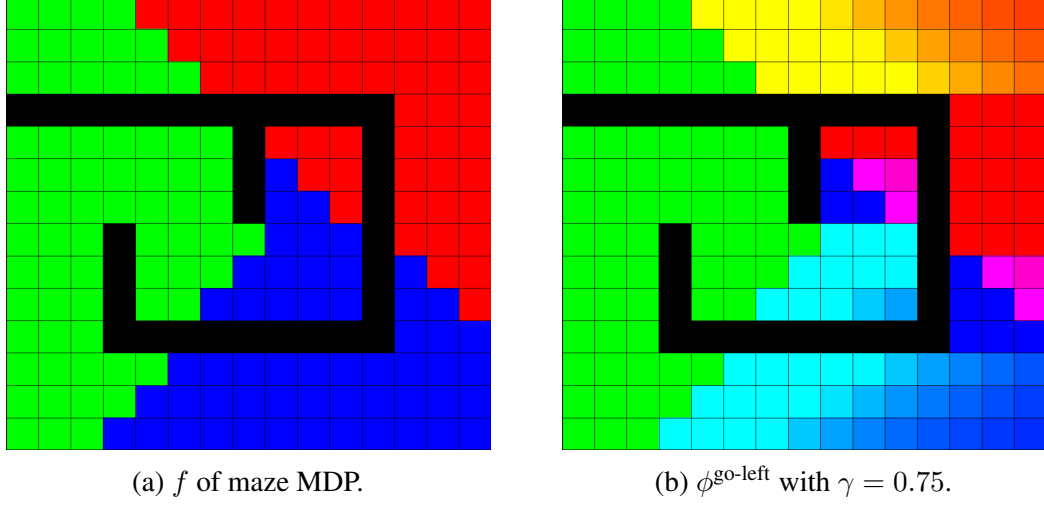


Figure 7.2: Maze environment example.

while successor-style representations are often used in model-free RL. However, as Lehnert and Littman [172] point out, the state of a PSR is already a prediction about the future, so incorporating successor features into these models makes sense.

In a POMDP, we have a belief state  $q \in \mathbb{R}^k$  instead of a fully-observed state. We define the immediate features of  $q$  to be the expected features of the latent state:

$$f(q, a) = \sum_{s=1}^k q(s) f(s, a)$$

In a PSR, we similarly allow any feature function that is linear in the predictive state vector  $q \in \mathbb{R}^k$ :

$$f(q, a) = F_a q$$

with one matrix  $F_a \in \mathbb{R}^{d \times k}$  for each action  $a$ . In either case, define the successor features to be

$$\phi^\pi(q) = \mathbb{E}_\pi \left[ \sum_{t=1}^{\infty} \gamma^{t-1} f(q_t, a_t) \mid \text{do } q_1 = q \right]$$

Interestingly, the function  $\phi^\pi$  is *linear* in  $q$ . That is, for each  $\pi$ , there exists a matrix  $A^\pi \in \mathbb{R}^{d \times k}$  such that  $\phi^\pi(q) = A^\pi q$ . We call  $A^\pi$  the *successor feature matrix* for  $\pi$ ; it is related to the parameters of the *Linear Successor Feature Model* of Lehnert and Littman [172].

We can compute  $A^\pi$  recursively by working backward in time (upward from the leaves of a policy tree): for a tree with root action  $a$ , the recursion is

$$A^\pi = F_a + \gamma \sum_o A^{\pi(o)} T_{ao}$$

This recursion works by splitting  $A^\pi$  into contributions from the first step ( $F_a$ ) and from steps  $2 \dots H$  (rest of RHS). We give a more detailed derivation, as well as a proof of linearity, in the supplementary material online. All the above works for MDPs as well by taking  $q_t = e_{st}$ , which lets us keep a uniform notation across MDPs, POMDPs, and PSRs.

It is worth noting the multiple feature representations that contribute to the function  $\phi^\pi(q)$ . First are the immediate features  $f(q, a)$ . Second is the PSR state, which can often be thought of as a feature representation for an underlying “uncompressed” model [119]. Finally, both of the above feature representations help define the *exact* value of  $\phi^\pi$ ; we can also *approximate*  $\phi^\pi$  using a third feature representation. Any of these feature representations could be related, or we could use separate features for all three purposes. We believe that an exploration of the roles of these different representations would be important and interesting, but we leave it for future work.

## 7.6 Successor Feature Sets

To reason about multiple policies, we can collect together multiple matrices: the *successor feature set* at horizon  $H$  is defined as the set of all possible successor feature matrices at horizon  $H$ ,

$$\Phi^{(H)} = \{A^\pi \mid \pi \text{ a policy with horizon } H\}$$

As we will detail below, we can also define an infinite-horizon successor feature set  $\Phi$ , which is the limit of  $\Phi^{(H)}$  as  $H \rightarrow \infty$ .

The successor feature set tells us *how the future depends* on our state and our choice of policy. It tells us the range of outcomes that are possible: for a state  $q$ , each point in  $\Phi q$  tells us about one policy, and gives us moments of the distribution of future states under that policy. The extreme points of  $\Phi q$  therefore tell us the limits of what we can achieve. (Here we use the shorthand of *broadcasting*: set arguments mean that we perform an operation all possible ways, substituting one element from each set. E.g., if  $X, Y$  are sets,  $X + Y$  means Minkowski sum  $\{x + y \mid x \in X, y \in Y\}$ .)

Note that  $\Phi^{(H)}$  is a convex, compact set: by linearity of expectation, the feature matrix for a stochastic policy will be a convex combination of the matrices for its component deterministic policies. Therefore,  $\Phi^{(H)}$  will be the convex hull of a finite set of matrices, one for each possible deterministic policy at horizon  $H$ .

Working with multiple policies at once provides a number of benefits: perhaps most importantly, it lets us define a Bellman backup that builds new policies combinatorially by combining existing policies at each iteration (Sec. 7.8). That way, we can reason about all possible policies instead of just a fixed list. Another benefit of  $\Phi$  is that, as we will see below,

it can help us compute optimal policies and feature-matching policies efficiently. On the other hand, because it contains so much information, the set  $\Phi$  is a complicated object; it can easily become impractical to work with. We return to this problem in Sec. 7.10.

## 7.7 Special Cases

In some useful special cases, successor feature matrices and successor feature sets have a simpler structure that can make them easier to reason about and work with. E.g., in an MDP, we can split the successor feature matrix into its columns, resulting in one vector per state — this is the ordinary successor feature vector  $\phi^\pi(s) = A^\pi e_s$ . Similarly, we can split  $\Phi$  into sets of successor feature vectors, one at each state, representing the range of achievable futures:

$$\phi(s) = \{\phi^\pi(s) \mid \pi \text{ a policy}\} = \Phi e_s$$

Fig. 7.3 visualizes these projections, along with the Bellman backups described below. Each projection tells us the discounted total feature vectors that are achievable from the corresponding state. For example, the top-left plot shows a set with five corners, each corresponding to a policy that is optimal in this state under a different reward function; the bottom-left corner corresponds to “always go down,” which is optimal under reward  $R(s, a) = (-1, -1)f(s, a)$ .

On the other hand, if we only have a single one-step feature ( $f(q, a) \in \mathbb{R}$ ), then we can only represent a 1d family of reward functions. All positive multiples of  $f$  are equivalent to one another, as are all negative multiples. In this case, our recursion effectively reduces to classic POMDP or PSR value iteration: each element of  $\Phi$  is now a vector  $\alpha^\pi \in \mathbb{R}^k$  instead of a matrix  $A^\pi \in \mathbb{R}^{d \times k}$ . This  $\alpha$ -vector represents the (linear) value function of policy  $\pi$ ; the pointwise

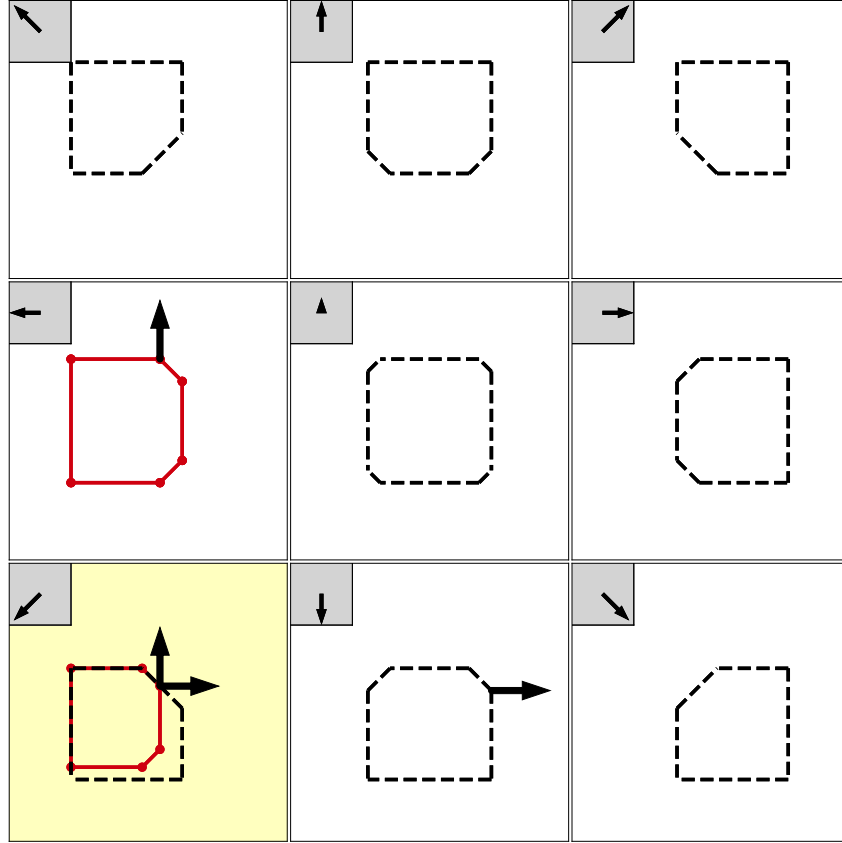


Figure 7.3: Visualization of the successor feature set  $\Phi$  for a  $3 \times 3$  gridworld MDP with 2d features. Start state is in yellow. Gray insets show one-step feature vectors, which depend only on the state, not the action. Each subplot shows one projection  $\Phi e_j$  (scale is arbitrary, so no axes are necessary). The red sets illustrate a Bellman backup at the bottom-left state, and the black arrows illustrate the feature-matching policy there. See text for details.

maximum of all these functions is the (piecewise linear and convex) optimal value function of the POMDP or PSR.

## 7.8 Bellman Equations

Each element of the successor feature set is a successor feature matrix for some policy, and as such, it satisfies the recursion given above. For efficiency, though, we would like to avoid running Bellman backups separately for too many possible policies. To this end, we can write a backup operator and Bellman equations that apply to all policies at once, and hence describe the

entire successor feature set.

The joint backup works by relating horizon- $H$  policies to horizon- $(H - 1)$  policies. Every horizon- $H$  policy tree can be constructed recursively, by choosing an action to perform at the root node and a horizon- $(H - 1)$  tree to execute after each possible observation. So, we can break down any horizon- $H$  policy (including stochastic ones) into a distribution over the initial action, followed by conditional distributions over horizon- $(H - 1)$  policy trees for each possible initial observation.

Therefore, if we have the successor feature set  $\Phi^{(H-1)}$  at horizon  $H - 1$ , we can construct the successor feature set at horizon  $H$  in two steps: first, for each possible initial action  $a$ , we construct

$$\Phi_a^{(H)} = F_a + \gamma \sum_o \Phi^{(H-1)} T_{ao}$$

This set tells us the successor feature matrices for all horizon- $H$  policies that begin with action  $a$ . Note that only the first action is deterministic:  $\Phi^{(H-1)}$  lets us assign any conditional distribution over horizon- $(H - 1)$  policy trees after each possible observation.

Second, since a general horizon- $H$  policy is a distribution over horizon- $H$  policies that start with different actions, each element of  $\Phi^{(H)}$  is a convex combination of elements of  $\Phi_a^{(H)}$  for different values of  $a$ . That is,

$$\Phi^{(H)} = \text{conv} \bigcup_a \Phi_a^{(H)}$$

The recursion bottoms out at horizon 0, where we have

$$\Phi^{(0)} = \{0\}$$



since the discounted sum of a length-0 trajectory is always the zero vector.

Fig. 7.3 shows a simple example of the Bellman backup. Since this is an MDP,  $\Phi$  is determined by its projections  $\Phi e_j$  onto the individual states. The action “up” takes us from the bottom-left state to the middle-left state. So, we construct  $\Phi_{\text{up}} e_{\text{bottom-left}}$  by shifting and scaling  $\Phi e_{\text{middle-left}}$  (red sets). The full set  $\Phi e_{\text{bottom-left}}$  is the convex hull of four sets  $\Phi_a e_{\text{bottom-left}}$ ; the other three are not shown, but for example, taking  $a = \text{right}$  gives us a shifted and scaled copy of the set from the bottom-center plot.

The update from  $\Phi^{(H-1)}$  to  $\Phi^{(H)}$  is a contraction: see the supplementary material online for a proof. So, as  $H \rightarrow \infty$ ,  $\Phi^{(H)}$  will approach a limit  $\Phi$ ; this set represents the achievable successor feature matrices in the infinite-horizon discounted setting.  $\Phi$  is a fixed point of the Bellman backup, and therefore satisfies the *stationary* Bellman equations

$$\Phi = \text{conv} \bigcup_a \left[ F_a + \gamma \sum_o \Phi T_{ao} \right]$$

## 7.9 Feature Matching and Optimal Planning

Once we have computed the successor feature set, we can return to the feature matching task described in Section 7.4. Knowing  $\Phi$  makes feature matching easier: for any target vector of discounted feature expectations  $\phi^d$ , we can efficiently either compute a policy that matches  $\phi^d$  or verify that matching  $\phi^d$  is impossible. We detail an algorithm for doing more detail in the extended details.

Fig. 7.3 shows the first steps of our feature-matching policy in a simple MDP. At the bottom-left state, the two arrows show the initial target feature vector (root of the arrows) and the computed policy (randomize between “up” and “right” according to the size of the arrows). The

target feature vector at the next step depends on the outcome of randomization: each destination state shows the corresponding target and the second step of the computed policy.

---

**Algorithm 5** Feature Matching Policy

---

```

1:  $t \leftarrow 1$ 
2: Initialize  $\phi_t^d$  to the target vector of expected discounted features.
3: Initialize  $q_t$  to the initial state of the environment.
4: while not done do
5:   Choose actions  $a_{it}$ , vectors  $\phi_{it} \in \Phi_{a_{it}} q_t$ , and
   convex combination weights  $p_{it}$  s.t.  $\phi_t^d = \sum_{i=1}^{\ell} p_{it} \phi_{it}$ .
   Choose an index  $i$  according to probabilities  $p_{it}$ ,
   and execute the corresponding action:  $a_t \leftarrow a_{it}$ .
   Write the corresponding  $\phi_{it}$  as
    $\phi_{it} = F_{a_t} q_t + \gamma \sum_o \phi_{ot}$  by choosing
    $\phi_{ot} \in \Phi T_{a_t o} q_t$  for each  $o$ .
6:   Receive observation  $o_t$ , and calculate  $p_t = P(o_t \mid q_t, a_t) = u^T T_{a_t o_t} q_t$ .
7:    $q_{t+1} \leftarrow T_{a_t o_t} q_t / p_t$ 
8:    $\phi_{t+1}^d \leftarrow \phi_{o_t t} / p_t$ 
9:    $t \leftarrow t + 1$ 
10: end while

```

---

We can also use the successor feature set to make optimal planning easier. In particular, if we are given a new reward function expressed in terms of our features, say  $R(q, a) = r^T f(q, a)$  for some coefficient vector  $r$ , then we can efficiently compute the optimal value function under  $R$ :

$$V^*(q) = \max_{\pi} r^T \phi^{\pi} q = \max \{r^T \psi q \mid \psi \in \Phi\}$$

As a by-product we get an optimal policy: there will always be a matrix  $\psi$  that achieves the max above and satisfies  $\psi \in \Phi_a$  for some  $a$ . Any such  $a$  is an optimal action.

## 7.10 Implementation

An exact representation of  $\Phi$  can grow faster than exponentially with the horizon. So, in our experiments below, we work with a straightforward approximate representation. We use two

tools: first, we store  $\Phi_{ao} = \Phi T_{ao}$  for all  $a, o$  instead of storing  $\Phi$ , since the former sets tend to be effectively lower-dimensional due to sparsity. Second, analogous to PBVI [209, 256], we fix a set of directions  $m_i \in \mathbb{R}^{d \times k}$ , and retain only the most extreme point of  $\Phi_{ao}$  in each direction. Our approximate backed-up set is then the convex hull of these retained points. Just as in PBVI, we can efficiently compute backups by passing the max through the Minkowski sum in the Bellman equation. That is, for each  $i$  and each  $a, o$ , we solve

$$\arg \max \langle m_i, \phi \rangle \text{ for } \phi \in \bigcup_{a'} [F_{a'} + \gamma \sum_{o'} \Phi_{a'o'}] T_{ao}$$

by solving, for each  $i, a, o, a', o'$

$$\arg \max \langle m_i, \phi \rangle \text{ for } \phi \in \Phi_{a'o'} T_{ao}$$

and combining the solutions.

There are a couple of useful variants of this implementation that we can use in *stoppable* problems (i.e., problems where we have an emergency-stop or a safety policy; see the supplemental material for more detail). First, we can update *monotonically*, i.e., keep the better of the horizon- $H$  or horizon- $(H + 1)$  successor feature matrices in each direction. Second, we can update *incrementally*: we can update any subset of our directions while leaving the others fixed.

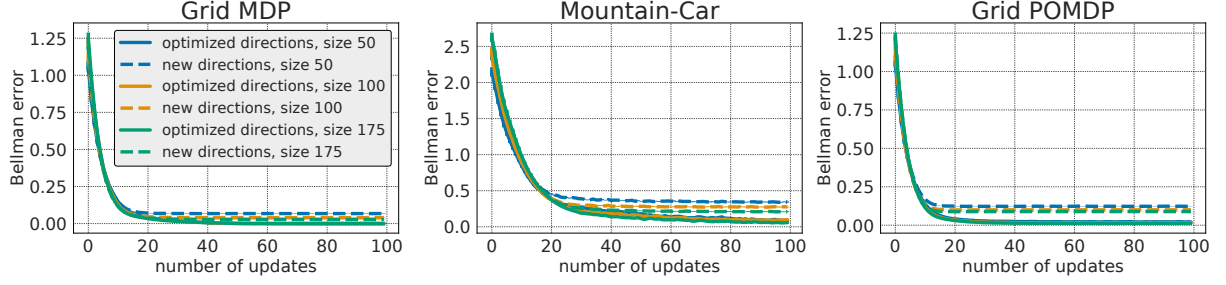


Figure 7.4: Bellman error v. iteration for three simple test domains, varying the amount of computation per iteration. We show error separately in directions we have optimized over and in new random directions. Average of 25 random seeds of the direction  $m_i$  with the highest bellman error per seed; all error bars are smaller than the line widths. The center panel shows the effect on Bellman error when we have higher- dimensional feature vectors. The rightmost panel shows the effect on Bellman error when the agent has less information about the exact state. In both cases the convergence rate stays similar, but we need more directions  $m_i$  to adequately sample the boundary of  $\Phi$  (i.e., to lower the asymptotic error on new directions).

## 7.11 More on Special Cases

### More on Special Cases

With the above pruning strategy, our dynamic programming iteration generalizes PBVI [210].

PBVI was defined originally for POMDPs, but it extends readily to PSRs as well: we just sample predictive states instead of belief states. To relate PBVI to our method, we look at a single task, with reward coefficient vector  $r$ . We sample a set of belief states or predictive states  $q_i$ ; these are the directions that PBVI will use to decide which value functions ( $\alpha$ -vectors) to retain. Based on these, we set the successor feature matrix directions to be  $m_i = r q_i^T$  for all  $i$ .

Now, when we search within our backed up set  $\Phi^{(H)}$  for the maximal element in direction  $m_i$ , we get some successor feature matrix  $\phi$ . Because  $\text{tr}(\phi^T r q_i^T)$  is maximal, we know that  $\text{tr}(q_i^T \phi^T r) = q_i^T (\phi^T r)$  is also maximal: that is,  $\phi^T r$  is as far as possible in the direction  $q_i$ . But  $\phi^T r$  is a backed-up value function under the reward  $r$ ; so,  $\phi^T r$  is exactly the value function that PBVI would retain when maximizing in the direction  $q_i$ .

## 7.12 Experiments: Dynamic Programming

### Experiments: Dynamic Programming

We tried our dynamic programming method on several small domains: the classic mountain-car domain and a random  $18 \times 18$  gridworld with full and partial observability. We evaluated both planning and feature matching; results for the former are discussed in this section, and an example of the latter is in Fig. 7.3. We give further details of our experimental setup in the supplementary material online. At a high level, our experiments show that the algorithms behave as expected, and that they are practical for small domains. They also tell us about limits on scaling: the tightest of these limits is our ability to represent  $\Phi$  accurately, governed by the number of boundary points that we retain for each  $\Phi_{ao}$ .

In mountain-car, the agent has two actions: accelerate left and accelerate right. The state is (position, velocity), in  $[-1.2, 0.6] \times [-0.07, 0.07]$ . We discretize to a  $12 \times 12$  mesh with piecewise-constant approximation. Our one-step features are radial basis functions of the state, with values in  $[0, 1]$ . We use 9 RBF centers evenly spaced on a  $3 \times 3$  grid. In the MDP gridworld, the agent has four deterministic actions: up, down, left, and right. The one-step features are  $(x, y)$  coordinates scaled to  $[-1, 1]$ , similar to Fig. 7.3. In the POMDP gridworld, the actions are stochastic, and the agent only sees a noisy indicator of state. In all domains, the discount is  $\gamma = 0.9$ .

Fig. 7.4 shows how Bellman error evolves across iterations of dynamic programming. Since  $\Phi$  is a set, we evaluate error by looking at random projections: how far do  $\Phi$  and the backup of  $\Phi$  extend in a given direction? We evaluate directions  $m_i$  that we optimized for during backups, as well as new random directions.

Note that the asymptotes for the new-directions lines are above zero; this persistent error is due to our limited-size representation of  $\Phi$ . The error decreases as we increase the number of boundary points that we store. It is larger in the domains with more features and more uncertainty (center and right panels), due to the higher-dimensional  $A^\pi$  matrices and the need to sample mixed (uncertain) belief states.

### 7.13 Conclusion

This work introduces *successor feature sets*, a new representation that generalizes successor features. Successor feature sets represent and reason about successor feature predictions for all policies at once, and respect the compositional structure of policies, in contrast to other approaches that treat each policy separately. The set represents the boundaries of what is achievable in the future, and how these boundaries depend on our initial state. This information lets us read off optimal policies or imitate a demonstrated behavior.

We give algorithms for working with successor feature sets, including a dynamic programming algorithm to compute them, as well as algorithms to read off policies from them. The dynamic programming update is a contraction mapping, and therefore convergent. We give both exact and approximate versions of the update. The exact version can be intractable, due to the so-called “curse of dimensionality” and “curse of history.” The approximate version mitigates these curses using point-based sampling.

Finally, we present computational experiments. These are limited to relatively small, known environments; but in these environments, we demonstrate that we can compute successor feature sets accurately, and that they aid generalization. We also explore how our approximations scale with environment complexity.

Overall we believe that our new representation can provide insight on how to reason about policies in a dynamical system. We know, though, that we have only scratched the surface of possible strategies for working with this representation, and we hope that our analysis can inform future work on larger-scale environments.

## 7.14 Extended Details

### 7.14.1 Feature matching

In this section we give the algorithm for imitation by feature matching, summarized as Alg. 5.

Our policy will be nonstationary: that is, its actions will depend on an internal policy state (defined below) as well as the environment’s current predictive state  $q_t$ .

Our algorithm updates its target feature vector over time in order to compensate for random outcomes (the action sampled from the policy and the next state sampled from the transition distribution). We write  $\phi_t^d$  for the target at time step  $t$ , and initialize  $\phi_1^d = \phi^d$ . Updates of this sort are necessary: we might by chance visit a state where it is impossible to achieve the original target  $\phi_1^d$ , but that does not mean that our policy has failed. Instead, the policy guarantees always to pick a target  $\phi_t^d$  that is achievable given the state  $q_t$  at step  $t$ , in a way that guarantees that on average we achieve the original target  $\phi_1^d$  from the initial state  $q_1$ .

To guarantee that the target is always achievable, our policy maintains the invariant that  $\phi_t^d \in \Phi q_t$ . By the definition of  $\Phi$ , the discounted feature vectors in  $\Phi q$  are exactly the ones that are achievable starting from state  $q$ , so this invariant is necessary and sufficient to ensure that  $\phi_t^d$  is achievable. At the first time step, we test whether  $\phi_1^d \in \Phi q_1$ . If yes, our invariant is satisfied

and we can proceed; if no, then we know that we have been given an impossible task. In the latter case we could raise an error, or we could raise a warning and look for the closest achievable vector  $\phi \in \Phi_q$  to  $\phi_1^d$ .

Our actions at step  $t$  and our targets at step  $t+1$  will be functions of the current environment state  $q_t$  and our current target  $\phi_t^d$ . As such,  $\phi_t^d$  is the internal policy state mentioned above.

We pick our actions and targets as follows. According to the Bellman equations, the successor feature set  $\Phi$  is equal to the convex hull of the union of  $\Phi_a$  over all  $a$ . Each matrix in  $\Phi$  can therefore be written as a convex combination of action-specific matrices, each one chosen from one of the sets  $\Phi_a$ . That means that each vector in  $\Phi_{q_t}$  can be written as a convex combination of vectors in  $\Phi_a q_t$ .

Write our target  $\phi_t^d$  in this way, say  $\phi_t^d = \sum_{i=1}^{\ell} p_{it} \phi_{it}$ , by choosing actions  $a_{it}$ , vectors  $\phi_{it} \in \Phi_{a_{it}} q_t$ , and weights  $p_{it} \geq 0$  with  $\sum_i p_{it} = 1$ . Then, at the current time step, our algorithm chooses an index  $i$  according to the probabilities  $p_{it}$ , and executes the corresponding action  $a_{it}$ .

Now let  $i$  be the chosen index, and write  $a = a_{it}$  for the chosen action. Again according to the Bellman equations, the point  $\phi_{it}$  is of the form  $[F_a + \gamma \sum_o \Phi T_{ao}] q_t$ . In particular, we can choose vectors  $\phi_{ot} \in \Phi T_{ao} q_t$  for each  $o$  such that

$$\phi_{it} = F_a q_t + \gamma \sum_o \phi_{ot}$$

Writing  $p_{ot} = P(o \mid q_t, a)$  for all  $o$ , we can multiply and divide by  $p_{ot}$  within the sum, and conclude

$$\phi_{it} = \mathbb{E}_o[F_a q_t + \gamma \phi_{ot} / p_{ot}]$$

That is, we can select our target for time step  $t+1$  as  $\phi_{t+1}^d = \phi_{ot} / p_{ot}$ , where  $o = o_t$  is our next



observation. To see why, note that our expected discounted feature vector at time  $t$  will remain the same: the LHS (the current target) is equal to the RHS (the expected one-step contribution plus discounted future target). And, note that the target at the next time step will always be feasible, maintaining our invariant: our state at the next time step will be

$$q_{t+1} = T_{ao}q_t/p_{ot}$$

and we have selected each  $\phi_{ot}$  to satisfy  $\phi_{ot} \in \Phi T_{ao}q_t$ , so

$$\phi_{ot}/p_{ot} \in \Phi T_{ao}q_t/p_{ot} = \Phi q_{t+1}$$

So, based on the observation that we receive, we can update our predictive state and target feature vector according to the equations above, and recurse. (In practice, numerical errors or incomplete convergence of  $\Phi$  could lead to an infeasible target; in this case we can project the target back onto the feasible set, which will result in some error in feature matching.)

Note that there may be more than one way to decompose  $\phi_t^d = \sum_{i=1}^{\ell} p_{it}\phi_{it}$ , or more than one way to decompose  $\phi_{it} = F_a q_t + \gamma \sum_o \phi_{ot}$ . If so, we can choose any valid decomposition arbitrarily.

### 7.14.2 Convergence of dynamic programming

We will show that the dynamic programming update for  $\Phi$  given in Section 7.8 is a contraction, which implies bounds on the convergence rate of dynamic programming. We will need a few definitions and facts about norms and metrics.

### 7.14.2.1 Norms

Given any symmetric, convex, compact set  $S$  with nonempty interior, we can construct a norm by treating  $S$  as the unit ball. The norm of a vector  $x$  is then the smallest multiple of  $S$  that contains  $x$ .

$$||x||_S = \inf_{x \in cS} c$$

This is a fully general way to define a norm: any norm can be constructed this way by using its own unit ball as  $S$ . That is, if  $B = \{x \mid ||x|| \leq 1\}$ , then

$$||x|| = ||x||_B$$

We will use the shorthand  $|| \cdot ||_p$  for an  $L_p$ -norm: e.g.,  $L_1$ ,  $L_2$  (Euclidean norm), or  $L_\infty$  (sup norm). If we start from an asymmetric set  $S$ , we can symmetrize it to get

$$\bar{S} = \{\alpha s + (1 - \alpha)s' \mid s \in S, s' \in -S, \alpha \in [0, 1]\}$$

(This is the convex hull of  $S \cup -S$ .) Given any norm  $|| \cdot ||$ , we can construct a *dual norm*  $|| \cdot ||^*$ :

$$||y||^* = \sup_{||x|| \leq 1} x \cdot y$$

This definition guarantees that dual norms satisfy Hölder's inequality:

$$x \cdot y \leq ||x|| ||y||^*$$

We will write  $S^*$  for the unit ball of the dual norm  $\|\cdot\|_S^*$ . Taking the dual twice returns to the original norm:  $\|\cdot\|_S^{**} = \|\cdot\|_S$  and  $S^{**} = S$ .

Given any two norms  $\|\cdot\|_P$  and  $\|\cdot\|_Q$  and their corresponding unit balls  $P$  and  $Q$ , the *operator norm* of a matrix  $A$  is

$$\|A\|_{P,Q} = \sup_{x \in P} \|Ax\|_Q = \sup_{x \in P, y \in Q^*} y^T Ax$$

This definition ensures that Hölder's inequality extends to operator norms:

$$\|Ax\|_Q \leq \|A\|_{P,Q} \|x\|_P$$

The norm of the transpose of a matrix can be expressed in terms of the duals of  $\|\cdot\|_P$  and  $\|\cdot\|_Q$ :

$$\|A^T\|_{Q^*,P^*} = \|A\|_{P,Q}$$

If  $P$  and  $Q$  are the same, we will shorten to

$$\|A^T\|_{P^*} = \|A\|_P$$

Given a norm, we can define the Hausdorff metric between sets:

$$d(X, Y) = \max(\bar{d}(X, Y), \bar{d}(Y, X))$$

$$\bar{d}(X, Y) = \sup_{x \in X} \inf_{y \in Y} \|x - y\|$$

If  $V$  is any real vector space (such as  $\mathbb{R}^{d \times k}$ ), the Hausdorff metric makes the set of non-empty

compact subsets of  $V$  into a complete metric space. Given a metric, a *contraction* is a function  $f$  that reduces the metric by a constant factor:

$$d(f(X), f(Y)) \leq \beta d(X, Y)$$

The factor  $\beta \in [0, 1)$  is called the *modulus*. If  $\beta = 1$  then  $f$  is called a *nonexpansion*. For a linear operator  $A$ , with metric  $d(x, y) = \|x - y\|_P$ , the modulus is the same as the operator norm  $\|A\|_{P,P}$ . The *Banach fixed-point theorem* guarantees the existence of a fixed point of any contraction on a complete metric space.

#### 7.14.2.2 Norms for POMDPs and PSRs

We can bound the transition operators  $T_{ao}$  for POMDPs and PSRs using operator norms that correspond to the set of valid states. In POMDPs, valid belief states are probability distributions, and therefore satisfy  $\|q\|_1 \leq 1$ . For PSRs, there is no single norm that works for all models. Instead, for each PSR, we only know that there exists a norm  $\|\cdot\|_{\bar{S}}$  such that all valid states are in the unit ball  $\bar{S}$ . (We can get  $\bar{S}$  by symmetrizing the PSR's set of valid states  $S$ .) We will write  $\|\cdot\|_{\bar{S}}$  in both cases, by taking  $S$  to be the probability simplex if our model is a POMDP. Given these definitions, we are guaranteed that, for each  $a$ ,

$$\|T_a\|_S \leq 1 \quad \text{where} \quad T_a = \sum_o T_{ao}$$

We also know that each transition operator  $T_{ao}$  maps states to unnormalized states: it maps  $S$  to the cone generated by  $S$ , i.e.,  $\{\lambda s \mid s \in S, \lambda \geq 0\}$ .

### 7.14.2.3 Convergence: key step

The key step in the proof of convergence is to analyze

$$\sum_o \Phi T_{ao}$$

for a fixed action  $a$ . We will show that this operation is a nonexpansion in the Hausdorff metric based on a particular norm. To build the appropriate norm, we can start from norms for our states and our features. For states we will use the norm that corresponds to our state space:  $\|\cdot\|_{\bar{S}}$ . For features we can use any norm  $\|\cdot\|_F$ . For elements of  $\Phi$  we can then use the operator norm for  $\bar{S}$  and  $F$ :  $\|\cdot\|_{F,\bar{S}}$ . For sets like  $\Phi$  we can use the Hausdorff metric based on  $\|\cdot\|_{F,\bar{S}}$ , which we will write as just  $d(\cdot, \cdot)$ .

For simplicity we will first analyze distance to a point: start by assuming  $d(\Phi, \{0\}) \leq k$  for some  $k$ . Now, for each  $a$ ,

$$\begin{aligned} d(\sum_o \Phi T_{ao}, \{0\}) &= \sup_{\psi_o \in \Phi T_{ao}} \|\sum_o \psi_o\|_{F,\bar{S}} \\ &= \sup_{\phi_o \in \Phi} \|\sum_o \phi_o T_{ao}\|_{F,\bar{S}} \\ &= \sup_{\phi_o \in \Phi} \sup_{f \in F^*, q \in \bar{S}} f^T \sum_o \phi_o T_{ao} q \end{aligned}$$

where we have written  $\sup_{\psi_o \in \Phi T_{ao}}$  as shorthand for  $\sup_{\psi_1 \in \Phi T_{a,1}} \sup_{\psi_2 \in \Phi T_{a,2}} \dots$ , i.e., one supremum per observation.

Since  $q$  is the solution to a linear optimization problem, we can assume it is an extreme point of the feasible region  $\bar{S}$ , which means either  $q \in S$  or  $q \in -S$ . Assume  $q \in S$ ; the other case is symmetric. This lets us replace  $\sup_{q \in \bar{S}}$  with  $\sup_{q \in S}$ .

We next want to simplify the supremum over  $f$ . We can do this in two steps: first, the supremum can only increase if we let the choice of  $f$  depend on  $o$  (which we write as  $\sup_{f_o}$ ). Second, Hölder's inequality tells us that  $\|\phi_o^T f_o\|_{\bar{S}^*} \leq k$ , since  $\|f_o\|_{F^*} \leq 1$  and  $\|\phi_o^T\|_{\bar{S}^*, F^*} \leq k$ . So, optimizing over  $k\bar{S}^*$  instead of just over vectors of the form  $\phi_o^T f_o$  can again only increase the supremum. We therefore have

$$\begin{aligned}
d(\sum_o \Phi T_{ao}, \{0\}) & \\
&\leq \sup_{\phi_o \in \Phi} \sup_{q \in S, f_o \in F^*} \sum_o f_o^T \phi_o T_{ao} q \\
&\leq \sup_{q \in S} \sup_{r_o \in k\bar{S}^*} \sum_o r_o^T T_{ao} q
\end{aligned}$$

We can now solve the optimizations over  $r_o$ . Note that the normalization vector  $u$  is in  $\bar{S}^*$ :  $u \cdot s = 1$  for every  $s \in S$ , so  $u \cdot \bar{s} \in [-1, 1]$  for every  $\bar{s} \in \bar{S}$ . And, for any valid state  $s$ , no vector in  $\bar{S}^*$  can have dot product larger than 1 with  $s$ , by definition of  $\bar{S}^*$ .  $T_{ao}q$  is a nonnegative multiple of a valid state for each  $o$ ; therefore,  $r_o = ku$  is an optimal solution for each  $o$ , and we have

$$\begin{aligned}
d(\sum_o \Phi T_{ao}, \{0\}) &\leq \sup_{q \in S} \sum_o k u^T T_{ao} q \\
&= k \sup_{q \in S} u^T T_a q \\
&= k = d(\Phi, \{0\})
\end{aligned}$$

To handle distances to a general set  $\Phi$ , we need to track a sup inf instead of just a sup. Assume wlog that

$$d(\sum_o \Phi T_{ao}, \sum_o \Psi T_{ao}) = \bar{d}(\sum_o \Phi T_{ao}, \sum_o \Psi T_{ao})$$

(the other ordering is symmetric). Then

$$\begin{aligned}
& \bar{d}(\sum_o \Phi T_{ao}, \sum_o \Psi T_{ao}) \\
&= \sup_{\phi_o \in \Phi} \inf_{\psi_o \in \Psi} \|\sum_o \phi_o T_{ao} - \sum_o \psi_o T_{ao}\|_{F, \bar{S}} \\
&= \sup_{\phi_o \in \Phi} \inf_{\psi_o \in \Psi} \|\sum_o (\phi_o - \psi_o) T_{ao}\|_{F, \bar{S}}
\end{aligned}$$

The argument proceeds from here exactly as above, since we know that  $\|\phi_o - \psi_o\|_{F, \bar{S}}$  is bounded by  $d(\Phi, \Psi)$  for each  $o$ .

#### 7.14.2.4 Convergence: rest of the proof

The remaining steps in our dynamic programming update are multiplying by  $\gamma$ , adding  $F_a$ , and taking the convex hull of the union over  $a$ . Multiplying the sets by  $\gamma$  changes the modulus from 1 to  $\gamma$ . Adding the same vector to both sets does not change the modulus. Finally, convex hull of union also leaves the modulus unchanged: more specifically, if  $f_1, f_2, \dots$  are all contractions of modulus  $\gamma$ , then the mapping

$$\Phi \rightarrow \text{conv} \bigcup_i f_i(\Phi)$$

is also a contraction of modulus  $\gamma$ . To see why, consider two sets  $\text{conv} \bigcup_i f_i(\Phi)$  and  $\text{conv} \bigcup_i f_i(\Psi)$ , with  $d(\Phi, \Psi) = 1$ . Consider a point in the former set: it can be written as  $\sum_j \alpha_j \phi_j$  with each  $\phi_j$  in one of the sets  $f_i(\Phi)$  and the  $\alpha_j$  a convex combination. For each  $j$ , we can find a point in the corresponding set  $f_i(\Psi)$  at distance at most  $\gamma$ , since  $f_i$  is a contraction. Using the triangle inequality on the convex combination, the final distance is therefore at most  $\gamma$ .

Putting everything together, we have that the dynamic programming update is a contraction of modulus  $\gamma < 1$ . From here, the Banach fixed-point theorem guarantees that there exists a unique fixed point of the update, and that each iteration of dynamic programming brings us closer to this fixed point by a factor  $\gamma$ , as long as we initialize with a nonempty compact subset of the set of matrices.

### 7.14.3 Background on PSRs

Here we describe a mechanical way to define a valid PSR, given some information about a controlled dynamical system. This method is fully general: if it is possible to express a dynamical system as a PSR, we can use this method to do so. And, PSRs constructed this way allow a nice interpretation of the otherwise-opaque PSR state vector. To describe this method, it will help to define a kind of experiment called a *test*.

#### 7.14.3.1 Tests

A test  $\tau$  consists of a sequence of actions  $A_\tau = (a_1, a_2, \dots, a_\ell)$  and a function  $F_\tau : \{1 \dots O\}^\ell \rightarrow \mathbb{R}$ . We execute  $\tau$  by executing  $a_1, a_2, \dots, a_\ell$  starting from some state  $q$ . We record the resulting observations  $o_t, o_{t+1}, \dots, o_{t+\ell-1}$ , and feed them as inputs to  $F_\tau$ ; the output is called the *test outcome*. The *test value* is the expected outcome

$$\tau(q) = \mathbb{E}(F(o_t, o_{t+1}, \dots, o_{t+\ell-1}) \mid q_t = q, \text{ do } A^\tau)$$

A *simple test* is one where the function  $F_\tau$  is the indicator of a given sequence of  $\ell$  observations; in this case the test value is also called the test *success probability*. Tests that are not simple are



*compound*. Below, we will use tests to construct PSRs. If we use exclusively simple tests, we will call the result a *simple PSR*; else it will be a *transformed PSR*.

We can express compound tests as linear combinations of simple tests: we can break the expectation into a sum over all possible sequences of  $\ell$  observations to get

$$\tau(q) = \sum_{o_1 \dots o_\ell} P(o_1 \dots o_\ell \mid q, \text{do } A^\tau) F^\tau(o_1, \dots, o_\ell)$$

and each term in the summation is a fixed multiple of a simple test probability.

In a PSR, for any test  $\tau$ , it turns out that the function  $\tau(q)$  is *linear*: for a simple test with actions  $a_1 \dots a_\ell$  and observations  $o_1 \dots o_\ell$ ,

$$\begin{aligned} \tau(q) &= P(o_1, \dots, o_\ell \mid q, \text{do } A^\tau) \\ &= u^T T_{a_\ell o_\ell} \cdot T_{a_{\ell-1} o_{\ell-1}} \cdots T_{a_2 o_2} \cdot T_{a_1 o_1} q \end{aligned}$$

which is linear in  $q$ . For a compound test, the value is linear because it is a linear combination of simple tests.

In fact, this linearity property is the defining feature of PSRs: a dynamical system can be described as a PSR exactly when we can define a state vector that makes all test values into linear functions. That is, we can write down a PSR iff there exist state extraction functions  $q_t = Q_t(a_1, o_1, a_2, o_2, \dots, a_{t-1}, o_{t-1}) \in \mathbb{R}^k$  such that, for all tests  $\tau$ , there exist prediction vectors  $m_\tau \in \mathbb{R}^k$  such that the value of  $\tau$  is  $\tau(q_t) = m_\tau \cdot q_t$ . There may be many ways to define a state vector for a given dynamical system; we are interested particularly in *minimal* state vectors, i.e., those with the smallest possible dimension  $k$ .

Above, we saw one direction of the equivalence between PSRs and dynamical systems

satisfying the linearity property: given a PSR, the state update equations define  $Q_t$ , and the expression above gives  $m_\tau$ . We will demonstrate the other direction in the next section below, by constructing a PSR given  $Q_t$  and  $m_\tau$ .

Given a test  $\tau$ , an action  $a$ , and an observation  $o$ , define the *one-step extension*  $\tau^{ao}$  as follows: let  $a_1, \dots, a_\ell$  be the sequence of actions for  $\tau$ , and let  $F(\cdot)$  be the statistic for  $\tau$ . Then the action sequence for  $\tau^{ao}$  is  $a, a_1, \dots, a_\ell$ , and the statistic for  $\tau^{ao}$  is  $F^o(\cdot)$ , defined as

$$F^o(o_1, \dots, o_{\ell+1}) = \mathbb{I}(o_1 = o)F(o_2, \dots, o_{\ell+1})$$

In words, the one-step extension tacks  $a$  onto the beginning of the action sequence. It then applies  $F(\cdot)$  on the observation sequence starting at the *second* time step in the future, but it either keeps the result or zeros it out, depending on the value of the first observation.

We can relate the value of a one-step extension test  $\tau^{ao}$  to the value of the original test  $\tau$ :

$$\tau^{ao}(q) = P(o \mid q, \text{do } a)\tau(q')$$

where  $q' = T_{ao}q/u^T T_{ao}q$  is the state we reach from  $q$  after executing  $a$  and observing  $o$ . (We can derive this expression by conditioning on whether we receive  $o$  or not: with probability  $P(o \mid q, \text{do } a)$  the outcome of  $\tau^{ao}$  is as if we executed  $\tau$  from  $q'$ , else the outcome of  $\tau^{ao}$  is zero.)

For example, in any PSR, we can define the *constant test*  $\tau_1$ , which has an empty action sequence and always has outcome equal to 1. The one-step extensions of this test give the probabilities of different observations at the current time step:

$$\tau_1^{ao}(q) = P(o \mid q, \text{do } a)$$

### 7.14.3.2 PSRs and tests

We can use tests to construct a PSR from a dynamical system, and to interpret the resulting state vector. This interpretation explains the terminology *predictive state*: our state is equivalent to a vector of predictions about the future. Crucially, these predictions are for observable outcomes of experiments that we could actually conduct. This is in contrast to a POMDP's state, which may be only partially observable.

In more detail, suppose we have a dynamical system with a minimal state  $q_t$  that satisfies the linearity property defined above. That is, suppose we have functions  $Q_t$  that compute minimal states  $q_t = Q_t(a_1, o_1, \dots, a_{t-1}, o_{t-1}) \in \mathbb{R}^k$ , and vectors  $m_\tau \in \mathbb{R}^k$  that predict test values  $\tau(q_t) = m_\tau \cdot q_t$ . We will show that each coordinate of  $q_t$  is a linear combination of test values, and we will define PSR parameters  $T_{ao,u}$  that let us update  $q_t$  recursively, instead of having to compute  $q_t$  from scratch at each time step using the state extraction functions  $Q_t$ .

Pick  $k$  tests  $\tau_1 \dots \tau_k$ , and define  $q'_t \in \mathbb{R}^k$  to have coordinates  $[q'_t]_i = m_{\tau_i} \cdot q_t$ . Equivalently, let  $S$  be the matrix with rows  $m_{\tau_i}$ , and write  $q'_t = Sq_t$ . We say that our set of tests is linearly independent if their prediction vectors  $m_{\tau_i}$  are linearly independent — equivalently, if the matrix  $S$  is invertible. If this happens to be true for  $\tau_1 \dots \tau_k$ , then  $q'_t$  is another minimal state vector for our dynamical system: the value of test  $\tau$  is  $m_\tau \cdot q_t = m_\tau \cdot S^{-1}q'_t$ , which is a linear function of  $q'_t$ . Furthermore, we have interpreted each coordinate of  $q_t$  as a linear combination of tests, as promised:  $q_t = S^{-1}q'_t$ .

It turns out that we can always pick  $k$  linearly independent tests. To see why: the empty list is linearly independent. For any list shorter than  $k$ , there will always exist another linearly independent test that we can add: if not, every possible  $m_\tau$  is a linear combination of our existing

vectors  $m_{\tau_i}$ , meaning that we can express  $m_{\tau} \cdot q_t$  as a linear function of  $m_{\tau_i} \cdot q_t$ . We could then define  $[q'_t]_i = m_{\tau_i} \cdot q_t$  as before, and get a state vector of dimension smaller than  $k$ , contradicting the minimality of  $q_t$ .

Now it just remains to show how to update our state vector recursively. We will describe first how to update  $q'_t$ , and then how to update the original state vector  $q_t$ .

For each of the tests  $\tau_i$  that make up  $q'_t$ , consider the one-step extensions  $\tau_i^{ao}$  for each  $a$  and  $o$ . Write  $m_i^{ao}$  for the corresponding prediction vectors, so that  $\tau_i^{ao}(q'_t) = m_i^{ao} \cdot q'_t$ . And, write  $m_1$  for the prediction vector of the constant test  $\tau_1$ .

We can now define PSR parameters in terms of these prediction vectors: let  $T_{ao}$  be the matrix with rows  $m_i^{ao}$ ,

$$[T_{ao}]_{ij} = [m_i^{ao}]_j$$

and define

$$u = m_1$$

If we now use  $T_{ao}$  to update  $q'_t$ , we get

$$\begin{aligned} [T_{ao}q'_t]_i &= m_i^{ao} \cdot q'_t \\ &= \tau_i^{ao}(q'_t) \\ &= P(o \mid q'_t, \text{do } a) \tau_i(q'_{t+1}) \\ &= P(o \mid q'_t, \text{do } a) [q'_{t+1}]_i \end{aligned}$$

or equivalently

$$q'_{t+1} = T_{ao}q'_t / P(o \mid q'_t, \text{do } a)$$

which is the correct update for  $q'_t$  after action  $a$  and observation  $o$ . And,

$$\begin{aligned}
u \cdot T_{ao} q'_t &= u \cdot q'_{t+1} P(o \mid q'_t, \text{do } a) \\
&= m_1 \cdot q'_{t+1} P(o \mid q'_t, \text{do } a) \\
&= P(o \mid q'_t, \text{do } a)
\end{aligned}$$

demonstrating that  $u$  correctly computes observation probabilities and lets us normalize our state vector.

Recapping, if we use the new state vector  $q'_t$ , each coordinate of our state is a test value, and we can interpret our parameter matrices in terms of tests. The rows of  $T_{ao}$  correspond to one-step extension tests, and the normalization vector  $u$  corresponds to the constant test.

For the original state vector  $q_t$ , we can make a similar interpretation. Define the one-step extension of a linear combination of tests by passing the extension through the linear combination: that is, given a linear combination  $\sigma = \sum_i a_i \tau_i$  for coefficients  $a_i$  and tests  $\tau_i$ , the one-step extension  $\sigma^{ao}$  is  $\sum_i a_i \tau_i^{ao}$ . With this definition, the exact same construction of  $T_{ao}$  and  $u$  works for our original state vector. That is, each component of  $q_t$  can be interpreted as a linear combination of tests; each row of  $T_{ao}$  is the prediction vector for a one-step extension of one of these linear combinations; and  $u$  is the prediction vector for the constant test.

#### 7.14.4 Background on policies

We can represent a horizon- $H$  deterministic policy  $\pi$  as a balanced tree of depth  $H$  (Fig. 7.1). We start at the root of the tree. At each node, we execute the corresponding action  $a$ , branch to a child node  $\pi(o)$  depending on the resulting observation  $o$ , and repeat.

We can write a horizon- $H$  stochastic policy as a mixture of horizon- $H$  deterministic policies — i.e., a convex combination of depth- $H$  trees. To execute a stochastic policy, we alternate between choosing actions and receiving observations, as follows. To choose an action, we look at the labels of the root nodes of all of the policy trees in our mixture: the probability of action  $a$  is the total weight of trees whose root label is  $a$ . Given the action  $a$ , we keep only the trees with root label  $a$ , and renormalize the mixture weights to sum to 1. To incorporate an observation, we branch to a child node within each tree according to the received observation. That is, we replace each tree  $\pi$  in our mixture by its child  $\pi(o)$ , keeping the same weight. We write  $\pi(a, o)$  for the resulting mixture after choosing action  $a$  and incorporating observation  $o$ .

#### 7.14.5 Successor feature matrices

In a POMDP or PSR, we do not want a separate successor feature vector at each state, since we do not have access to a fully observable state. Instead, the successor feature representation is a function of the continuous predictive state or belief state  $q$ . Here, we show that this function is *linear* in  $q$ : that is, we can represent it as

$$\phi^\pi(q) = A^\pi q$$

for some matrix  $A^\pi \in \mathbb{R}^{d \times k}$  that depends on the policy  $\pi$ . We also show how to compute the successor feature matrix  $A^\pi$ .

We can show linearity, and at the same time compute  $A^\pi$ , by induction over the horizon. In the base case (a horizon of  $H = 1$ ), our total discounted features are the same as our one-step

features:

$$\phi^\pi(q) = \mathbb{E}_\pi[f(q, a)] = \sum_a P(a \mid \pi) F_a q$$

Note that the RHS is a linear function of  $q$ , as claimed.

In the inductive case (horizon  $H > 1$ ), we can split our our expected total discounted features into contributions from the present and the future:

$$\phi^\pi(q_t) = \mathbb{E}_\pi[f(q_t, a_t) + \gamma \phi^{\pi(a_t, o_t)}(q_{t+1})]$$

In the the contribution of future time steps, note both the one-step updated policy  $\pi(a_t, o_t)$  and the one-step updated predictive state  $q_{t+1}$ . Expanding the expectation and substituting our expression for  $f(\dots)$ , we get

$$\phi^\pi(q_t) = \sum_{a, o} P(a \mid \pi) P(o \mid q_t, \text{do } a) [F_a q_t + \gamma \phi^{\pi(a, o)}(q_{t+1})]$$

We can inductively assume that  $\phi^{\pi(a, o)}$  is linear, since  $\pi(a, o)$  is a shorter-horizon policy than  $\pi$ . That is, we can write  $\phi^{\pi(a, o)}(q) = A^{\pi(a, o)} q$ . Substituting this expression, and using  $q_{t+1} = T_{ao} q_t / P(o \mid q_t, \text{do } a)$ , we see that  $P(o \mid q_t, \text{do } a)$  cancels:

$$\phi^\pi(q_t) = \sum_a P(a \mid \pi) \left[ F_a q_t + \gamma \sum_o A^{\pi(a, o)} T_{ao} q_t \right]$$

We can observe that the RHS is a linear function of  $q_t$ , which completes our inductive proof of linearity.

Because of linearity, there exists a matrix  $A^\pi$  such that  $\phi^\pi(q) = A^\pi q$ . With this notation,

$$A^\pi q_t = \sum_a P(a \mid \pi) \left[ F_a q_t + \gamma \sum_o A^{\pi(a,o)} T_{ao} q_t \right]$$

Because the above equation must hold for any predictive state  $q_t$ , we get

$$A^\pi = \sum_a P(a \mid \pi) \left[ F_a + \gamma \sum_o A^{\pi(a,o)} T_{ao} \right]$$

This equation defines  $A^\pi$  recursively in terms of matrices for shorter-horizon policies. So, we can compute  $A^\pi$  by dynamic programming, working backward from horizon 1: we start by computing the matrices for all 1-step policies that we can get from  $\pi$  by fixing the first  $H - 1$  actions and observations, then combine these to compute the matrices for all 2-step policies that we can get from  $\pi$  by fixing the first  $H - 2$  actions and observations, and so forth.

For a deterministic policy with root action  $a$ , the recursion simplifies to

$$A^\pi = F_a + \gamma \sum_o A^{\pi(o)} T_{ao}$$

We can think of this recursion as working upward from the leaves of a single policy tree.

### 7.14.6 Implementation and experimental setup

In the following sections we discuss experimental details for computing the successor feature sets and using them for feature matching.



#### 7.14.6.1 Successor Feature Sets Implementation

We start by initializing each  $\Phi_{ao}$  to the set consisting of the zero matrix with dimension  $d \times k$ . We sample a fixed set of directions  $m_i \in \mathbb{R}^{d \times k}$  in a  $dk$ -sphere by sampling from a Gaussian and normalizing. To make computation more regular and GPU-friendly, we pre-allocate  $|A|$  tensors whose dimensions are  $\hat{n} \times d \times k$ ; we group the  $\Phi_{ao}$  matrices for all  $o$  and store them into the tensors.  $\hat{n}$  corresponds to the max number of boundary points that we store for each  $\Phi_{ao}$ . These tensors allow us efficiently solve

$$\arg \max \langle m_i, \phi \rangle \text{ for } \phi \in \bigcup_{a'} [F_{a'} + \gamma \sum_{o'} \Phi_{a'o'}] T_{ao}$$

because  $[F_{a'} + \gamma \sum_{o'} \Phi_{a'o'}] T_{ao}$  becomes a series of matrix multiplications which we can efficiently compute in parallel using a GPU. We try three different numbers of random projections: 50, 100 and 175. We prune the resulting boundary points to keep only the unique ones.

#### 7.14.6.2 Mountain-Car Implementation

In the mountain-car environment, the one-step features are radial basis functions of the state with values in  $[0, 1]$ . In particular, if we rescale the state space to  $[-1, 1] \times [-1, 1]$ , we set the 9 RBF centers to be at  $\{-0.8, 0, 0.8\} \times \{-0.8, 0, 0.8\}$ , a  $3 \times 3$  grid. The RBF widths in the rescaled state space are  $\sigma = 0.8$ .

#### 7.14.6.3 Grid POMDP Implementation

In the Grid POMDP environment the agent has 0.05 probability of transitioning to a random neighboring state, and an 0.05 probability of observing a random neighboring state

instead of the current state that it is in. We experimented as well with various amounts of noise (not shown); increasing the noise increases the effective dimensionality of the  $\Phi_{ao}$  sets, and we start to need more and more boundary points. Decreasing the noise makes the POMDP solution approach the MDP solution.

#### 7.14.6.4 Feature Matching Implementation

To implement step 5 or step 7 in Algorithm 5, we need to solve a small convex program. The best way to do so depends on the data structures we use to represent  $\Phi$  and  $\Phi_a$ . With our  $\Phi_{ao}$  decomposition, the sets  $\Phi_a$  are the convex hull of a finite set of vertices, with the number of vertices bounded by  $\hat{m}^{|O|}$ .

With this representation, for step 5, a reasonable approach is to use the Frank-Wolfe algorithm to find  $\phi_{it}$  and  $p_{it}$ : if we minimize the squared error between the LHS and RHS of the equation in step 5, the Frank-Wolfe method will naturally produce its output in the form of a convex combination of vertices of  $\Phi_a q_t$ .

Note that if we use Frank-Wolfe in step 5, every  $\phi_{it}$  we need to decompose in step 7 will be a vertex of one of the sets  $\Phi_a q_t$ . So, a reasonable approach is to annotate the vertices of  $\Phi_a$  as we compute them. Each vertex of  $\Phi_a$  will be constructed from some list of vertices of  $\Phi_{ao}$  for different  $o$ 's; we can just record which vertices of the sets  $\Phi_{ao}$  were used to construct each vertex of  $\Phi_a$ . We can multiply the current state  $q_t$  into the vertices of  $\Phi_a$  and  $\Phi_{ao}$  to get vertices of  $\Phi_a q_t$  and  $\Phi_{ao} q_t$ .

Note that if we use the above approach to decompose  $\phi_{it}$ , on the next time step  $\phi_{t+1}^d$  will be a vertex of  $\Phi q_{t+1}$ . So, we will not need to run Frank-Wolfe in step 5 on any subsequent time steps, unless numerical errors or incomplete convergence of the dynamic programming iteration

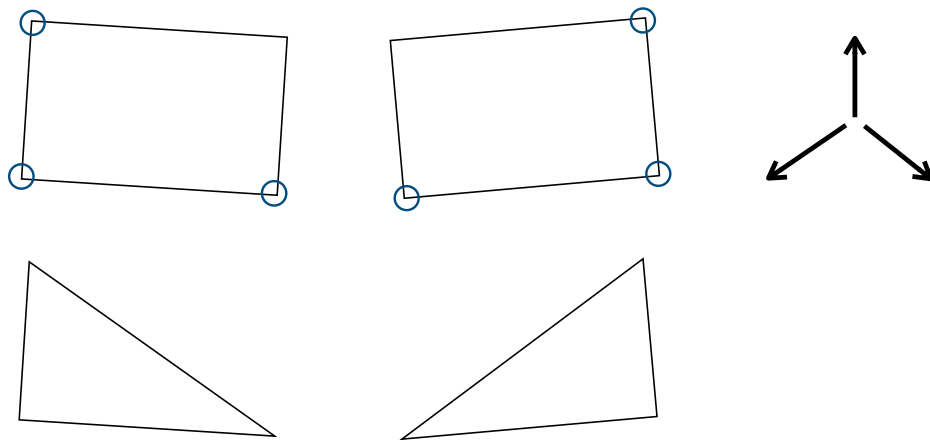


Figure 7.5: Example of the behavior of point-based approximation. Two convex sets (top row) are very similar. If we retain the maximal points (blue circles) in the indicated directions (arrows, top right), the convex hulls of the two sets of retained points are very different (bottom row).

cause us to drift away from being an exact vertex.

#### 7.14.7 Convergence of point-based approximations

While the exact dynamic programming update is a contraction, the point-based approximate dynamic programming update might not be. Fig. 7.5 shows why: an arbitrarily small change in a backed up set can lead to a large change in the point-based approximation of that set. Despite this fact, in practice we observe rapid convergence of the point-based approximate iteration.

Nonetheless, we can show that a small modification of our point-based approximate method converges and has bounded error. In particular, we analyze *monotone* point-based backups. Our analysis is similar to a corresponding analysis for monotone point-based value iteration in POMDPs.

For the modification, suppose that we are in a *stoppable* process: that is, suppose there is a designated *stop* action that ends the process, giving us some (possibly bad) terminal reward that can depend on the current state. In this case we can initialize our dynamic programming iteration with  $\{\phi^{\text{stop}}\}$ , the singleton set containing the successor feature matrix of the policy that

always takes the stop action. One common way that stoppable processes arise is if we have an *emergency* or *safety* policy — the equivalent of a big red button that causes our robot to shut down or retreat to a safe state. If we have an *idle* action, one that does not change our state but also does not yield a good reward, then we can use the always-idle policy as our safety policy.

In stoppable problems, with the given initialization, we know that our point-based backup will compute only *achievable* successor feature matrices — i.e., only those  $\phi$  that correspond to policies that we can always execute. So, we can use monotone backups: we can keep at each step the better of the existing (horizon  $H$ ) and the backed up (horizon  $H + 1$ ) successor feature matrix in each direction. (We can make the same modification to the exact backup operator as well: we merge together the current successor feature set  $\Phi^{(H)}$  with the backed up successor feature set  $\Phi^{(H+1)}$ , by taking the convex hull of their union. This modification does not affect the convergence proof or error bound given above.)

We can now analyze the monotone backup. First, note that the point-based backup of any set is a subset of the exact backup of that set, since we get the point-based backup by dropping elements of the exact backup. Second, note that both the point-based and the exact backup operators are monotone with respect to set inclusion: if  $P \subseteq Q$  then the backup of  $P$  is a subset of the backup of  $Q$ . So, the iterates from either are monotonically increasing. For the point-based backup, this means that the convex hull of our retained  $\phi$  matrices at each horizon always contains the convex hull at shorter horizons.

The exact backup sequence converges to the exact successor feature set, which is therefore an upper bound on the approximate backup sequence. By the monotone convergence theorem, this means that the monotone point-based iteration must converge to a subset of the exact successor feature set.

We can use this same argument to get a simple error bound: write  $\Phi^{PB}$  for the convergence point of the point-based iteration, and write  $\Phi^{PB+}$  for its one-step exact backup. Suppose that these two sets differ by at most  $\epsilon$  in Hausdorff metric. Then a standard argument shows that  $\Phi^{PB}$  cannot be farther than  $\frac{\epsilon}{1-\gamma}$  from the exact successor feature set. We know that  $\epsilon$  can be at most the size of the exact successor feature set (which is bounded by  $\frac{R_{\max}}{1-\gamma}$ ), but it may be much smaller.

## Chapter 8: Constraint Feedback

Imitation learning typically considers the problem of learning to optimize the behavior of an agent in an unknown environment using demonstration data or an interactive expert. This can be sufficient for simple tasks, but boiling down the learning goal only using demonstration data can be challenging for complex tasks. Many critical aspects of the desired behavior are more naturally expressed as constraints. For instance, the designer may want to limit unsafe actions, increase the diversity of trajectories to enable exploration, or approximate expert trajectories when rewards are sparse. Moreover, demonstration data or query an online expert might not be a natural formalism for stating certain learning objectives, such as safety desires (“avoid dangerous situations”) or exploration suggestions (“maintain a distribution over visited states that is as close to uniform as possible”).

In this chapter, we propose an algorithmic scheme that can handle a wide class of constraints in reinforcement learning and imitation learning tasks, specifically, any constraints that require expected values of some vector measurements (such as the use of an action) to lie in a convex set. This captures previously studied constraints (such as safety and proximity to an expert) and enables new classes of constraints (such as diversity). Our approach comes with rigorous theoretical guarantees and only relies on solving standard reinforcement learning tasks approximately. As a result, it can be easily adapted to work with any model-free reinforcement learning algorithm or model-based reinforcement learning algorithm, or imitation learning algorithm. Our experiments show that it matches previous algorithms that enforce safety via

constraints but can also enforce new properties that these algorithms cannot incorporate, such as diversity.

## 8.1 Introduction

Reinforcement learning (RL) typically considers the problem of learning to optimize the behavior of an agent in an unknown environment against a single scalar reward function. For simple tasks, this can be sufficient, but for complex tasks, boiling down the learning goal into a single scalar reward can be challenging. Moreover, a scalar reward might not be a natural formalism for stating certain learning objectives, such as safety desires (“avoid dangerous situations”) or exploration suggestions (“maintain a distribution over visited states that is as close to uniform as possible”). In these settings, it is much more natural to define the learning goal in terms of a vector of *measurements* over the behavior of the agent, and to learn a policy whose measurement vector is inside a target set.

We derive an algorithm, *approachability-based policy optimization* (APPROPO, pronounced like “apropos”), for solving such problems. Given a Markov decision process with vector-valued measurements, and a target constraint set, APPROPO learns a stochastic policy whose expected measurements fall in that target set (akin to Blackwell approachability in single-turn games, 33). We derive our algorithm from a game-theoretic perspective, leveraging recent results in online convex optimization. APPROPO is implemented as a *reduction* to any off-the-shelf reinforcement learning algorithm that can return an approximately optimal policy, and so can be used in conjunction with the algorithms that are the most appropriate for any given domain.

Our approach builds on prior work for reinforcement learning under constraints, such as the formalism of constrained Markov decision processes (CMDPs) introduced by Altman

[7]. In CMDPs, the agent’s goal is to maximize reward while satisfying some linear constraints over auxiliary costs (akin to our measurements). Altman [7] gave an LP-based approach when the CMDP is fully known, and more recently, model-free approaches have been developed for CMDPs in high-dimensional settings. For instance, Achiam et al. [3] constrained policy optimization (CPO) focuses on safe exploration and seeks to ensure approximate constraint satisfaction during the learning process. Tessler et al. [280] reward constrained policy optimization (RCPO) follows a two-timescale primal-dual approach, giving guarantees for the convergence to a fixed point. Le et al. [166] describe a batch off-policy algorithm with PAC-style guarantees for CMDPs using a similar game-theoretic formulation to ours.

While all of these works are only applicable to *orthant* constraints, our algorithm can work with arbitrary convex constraints. This enables APPROPO to incorporate previously studied constraint types, such as inequality constraints that represent safety or that keep the policy’s behavior close to that of an expert [277], as well as constraints like the aforementioned exploration suggestion, implemented as an entropy constraint on the policy’s state visitation vector. The entropy of the visitation vector was recently studied as the objective by Hazan et al. [115], who gave an algorithm capable of maximizing a concave function (e.g., entropy) over such vectors. However, it is not clear whether their approach can be adapted to the convex constraints setting studied here.

Our main contributions are: (1) a new algorithm, APPROPO, for solving reinforcement learning problems with arbitrary convex constraints; (2) a rigorous theoretical analysis that demonstrates that it can achieve sublinear regret under mild assumptions; and (3) a preliminary experimental comparison with RCPO [280], showing that our algorithm is competitive with RCPO on orthant constraints, while also handling a diversity constraint.



## 8.2 Background

We begin with a description of our learning setting. A *vector-valued Markov decision process* is a tuple  $M = (\mathcal{S}, \mathcal{A}, \beta, P_s, P_z)$ , where  $\mathcal{S}$  is the set of states,  $\mathcal{A}$  is the set of actions and  $\beta$  is the initial-state distribution. Each episode starts by drawing an initial state  $s_0$  from the distribution  $\beta$ . Then in each step  $i = 1, 2, \dots$ , the agent observes its current state  $s_i$  and takes action  $a_i \in \mathcal{A}$  causing the environment to move to the next state  $s_{i+1} \sim P_s(\cdot | s_i, a_i)$ . The episode ends after a certain number of steps (called the horizon) or when a terminal state is reached. However, in our setting, instead of receiving a scalar reward, the agent observes a  $d$ -dimensional *measurement* vector  $\mathbf{z}_i \in \mathbb{R}^d$ , which, like  $s_{i+1}$ , is dependent on both the current state  $s_i$  and the action  $a_i$ , that is,  $\mathbf{z}_i \sim P_z(\cdot | s_i, a_i)$ . (Although not explicit in our setting, reward could be incorporated in the measurement vector.)

Typically, actions are selected according to a (stationary) policy  $\pi$  so that  $a_i \sim \pi(s_i)$ , where  $\pi$  maps states to distributions over actions. We assume we are working with policies from some candidate space  $\Pi$ . For simplicity of presentation, we assume this space is finite, though possibly extremely large. For instance, if  $\mathcal{S}$  and  $\mathcal{A}$  are finite, then  $\Pi$  might consist of all deterministic policies. (Our results hold also when  $\Pi$  is infinite with minor technical adjustments.)

Our aim is to control the MDP so that measurements satisfy some constraints. For any policy  $\pi$ , we define the *long-term measurement*  $\mathbb{Z}(\pi)$  as the expected sum of discounted measurements:

$$\mathbb{Z}(\pi) \triangleq \mathbb{E} \left[ \sum_{i=0}^{\infty} \gamma^i \mathbf{z}_i \mid \pi \right] \quad (8.1)$$

for some discount factor  $\gamma \in [0, 1)$ , and where expectation is over the random process described above (including randomness inherent in  $\pi$ ).

Later, we will also find it useful to consider *mixed policies*  $\mu$ , which are distributions over finitely many stationary policies. The space of all such mixed policies over  $\Pi$  is denoted  $\Delta(\Pi)$ . To execute a mixed policy  $\mu$ , before taking any actions, a single policy  $\pi$  is randomly selected according to  $\mu$ ; then all actions henceforth are chosen from  $\pi$ , for the entire episode. The long-term measurement of a mixed policy  $\mathbb{Z}(\mu)$  is defined accordingly:

$$\mathbb{Z}(\mu) \triangleq \mathbb{E}_{\pi \sim \mu} [\mathbb{Z}(\pi)] = \sum_{\pi} \mu(\pi) \mathbb{Z}(\pi). \quad (8.2)$$

Our learning problem, called the *feasibility problem*, is specified by a convex *target set*  $\mathcal{C}$ . The goal is to find a mixed policy  $\mu$  whose long-term measurements lie in the set  $\mathcal{C}$ :

$$\text{Feasibility Problem: Find } \mu \in \Delta(\Pi) \text{ such that } \mathbb{Z}(\mu) \in \mathcal{C}. \quad (8.3)$$

For instance, in our experiments we consider a grid-world environment where the measurements include the distance traveled, an indicator of hitting a rock, and indicators of visiting various locations on the grid. The feasibility goal is to achieve at most a certain trajectory length while keeping the probability of hitting the rock below a threshold for safety reasons, and maintaining a distribution over visited states close to the uniform distribution to enable exploration. We can potentially also handle settings where the goal is to maximize one measurement (e.g., “reward”) subject to others by performing a binary search over the maximum attainable value of the reward (see §8.3.4).

### 8.3 Our Approach: APPROPO

Before giving details of our approach, we overview the main ideas, which, to a large degree, follow the work of Abernethy et al. [2], who considered the problem of solving two-player games; we extend these results to solve our feasibility problem (8.3).

Although feasibility is our main focus, we actually solve the stronger problem of finding a mixed policy  $\mu$  that minimizes the Euclidean distance between  $\mathbb{Z}(\mu)$  and  $\mathcal{C}$ , meaning the Euclidean distance between  $\mathbb{Z}(\mu)$  and its closest point in  $\mathcal{C}$ . That is, we want to solve

$$\min_{\mu \in \Delta(\Pi)} \text{dist}(\mathbb{Z}(\mu), \mathcal{C}) \quad (8.4)$$

where  $\text{dist}$  denotes the Euclidean distance between a point and a set.

Our main idea is to take a game-theoretic approach, formulating this problem as a game and solving it. Specifically, suppose we can express the distance function in Eq. (8.4) as a maximization of the form

$$\text{dist}(\mathbb{Z}(\mu), \mathcal{C}) = \max_{\lambda \in \Lambda} \lambda \cdot \mathbb{Z}(\mu) \quad (8.5)$$

for some convex, compact set  $\Lambda$ .<sup>1</sup> Then Eq. (8.4) becomes

$$\min_{\mu \in \Delta(\Pi)} \max_{\lambda \in \Lambda} \lambda \cdot \mathbb{Z}(\mu). \quad (8.6)$$

This min-max form immediately evokes interpretation as a two-person zero-sum game: the first player chooses a mixed policy  $\mu$ , the second player responds with a vector  $\lambda$ , and  $\lambda \cdot \mathbb{Z}(\mu)$  is the

---

<sup>1</sup>Note that the distance between a point and a set is defined as a minimization of the distance function over all points in the set  $\mathcal{C}$ , but here we require that it be rewritten as a maximization of a linear function over some other set  $\Lambda$ . We will show how to achieve this in §8.3.2.

amount that the first player is then required to pay to the second player. Assuming this game satisfies certain conditions, the final payout under the optimal play, called the *value* of the game, is the same even when the order of the players is reversed:

$$\max_{\lambda \in \Lambda} \min_{\mu \in \Delta(\Pi)} \lambda \cdot \mathbb{Z}(\mu). \quad (8.7)$$

Note that the policy  $\mu$  we are seeking is the *solution* of this game, that is, the policy realizing the minimum in Eq. (8.6). Therefore, to find that policy, we can apply general techniques for solving a game, namely, to let a no-regret learning algorithm play the game repeatedly against a best-response player. When played in this way, it can be shown that the averages of their plays converge to the solution of the game (details in §8.3.1).

In our case, we can use a no-regret algorithm for the  $\lambda$ -player, and best response for the  $\mu$ -player. Importantly, in our context, computing best response turns out to be an especially convenient task. Given  $\lambda$ , best response means finding the mixed policy  $\mu$  minimizing  $\lambda \cdot \mathbb{Z}(\mu)$ . As we show below, this can be solved by treating the problem as a standard reinforcement learning task where in each step  $i$ , the agent accrues a scalar reward  $r_i = -\lambda \cdot \mathbf{z}_i$ . We refer to any algorithm for solving the problem of scalar reward maximization as the *best-response oracle*. During the run of our algorithm, we invoke this oracle for different vectors  $\lambda$  corresponding to different definitions of a scalar reward. Although the oracle is only capable of solving RL tasks with a scalar reward, our algorithm can leverage this capability to solve the multi-dimensional feasibility (or distance minimization) problem.

In the remainder of this section, we provide the details of our approach, leading to our main algorithm and its analysis, and conclude with a discussion of steps for making a practical

implementation. We begin by discussing game-playing techniques in general, which we then apply to our setting.

### 8.3.1 Solving zero-sum games using online learning

At the core of our approach, we use the general technique of Freund and Schapire [94] for solving a game by repeatedly playing a no-regret online learning algorithm against best response.

For this purpose, we first briefly review the framework of online convex optimization, which we will soon use for one of the players: At time  $t = 1, \dots, T$ , the learner makes a decision  $\lambda_t \in \Lambda$ , the environment reveals a convex loss function  $\ell_t : \Lambda \rightarrow \mathbb{R}$ , and the learner incurs loss  $\ell_t(\lambda_t)$ . The learner seeks to achieve small *regret*, the gap between its loss and the best in hindsight:

$$\text{Regret}_T \triangleq \left[ \sum_{t=1}^T \ell_t(\lambda_t) \right] - \min_{\lambda \in \Lambda} \left[ \sum_{t=1}^T \ell_t(\lambda) \right]. \quad (8.8)$$

An online learning algorithm is *no-regret* if  $\text{Regret}_T = o(T)$ , meaning its average loss approaches the best in hindsight. An example of such an algorithm is *online gradient descent (OGD)* of Zinkevich [328] (see section 8.6.1). If the Euclidean diameter of  $\Lambda$  is at most  $D$ , and  $\|\nabla \ell_t(\lambda)\| \leq G$  for any  $t$  and  $\lambda \in \Lambda$ , then the regret of OGD is at most  $DG\sqrt{T}$ .

Now consider a two-player zero-sum game in which two players select, respectively,  $\lambda \in \Lambda$  and  $\mathbf{u} \in \mathcal{U}$ , resulting in a payout of  $g(\lambda, \mathbf{u})$  from the  $\mathbf{u}$ -player to the  $\lambda$ -player. The  $\lambda$ -player wants to maximize this quantity and the  $\mathbf{u}$ -player wants to minimize it. Assuming  $g$  is concave in  $\lambda$  and convex in  $\mathbf{u}$ , if both spaces  $\Lambda$  and  $\mathcal{U}$  are convex and compact, then the minimax theorem [199, 264] implies that

$$\max_{\lambda \in \Lambda} \min_{\mathbf{u} \in \mathcal{U}} g(\lambda, \mathbf{u}) = \min_{\mathbf{u} \in \mathcal{U}} \max_{\lambda \in \Lambda} g(\lambda, \mathbf{u}). \quad (8.9)$$

This means that the  $\lambda$ -player has an “optimal” strategy which realizes the maximum on the left and guarantees payoff of at least the *value* of the game, i.e., the value given by this expression; a similar statement holds for the  $u$ -player.

We can solve this game (find these optimal strategies) by playing it repeatedly. We use a no-regret online learner as the  $\lambda$ -player. At each time  $t = 1, \dots, T$ , the learner chooses  $\lambda_t \in \Lambda$ . In response, the  $u$ -player, who in this setting is permitted knowledge of  $\lambda_t$ , selects  $u_t$  to minimize the payout, that is,  $u_t = \operatorname{argmin}_{u \in \mathcal{U}} g(\lambda_t, u)$ . This is called *best response*. The online learning algorithm is then updated by setting its loss function to be  $\ell_t(\lambda) = -g(\lambda, u_t)$ . (See Algorithm 6.) As stated in Theorem 8.3.1,  $\bar{\lambda}$  and  $\bar{u}$ , the averages of the players’ decisions, converge to the solution of the game (see section 8.6.2 for the proof).

---

**Algorithm 6** Solving a game with repeated play

---

```

1: input concave-convex function  $g : \Lambda \times \mathcal{U} \rightarrow \mathbb{R}$ , online learning algorithm LEARNER
2: for  $t = 1$  to  $T$  do
3:   LEARNER makes a decision  $\lambda_t \in \Lambda$ 
4:    $u_t \leftarrow \operatorname{argmin}_{u \in \mathcal{U}} g(\lambda_t, u)$ 
5:   LEARNER observes loss function  $\ell_t(\lambda) = -g(\lambda, u_t)$ 
6: end for
7: return  $\bar{\lambda} = \frac{1}{T} \sum_{t=1}^T \lambda_t$  and  $\bar{u} = \frac{1}{T} \sum_{t=1}^T u_t$ 

```

---

**Theorem 8.3.1.** *Let  $v$  be the value of the game in Eq. (8.9) and let  $\operatorname{Regret}_T$  be the regret of the  $\lambda$ -player. Then for  $\bar{\lambda}$  and  $\bar{u}$  we have*

$$\min_{u \in \mathcal{U}} g(\bar{\lambda}, u) \geq v - \delta \quad \text{and} \quad \max_{\lambda \in \Lambda} g(\lambda, \bar{u}) \leq +\delta, \quad \text{where } \delta = \frac{1}{T} \operatorname{Regret}_T. \quad (8.10)$$

### 8.3.2 Algorithm and main result

We can now apply this game-playing framework to the approach outlined at the beginning of this section. First, we show how to write distance as a maximization, as in Eq. (8.5). For

now, we assume that our target set  $\mathcal{C}$  is a *convex cone*, that is, closed under summation and also multiplication by non-negative scalars (we will remove this assumption in §8.3.3). With this assumption, we can apply the following lemma (Lemma 13 of 2), in which distance to a convex cone  $\mathcal{C} \subseteq \mathbb{R}^d$  is written as a maximization over a dual convex cone  $\mathcal{C}^\circ$  called the *polar cone*:

$$\mathcal{C}^\circ \triangleq \{\boldsymbol{\lambda} : \boldsymbol{\lambda} \cdot \mathbf{x} \leq 0 \text{ for all } \mathbf{x} \in \mathcal{C}\}. \quad (8.11)$$

**Lemma 5.** *For a convex cone  $\mathcal{C} \subseteq \mathbb{R}^d$  and any point  $\mathbf{x} \in \mathbb{R}^d$*

$$\text{dist}(\mathbf{x}, \mathcal{C}) = \max_{\boldsymbol{\lambda} \in \mathcal{C}^\circ \cap \mathcal{B}} \boldsymbol{\lambda} \cdot \mathbf{x}, \quad (8.12)$$

where  $\mathcal{B} \triangleq \{\mathbf{x} : \|\mathbf{x}\| \leq 1\}$  is the Euclidean ball of radius 1 at the origin.

Thus, Eq. (8.5) is immediately achieved by setting  $\Lambda = \mathcal{C}^\circ \cap \mathcal{B}$ , so the distance minimization problem (8.4) can be cast as the min-max problem (8.6). This is a special case of the zero-sum game (8.9), with  $\mathcal{U} = \{\mathbb{Z}(\mu) : \mu \in \Delta(\Pi)\}$  and  $g(\boldsymbol{\lambda}, \mathbf{u}) = \boldsymbol{\lambda} \cdot \mathbf{u}$ , which can be solved with Algorithm 6. Note that the set  $\mathcal{U}$  is convex and compact, because it is a linear transformation of a convex and compact set  $\Delta(\Pi)$ .

We will see below that the best responses  $\mathbf{u}_t$  in Algorithm 6 can be expressed as  $\mathbb{Z}(\pi_t)$  for some  $\pi_t \in \Pi$ , and so Algorithm 6 returns

$$\bar{\mathbf{u}} = \frac{1}{T} \sum_{t=1}^T \mathbb{Z}(\pi_t) = \mathbb{Z}\left(\frac{1}{T} \sum_{t=1}^T \pi_t\right),$$

which is exactly the long-term measurement vector of the mixed policy  $\bar{\mu} = \frac{1}{T} \sum_{t=1}^T \pi_t$ . For this

mixed policy, Theorem 8.3.1 immediately implies

$$\text{dist}(\mathbb{Z}(\bar{\mu}), \mathcal{C}) \leq \min_{\mu \in \Delta(\Pi)} \text{dist}(\mathbb{Z}(\mu), \mathcal{C}) + \frac{1}{T} \text{Regret}_T. \quad (8.13)$$

If the problem is feasible, then  $\min_{\mu \in \Delta(\Pi)} \text{dist}(\mathbb{Z}(\mu), \mathcal{C}) = 0$ , and since  $\text{Regret}_T = o(T)$ , our long-term measurement  $\mathbb{Z}(\bar{\mu})$  converges to the target set and solves the feasibility problem (8.3).

It remains to specify how to implement the no-regret learner for the  $\lambda$ -player and best response for the  $u$ -player. We discuss these next, beginning with the latter.

The best-response player, for a given  $\lambda$ , aims to minimize  $\lambda \cdot \mathbb{Z}(\mu)$  over mixed policies  $\mu$ , but since this objective is linear in the mixture weights  $\mu(\pi)$  (see Eq. 8.2), it suffices to minimize  $\lambda \cdot \mathbb{Z}(\pi)$  over stationary policies  $\pi \in \Pi$ . The key point, as already mentioned, is that this is the same as finding a policy that maximizes long-term reward in a standard reinforcement learning task if we define the scalar reward to be  $r_i = -\lambda \cdot \mathbf{z}_i$ . This is because the reward of a policy  $\pi$  is given by

$$R(\pi) \triangleq \mathbb{E} \left[ \sum_{i=0}^{\infty} \gamma^i r_i \mid \pi \right] = \mathbb{E} \left[ \sum_{i=0}^{\infty} \gamma^i (-\lambda \cdot \mathbf{z}_i) \mid \pi \right] = -\lambda \cdot \mathbb{E} \left[ \sum_{i=0}^{\infty} \gamma^i \mathbf{z}_i \mid \pi \right] = -\lambda \cdot \mathbb{Z}(\pi). \quad (8.14)$$

Therefore, maximizing  $R(\pi)$ , as in standard RL, is equivalent to minimizing  $\lambda \cdot \mathbb{Z}(\pi)$ .

Thus, best response can be implemented using any one of the many well-studied RL algorithms that maximize a scalar reward. We refer to such an RL algorithm as the *best-response oracle*. For robustness, we allow this oracle to return an approximately optimal policy.

Best-response oracle:  $\text{BESTRESPONSE}(\lambda)$ .

Given  $\lambda \in \mathbb{R}^d$ , return a policy  $\pi \in \Pi$  that satisfies  $R(\pi) \geq \max_{\pi' \in \Pi} R(\pi') - \epsilon_0$ , where

$R(\pi)$  is the long-term reward of policy  $\pi$  with scalar reward defined as  $r = -\lambda \cdot \mathbf{z}$ .



For the  $\lambda$ -player, we do our analysis using online gradient descent [328], an effective no-regret learner. For its update, OGD needs the gradient of the loss functions  $\ell_t(\lambda) = -\lambda \cdot \mathbb{Z}(\pi_t)$ , which is just  $-\mathbb{Z}(\pi_t)$ . With access to the MDP,  $\mathbb{Z}(\pi)$  can be estimated simply by generating multiple trajectories using  $\pi$  and averaging the observed measurements. We formalize this by assuming access to an *estimation oracle* for estimating  $\mathbb{Z}(\pi)$ .

Estimation oracle:  $\text{EST}(\pi)$ .

Given policy  $\pi$ , return  $\hat{\mathbf{z}}$  satisfying  $\|\hat{\mathbf{z}} - \mathbb{Z}(\pi)\| \leq \epsilon_1$ .

OGD also requires projection to the set  $\Lambda = \mathcal{C}^\circ \cap \mathcal{B}$ . In fact, if we can simply project onto the target set  $\mathcal{C}$ , which is more natural, then it is possible to also project onto  $\Lambda$ . Consider an arbitrary  $\mathbf{x}$  and denote its projection onto  $\mathcal{C}$  as  $\Gamma_{\mathcal{C}}(\mathbf{x})$ . Then the projection of  $\mathbf{x}$  onto the polar cone  $\mathcal{C}^\circ$  is  $\Gamma_{\mathcal{C}^\circ}(\mathbf{x}) = \mathbf{x} - \Gamma_{\mathcal{C}}(\mathbf{x})$  [132]. Given the projection  $\Gamma_{\mathcal{C}^\circ}(\mathbf{x})$  and further projecting onto  $\mathcal{B}$ , we obtain  $\Gamma_{\Lambda}(\mathbf{x}) = (\mathbf{x} - \Gamma_{\mathcal{C}}(\mathbf{x})) / \max\{1, \|\mathbf{x} - \Gamma_{\mathcal{C}}(\mathbf{x})\|\}$  (because Dykstra’s projection algorithm converges to this point after two steps, 40). Therefore, it suffices to require access to a *projection oracle* for  $\mathcal{C}$ :

Projection oracle:  $\Gamma_{\mathcal{C}}(\mathbf{x}) = \operatorname{argmin}_{\mathbf{x}' \in \mathcal{C}} \|\mathbf{x} - \mathbf{x}'\|$ .

Pulling these ideas together and plugging into Algorithm 6, we obtain our main algorithm, called APPROPO (Algorithm 7), for *approachability-based policy optimization*. The algorithm provably yields a mixed policy that approximately minimizes distance to the set  $\mathcal{C}$ , as shown in Theorem 8.3.2 (proved in section 8.6.3).

**Theorem 8.3.2.** *Assume that  $\mathcal{C}$  is a convex cone and for all measurements we have  $\|\mathbf{z}\| \leq B$ .*

*Suppose we run Algorithm 7 for  $T$  rounds with  $\eta = \left(\frac{B}{1-\gamma} + \epsilon_1\right)^{-1} T^{-1/2}$ . Then*

$$\operatorname{dist}(\mathbb{Z}(\bar{\mu}), \mathcal{C}) \leq \min_{\mu \in \Delta(\Pi)} \operatorname{dist}(\mathbb{Z}(\mu), \mathcal{C}) + \left(\frac{B}{1-\gamma} + \epsilon_1\right) T^{-1/2} + \epsilon_0 + 2\epsilon_1, \quad (8.15)$$

---

**Algorithm 7** APPROPO

---

```
1: input projection oracle  $\Gamma_{\mathcal{C}}(\cdot)$  for target set  $\mathcal{C}$  which is a convex cone,
    best-response oracle  $\text{BESTRESPONSE}(\cdot)$ , estimation oracle  $\text{EST}(\cdot)$ ,
    step size  $\eta$ , number of iterations  $T$ 
2: define  $\Lambda \triangleq \mathcal{C}^\circ \cap \mathcal{B}$ , and its projection operator  $\Gamma_\Lambda(\mathbf{x}) \triangleq$ 
     $(\mathbf{x} - \Gamma_{\mathcal{C}}(\mathbf{x})) / \max\{1, \|\mathbf{x} - \Gamma_{\mathcal{C}}(\mathbf{x})\|\}$ 
3: initialize  $\boldsymbol{\lambda}_1$  arbitrarily in  $\Lambda$ 
4: for  $t = 1$  to  $T$  do
5:   Compute an approximately optimal policy for standard RL with scalar reward  $r = -\boldsymbol{\lambda}_t \cdot \mathbf{z}$ :
      $\pi_t \leftarrow \text{BESTRESPONSE}(\boldsymbol{\lambda}_t)$ 
6:   Call the estimation oracle to approximate long-term measurement for  $\pi_t$ :
      $\hat{\mathbf{z}}_t \leftarrow \text{EST}(\pi_t)$ 
7:   Update  $\boldsymbol{\lambda}_t$  using online gradient descent with the loss function  $\ell_t(\boldsymbol{\lambda}) = -\boldsymbol{\lambda} \cdot \hat{\mathbf{z}}_t$ :
      $\boldsymbol{\lambda}_{t+1} \leftarrow \Gamma_\Lambda(\boldsymbol{\lambda}_t + \eta \hat{\mathbf{z}}_t)$ 
8: end for
9: return  $\bar{\mu}$ , a uniform mixture over  $\pi_1, \dots, \pi_T$ 
```

---

where  $\bar{\mu}$  is the mixed policy returned by the algorithm.

When the goal is to solve the feasibility problem (8.3) rather than the stronger distance minimization (8.4), we can make use of a weaker reinforcement learning oracle, which only needs to find a policy that is “good enough” in the sense of providing long-term reward above some threshold:

Positive-response oracle:  $\text{POSRESPONSE}(\boldsymbol{\lambda})$ .

Given  $\boldsymbol{\lambda} \in \mathbb{R}^d$ , return  $\pi \in \Pi$  that satisfies  $R(\pi) \geq -\epsilon_0$  if  $\max_{\pi' \in \Pi} R(\pi') \geq 0$  (and arbitrary  $\pi$  otherwise), where  $R(\pi)$  is the long-term reward of  $\pi$  with scalar reward  $r = -\boldsymbol{\lambda} \cdot \mathbf{z}$ .

When the problem is feasible, it can be shown that there must exist  $\pi \in \Pi$  with  $R(\pi) \geq 0$ , and furthermore, that  $\ell_t(\boldsymbol{\lambda}_t) \geq -(\epsilon_0 + \epsilon_1)$  (from Lemma 7 in section 8.6.3). This means, if the goal is feasibility, we can modify Algorithm 7, replacing  $\text{BESTRESPONSE}$  with  $\text{POSRESPONSE}$ , and adding a test at the end of each iteration to report infeasibility if  $\ell_t(\boldsymbol{\lambda}_t) < -(\epsilon_0 + \epsilon_1)$ . The pseudocode is provided in Algorithm 9 in section 8.6.4 along with the proof of the following

convergence bound:

**Theorem 8.3.3.** *Assume that  $\mathcal{C}$  is a convex cone and for all measurements we have  $\|\mathbf{z}\| \leq B$ . Suppose we run Algorithm 9 for  $T$  rounds with  $\eta = \left(\frac{B}{1-\gamma} + \epsilon_1\right)^{-1} T^{-1/2}$ . Then either the algorithm reports infeasibility or returns  $\bar{\mu}$  such that*

$$\text{dist}(\mathbb{Z}(\bar{\mu}), \mathcal{C}) \leq \left(\frac{B}{1-\gamma} + \epsilon_1\right) T^{-1/2} + \epsilon_0 + 2\epsilon_1. \quad (8.16)$$

### 8.3.3 Removing the cone assumption

Our results so far have assumed the target set  $\mathcal{C}$  is a convex cone. If instead  $\mathcal{C}$  is an arbitrary convex, compact set, we can use the technique of Abernethy et al. [2] and apply our algorithm to a specific convex cone  $\tilde{\mathcal{C}}$  constructed from  $\mathcal{C}$  to obtain a solution with provable guarantees.

In more detail, given a compact, convex target set  $\mathcal{C} \subseteq \mathbb{R}^d$ , we augment every vector in  $\mathcal{C}$  with a new coordinate held fixed to some value  $\kappa > 0$ , and then let  $\tilde{\mathcal{C}}$  be its conic hull. Thus,

$$\tilde{\mathcal{C}} = \text{cone}(\mathcal{C} \times \{\kappa\}), \quad \text{where } \text{cone}(\mathcal{X}) = \{\alpha \mathbf{x} \mid \mathbf{x} \in \mathcal{X}, \alpha \geq 0\}. \quad (8.17)$$

Given our original vector-valued MDP  $M = (\mathcal{S}, \mathcal{A}, \beta, P_s, P_z)$ , we define a new MDP  $M' = (\mathcal{S}, \mathcal{A}, \beta, P_s, P'_{z'})$  with  $(d+1)$ -dimensional measurement  $\mathbf{z}' \in \mathbb{R}^{d+1}$ , defined (and generated) by

$$\mathbf{z}'_i = \mathbf{z}_i \oplus \langle (1-\gamma)\kappa \rangle, \quad \mathbf{z}_i \sim P_z(\cdot \mid s_i, a_i) \quad (8.18)$$

where  $\oplus$  denotes vector concatenation. Writing long-term measurement for  $M$  and  $M'$  as  $\mathbb{Z}$  and  $\mathbb{Z}'$  respectively,  $\mathbb{Z}'(\pi) = \mathbb{Z}(\pi) \oplus \langle \kappa \rangle$ , for any policy  $\pi \in \Pi$ , and similarly for any mixed policy  $\mu$ .

The main idea is to apply the algorithms described above to the modified MDP  $M'$  using

the cone  $\tilde{\mathcal{C}}$  as target set. For an appropriate choice of  $\kappa > 0$ , we show that the resulting mixed policy will approximately minimize distance to  $\mathcal{C}$  for the original MDP  $M$ . This is a consequence of the following lemma, an extension of Lemma 14 of Abernethy et al. [2], which shows that distances are largely preserved in a controllable way under this construction. The proof is in section 8.6.5.

**Lemma 6.** *Consider a compact, convex set  $\mathcal{C}$  in  $\mathbb{R}^d$  and  $\mathbf{x} \in \mathbb{R}^d$ . For any  $\delta > 0$ , let  $\tilde{\mathcal{C}} = \text{cone}(\mathcal{C} \times \{\kappa\})$ , where  $\kappa = \frac{\max_{\mathbf{x} \in \mathcal{C}} \|\mathbf{x}\|}{\sqrt{2\delta}}$ . Then  $\text{dist}(\mathbf{x}, \mathcal{C}) \leq (1 + \delta)\text{dist}(\mathbf{x} \oplus \langle \kappa \rangle, \tilde{\mathcal{C}})$ .*

**Corollary 8.3.4.** *Assume that  $\mathcal{C}$  is a convex, compact set and for all measurements we have  $\|\mathbf{z}\| \leq B$ . Then by putting  $\eta = \left(\frac{B+\kappa}{1-\gamma} + \epsilon_1\right)^{-1} T^{-1/2}$  and running Algorithm 7 for  $T$  rounds with  $M'$  as the MDP and  $\tilde{\mathcal{C}}$  as the target set, the mixed policy  $\bar{\mu}$  returned by the algorithm satisfies*

$$\text{dist}(\mathbb{Z}(\bar{\mu}), \mathcal{C}) \leq (1 + \delta) \left( \min_{\mu \in \Delta(\Pi)} \text{dist}(\mathbb{Z}(\mu), \mathcal{C}) + \left(\frac{B+\kappa}{1-\gamma} + \epsilon_1\right) T^{-1/2} + \epsilon_0 + 2\epsilon_1 \right), \quad (8.19)$$

where  $\kappa = \frac{\max_{\mathbf{x} \in \mathcal{C}} \|\mathbf{x}\|}{\sqrt{2\delta}}$  for an arbitrary  $\delta > 0$ . Similarly for Algorithm 9, we either have

$$\text{dist}(\mathbb{Z}(\bar{\mu}), \mathcal{C}) \leq (1 + \delta) \left( \left(\frac{B+\kappa}{1-\gamma} + \epsilon_1\right) T^{-1/2} + \epsilon_0 + 2\epsilon_1 \right) \quad (8.20)$$

or the algorithm reports infeasibility.

### 8.3.4 Practical implementation of the positive response and estimation oracles

We next briefly describe a few techniques for the practical implementation of our algorithm.

As discussed in §8.3.2, when our aim is to solve a feasibility problem, we only need access to a positive response oracle. In episodic environments, it is straightforward to use any standard

iterative RL approach as a positive response oracle: As the RL algorithm runs, we track its accrued rewards, and when the trailing average of the last  $n$  trajectory-level rewards goes above some level  $-\epsilon$ , we return the current policy (possibly specified implicitly as a  $Q$ -function).<sup>2</sup> Furthermore, the average of the measurement vectors  $\mathbf{z}$  collected over the last  $n$  trajectories can serve as the estimate  $\hat{\mathbf{z}}_t$  of the long-term measurement required by the algorithm, side-stepping the need for an additional estimation oracle.

The hyperparameters  $\epsilon$  and  $n$  influence the oracle quality; specifically, assuming that the rewards are bounded and the overall number of trajectories until the oracle terminates is at most polynomial in  $n$ , we have  $\epsilon_0 = \epsilon - O(\sqrt{(\log n)/n})$  and  $\epsilon_1 = O(\sqrt{(\log n)/n})$ . In principle, we could use [Theorem 8.3.3](#) to select a value  $T$  at which to stop; in practice, we run until the running average of the measurements  $\hat{\mathbf{z}}_t$  gets within a small distance of the target set  $\mathcal{C}$ . If the RL algorithm runs for too long without achieving non-negative rewards, we stop and declare that the underlying problem is “empirically infeasible.” (Actual infeasibility would hold if it is truly not possible to reach non-negative expected reward.)

An important mechanism to further speed up our algorithm is to maintain a “cache” of all the policies returned by the positive response oracle so far. Each of the cached policies  $\pi$  is stored with the estimate of its expected measurement vector  $\hat{\mathbf{z}}(\pi) \approx \bar{\mathbf{z}}(\pi)$ , based on its last  $n$  iterations (as above). In each outer-loop iteration of our algorithm, we first check if the cache contains a policy that already achieves a reward at least  $-\epsilon$  under the new  $\boldsymbol{\lambda}$ ; this can be determined from the cached  $\hat{\mathbf{z}}(\pi)$  since the reward is just a linear function of the measurement vector. If such a policy is found, we return it, alongside  $\hat{\mathbf{z}}(\pi)$ , instead of calling the oracle. Otherwise, we pick the policy from the cache with the largest reward (below  $-\epsilon$  by assumption) and use it to

---

<sup>2</sup>This assumes that the last  $n$  trajectories accurately estimate the performance of the final iterate. If that is not the case, the oracle can instead return the mixture of the policies corresponding to the last  $n$  iterates.

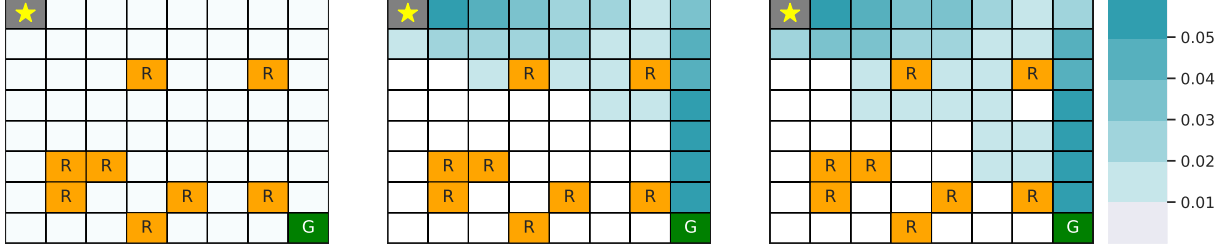


Figure 8.1: *Left*: The Mars rover environment. The agent starts in top-left and needs to reach the goal in bottom-right while avoiding rocks. *Middle, Right*: Visitation probabilities of APPROPO (middle) and APPROPO with a diversity constraints (right) at 12k samples. Both plots based on a single run.

warm-start the RL algorithm implementing the oracle. The cache can be initialized with a few random policies (as we do in our experiments), effectively implementing randomized weight initialization.

The cache interacts well with a straightforward binary-search scheme that can be used when the goal is to maximize some reward (possibly subject to additional constraints), rather than only satisfy a set of constraints. The feasibility problems corresponding to iterates of binary search only differ in the constraint values, but use the same measurements, so the same cache can be reused across all iterations.

**Running time.** Note that APPROPO spends the bulk of its running time executing the best-response oracle. It additionally performs updates of  $\lambda$ , but these tend to be orders of magnitude cheaper than any per-episode (or per-transition) updates within the oracle. For example, in our experiments, the dimension of  $\lambda$  is either 2 or 66 (without or with the diversity constraint, respectively), whereas the policies  $\pi$  trained by the oracle are two-layer networks described by 8,704 floating-point numbers.

## 8.4 Experiments

We next evaluate the performance of APPROPO and demonstrate its ability to handle a variety of constraints. For simplicity, we focus on the feasibility version (Algorithm 9 in section 8.6.4). We compare APPROPO with the RCPO approach of Tessler et al. [280], which adapts policy gradient, specifically, asynchronous actor-critic (A2C) [191], to find a fixed point of the Lagrangian of the constrained policy optimization problem. RCPO maintains and updates a vector of Lagrange multipliers, which is then used to derive a reward for A2C. The vector of Lagrange multipliers serves a similar role as our  $\lambda$ , and the overall structure of RCPO is similar to APPROPO, so RCPO is a natural baseline for a comparison. Unlike APPROPO, RCPO only allows orthant constraints and it seeks to maximize reward, whereas APPROPO solves the feasibility problem.

For a fair comparison, APPROPO uses A2C as a positive-response oracle, with the same hyperparameters as used in RCPO. Online learning in the outer loop of APPROPO was implemented via online gradient descent with momentum. Both RCPO and APPROPO have an outer-loop learning rate parameter, which we tuned over a grid of values  $10^{-i}$  with integer  $i$  (see section 8.6.6 for the details). Here, we report the results with the best learning rate for each method.

We ran our experiments on a small version of the *Mars rover* grid-world environment, used previously for the evaluation of RCPO [280]. In this environment, depicted in Figure 8.1 (left), the agent must move from the starting position to the goal without crashing into rocks. The episode terminates when a rock or the goal is reached, or after 300 steps. The environment is stochastic: with probability  $\delta = 0.05$  the agent’s action is perturbed to a random action. The

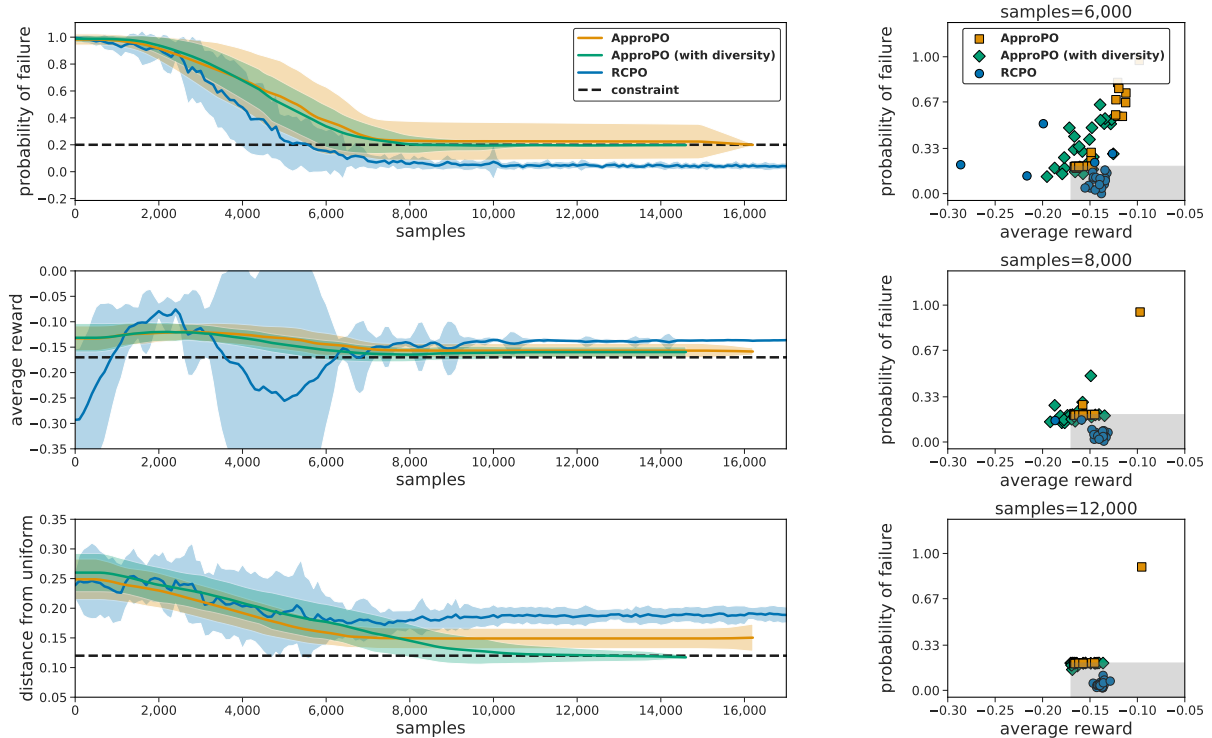


Figure 8.2: *Left*: The performance of the algorithms as a function of the number of samples (steps in the environment); showing average and standard deviation over 25 runs. The vertical axes correspond to the three constraints, with thresholds shown as a dashed line; for reward (middle) this is a lower bound; for the others it is an upper bound. *Right*: Each point in the scatter plot represents the reward and the probability of failure obtained by the policy learnt by the algorithm at the specified number of samples. The grey region is the target set. Different points represent different random runs.



agent receives small negative reward each time step and zero for terminating, with  $\gamma = 0.99$ . We used the same safety constraint as Tessler et al. [280]: ensure that the (discounted) probability of hitting a rock is at most a fixed threshold (set to 0.2). RCPO seeks to maximize reward subject to this constraint. APPROPO solves a feasibility problem with the same safety constraint, and an additional constraint requiring that the reward be at least  $-0.17$  (this is slightly lower than the final reward achieved by RCPO). We also experimented with including the exploration suggestion as a “diversity constraint,” requiring that the Euclidean distance between our visitation probability vector (across the cells of the grid) and the uniform distribution over the upper-right triangle cells of the grid (excluding rocks) be at most 0.12.<sup>3</sup>

In Figure 8.2 (left), we show how the probability of failure, the average reward, and the distance to the uniform distribution over upper triangle vary as a function of the number of samples seen by each algorithm. Both variants of our algorithm are able to satisfy the safety constraints and reach similar reward as RCPO with a similar number of samples (around 8k samples). Furthermore, including the diversity constraint, which RCPO is not capable of enforcing, allowed our method to reach a more diverse policy as depicted in both Figure 8.2 (bottom-left) and Figure 8.1 (right).

## 8.5 Conclusion

In this paper, we introduced APPROPO, an algorithm for solving reinforcement learning problems with arbitrary convex constraints. APPROPO can combine any no-regret online learner with any standard RL algorithm that optimizes a scalar reward. Theoretically, we showed that for the specific case of online gradient descent, APPROPO learns to approach the constraint

---

<sup>3</sup>This number ensures that APPROPO without the diversity constraint does not satisfy it automatically.

set at a rate of  $1/\sqrt{T}$ , with an additive non-vanishing term that measures the optimality gap of the reinforcement learner. Experimentally, we demonstrated that APPROPO can be applied with well-known RL algorithms for discrete domains (like actor-critic), and achieves similar performance as RCPO [280], while being able to satisfy additional types of constraints. In sum, this yields a theoretically justified, practical algorithm for solving the approachability problem in reinforcement learning.

## 8.6 Extended Details

### 8.6.1 Online gradient descent (OGD)

---

**Algorithm 8** Online gradient descent (OGD)

---

```

1: input: projection oracle  $\Gamma_\Lambda$   $\{\Gamma_\Lambda(\boldsymbol{\lambda}) = \operatorname{argmin}_{\boldsymbol{\lambda}' \in \Lambda} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\|\}$ 
2: init:  $\boldsymbol{\lambda}_1$  arbitrarily
3: parameters: step size  $\eta_t$ 
4: for  $t = 1$  to  $T$  do
5:   observe convex loss function  $\ell_t : \Lambda \rightarrow \mathbb{R}$ 
6:    $\boldsymbol{\lambda}'_{t+1} = \boldsymbol{\lambda}_t - \eta_t \nabla \ell_t(\boldsymbol{\lambda}_t)$ 
7:    $\boldsymbol{\lambda}_{t+1} = \Gamma_\Lambda(\boldsymbol{\lambda}'_{t+1})$ 
8: end for

```

---

**Theorem 8.6.1.** [328] Assume that for any  $\boldsymbol{\lambda}, \boldsymbol{\lambda}' \in \Lambda$  we have  $\|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\| \leq D$  and also

$\|\nabla \ell_t(\boldsymbol{\lambda})\| \leq G$ . Let  $\eta_t = \eta = \frac{D}{G\sqrt{T}}$ . Then the regret of OGD is

$$\operatorname{Regret}_T(\text{OGD}) = \sum_{t=1}^T \ell_t(\boldsymbol{\lambda}_t) - \min_{\boldsymbol{\lambda}} \sum_{t=1}^T \ell_t(\boldsymbol{\lambda}) \leq DG\sqrt{T}.$$

### 8.6.2 Proof of Theorem 8.3.1

We have that

$$\frac{1}{T} \sum_{t=1}^T g(\boldsymbol{\lambda}_t, \mathbf{u}_t) = \frac{1}{T} \sum_{t=1}^T \min_{\mathbf{u} \in \mathcal{U}} g(\boldsymbol{\lambda}_t, \mathbf{u}) \tag{8.21}$$

$$\leq \frac{1}{T} \min_{\mathbf{u} \in \mathcal{U}} \sum_{t=1}^T g(\boldsymbol{\lambda}_t, \mathbf{u}) \tag{8.22}$$

$$\leq \min_{\mathbf{u} \in \mathcal{U}} g\left(\frac{1}{T} \sum_{t=1}^T \boldsymbol{\lambda}_t, \mathbf{u}\right) \tag{8.23}$$

$$\leq \max_{\boldsymbol{\lambda} \in \Lambda} \min_{\mathbf{u} \in \mathcal{U}} g(\boldsymbol{\lambda}, \mathbf{u}). \tag{8.24}$$

equation 8.21 is because the  $\mathbf{u}$ -player is playing best response so that  $\mathbf{u}_t = \operatorname{argmin}_{\mathbf{u} \in \mathcal{U}} g(\boldsymbol{\lambda}_t, \mathbf{u})$ . equation 8.22 is because taking the minimum of each term of a sum cannot exceed the minimum of the sum as a whole. Eqs. (8.23) and (8.24) use the concavity of  $g$  with respect to  $\boldsymbol{\lambda}$ , and the definition of  $\max$ , respectively. By letting  $\delta = \frac{1}{T} \operatorname{Regret}_T$ , writing the definition of regret for the  $\boldsymbol{\lambda}$ -player, and using  $\ell_t(\boldsymbol{\lambda}) = -g(\boldsymbol{\lambda}, \mathbf{u}_t)$ , we have

$$\frac{1}{T} \sum_{t=1}^T g(\boldsymbol{\lambda}_t, \mathbf{u}_t) + \delta = \frac{1}{T} \max_{\boldsymbol{\lambda} \in \Lambda} \sum_{t=1}^T g(\boldsymbol{\lambda}, \mathbf{u}_t) \geq \max_{\boldsymbol{\lambda} \in \Lambda} g\left(\boldsymbol{\lambda}, \frac{1}{T} \sum_{t=1}^T \mathbf{u}_t\right) \geq \min_{\mathbf{u} \in \mathcal{U}} \max_{\boldsymbol{\lambda} \in \Lambda} g(\boldsymbol{\lambda}, \mathbf{u}),$$

where the second and third inequalities use convexity of  $g$  with respect to  $\mathbf{u}$  and definition of  $\min$ , respectively. Combining yields

$$\min_{\mathbf{u} \in \mathcal{U}} g\left(\frac{1}{T} \sum_{t=1}^T \boldsymbol{\lambda}_t, \mathbf{u}\right) \geq \min_{\mathbf{u} \in \mathcal{U}} \max_{\boldsymbol{\lambda} \in \Lambda} g(\boldsymbol{\lambda}, \mathbf{u}) - \delta,$$

and also

$$\max_{\boldsymbol{\lambda} \in \Lambda} g\left(\boldsymbol{\lambda}, \frac{1}{T} \sum_{t=1}^T \mathbf{u}_t\right) \leq \max_{\boldsymbol{\lambda} \in \Lambda} \min_{\mathbf{u} \in \mathcal{U}} g(\boldsymbol{\lambda}, \mathbf{u}) + \delta,$$

completing the proof.

### 8.6.3 Proof of Theorem 8.3.2

Let  $v$  be the value of the game in equation 8.7:

$$v = \min_{\mu \in \Delta(\Pi)} \operatorname{dist}(\mathbb{Z}(\mu), \mathcal{C}), \quad (8.25)$$

and let  $\ell_t(\boldsymbol{\lambda}) = -\boldsymbol{\lambda} \cdot \hat{\mathbf{z}}_t$  (i.e., the loss function that OGD observes).

**Lemma 7.** For  $t = 1, 2, \dots, T$  we have

$$\ell_t(\boldsymbol{\lambda}_t) = -\boldsymbol{\lambda}_t \cdot \hat{\mathbf{z}}_t \geq -v - (\epsilon_0 + \epsilon_1).$$

*Proof.* By equation 8.5 (which must hold by Lemma 5), and by equation 8.25, there exists  $\mu^* \in \Delta(\Pi)$  such that

$$v = \text{dist}(\mathbb{Z}(\mu^*), \mathcal{C}) = \max_{\boldsymbol{\lambda} \in \Lambda} \boldsymbol{\lambda} \cdot \mathbb{Z}(\mu^*).$$

Thus,  $\boldsymbol{\lambda}_t \cdot \mathbb{Z}(\mu^*) \leq v$  since  $\boldsymbol{\lambda}_t \in \Lambda$  for all  $t$ . By our assumed guarantee for the policy  $\pi_t$  returned by the planning oracle, we have

$$-\boldsymbol{\lambda}_t \cdot \mathbb{Z}(\pi_t) \geq -\boldsymbol{\lambda}_t \cdot \mathbb{Z}(\mu^*) - \epsilon_0 \geq -v - \epsilon_0.$$

Now using the error bound of the estimation oracle,

$$\|\mathbb{Z}(\pi_t) - \hat{\mathbf{z}}_t\| \leq \epsilon_1, \tag{8.26}$$

and the fact that  $\|\boldsymbol{\lambda}_t\| \leq 1$ , we have

$$(-\boldsymbol{\lambda}_t \cdot \hat{\mathbf{z}}_t) + \epsilon_1 \geq -\boldsymbol{\lambda}_t \cdot \mathbb{Z}(\pi_t).$$

Combining completes the proof. □

Now we are ready to prove Theorem 8.3.2. Using the definition of mixed policy  $\bar{\mu}$  returned by the algorithm we have

$$\begin{aligned} \text{dist}(\mathbb{Z}(\bar{\mu}), \mathcal{C}) &= \text{dist}\left(\frac{1}{T} \sum_{t=1}^T \mathbb{Z}(\pi_t), \mathcal{C}\right) \\ &= \max_{\lambda \in \Lambda} \lambda \cdot \left(\frac{1}{T} \sum_{t=1}^T \mathbb{Z}(\pi_t)\right) \end{aligned} \quad (8.27)$$

$$\begin{aligned} &= \frac{1}{T} \max_{\lambda \in \Lambda} \sum_{t=1}^T \lambda \cdot \mathbb{Z}(\pi_t) \\ &\leq \frac{1}{T} \max_{\lambda \in \Lambda} \sum_{t=1}^T \lambda \cdot \hat{\mathbf{z}}_t + \epsilon_1 \end{aligned} \quad (8.28)$$

$$= -\frac{1}{T} \min_{\lambda \in \Lambda} \sum_{t=1}^T \ell_t(\lambda) + \epsilon_1 \quad (8.29)$$

$$\leq -\frac{1}{T} \min_{\lambda \in \Lambda} \sum_{t=1}^T \ell_t(\lambda) + \epsilon_1 + \frac{1}{T} \sum_{t=1}^T (\ell_t(\lambda_t) + \epsilon_1 + \epsilon_0 + v) \quad (8.30)$$

$$\begin{aligned} &= v + \left(-\frac{1}{T} \min_{\lambda \in \Lambda} \sum_{t=1}^T \ell_t(\lambda) + \frac{1}{T} \sum_{t=1}^T \ell_t(\lambda_t)\right) + 2\epsilon_1 + \epsilon_0 \\ &= v + \frac{\text{Regret}_T(\text{OGD})}{T} + 2\epsilon_1 + \epsilon_0. \end{aligned}$$

Here, equation 8.27 is by equation 8.5. equation 8.28 uses equation 8.26 and the fact that  $\|\lambda\| \leq 1$ . equation 8.31 uses Lemma 7.

The diameter of decision set  $\Lambda = \mathcal{C}^\circ \cap \mathcal{B}$  is at most 1. The gradient of the loss function  $\nabla(\ell_t(\lambda)) = -\hat{\mathbf{z}}_t$  has norm at most  $\|\mathbb{Z}(\pi_t)\| + \epsilon_1 \leq \frac{B}{1-\gamma} + \epsilon_1$ . Therefore, setting  $\eta = \left(\left(\frac{B}{1-\gamma} + \epsilon_1\right)\sqrt{T}\right)^{-1}$  based on Theorem 8.6.1, we get

$$\frac{\text{Regret}_T(\text{OGD})}{T} \leq \left(\frac{B}{1-\gamma} + \epsilon_1\right) T^{-1/2}$$

### 8.6.4 APPROPO for feasibility

---

**Algorithm 9** APPROPO – Feasibility

---

```

1: input projection oracle  $\Gamma_{\mathcal{C}}(\cdot)$  for target set  $\mathcal{C}$  which is a convex cone,
   positive response oracle  $\text{PosPlan}(\cdot)$ , estimation oracle  $\text{Est}(\cdot)$ ,
   step size  $\eta$ , number of iterations  $T$ 
2: define  $\Lambda \triangleq \mathcal{C}^\circ \cap \mathcal{B}$ , and its projection operator  $\Gamma_\Lambda(\mathbf{x}) \triangleq$ 
    $(\mathbf{x} - \Gamma_{\mathcal{C}}(\mathbf{x})) / \max\{1, \|\mathbf{x} - \Gamma_{\mathcal{C}}(\mathbf{x})\|\}$ 
3: initialize  $\boldsymbol{\lambda}_1$  arbitrarily in  $\Lambda$ 
4: for  $t = 1$  to  $T$  do
5:   Call positive response oracle for the standard RL with scalar reward  $r = -\boldsymbol{\lambda}_t \cdot \mathbf{z}$ :
      $\pi_t \leftarrow \text{PosPlan}(\boldsymbol{\lambda}_t)$ 
6:   Call the estimation oracle to approximate long-term measurement for  $\pi_t$ :
      $\hat{\mathbf{z}}_t \leftarrow \text{Est}(\pi_t)$ 
7:   Update using online gradient descent with the loss function  $\ell_t(\boldsymbol{\lambda}) = -\boldsymbol{\lambda} \cdot \hat{\mathbf{z}}_t$ :
      $\boldsymbol{\lambda}_{t+1} \leftarrow \Gamma_\Lambda(\boldsymbol{\lambda}_t + \eta \hat{\mathbf{z}}_t)$ 
8:   if  $\ell_t(\boldsymbol{\lambda}_t) < -(\epsilon_0 + \epsilon_1)$  then
9:     return problem is infeasible
10:  end if
11: end for
12: return  $\bar{\mu}$ , a uniform mixture over  $\pi_1, \dots, \pi_T$ 

```

---

#### 8.6.4.1 Proof of Theorem 8.3.3

**Lemma 8.** *If the problem is feasible, then for  $t = 1, 2, \dots, T$  we have*

$$\ell_t(\boldsymbol{\lambda}_t) = -\boldsymbol{\lambda}_t \cdot \hat{\mathbf{z}}_t \geq -(\epsilon_0 + \epsilon_1).$$

*Proof.* If the problem is feasible, then there exists  $\mu^*$  such that  $\mathbb{Z}(\mu^*) \in \mathcal{C}$ . Since all  $\boldsymbol{\lambda}_t \in \mathcal{C}^\circ$ , they all have non-positive inner product with every point in  $\mathcal{C}$  including  $\mathbb{Z}(\mu^*)$ . Since  $-\boldsymbol{\lambda}_t \cdot \mathbb{Z}(\mu^*) \geq 0$ , we can conclude that  $\max_{\pi \in \Pi} R(\pi) = \max_{\pi \in \Pi} -\boldsymbol{\lambda}_t \cdot \mathbb{Z}(\pi) \geq 0$ . Therefore, by our guarantee for the positive response oracle,

$$R(\pi_t) = -\boldsymbol{\lambda}_t \cdot \mathbb{Z}(\pi) \geq -\epsilon_0.$$

Now using equation 8.26 and the fact that  $\|\boldsymbol{\lambda}_t\| \leq 1$ , we have

$$(-\boldsymbol{\lambda}_t \cdot \hat{\mathbf{z}}_t) + \epsilon_1 \geq -\boldsymbol{\lambda}_t \cdot \mathbb{Z}(\pi_t).$$

Combining completes the proof.  $\square$  The proof of Theorem 8.3.3 is similar to that of Theorem 8.3.2. If the algorithm reports infeasibility then the problem is infeasible as a result of Lemma 8. Otherwise, we have

$$\frac{1}{T} \sum_{t=1}^T (\ell_t(\boldsymbol{\lambda}_t) + \epsilon_1 + \epsilon_0) \geq 0,$$

which can be combined with equation 8.29 as before. Continuing this argument as before yields

$$\text{dist}(\mathbb{Z}(\mu), \mathcal{C}) \leq \left( \frac{B}{1-\gamma} + \epsilon_1 \right) T^{-1/2} + 2\epsilon_1 + \epsilon_0,$$

completing the proof.

### 8.6.5 Proof of Lemma 6

Let  $\mathcal{C}' = \mathcal{C} \times \{\kappa\}$  and  $\mathbf{q}$  be the projection of  $\tilde{\mathbf{x}} = \mathbf{x} \oplus \langle \kappa \rangle$  onto  $\tilde{\mathcal{C}} = \text{cone}(\mathcal{C}')$ , i.e.,

$$\mathbf{q} = \arg \min_{\mathbf{y} \in \tilde{\mathcal{C}}} \|\tilde{\mathbf{x}} - \mathbf{y}\|.$$

Let  $r$  be the last coordinate of  $\mathbf{q}$ . We prove the lemma in cases based on the value of  $r$  (which cannot be negative by construction).



Case 1 ( $r > \kappa$ ): Since  $\mathbf{q} \in \text{cone}(\mathcal{C}')$  with  $r > 0$ , there exists  $\alpha > 0$  and  $\mathbf{q}' \in \mathcal{C}'$  so that  $\mathbf{q} = \alpha\mathbf{q}'$ . Consider the plane defined by the three points  $\tilde{\mathbf{x}}, \mathbf{q}, \mathbf{q}'$ . Since the origin  $\mathbf{0}$  is on the line passing through  $\mathbf{q}$  and  $\mathbf{q}'$ , it must also be in this plane. Now consider the line that passes through  $\tilde{\mathbf{x}}$  and  $\mathbf{q}'$ . Note that all points on this line have last coordinate equal to  $\kappa$ , and they are all also in the aforementioned plane. Let  $\mathbf{v} \oplus \langle \kappa \rangle$  be the projection of  $\mathbf{0}$  onto this line ( $\mathbf{v} \in \mathbb{R}^d$ ).

Note that the two triangles  $\Delta(\tilde{\mathbf{x}}, \mathbf{q}, \mathbf{q}')$  and  $\Delta(\mathbf{0}, \mathbf{v} \oplus \langle \kappa \rangle, \mathbf{q}')$  are similar since they are right triangles with opposite angles at  $\mathbf{q}'$ . Therefore, by triangle similarity,

$$\frac{\|\mathbf{q}'\|}{\|\mathbf{v} \oplus \langle \kappa \rangle\|} = \frac{\|\tilde{\mathbf{x}} - \mathbf{q}'\|}{\|\tilde{\mathbf{x}} - \mathbf{q}\|} \geq \frac{\text{dist}(\tilde{\mathbf{x}}, \mathcal{C}')}{\text{dist}(\tilde{\mathbf{x}}, \tilde{\mathcal{C}})} = \frac{\text{dist}(\mathbf{x}, \mathcal{C})}{\text{dist}(\tilde{\mathbf{x}}, \tilde{\mathcal{C}})}.$$

Since  $\mathbf{q}' \in \mathcal{C}'$ , we have  $\|\mathbf{q}'\| \leq \sqrt{(\max_{\mathbf{x} \in \mathcal{C}} \|\mathbf{x}\|)^2 + \kappa^2}$ , resulting in

$$\frac{\|\mathbf{q}'\|}{\|\mathbf{v} \oplus \langle \kappa \rangle\|} \leq \frac{\sqrt{(\max_{\mathbf{x} \in \mathcal{C}} \|\mathbf{x}\|)^2 + \kappa^2}}{\kappa} = \sqrt{1 + 2\delta} \leq 1 + \delta$$

by the choice of  $\kappa$  given in the lemma. Combining completes the proof for this case.

Case 2 ( $r = \kappa$ ): Since  $\mathbf{q} \in \text{cone}(\mathcal{C}')$  with  $\kappa$  as last coordinate, we have  $\mathbf{q} \in \mathcal{C}'$ . Thus,

$$\text{dist}(\mathbf{x}, \mathcal{C}) = \text{dist}(\tilde{\mathbf{x}}, \mathcal{C}') \leq \|\tilde{\mathbf{x}} - \mathbf{q}\| = \text{dist}(\tilde{\mathbf{x}}, \tilde{\mathcal{C}})$$

which completes the proof for this case.

Case 3 ( $0 < r < \kappa$ ): The proof for this case is formally identical to that of Case 1, except that, in this case, the two triangles  $\Delta(\tilde{\mathbf{x}}, \mathbf{q}, \mathbf{q}')$  and  $\Delta(\mathbf{0}, \mathbf{v} \oplus \langle \kappa \rangle, \mathbf{q}')$  are now similar as a result of being right triangles with a shared angle at  $\mathbf{q}'$ .

Case 4 ( $r = 0$ ): Since  $\mathbf{q} \in \text{cone}(\mathcal{C}')$ ,  $\mathbf{q}$  must have been generated by multiplying some  $\alpha \geq 0$  by some point in  $\mathcal{C}'$ . Since all points in  $\mathcal{C}'$  have last coordinate equal to  $\kappa > 0$ , and since  $r = 0$ , it must be the case that  $\alpha = 0$ , and thus,  $\mathbf{q} = \mathbf{0}$ . Let  $\mathbf{q}'$  be the projection of  $\tilde{\mathbf{x}}$  onto  $\mathcal{C}'$ . Consider the plane defined by the three points  $\tilde{\mathbf{x}}, \mathbf{q} = \mathbf{0}, \mathbf{q}'$ . Let  $\mathbf{q}''$  be the projection of  $\tilde{\mathbf{x}}$  onto the line passing through  $\mathbf{q}$  and  $\mathbf{q}'$ . Then

$$\|\tilde{\mathbf{x}} - \mathbf{q}''\| \leq \|\tilde{\mathbf{x}}\| = \text{dist}(\tilde{\mathbf{x}}, \tilde{\mathcal{C}}).$$

Now consider the line passing through  $\tilde{\mathbf{x}}$  and  $\mathbf{q}'$ . Note that all points on this line have last coordinate equal to  $\kappa$  and are also in the aforementioned plane. Let  $\mathbf{v} \oplus \langle \kappa \rangle$  be the projection of  $\mathbf{0}$  onto this line ( $\mathbf{v} \in \mathbb{R}^d$ ). Note that the two triangles  $\Delta(\tilde{\mathbf{x}}, \mathbf{q}'', \mathbf{q}')$  and  $\Delta(\mathbf{0}, \mathbf{v} \oplus \langle \kappa \rangle, \mathbf{q}')$  are similar since they are right triangles with a shared angle at  $\mathbf{q}'$ . Therefore, by triangle similarity,

$$\frac{\|\mathbf{q}'\|}{\|\mathbf{v} \oplus \langle \kappa \rangle\|} = \frac{\|\tilde{\mathbf{x}} - \mathbf{q}'\|}{\|\tilde{\mathbf{x}} - \mathbf{q}''\|} \geq \frac{\text{dist}(\tilde{\mathbf{x}}, \mathcal{C}')}{\text{dist}(\tilde{\mathbf{x}}, \tilde{\mathcal{C}})} = \frac{\text{dist}(\mathbf{x}, \mathcal{C})}{\text{dist}(\tilde{\mathbf{x}}, \tilde{\mathcal{C}})}.$$

The rest of the proof for this case is exactly as in Case 1.

### 8.6.6 Additional experimental details

All the models were trained using the following hyperparameters: policy network consists of 2-layer fully-connected MLP with ReLU activation and 128 hidden units and a A2C learning rate of  $10^{-2}$ . For APPROPO, the constant  $\kappa$  (§8.3.3) is set to be 20. In the following figures, the performance of the algorithms has been depicted using different hyperparamters; showing average and standard deviation over 25 runs,.

# Chapter 9: Conclusion

## 9.1 Summary of the Thesis

The primary contribution of this thesis was introducing empirically and theoretically justified algorithms that use an expert-in-the-loop to solve sequential decision and prediction problems. Furthermore, we draw a connection between modern imitation learning algorithms and modern structured prediction problems in natural language processing.

Chapter 3 We investigated issues in interactive imitation learning regarding being query-inefficient to an online expert. The problem in interactive imitation learning is that it assumes you can query an online expert at every state. We address this issue by considering access to an additional nosy heuristic labeling function (e.g., rule-based) to reduce expert queries. We introduce a new algorithm, LEAQI, that takes advantage of this nosy heuristic labeling function. Our results show compared to interactive imitation learning and naive active, interactive imitation learning techniques, LEAQI reduces online expert queries the most without reducing performance.

Chapter 4 We introduced a setting where the query-inefficient issue of interactive imitation learning is not a problem. In this setting, we study the problem of neural language models generating sequences in a purely left-to-right, monotonic order; instead of some other arbitrarily order. We cast learning generation order as an interactive imitation learning problem where the expert is a computational oracle instead of a human, making querying the expert at every state more reasonable. Although our models did not perform as well as the monotonic order models,

we gained insight into the trade-off of different generation orders.

Chapter 5 We introduce an algorithm DRIL to address the covariate shift issue in imitation learning without querying an online expert at every state. Instead, this algorithm only assumes that an expert provides demonstration data. This algorithm uses an ensemble of policies trained on the demonstration data variance as a reward function for reinforcement learning. Our algorithm DRIL has good empirical performance, and in some cases, provably performs better than naive imitation learning techniques.

Chapter 6 We extend the ideas of DRIL and perform a large-scale study of modern imitation learning algorithms. These algorithms learn a reward function offline or online and optimize this reward function using reinforcement learning. We perform a fair comparison by combining all of the tricks introduced by various algorithms to all algorithms considered in this large-scale study. We notice that one of the most important things in this class of algorithms is to interleave behavior cloning updates. We also related this new category of imitation learning techniques to modern structured prediction natural language processing algorithms.

Chapter 7 We introduced a framework for performing exact imitation learning. In particular, we learned a rich embedding space encompassing all possible policies, rewards, and expected features. This embedding space builds on ideas from success features, POMDP value iteration, and PSRs. Given how complex the embedding space is, we show the framework’s robustness on small and medium tabular problems. Once the embedding space is learned, we can look up a policy in our embedding space and read it off in constant time.

Chapter 8 We presented an algorithm that allows an expert to provide constraints to solve sequential decision and prediction problems, unlike providing demonstrations or being queried at every state, some settings where an expert providing constraints is more natural. We provide

theoretical and empirical results justifying the robustness of our presented algorithm compared to previous ideas.

## 9.2 Future Work

This section mentions a few of the possible directions for extending and building upon this work.

## 9.3 Imitation learning for Structured-Prediction

Chapter 6 discussed the connection between modern imitation learning algorithms and the most successful reinforcement/imitation learning algorithms in structure prediction NLP problems. However, chapter 6 only connects both fields but does not experiment with modern imitation learning algorithms to solve modern structure prediction for NLP problems.

While this thesis focuses strictly on studying modern imitation learning techniques and thoroughly investigating what aspects of the algorithms are essential for successfully developing and implementing them. A significant future step would be combining modern imitation learning ideas learned in chapter 6 with concepts from algorithms in modern structure prediction NLP to solve degenerate issues in text generation. Some algorithms, such as GAIL as already been applied to modern structure prediction NLP problems, as seen in Wu et al. [308], but as seen in chapter 6, GAIL is not the best performing algorithm. Modern structured prediction for NLP problems brings a different set of tricks and issues than ideas studied in chapter 6 structured prediction problems. For example, the models we used to train structured prediction problems in chapter 6 assumed pretrained embedding and were much smaller in size than typical modern NLP models. That means training an ensemble of policies in DRIL becomes infeasible because you

would not have the compute resources to train with multiple copies of a large NLP model. Even if you had the compute resources to train multiple copies of a large NLP model together, training time would be incredibly slow. There are more issues when lifting shallow modern imitation learning techniques to very large NLP models. An important future direction is formalizing the exposure-bias problems in modern structure-prediction NLP models and understanding what modern imitation learning algorithms are feasible to run.

## 9.4 Active Imitation Learning

We introduced an algorithm to reduce online expert queries discussed in chapter 3. Active imitation learning is the most popular category of algorithms that attempts to solve this issue. In particular, most of these algorithms apply active learning ideas to interactive imitation learning algorithms. A future direction is to reduce the amount of demonstration data needed for modern imitation learning algorithms that use demonstration data and environment interactions, similar to active imitation learning techniques that mitigate online expert queries. Though simply applying active learning to these algorithms will not be sufficient because these algorithms learn a reward function with the data provided, not a policy. This subtle difference means that borrowing ideas from batch active learning may be more relevant than online active learning ideas seen in current active imitation learning.

## 9.5 Imitation learning with Successor Features

As deep learning models continue to learn good features for solving downstream tasks, having imitation learning algorithms that directly learn from these features will become necessary. Chapter 7 introduces a framework that learns a complex embedding space for exact imitation.

The framework assumes that the behavior policy's expected features are not known in advance. Instead, we learned the embedding space first, which includes all possible approaches, rewards, and expected features in a given MDP. But the framework encompasses more information than needed for simply imitation featuring a particular behavior policy. Instead of imitating any possible behavior policy, a future direction could be relaxing the framework to imitate one specific behavior policy when given access to a sample of the behavior policy features in advance. The future direction is to develop algorithms that match the expected features of a behavior policy when given access to the behavior policy's expected features in advance. Furthermore, by simplifying the problem, the hope is that this new framework would scale beyond tabular settings.

## Bibliography

- [1] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1.
- [2] Jacob Abernethy, Peter L Bartlett, and Elad Hazan. 2011. Blackwell approachability and no-regret learning are equivalent. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 27–46.
- [3] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31.
- [4] Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. 2019. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep.*
- [5] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. 2021. [Deep reinforcement learning at the edge of the statistical precipice](#).
- [6] Roei Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. *arXiv preprint arXiv:1704.04743*.
- [7] Eitan Altman. 1999. *Constrained Markov decision processes*, volume 7. CRC Press.
- [8] David Alvarez-Melis and Tommi S Jaakkola. 2016. Tree-structured decoding with doubly-recurrent neural networks.
- [9] Marcin Andrychowicz, Anton Raichuk, Piotr Stanczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. 2020. What matters in on-policy reinforcement learning. *A large-scale empirical study. CoRR, abs/2006.05990*.
- [10] Kai Arulkumaran and Dan Ogawa Lillrank. 2021. A pragmatic look at deep imitation learning. *arXiv preprint arXiv:2108.01867*.
- [11] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. 1997. Locally weighted learning for control. *Lazy learning*, pages 75–113.
- [12] Les E Atlas, David A Cohn, and Richard E Ladner. 1990. Training connectionist networks with queries and selective sampling. In *NeurIPS*.
- [13] Isabelle Augenstein, Mrinal Das, Sebastian Riedel, Lakshmi Vikraman, and Andrew McCallum. 2017. Semeval 2017 task 10: Scienceie - extracting keyphrases and relations from scientific publications. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*.
- [14] Drew Bagnell. 2016. Feedback in machine learning. In *Workshop on Safety and control for artificial intelligence, CMU*.



- [15] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*.
- [16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.
- [17] Dzmitry Bahdanau, Dmitriy Serdyuk, Philémon Brakel, Nan Rosemary Ke, Jan Chorowski, Aaron Courville, and Yoshua Bengio. 2015. Task loss estimation for sequence prediction. *arXiv preprint arXiv:1511.06456*.
- [18] Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, (2):179–190.
- [19] Nina Balcan, Alina Beygelzimer, and John Langford. 2006. Agnostic active learning. In *ICML*.
- [20] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. 2018. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*.
- [21] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. 2018. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*, pages 501–510. PMLR.
- [22] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. 2017. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pages 4055–4065.
- [23] André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. 2020. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*.
- [24] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.
- [25] Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. Scibert: Pretrained language model for scientific text. In *EMNLP*.
- [26] Vassil Chatalbashev Ben Taskar, Carlos Guestrin and Daphne Koller. Max-margin markov networks.
- [27] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *NeurIPS*.
- [28] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *arXiv preprint arXiv:1506.03099*.
- [29] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.
- [30] Mathias Berglund, Tapani Raiko, Mikko Honkala, Leo Kärkkäinen, Akos Vetek, and Juha T Karhunen. 2015. Bidirectional recurrent neural networks as generative models. In *Advances in Neural Information Processing Systems*, pages 856–864.

- [31] Alina Beygelzimer, Sanjoy Dasgupta, , and John Langford. 2009. Importance weighted active learning. In *ICML*.
- [32] Alina Beygelzimer, Daniel Hsu, John Langford, and Tong Zhang. 2010. Agnostic active learning without constraints. In *NeurIPS*.
- [33] David Blackwell. 1956. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8.
- [34] Lionel Blondé and Alexandros Kalousis. 2019. Sample-efficient imitation learning via generative adversarial nets. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3138–3148. PMLR.
- [35] Lionel Blondé, Pablo Strasser, and Alexandros Kalousis. 2020. Lipschitzness is all you need to tame off-policy generative adversarial imitation learning. *arXiv preprint arXiv:2006.16785*.
- [36] Lionel Blondé, Pablo Strasser, and Alexandros Kalousis. 2020. [Lipschitzness is all you need to tame off-policy generative adversarial imitation learning](#). *CoRR*, abs/2006.16785.
- [37] Michael Bloodgood and Chris Callison-Burch. 2010. Bucking the trend: Large-scale cost-focused active learning for statistical machine translation. In *ACL*.
- [38] Diana Borsa, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado van Hasselt, David Silver, and Tom Schaul. 2018. Universal successor features approximators. *arXiv preprint arXiv:1812.07626*.
- [39] Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*.
- [40] James P Boyle and Richard L Dykstra. 1986. A method for finding projections onto the intersection of convex sets in hilbert spaces. In *Advances in order restricted statistical inference*, pages 28–47. Springer.
- [41] Kianté Brantley, Soroush Mehri, and Geoffrey J Gordon. 2021. Successor feature sets: Generalizing successor representations across policies. *arXiv preprint arXiv:2103.02650*.
- [42] Kianté Brantley, Amr Sharaf, and Hal Daumé III. 2020. Active imitation learning with noisy guidance. *arXiv preprint arXiv:2005.12801*.
- [43] Kianté Brantley, Wen Sun, and Mikael Henaff. 2019. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*.
- [44] Kianté Brantley, Wen Sun, and Mikael Henaff. 2020. [Disagreement-regularized imitation learning](#). In *International Conference on Learning Representations*.
- [45] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. [Openai gym](#).
- [46] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

- [47] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. [A statistical approach to machine translation](#). *Comput. Linguist.*, 16(2):79–85.
- [48] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2019. [Exploration by random network distillation](#). In *International Conference on Learning Representations*.
- [49] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, Laurent Charlin MILA, and Hec Montréal. 2018. [Language gans falling short](#). *arXiv preprint 1811.02549*.
- [50] Chris Callison-Burch, Miles Osborne, and Philipp Koehn. 2006. Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*.
- [51] Erin Catto. 2011. Box2d: A 2d physics engine for games. URL: <http://www.box2d.org>.
- [52] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. 2006. Worst-case analysis of selective sampling for linear classification. *JMLR*.
- [53] Jonathan D Chang, Masatoshi Uehara, Dhruv Sreenivas, Rahul Kidambi, and Wen Sun. 2021. Mitigating covariate shift in imitation learning via offline data without great coverage. *arXiv preprint arXiv:2106.03207*.
- [54] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. 2015. Learning to search better than your teacher. *arXiv preprint arXiv:1502.02206*.
- [55] Ching-An Cheng and Byron Boots. 2018. Convergence of value aggregation for imitation learning. *arXiv preprint arXiv:1801.07292*.
- [56] Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. 2018. [Fast Policy Learning through Imitation and Reinforcement](#). *arXiv preprint 1805.10413*.
- [57] David Chiang. 2012. Hope and fear for discriminative training of statistical translation models. *JMLR*, 13.
- [58] Ting-Rui Chiang and Yun-Nung Chen. 2021. Relating neural text degeneration to exposure bias. *arXiv preprint arXiv:2109.08705*.
- [59] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. 2015. Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, 17(11):1875–1886.
- [60] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. [Learning phrase representations using RNN encoder-decoder for statistical machine translation](#). In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- [61] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*.
- [62] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

- [63] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. 2015. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585.
- [64] Massimiliano Ciaramita and Yasemin Altun. 2006. Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 594–602.
- [65] Axel Cleeremans, David Servan-Schreiber, and James L McClelland. 1989. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381.
- [66] William S Cleveland and Susan J Devlin. 1988. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83(403):596–610.
- [67] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. 2019. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9329–9338.
- [68] Samuel Cohen, Brandon Amos, Marc Peter Deisenroth, Mikael Henaff, Eugene Vinitzky, and Denis Yarats. 2021. Imitation learning from pixel observations for continuous control. In *Deep RL Workshop NeurIPS 2021*.
- [69] Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL*.
- [70] Erwin Coumin and Yunfei Bai. 2017. pybullet, a python module for physics simulation in robotics, games and machine learning.
- [71] Jan Christian Blaise Cruz and Charibeth Cheng. 2019. Evaluating language model finetuning techniques for low-resource languages. *arXiv preprint arXiv:1907.00409*.
- [72] Aron Culotta and Andrew McCallum. 2005. Reducing labeling effort for structured prediction tasks. In *AAAI*.
- [73] Robert Dadashi, Leonard Hussenot, Matthieu Geist, and Olivier Pietquin. 2021. [Primal wasserstein imitation learning](#). In *International Conference on Learning Representations*.
- [74] Steven Dalton, Iuri Frosio, and Michael Garland. 2019. Accelerating reinforcement learning through gpu atari emulation. *arXiv preprint arXiv:1907.08467*.
- [75] Hal Daumé, John Langford, and Daniel Marcu. 2009. [Search-based structured prediction](#). *CoRR*, abs/0907.0786.
- [76] Hal Daumé and Daniel Marcu. 2009. [Learning as search optimization: Approximate large margin methods for structured prediction](#). *CoRR*, abs/0907.0809.
- [77] Hal Daumé, III. 2009. Unsupervised search-based structured prediction. In *International Conference on Machine Learning*, Montreal, Canada.
- [78] Hal Daumé, III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning Journal*.

- [79] Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.
- [80] Peter Dayan. 1993. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624.
- [81] Ernesto De Vito, Lorenzo Rosasco, and Alessandro Toigo. 2014. A universally consistent spectral estimator for the support of a distribution. *Appl Comput Harmonic Anal*, 37:185–217.
- [82] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.
- [83] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*.
- [84] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*.
- [85] Ahmad Emami and Frederick Jelinek. 2005. A neural syntactic language model. *Machine learning*, 60(1-3):195–227.
- [86] Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. *arXiv preprint arXiv:1702.03525*.
- [87] Haiyan Fan and Marshall Scott Poole. 2006. What is personalization? perspectives on the design and implementation of personalization in information systems. *Journal of Organizational Computing and Electronic Commerce*, 16(3-4):179–202.
- [88] Meng Fang and Trevor Cohn. 2016. Learning when to trust distant supervision: An application to low-resource pos tagging using cross-lingual projection. *arXiv preprint arXiv:1607.01133*.
- [89] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. 2016. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*.
- [90] Corina Florescu and Cornelia Caragea. 2017. PositionRank: An unsupervised approach to keyphrase extraction from scholarly documents. In *ACL*.
- [91] Mikel L Forcada and Ramón Neco. 1997. Recursive hetero-associative memories for translation. In *International Work-Conference on Artificial Neural Networks*.
- [92] Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George E Dahl. 2018. The importance of generation order in language modeling. *arXiv preprint arXiv:1808.07910*.
- [93] Pedro Freire, Adam Gleave, Sam Toyer, and Stuart Russell. 2020. [Derail: Diagnostic environments for reward and imitation learning](#).
- [94] Yoav Freund and Robert E Schapire. 1999. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103.
- [95] Justin Fu, Katie Luo, and Sergey Levine. 2017. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*.

- [96] Justin Fu, Katie Luo, and Sergey Levine. 2018. [Learning robust rewards with adversarial inverse reinforcement learning](#). In *International Conference on Learning Representations*.
- [97] Scott Fujimoto and Shixiang Shane Gu. 2021. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*.
- [98] Yarin Gal and Zoubin Ghahramani. 2016. [Dropout as a bayesian approximation: Representing model uncertainty in deep learning](#). In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA. PMLR.
- [99] Matthew PH Gardner, Geoffrey Schoenbaum, and Samuel J Gershman. 2018. Rethinking dopamine as generalized prediction error. *Proceedings of the Royal Society B*, 285(1891):20181645.
- [100] Dan Garrette and Jason Baldridge. 2013. Learning a part-of-speech tagger from two hours of annotation. In *Proceedings of the 2013 conference of the North American chapter of the association for computational linguistics: Human language technologies*, pages 138–147.
- [101] Samuel J Gershman. 2018. The successor representation: its computational logic and neural substrates. *Journal of Neuroscience*, 38(33):7193–7200.
- [102] Samuel J Gershman, Christopher D Moore, Michael T Todd, Kenneth A Norman, and Per B Sederberg. 2012. The successor representation and temporal context. *Neural Computation*, 24(6):1553–1568.
- [103] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. 2020. A divergence minimization perspective on imitation learning methods. In *Conference on Robot Learning*, pages 1259–1277. PMLR.
- [104] Seyed Kamyar Seyed Ghasemipour, Richard S. Zemel, and Shixiang Gu. 2019. [A divergence minimization perspective on imitation learning methods](#). *CoRR*, abs/1911.02256.
- [105] Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *NAACL*.
- [106] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*, volume 1. MIT Press.
- [107] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.
- [108] Kartik Goyal, Chris Dyer, and Taylor Berg-Kirkpatrick. 2017. Differentiable scheduled sampling for credit assignment. *arXiv preprint arXiv:1704.06970*.
- [109] Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.
- [110] Ben Hachey, Beatrice Alex, and Markus Becker. 2005. Investigating the effects of selective sampling on the annotation task. In *CoNLL*.
- [111] Robbie Haertel, Eric K. Ringger, Kevin D. Seppi, James L. Carroll, and Peter McClanahan. 2008. Assessing the costs of sampling methods in active learning for annotation. In *ACL*.



- [112] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2019. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR.
- [113] Aria Haghighi and Dan Klein. 2006. Prototype-driven learning for sequence models.
- [114] Steven Hansen, Will Dabney, Andre Barreto, Tom Van de Wiele, David Warde-Farley, and Volodymyr Mnih. 2019. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*.
- [115] Elad Hazan, Sham M Kakade, Karan Singh, and Abby Van Soest. 2018. Provably efficient maximum entropy exploration. *arXiv preprint arXiv:1812.02690*.
- [116] Tamir Hazan, Joseph Keshet, and David A McAllester. 2010. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, pages 1594–1602.
- [117] He He, Jason Eisner, and Hal Daume. 2012. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*, pages 3149–3157.
- [118] Michael A Hedderich, Lukas Lange, Heike Adel, Jannik Strötgen, and Dietrich Klakow. 2020. A survey on recent approaches for natural language processing in low-resource scenarios. *arXiv preprint arXiv:2010.12309*.
- [119] Ahmed Hefny, Carlton Downey, and Geoffrey Gordon. 2015. Supervised learning for dynamical system learning. In *Advances in neural information processing systems*.
- [120] David P. Helmbold, Nicholas Littlestone, and Philip M. Long. 2000. Apple tasting. *Information and Computation*.
- [121] Mikael Henaff. 2019. [Explicit explore-exploit algorithms in continuous state spaces](#). In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alche-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 9377–9387. Curran Associates, Inc.
- [122] Mikael Henaff, Alfredo Canziani, and Yann LeCun. 2019. [Model-predictive policy learning with uncertainty regularization for driving in dense traffic](#). In *International Conference on Learning Representations*.
- [123] Matteo Hessel, Joseph Modayil, Hado P. van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*.
- [124] Todd Hester, Matej Vecerík, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. 2018. [Deep q-learning from demonstrations](#). In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3223–3230.
- [125] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable baselines. <https://github.com/hill-a/stable-baselines>.

- [126] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc.
- [127] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*.
- [128] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.
- [129] Daniel Hsu. 2019. A new framework for query efficient active imitation learning. *arXiv preprint arXiv:1912.13037*.
- [130] Ferenc Huszár. 2015. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*.
- [131] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. 2002. Learning attractor landscapes for learning motor primitives. Technical report.
- [132] John M Ingram and MM Marsh. 1991. Projections onto convex cones in hilbert space. *Journal of approximation theory*, 64(3):343–350.
- [133] Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. 2010. Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 944–952. Association for Computational Linguistics.
- [134] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. 2017. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *arXiv preprint arXiv:1704.05519*.
- [135] Rohit Jena, Siddharth Agrawal, and Katia Sycara. 2020. Addressing reward bias in adversarial imitation learning with neutral reward functions. *arXiv preprint arXiv:2009.09467*.
- [136] Rohit Jena, Changliu Liu, and Katia Sycara. 2020. Augmenting gail with bc for sample efficient imitation learning. *arXiv preprint arXiv:2001.07798*.
- [137] Rohit Jena, Changliu Liu, and Katia Sycara. 2020. Augmenting gail with bc for sample efficient imitation learning. *arXiv preprint arXiv:2001.07798*.
- [138] William B Johnson and Joram Lindenstrauss. 1984. Extensions of lipschitz mappings into a hilbert space 26. *Contemporary mathematics*, 26.
- [139] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.
- [140] Kshitij Judah, Alan Fern, and Thomas Dietterich. 2011. Active imitation learning via state queries. In *ICML Workshop on Combining Strategies for Reducing the Label Cost*.
- [141] Kshitij Judah, Alan Paul Fern, and Thomas Glenn Dietterich. 2012. Active imitation learning via reduction to iid active learning. In *AAAI*.
- [142] Matti Kääriäinen. 2006. Lower bounds for reductions. In *Atomic Learning Workshop*.
- [143] Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer.



- [144] Sham M Kakade. 2001. A natural policy gradient. *Advances in neural information processing systems*, 14.
- [145] Anssi Kanervisto, Joonas Pussinen, and Ville Hautamäki. 2020. [Benchmarking end-to-end behavioural cloning on video games](#).
- [146] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.
- [147] Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee, and Siddhartha Srinivasa. 2020. Imitation learning as f-divergence minimization. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 313–329. Springer.
- [148] Daniel Khashabi, Mark Sammons, Ben Zhou, Tom Redman, Christos Christodoulopoulos, Vivek Srikumar, Nicholas Rizzolo, Lev Ratnov, Guanheng Luo, Quang Do, Chen-Tse Tsai, Subhro Roy, Stephen Mayhew, Zhili Feng, John Wieting, Xiaodong Yu, Yangqiu Song, Shashank Gupta, Shyam Upadhyay, Naveen Arivazhagan, Qiang Ning, Shaoshi Ling, and Dan Roth. 2018. CogCompNLP: Your swiss army knife for NLP. In *LREC*.
- [149] Beomjoon Kim and Joelle Pineau. 2013. Maximum mean discrepancy imitation learning. In *Robotics: Science and systems*.
- [150] Kee-Eung Kim and Hyun Soo Park. 2018. Imitation learning via kernel mean embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.
- [151] Kuno Kim, Akshat Jindal, Yang Song, Jiaming Song, Yanan Sui, and Stefano Ermon. 2021. Imitation with neural density models. In *Thirty-Fifth Conference on Neural Information Processing Systems*.
- [152] Diederik P. Kingma and Jimmy Ba. 2014. [Adam: A method for stochastic optimization](#). Cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
- [153] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [154] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics.
- [155] Ilya Kostrikov. 2018. `pytorch-a2c-ppo-acktr`.
- [156] Ilya Kostrikov. 2018. Pytorch implementations of reinforcement learning algorithms. <https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail>.
- [157] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. 2019. [Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning](#). In *International Conference on Learning Representations*.

- [158] Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. 2019. Imitation learning via off-policy distribution matching. *arXiv preprint arXiv:1912.05032*.
- [159] Ilya Kostrikov, Denis Yarats, and Rob Fergus. 2020. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*.
- [160] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. 2017. Imitating driver behavior with generative adversarial networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 204–211. IEEE.
- [161] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. 2016. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*.
- [162] Michael Laskey, Caleb Chuck, Jonathan Lee, Jeffrey Mahler, Sanjay Krishnan, Kevin Jamieson, Anca Dragan, and Ken Goldberg. 2017. Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 358–365. IEEE.
- [163] Michael Laskey, Sam Staszak, Wesley Yu-Shu Hsieh, Jeffrey Mahler, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. 2016. Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 462–469. IEEE.
- [164] Alon Lavie and Abhaya Agarwal. 2007. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 228–231. Association for Computational Linguistics.
- [165] Hoang M Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. 2018. Hierarchical imitation and reinforcement learning. *arXiv preprint arXiv:1803.00590*.
- [166] Hoang M Le, Cameron Voloshin, and Yisong Yue. 2019. Batch policy learning under constraints. *arXiv preprint arXiv:1903.08738*.
- [167] Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. 2018. SeaRNN: Training RNNs with global-local losses. In *ICLR*.
- [168] Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. 2018. SEARNN: Training RNNs with global-local losses. In *ICLR*.
- [169] Donghun Lee, Srivatsan Srinivasan, and Finale Doshi-Velez. 2019. Truly batch apprenticeship learning with deep successor features. *arXiv preprint arXiv:1903.10077*.
- [170] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2011. Stanford’s multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*.
- [171] Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*.
- [172] Lucas Lehnert and Michael L. Littman. 2019. Successor features combine elements of model-free and model-based reinforcement learning. Technical Report arXiv:1901.11437.

- [173] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*.
- [174] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*.
- [175] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.
- [176] Chin-Yew Lin and Eduard Hovy. 2003. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 150–157.
- [177] N. Littlestone and M. K. Warmuth. 1989. The weighted majority algorithm. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*.
- [178] Michael L Littman, Richard S Sutton, and Satinder P Singh. 2001. Predictive representations of state. In *NIPS*, volume 14, page 30.
- [179] Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023*.
- [180] Minghuan Liu, Tairan He, Minkai Xu, and Weinan Zhang. 2020. Energy-based imitation learning. *arXiv preprint arXiv:2004.09395*.
- [181] Chi-kiu Lo. 2018. [YiSi: A semantic machine translation evaluation metric for evaluating languages with different levels of available resources](#).
- [182] Yuping Luo, Huazhe Xu, and Tengyu Ma. 2019. [Learning self-correctable policies and value functions from demonstrations with negative sampling](#). *CoRR*, abs/1907.05634.
- [183] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics.
- [184] Chen Ma, Junfeng Wen, and Yoshua Bengio. 2018. Universal successor representations for transfer reinforcement learning. *arXiv preprint arXiv:1804.03758*.
- [185] Marlos C Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauero, and Murray Campbell. 2017. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*.
- [186] Francis Maes, Ludovic Denoyer, and Patrick Gallinari. 2009. Structured prediction with reinforcement learning. *Machine learning*, 77(2):271–301.
- [187] Elman Mansimov and Kyunghyun Cho. 2018. Simple nearest neighbor policy method for continuous control tasks.
- [188] Raphael Memmesheimer, Ivanna Kramer, Viktor Seib, and Dietrich Paulus. 2019. Simitate: A hybrid imitation learning benchmark. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5243–5249. IEEE.

- [189] Kunal Menda, Katherine Rose Driggs-Campbell, and Mykel J. Kochenderfer. 2018. Ensembledagger: A bayesian approach to safe imitation learning. *ArXiv*, abs/1807.08364.
- [190] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. [Asynchronous methods for deep reinforcement learning](#). In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA. PMLR.
- [191] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.
- [192] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. [Human-level control through deep reinforcement learning](#). *Nature*, 518(7540):529–533.
- [193] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.
- [194] Aaqib Parvez Mohammed and Matias Valdenegro-Toro. 2020. Can reinforcement learning for continuous control generalize across physics engines? *arXiv preprint arXiv:2010.14444*.
- [195] Ida Momennejad, Evan M Russek, Jin H Cheong, Matthew M Botvinick, Nathaniel Douglass Daw, and Samuel J Gershman. 2017. The successor representation in human reinforcement learning. *Nature Human Behaviour*, 1(9):680–692.
- [196] Rémi Munos and Csaba Szepesvári. 2008. Finite-time bounds for fitted value iteration. *J. Mach. Learn. Res.*, 9:815–857.
- [197] Ofir Nachum and Mengjiao Yang. 2021. Provable representation learning for imitation with contrastive fourier features. *arXiv preprint arXiv:2105.12272*.
- [198] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. 2017. Overcoming exploration in reinforcement learning with demonstrations. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299.
- [199] J. von Neumann. 1928. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100:295–320.
- [200] Joakim et. al Nivre. 2018. Universal dependencies v2.5. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University.
- [201] Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C Cobo, Florian Stimberg, et al. 2017. Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433*.
- [202] Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. 2021. [What matters for adversarial imitation learning?](#)

- [203] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. 2018. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*.
- [204] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. [Deep exploration via bootstrapped DQN](#). *CoRR*, abs/1602.04621.
- [205] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.
- [206] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. 2019. Self-supervised exploration via disagreement. In *ICML*.
- [207] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.
- [208] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. [Glove: Global vectors for word representation](#). In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.
- [209] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*.
- [210] Joelle Pineau, Geoffrey J. Gordon, and Sebastian B. Thrun. 2003. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025–1030.
- [211] D Pomerleau. 1998. An autonomous land vehicle in a neural network. *Advances in Neural Information Processing Systems; Morgan Kaufmann Publishers Inc.: Burlington, MA, USA*.
- [212] Dean A Pomerleau. 1989. Alvin: An autonomous land vehicle in a neural network. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY ....
- [213] Dean A. Pomerleau. 1989. [Alvin: An autonomous land vehicle in a neural network](#). In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan-Kaufmann.
- [214] P. K. Pook and D. H. Ballard. 1993. [Recognizing teleoperated manipulations](#). In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 578–585 vol.2.
- [215] Polly K Pook and Dana H Ballard. 1993. Recognizing teleoperated manipulations. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 578–585. IEEE.
- [216] Dev Bahadur Poudel. 2013. Coordinating hundreds of cooperative, autonomous robots in a warehouse. *Jan*, 27(1-13):26.
- [217] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2013. Towards robust linguistic analysis using ontonotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152.

- [218] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. 2017. Neural episodic control. In *International Conference on Machine Learning*, pages 2827–2836. PMLR.
- [219] Martin L. Puterman. 1990. Markov decision processes. *Handbooks in Operations Research and Management Science*, 2:331–434.
- [220] Rafael Rafailov, Tianhe Yu, Aravind Rajeswaran, and Chelsea Finn. 2021. Visual adversarial imitation learning using variational models. *arXiv preprint arXiv:2107.08829*.
- [221] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. 2019. Stable baselines3. <https://github.com/DLR-RM/stable-baselines3>.
- [222] Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. 2020. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*.
- [223] Nived Rajaraman, Lin F Yang, Jiantao Jiao, and Kannan Ramachandran. 2020. Toward the fundamental limits of imitation learning. *arXiv preprint arXiv:2009.05990*.
- [224] Rajkumar Ramamurthy, Rafet Sifa, and Christian Bauckhage. 2020. [Nlpgym – a toolkit for evaluating rl agents on natural language processing tasks](#).
- [225] Marc’Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.
- [226] Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *EMNLP*.
- [227] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. 2019. [SQL: imitation learning via regularized behavioral cloning](#). *CoRR*, abs/1905.11108.
- [228] Larry Rendell. 1986. A general framework for induction and a study of selective induction. *Machine Learning Journal*.
- [229] Ellen Riloff and Janyce Wiebe. 2003. Learning extraction patterns for subjective expressions. In *EMNLP*.
- [230] Eric Ringger, Peter McClanahan, Robbie Haertel, George Busby, Marc Carmen, James Carroll, Kevin Seppi, and Deryle Lonsdale. 2007. Active learning for part-of-speech tagging: Accelerating corpus annotation. In *Proceedings of the Linguistic Annotation Workshop*.
- [231] Stephane Ross and Drew Bagnell. 2010. [Efficient reductions for imitation learning](#). In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy. PMLR.
- [232] Stéphane Ross and Drew Bagnell. 2010. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668.
- [233] Stephane Ross and J Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*.



- [234] Stéphane Ross, Geoff J. Gordon, and J. Andrew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *AI-Stats*.
- [235] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.
- [236] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. [A reduction of imitation learning and structured prediction to no-regret online learning](#). In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA. PMLR.
- [237] Evan M Russek, Ida Momennejad, Matthew M Botvinick, Samuel J Gershman, and Nathaniel D Daw. 2017. Predictive representations can link model-based reinforcement learning to model-free mechanisms. *PLoS computational biology*, 13(9):e1005768.
- [238] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76.
- [239] Erik F Sang and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task: Chunking. *arXiv preprint cs/0009008*.
- [240] Erik F Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*.
- [241] Ruhi Sarikaya. 2017. The technology behind personal digital assistants: An overview of the system architecture and key components. *IEEE Signal Processing Magazine*, 34(1):67–81.
- [242] Fumihiko Sasaki, Tetsuya Yohira, and Atsuo Kawaguchi. 2019. [Sample efficient imitation learning for continuous control](#). In *International Conference on Learning Representations*.
- [243] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. 2015. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320.
- [244] Allen Schmalz, Alexander M. Rush, and Stuart Shieber. 2016. [Word ordering without syntax](#). In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2319–2324. Association for Computational Linguistics.
- [245] Jürgen Schmidhuber. 1991. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463.
- [246] Jürgen Schmidhuber. 1991. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227.
- [247] Florian Schmidt. 2019. Generalization in generation: A closer look at exposure bias. *arXiv preprint arXiv:1910.00292*.
- [248] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319.

- [249] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.
- [250] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.
- [251] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. [Proximal policy optimization algorithms](#). *CoRR*, abs/1707.06347.
- [252] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- [253] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. 2014. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*.
- [254] David Sculley. 2007. Practical learning from one-sided feedback. In *KDD*.
- [255] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.
- [256] Guy Shani, Joelle Pineau, and Robert Kaplow. 2013. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27:1–51.
- [257] Vighnesh Leonardo Shiv and Chris Quirk. 2019. [Novel positional encodings to enable tree-structured transformers](#).
- [258] Pranav Shyam, Wojciech Jaskowski, and Faustino Gomez. 2018. Model-based active exploration. *CoRR*, abs/1810.12162.
- [259] Natalia Silveira, Timothy Dozat, Marie-Catherine De Marneffe, Samuel R Bowman, Miriam Connor, John Bauer, and Christopher D Manning. 2014. A gold standard dependency corpus for english. In *LREC*, pages 2897–2904. Citeseer.
- [260] David Silver. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503.
- [261] B. W. Silverman and M. C. Jones. 1989. E. fix and j.l. hedges (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on fix and hedges (1951). *International Statistical Review / Revue Internationale de Statistique*, 57(3):233–238.
- [262] Satinder Singh, Michael James, and Matthew Rudary. 2012. Predictive state representations: A new theory for modeling dynamical systems. *arXiv preprint arXiv:1207.4167*.
- [263] Vikash Singh. 2017. Replace or retrieve keywords in documents at scale. *CoRR*, abs/1711.00046.
- [264] Maurice Sion. 1958. On general minimax theorems. *Pacific Journal of mathematics*, 8(1):171–176.
- [265] Peter Smit, Sami Virpioja, Stig-Arne Grönroos, and Mikko Kurimo. 2014. Morfessor 2.0: Toolkit for statistical morphological segmentation. In *EACL*.



- [266] Jonathan Spencer, Sanjiban Choudhury, Arun Venkatraman, Brian Ziebart, and J. Andrew Bagnell. 2021. [Feedback in imitation learning: The three regimes of covariate shift](#).
- [267] Jonathan Spencer, Sanjiban Choudhury, Arun Venkatraman, Brian Ziebart, and J. Andrew Bagnell. 2021. [Feedback in imitation learning: The three regimes of covariate shift](#).
- [268] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.
- [269] Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. 2017. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643.
- [270] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, pages 10107–10116.
- [271] Veselin Stoyanov and Jason Eisner. 2012. Easy-first coreference resolution. *Proceedings of COLING 2012*, pages 2519–2534.
- [272] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. 2017. Deeply aggravated: Differentiable imitation learning for sequential prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3309–3318. JMLR. org.
- [273] Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.
- [274] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NeurIPS*.
- [275] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
- [276] Gokul Swamy, Sanjiban Choudhury, Zhiwei Steven Wu, and J Andrew Bagnell. 2021. Of moments and matching: Trade-offs and treatments in imitation learning. *arXiv preprint arXiv:2103.03236*.
- [277] Umar Syed and Robert E. Schapire. 2008. A game-theoretic approach to apprenticeship learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- [278] Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.
- [279] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. 2018. Deepmind control suite. *arXiv preprint arXiv:1801.00690*.
- [280] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. 2019. [Reward constrained policy optimization](#). In *International Conference on Learning Representations*.
- [281] Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. 1999. Active learning for natural language parsing and information extraction. In *ICML*.

- [282] Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *NAACL/HLT*.
- [283] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.
- [284] Mikolov Tomas. 2012. Statistical language models based on neural networks. *Brno University of Technology*.
- [285] Sam Toyer, Rohin Shah, Andrew Critch, and Stuart Russell. 2020. [The magical benchmark for robust imitation](#).
- [286] Yoshimasa Tsuruoka and Jun’ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 467–474. Association for Computational Linguistics.
- [287] Aaron Tucker, Adam Gleave, and Stuart Russell. 2018. Inverse reinforcement learning for video games. *arXiv preprint arXiv:1810.10593*.
- [288] Vladimir Vapnik. 1982. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag, Berlin, Heidelberg.
- [289] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#). In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.
- [290] Arun Venkatraman, Martial Hebert, and J. Andrew Bagnell. 2015. [Improving multi-step prediction of learned time series models](#). In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI’15*, pages 3024–3030. AAAI Press.
- [291] Eszter V rt s and Maneesh Sahani. 2019. A neurally plausible model learns successor representations in partially observable environments. In *Advances in Neural Information Processing Systems*, pages 13714–13724.
- [292] Oriol Vinyals and Quoc Le. 2015. A neural conversational model. *arXiv preprint arXiv:1506.05869*.
- [293] Andreas Vlachos and Stephen Clark. 2014. A new corpus and imitation learning framework for context-dependent semantic parsing. *Transactions of the Association for Computational Linguistics*, 2:547–560.
- [294] Matthew P Wand and William R Schucany. 1990. Gaussian-based kernels. *Canadian Journal of Statistics*, 18(3):197–204.
- [295] Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. *arXiv preprint arXiv:1808.08583*.
- [296] Ruohan Wang, Carlo Ciliberto, Pierlugi Amadori, and Yiannis Demiris. 2019. Random expert distillation: Imitation learning via expert policy support estimation. In *Proceedings of International Conference on Machine Learning*. ACM.

- [297] Ruohan Wang, Carlo Ciliberto, Pierluigi Amadori, and Yiannis Demiris. 2020. Support-weighted adversarial imitation learning. *arXiv preprint arXiv:2002.08803*.
- [298] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2020. Automated concatenation of embeddings for structured prediction. *arXiv preprint arXiv:2010.05006*.
- [299] Sean Welleck, Kianté Brantley, Hal Daumé, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. In *ICML*.
- [300] Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. 2019. [Non-monotonic sequential text generation](#). *CoRR*, abs/1902.02192.
- [301] Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2019. Neural text generation with unlikelihood training. *arXiv preprint arXiv:1908.04319*.
- [302] Sean Welleck, Zixin Yao, Yu Gai, Jialin Mao, Zheng Zhang, and Kyunghyun Cho. 2018. Loss functions for multiset prediction. In *Advances in Neural Information Processing Systems*, pages 5788–5797.
- [303] Steven Whitehead. 1991. A study of cooperative mechanisms for faster reinforcement learning. Technical report, University of Rochester.
- [304] Yorrick Wilks. 2008. *Machine translation: its scope and limits*. Springer Science & Business Media.
- [305] Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*.
- [306] George Wolberg. 1990. *Digital image warping*, volume 10662. IEEE computer society press Los Alamitos, CA.
- [307] Lijun Wu, Yingce Xia, Fei Tian, Li Zhao, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. Adversarial neural machine translation. In *Asian Conference on Machine Learning*, pages 534–549. PMLR.
- [308] Qingyang Wu, Lei Li, and Zhou Yu. 2020. Textgail: Generative adversarial imitation learning for text generation. *arXiv preprint arXiv:2004.13796*.
- [309] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.
- [310] Huang Xiao, Michael Herman, Joerg Wagner, Sebastian Ziesche, Jalal Etesami, and Thai Hong Linh. 2019. Wasserstein adversarial imitation learning. *arXiv preprint arXiv:1906.08113*.
- [311] Danfei Xu and Misha Denil. 2019. Positive-unlabeled reward learning. *arXiv preprint arXiv:1911.00459*.
- [312] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. 2020. Luke: deep contextualized entity representations with entity-aware self-attention. *arXiv preprint arXiv:2010.01057*.

- [313] Denis Yarats, Ilya Kostrikov, and Rob Fergus. 2021. [Image augmentation is all you need: Regularizing deep reinforcement learning from pixels](#). In *International Conference on Learning Representations*.
- [314] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2016. [Seqgan: Sequence generative adversarial nets with policy gradient](#). *CoRR*, abs/1609.05473.
- [315] Xingrui Yu, Yueming Lyu, and Ivor Tsang. 2020. Intrinsic reward driven imitation learning via generative model. In *International Conference on Machine Learning*, pages 10925–10935. PMLR.
- [316] Torsten Zesch, Christof Müller, and Iryna Gurevych. 2008. Extracting lexical semantic knowledge from Wikipedia and Wiktionary. In *LREC*.
- [317] Chicheng Zhang and Kamalika Chaudhuri. 2015. Active learning from weak and strong labelers. In *NeurIPS*.
- [318] Jiakai Zhang and Kyunghyun Cho. 2017. Query-efficient imitation learning for end-to-end simulated driving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.
- [319] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. 2017. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2371–2378. IEEE.
- [320] Ruohan Zhang, Zhuode Liu, Lin Guan, Luxin Zhang, Mary M. Hayhoe, and Dana H. Ballard. 2019. [Atari-head: Atari human eye-tracking and demonstration dataset](#). *CoRR*, abs/1903.06754.
- [321] Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2204–2213, Melbourne, Australia. Association for Computational Linguistics.
- [322] Shangdong Zhang. 2018. Modularized implementation of deep rl algorithms in pytorch. <https://github.com/ShangdongZhang/DeepRL>.
- [323] Wen Zhang, Yang Feng, Fandong Meng, Di You, and Qun Liu. 2019. Bridging the gap between training and inference for neural machine translation. *arXiv preprint arXiv:1906.02448*.
- [324] Xingxing Zhang, Liang Lu, and Mirella Lapata. 2015. Top-down tree long short-term memory networks. *arXiv preprint arXiv:1511.00060*.
- [325] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texus: A benchmarking platform for text generation models. *SIGIR*.
- [326] Yuke Zhu, Daniel Gordon, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. 2017. Visual semantic planning using deep successor representations. In *Proceedings of the IEEE international conference on computer vision*, pages 483–492.
- [327] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. 2008. Maximum entropy inverse reinforcement learning. In *AaaI*, volume 8, pages 1433–1438. Chicago, IL, USA.
- [328] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*.