Practical Techniques for Leveraging Experts for Sequential Decisions and Predictions

Preliminary Oral Exam (Thesis Proposal)

Kianté Brantley
Department of Computer Science
University of Maryland, College Park
`kdbrant@umd.edu`

Dissertation Proposal submitted to:
Department of Computer Science
University of Maryland, College Park, MD 20742

November 26, 2021

Advisory Committee:

Dr. Hal Daumé III	Chair	University of Maryland, College Park
Dr. Tom Goldstein	Dept's Rep	University of Maryland, College Park
Dr. John Baras	Dean's Rep	University of Maryland, College Park
Dr. Philip Resnik	Member	University of Maryland, College Park
Dr. Geoff Gordon	Member	Carnegie Mellon University
Dr. Kyunghyun Cho	Member	New York University

Abstract

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

Our primary goal is to make these algorithms more data-efficient using techniques related to imitation learning. Imitation learning is a technique for using expert feedback in sequential decision and prediction-making problems. Naive imitation learning has a covariate shift problem (i.e. training distribution is different than test distribution). We propose methods and ideas to address this issue and address other issues that arise in other styles of imitation learning. In particular, we study two types of imitation learning style feedbacks, 'interactive' feedback and 'demonstration' feedback. We propose two ideas to help move towards learning in real-world settings.

For my first proposed work, I plan to study Imitation learning problems in the context of multi-player games. Modern multi-player games are a setting for studying imitation learning because of the amount of potential demonstration data available. Reinforcement learning techniques are hard to run in these settings because the reward is often sparse and the state space is usually large. Imitation learning techniques can help, but simply running imitation learning techniques out of the box could exaggerated issues already present. In this work, we use the fact that we are playing a game and that the expert provides good state coverage in the 'demonstration' feedback setting — and propose a new technique to learn from the 'demonstration' feedback.

Extending beyond multi-player games, we propose to benchmark the current state of imitation learning algorithms. To understand how much progress we have made over the years, we plan to standardize comparing imitation learning algorithms with each other. At its simplest form imitation learning is supervised learning and a lot of progress has been made from benchmarks in supervised learning. We go a step further by introducing more standard 'baselines' besides behavior cloning and standardize ways of training behavior cloning. This benchmark will provide us with an understanding of how recent ideas hold against various situations, giving us an indicator of how practical and feasible it would be to run these algorithms in real-world settings.

Contents

1 Introduction

1.1 Motivation

As the impact of machine learning on all aspects of our lives continues to grow, having systems that can learn through interaction with users and the world becomes more and more pressing. Unfortunately, today's approaches to learning through interaction–typically driven by reinforcement learning algorithms–require a very large number of interactions, which is feasible only for systems we can *fully* simulate or for very narrow applications like ad placement or news recommendations because a lot of feedback is available (i.e. billions of ads are displayed every day). But for many real-world systems, simulators do not already exist, 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 so that they can learn in a reasonable amount of time.

1.2 Research questions

The first attempt to develop agents to interact with real systems using expert feedback was Alvinn: (An autonomous land vehicle in a neural network) [39], which occurred 3 decades ago. Instead of training an autonomous using a reward signal (i.e. with reinforcement learning), Alvinn was trained using imitation learning with pre-collected data from an expert, denoted 'demonstration feedback' to reduce sample complexity. But the Alvinn experiment introduced a fundamental issue in imitation learning called the 'covariate shift' [81]. Essentially, if the set of states (i.e. roads maps in the Alvinn example) from the expert demonstrations are different than the set of states the agent sees on the road, then the agent will perform poorly because it was not trained on these states. There have been several proposed ideas but they all come with their set of trade-offs. For example, 'interactive' feedback assumes you query an expert at any time instead of learning from pre-collected demonstrations in 'demonstration' feedback. But assuming you have an expert to query at any time is a strong assumption. 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. Unfortunately, providing demonstrations is often both too much (requiring too much effort on the part of the expert) and too little (providing an insufficient signal to learn). In prior work, the most prevalent framework in which to leverage an expert is to ask the expert to demonstrate the desired behavior and then learn to mimic them: the *imitation learning* paradigm. My work builds new methods that improve on and go beyond *traditional* imitation learning. My research question is: How do we use expert feedback to develop algorithms for real-world systems?

1.3 Proposed solutions

In order move to closer to developing learning algorithms that potentially work in real-world systems we propose two ideas. Games provide a complex real-world environment where interactive learning is possible because the cost of making a mistake controlling a player interacting in a game compared to controlling a car interacting in real life (i.e. self-driving car) is much lower. The state-space in modern high-budget 'AAA' video games is huge which means that developing techniques with low sample complexity is important. We propose a technique that uses expert feedback for exploration instead of mimicking like ‘tradition’ imitation learning. The second idea I propose to move the field of imitation learning closer to real-world systems is to develop a benchmark to rigorously compare all the most recent methods in imitation learning. At the moment even though imitation learning is similar in spirit to supervised learning, there are no standard benchmarks for testing algorithms to measure progress.

1.4 Organization

Chapter 2 presents an overview of fundamental frameworks that my work builds on. Chapter 3 focuses on a setting where using interactive feedback works well. Chapter 4 reduces the amount of interactive feedback needed. Chapter 5 completely removes interactive feedback but instead uses demonstration feedback and maintains good performance in theory and practice. Chapter 6 uses demonstration feedback to learn rich embedding (i.e. representations) for planning. Chapter 7 describe my two proposed work extending my preliminary work in learning with expert feedback. Chapter 8 concludes my proposal and describes a timeline for my remaining proposed work.

2 Background

This chapter introduces the fundamental Markov Decision Process (MDP) which is a model that allows agents proposed in this thesis to learn through interaction. We introduce 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* $T_a \in \mathbb{R}^{k \cdot k}$ 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 do a to indicate that choosing an action is an *intervention*.

2.2 Partially Observable Markov Decision Process

In an MDP, the agent gets to know 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* $D \in \mathbb{R}^{O \cdot k}$ 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.

Just 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

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

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 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) [61] 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 require 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. Whereas a PSR uses the sequence of actions and observations of the agent to make predictions about future observations. PSRs representation can potentially be more compact than POMDPs, which is important for planning in large domains [96].

In more detail, we are given a starting state vector q_1, matrices $T_{ao} \in \mathbb{R}^{k \cdot k}$, 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,

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

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* [10, 32] as:

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

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 allow us to learn representation of the 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. But most sequential (structured) prediction problems can be framed in terms of MDPs as well [28]. The "learning to search" approach to structured prediction casts the joint prediction problem of producing complex output as a sequence of smaller predictions [26, 30, 77]. Meaning that MDPs can be viewed as a framework for modeling sequential decisions and predictions of agents in an environment. For sequential structured predictions, 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 a tuple* $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C}, \rho_0, \mathcal{T})$ *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* $a \in \mathcal{A}$*;* $\mathcal{C} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *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* $\rho_0 : \mathcal{S} \to [0,1]$ *is the distribution of the initial state.*

We consider episodic finite horizon MDP in this work. Let Π the class of policies the learner is considering and $\pi^\star$ the expert policy whose behavior the learner is trying to mimic. For any policy π, let d_π denote the distribution over states induced by following π. We assume the $C(s,a)$ is bounded in $[0,1]$. In the imitation learning setting, we do not necessarily know the true costs $C(s,a)$, and instead, we observe expert demonstrations. Our goal is to find a policy π which minimizes an observed surrogate loss ℓ between its actions and the actions of the expert under its induced distribution of states, i.e.

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

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

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

2.4.1 Behavior Cloning

Unfortunately, it is often not possible to optimize J_{exp} directly, since it requires evaluating the expert policy on the states induced by following the *current* policy. The supervised behavioral cloning cost J_{BC}, which is computed on states induced by the expert, is often used instead:

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

Minimizing this loss within ϵ yields a quadratic regret bound on regret:

Theorem 2.4.1. *[81] Let* $J_{\text{BC}}(\pi) = \epsilon$*, then* $J_{\text{C}}(\pi) \leq J_{\text{C}}(\pi^\star) + T^2\epsilon$*.*

The drawback of the supervised learning approach (i.e. behavior cloning) is that it does not take into account that the learner π and expert π^* state-distribution are different leading to the covariate shift problem. Essentially, there is a mismatch between the distribution of states trained on and tested on. In particular, during training, the learner π is trained on state-action pairs from the expert policy π^*. 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 π^* on its state-distribution, so that the train and test time mismatch problem is resolved. The key idea of DAgger is to iterative learner a policy by teaching the learner the optimal action to take from its own state-distribution. The tradeoff is that we are 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 π to learn from mistakes on its state-distribution because it knows what the expert would do unlike in behavior cloning.

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

```
1:  initialize dataset $D = \{\}$
2:  initialize policy $\hat{\pi}_1$ to any policy in $\Pi$
3:  for $i = 1 \dots N$ do
4:      ▷ stochastic mixture policy
5:      Let $\pi_i = \beta_i \pi^\star + (1-\beta_i)\hat{\pi}_i$
6:      Generate a $T$-step trajectory using $\pi_i$
7:      Accumulate data $D \leftarrow D \cup \{(s, \pi^\star(s))\}$ 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 π, which is defined as $Q_t^\pi(s,a) = \mathcal{C}(s,a) + \gamma \mathbb{E}_{s\ \mathcal{P}(\cdot|s,a)}\left[\mathcal{V}_{t-1}^\pi(s')\right]$ where $\mathcal{V}_t^\pi(s)$ represented the cost of executing π. The following result for DAgger shows that if ℓ is an upper bound on the $0-1$ loss and C satisfies certain smoothness conditions, then minimizing this loss within ϵ translates into an $\mathcal{O}(\epsilon T)$ regret bound on the true task cost $J_{\mathrm{C}}(\pi) = \mathbb{E}_{s,a\sim d_\pi}[C(s,a)]$:

Theorem 2.4.2. *[85] Let π be such that $J_{\exp}(\pi) = \epsilon$, and $Q_{T-t+1}^{\pi^\star}(s,a) - Q_{T-t+1}^{\pi^\star}(s,\pi^\star) \leq u$ for all $a \in \mathcal{A}, t \in \{1,2,...,T\}, d_\pi^t(s) > 0$. Then $J_{\mathrm{C}}(\pi) \leq J_{\mathrm{C}}(\pi^\star) + uT\epsilon$.*

The theorem means if the supervised learning loss is ϵ, then loss with respect to the expert is $\mathcal{O}(u\mathcal{T}\epsilon)$, 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 π is to be trained. In most formal analyses, the roll-in policy is a stochastic mixture of the learned policy π and the oracle policy π^*, ensuring that π is eventually trained on its state distribution [23, 31, 82, 85]. *Experimentally*, it has been found that using the oracle's state distribution is optimal [56, 76].

2.5 Active Learning

Active learning is way of reducing expert feedback in both interactive and demonstration feedback learning [5, 78]. At each round, the learning algorithm is presented an example $\boldsymbol{x}$ 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. [22] 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.

3 Text Generation using Interactive Feedback

DAgger (see background section) assumes that an agent can query an expert for every state that it visits to ask for the optimal action. This assimption is unrealistic in most setting except for when the expert is a dynamic oracle. The dynamic oracle can learn the near-(optimal) as the expensive of e.g. expanding a game-tree or search-tree. Most sequence-generation models, from n-grams [9] to neural language models [14] generate sequences in a purely left-to-right, monotonic order. This raises the question of whether alternative, non-monotonic orders are worth considering [35], especially given the success of "easy first" techniques in natural language tagging [102], parsing [38], and coreference [98], which allow a model to effectively learn their ordering. In investigating this question, we are solely interested in considering non-monotonic generation that does not rely on external supervision.

We introduce a framework for training sequential text generation models which learn a generation order without having to specifying an order in advance [105]. An example generation from our model is shown in Figure 3.1. We frame the learning problem as an *imitation learning* problem using DAgger [85], in which we aim to learn a generation policy that mimics the actions of an oracle generation policy.

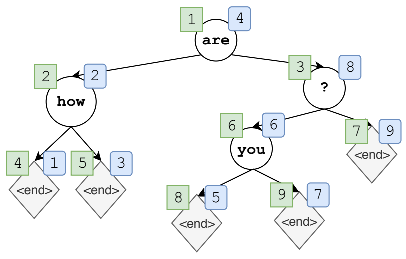


Figure 3.1: A sequence, "how are you ?", generated by the proposed approach trained on utterances from a dialogue dataset. The model first generated the word "are" and then recursively generated left and right subtrees ("how" and "you ?", respectively) of this word. At each production step, the model 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 (left) denote the order in which the nodes were generated (level-order); those in rounded blue squares (right) denote the nodes' location in the final sequence (in-order).

3.1 Non-Monotonic Generation

We consider the problem of sequentially generating a sequence of discrete tokens $Y = (w_1, \ldots, w_N)$, 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 3.1.

We view the generation process as deterministically navigating a state space $\mathcal{S} = \tilde{V}^\star$ here 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 ⟨end⟩ terminates a subtree. The initial state s_0 is the empty sequence. For example, in Figure 3.1, $s_1 = \langle\text{are}\rangle$, $s_2 = \langle\text{are}, \text{how}\rangle$, ..., $s_4 = \langle\text{are}, \text{how}, ?\ , \langle\text{end}\rangle\rangle$. 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 ⟨end⟩'ed. If a terminal state s_T is reached, we have that $T = 2N + 1$, where N is the number of words (non-⟨end⟩ 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 $\langle a_{\tau(1)}, \ldots, a_{\tau(T)} \rangle$ 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 3.1, τ maps from the numbers in the blue squares to those in the green squares.

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).

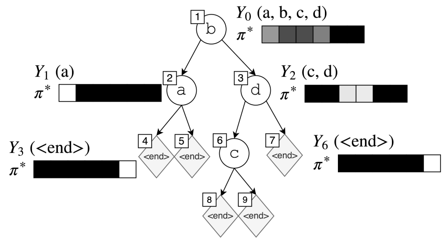


Figure 3.2: A sampled tree for the sentence "a b c d" with an action space $\tilde{V}$ = (a,b,c,d,e, $\langle \text{end} \rangle$), showing an oracle's distribution π^* and consecutive subsequences ("valid actions") Y_t for $t \in \{0, 1, 2, 3, 6\}$. Each oracle distribution is depicted as 6 boxes showing $\pi^*(a_{t+1}|s_t)$ (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, π^* only assigns positive probability to tokens in Y_t

3.1.1 Learning Non-Monotonic Generation

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 3.2 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 [42] and related techniques [24, 43].

3.1.2 Oracle Policies

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. An oracle policy $\pi^*(\cdot|s_t)$ is defined with respect to Y_t, a consecutive subsequence of Y. At $s_0 = \langle\rangle$, π^* 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 3.2, 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 $Y_t = (w'_1, \ldots, w'_{N'})$, an oracle policy is defined as:

$$\pi^*(a|s_t) = \begin{cases} 1 & \text{if } a = \langle \text{end} \rangle \text{ and } Y_t = \langle\rangle \\ p_a & \text{if } a \in Y_t \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where the p_as 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. There are many possible oracle policies, and each of them is characterized by how p_a in Eq. Eq. (3.1) is defined. Specifically, we propose three variants.

Uniform Oracle. 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, π^*_{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 ([29, 106]).

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. [43]. Motivated by this, we design a coaching oracle as the product of the uniform oracle and current policy π: $\pi^*_{\text{coaching}}(a|s) \propto \pi^*_{\text{uniform}}(a|s)\ \pi(a|s)$. This coaching oracle prefers actions of 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: $\pi^*_{\text{annealed}}(a|s) = \beta\pi^*_{\text{uniform}}(a|s) + (1-\beta)\pi^*_{\text{coaching}}(a|s)$

Deterministic Left-to-Right Oracle. 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.

	Test		
Oracle	**Bleu**	**F1**	**EM**
left-right	46.3	0.903	0.208
uniform	44.3	0.960	0.197
annealed	46.0	0.950	0.212

Table 3.1: Word Reordering results on Persona-Chat, reported according to Bleu score, F1 score, and percent exact match on validation and test data.

3.2 Experiments

In this section we experiment with our non-monotone sequence generation model across two tasks (see [105] for more experiments and details) .

3.2.1 Word Reordering

We first evaluate the proposed models for conditional generation on the Word Reordering task, also known as Bag Translation [20] or Linearization [87]. In this task, a sentence $Y = (w_1, ..., w_N)$ is given as an unordered collection $X = \{w_1, ..., w_N\}$, and the task is to reconstruct Y from X. We assemble a dataset of (X, Y) pairs using sentences Y from the Persona-Chat [111] dialogue dataset, which consists of multi-turn dialogues between two agents. In our approach, we do not explicitly force the policies trained with our non-monotonic oracles to produce a permutation of the input and instead let them learn this automatically.

Model. For encoding each unordered input $x = \{w_1, ..., w_N\}$, we use a simple bag-of-words encoder: $f^{\text{enc}}(\{w_1, ..., w_N\}) = \frac{1}{T}\sum_{i=1}^{N} \text{emb}(w_i)$. We implement $\text{emb}(w_i)$ using an embedding layer followed by a linear transformation. The embedding layer is initialized with GloVe [72] vectors and updated during training. As the policy we use a flat LSTM with 2 layers of 1024 LSTM units.

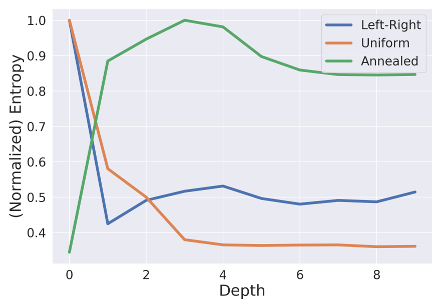


Figure 3.3: Normalized entropy of $\pi(\cdot|s)$ as a function of tree depth for policies trained with each of the oracles.

Results. Table 3.1 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.

Easy-First Analysis. Figure 3.3 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 **??**) and defers harder decisions until later.

3.2.2 Machine Translation

Data and Preprocessing. We evaluate the proposed models on IWSLT'16 German $\rightarrow$ English (196k pairs) translation task. The data sets consist of TED talks. We use the TED tst2013 as a validation dataset and tst-2014 as test. We use the default Moses tokenizer script [52] and segment each word into a subword using BPE [91] creating 40k tokens for both source and target.

	Test				
Oracle	**Bleu**	(BP)	**Meteor**	**YiSi**	**Ribes**
left-right	28.00	(1.00)	30.10	65.22	82.29
uniform	21.40	(0.86)	26.40	62.41	80.00
annealed	23.30	(0.91)	27.96	63.38	80.91
+⟨end⟩-tuning	24.60	(1.00)	29.30	64.18	80.53

Table 3.2: Results of machine translation experiments for different training oracles across four different evaluation metrics.

Model & Training. We use a Transformer policy, following the architecture of [103]. We use auxiliary ⟨end⟩ prediction by introducing an additional output head, after observing a low brevity penalty in preliminary experiments.

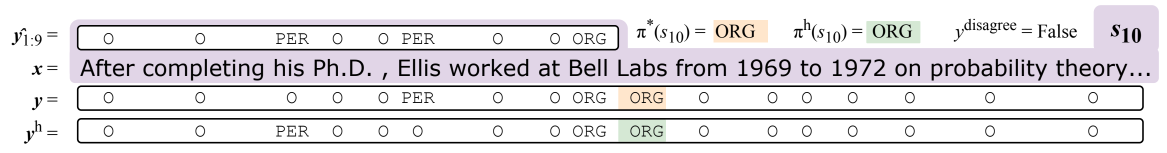
Results. Results on validation and test data are in Table 3.2 according to four (very) different evaluation measures: Bleu, Meteor [55], YiSi [62], and Ribes [49]. First focusing on the non-monotonic models, we see that the annealed policy outperforms the uniform policy on all metrics, with tree-encoding yielding further gains. Adding ⟨end⟩ tuning to the tree-encoding model decreases the Ribes score but improves the other metrics, notably increasing the Bleu brevity penalty.

The left-to-right model has superior performance according to Bleu As previously observed [21, 107], 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 Reducing Interactive Feedback

[1] DAgger address the issues with in behavior cloning when doing imitation learning (see background section). But it does this with a trader off of assuming that we query an expert at every state (which is not sample efficient to the expert). 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, a natural starting point for this is imitation learning-based "learning to search" approaches to structured prediction [13, 30, 57, 84]. 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 introduce Learning to Query for Imitation (LeaQI, /ˈliːˌtʃiː/) to reduce annotation cost by only asking an expert for labels that are truly needed. LeaQI assumes access to a *noisy heuristic* labeling function (for instance, a rule-based model) that can provide low-quality labels. It trains, simultaneously, a *difference classifier* [110] that predicts disagreements between the expert and the heuristic (see Figure 4.1). 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.



$\hat{y}_{1:9}$ = O O PER O O PER O O ORG
$\pi^*(s_{10})$ = ORG
$\pi^h(s_{10})$ = ORG
$y^{disagree}$ = False
s_{10}
x = After completing his Ph.D. , Ellis worked at Bell Labs from 1969 to 1972 on probability theory...
y = O O O O O PER O O ORG ORG O O O O O O O
y^h = O O PER O O O O O ORG ORG O O O O O O O

Figure 4.1: A named entity recognition example (from the Wikipedia page for Clarence Ellis). $\boldsymbol{x}$ is the input sentence and $\boldsymbol{y}$ is the (unobserved) ground truth. The predictor π 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 $\boldsymbol{x}$ with $\hat{\boldsymbol{y}}_{1:9}$. The heuristic makes two errors at $t = 4$ and $t = 6$. The heuristic label at $t = 10$ is y^h_{10} =`ORG`. Under Hamming loss, the cost at $t = 10$ is minimized for $a =$ `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.

4.1 LeaQI

As a concrete example, return to Figure 4.1: at s_{10}, π must predict the label of the tenth word. If π is confident in its own prediction, LeaQI can avoid any query, similar to traditional active learning. If π 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.

4.1.1 Learning to Query for Imitation

Our algorithm, LeaQI, is specified in Algorithm 2. As input, LeaQI takes a policy class Π, a hypothesis class $\mathcal{H}$ for the difference classifier (assumed to be symmetric and to contain the

[1]Paper published at ICML 2019 [105]

"constant one" function), a number of episodes N, an expert policy $\pi^\star$, a heuristic policy π^{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 $\pi^\star$ is only queried if it is predicted to disagree with the heuristic π^{h} at s by the difference classifier, or if apple tasting method switches the difference classifier label (see subsection 4.1.2).

Algorithm 2 LeaQI$(\Pi, \mathcal{H}, N, \pi^\star, \pi^{\mathrm{h}}, b)$

```
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 $\boldsymbol{x}$
7:      ▷ generate a $T$-step trajectory using $\pi_i$
8:      Generate output $\hat{\boldsymbol{y}}$ using $\pi_i$
9:      for each $s$ in $\hat{\boldsymbol{y}}$ do
10:         ▷ draw bernouilli random variable
11:         $Z_i \sim \text{Bern}\left(\frac{b}{b+\text{certainty}(\pi_i,s)}\right)$; see ??
12:         if $Z_i = 1$ then
13:             ▷ set difference classifier prediction
14:             $\hat{d}_i = h_i(s)$
15:             if AppleTaste($s$, $\pi^{\mathrm{h}}(s)$, $\hat{d}_i$) then
16:                 ▷ predict agree query heuristic
17:                 $D \leftarrow D \cup \{\ (s, \pi^{\mathrm{h}}(s))\ \}$
18:             else
19:                 ▷ predict disagree query expert
20:                 $D \leftarrow D \cup \{\ (s, \pi^\star(s))\ \}$
21:                 $d_i = \mathbb{1}[\pi^\star(s) = \pi^{\mathrm{h}}(s)]$
22:             end if
23:         end if
24:     end for
25:     Train policy $\pi_{i+1} \in \Pi$ on $D$
26:     Train difference classifier $h_{i+1} \in \mathcal{H}$ on $S$
27: end for
28: return best (or random) $\pi_i$
```

In particular, at each state visited by π_i, LeaQI estimates z, the certainty of π_i's prediction at that state. A sampling probability ρ is set to $b/(b+z)$ where z is the certainty, and so if the model is very uncertain then ρ tends to zero, following [22]. With probability ρ, 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 $\pi^\star$ and π^{h} are likely to disagree on the correct action. The difference classifier's prediction, $\hat{d}_i$, is passed to an *apple tasting* method.Intuitively, most apple tasting procedures (including the one we use, STAP) return $\hat{d}_i$, unless the difference classifier is making many Type II errors, in which case it may return $\neg\hat{d}_i$.

4.1.2 Apple Tasting for One-Sided Learning

The difference classifier $h \in \mathcal{H}$ must be trained 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 [44]'s algorithms, STAP, which works by random sampling from apples that are predicted to not be tasted and tasting them anyway (line ??). 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 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).

4.2 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 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: 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.

Algorithm 3 AppleTaste_STAP($S, a_i^{\mathrm{h}}, \hat{d}_i$)

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

4.2.1 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.

4.2.2 Expert Policy and Heuristics

In all experiments, the expert $\pi^\star$ is a simulated human annotator who annotates one word at a time. We take the annotation cost to be the total number of words labeled.

NER: The heuristic we implement for named entity recognition is a high-precision gazeteer-based string matching approach. This heuristic achieves a precision of 0.88, recall of 0.27 and F-score of 0.41 on the training data.

Keyphrase: The keyphrase extraction heuristic is the output of an "unsupervised keyphrase extraction" approach [34]. This heuristic achieves a precision of 0.20, recall of 0.44 and F-score of 0.27 on the training data.

POS: The part of speech tagging heuristic is based on a small dictionary compiled from Wiktionary. Following Haghighi and Klein [40] and Zesch et al. [108], we extract this dictionary

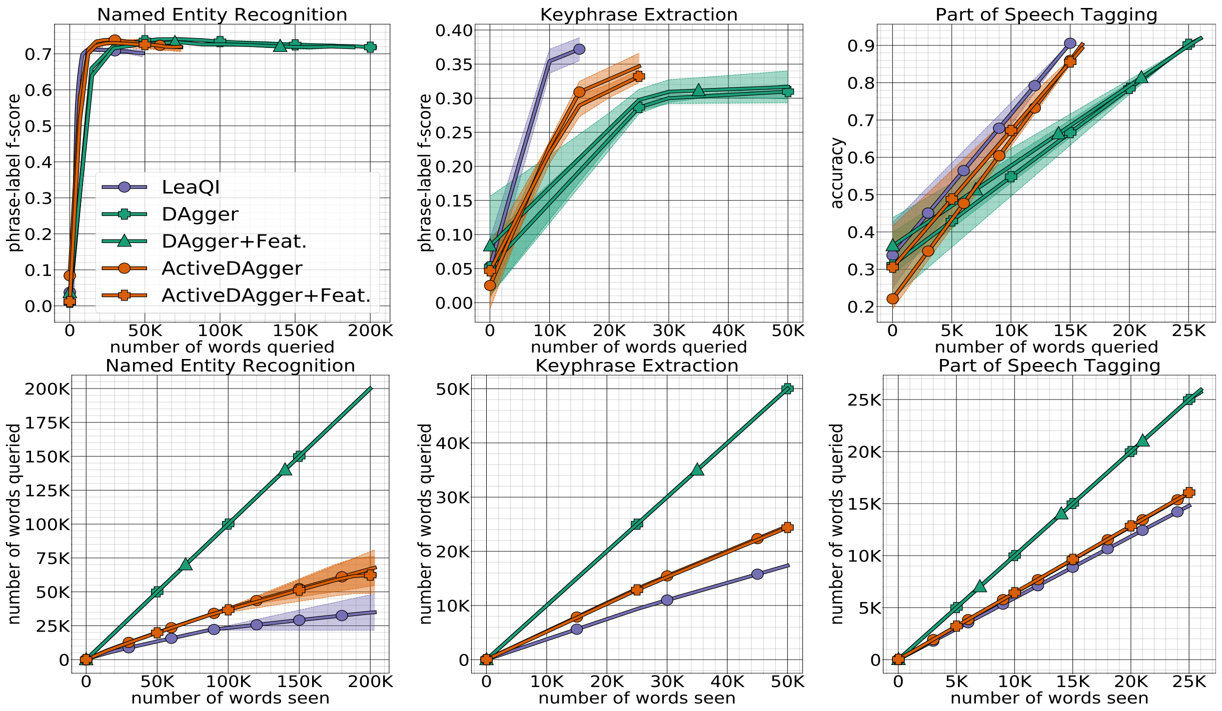


Figure 4.2: The top rows shows performance (f-score or accuracy) with respect to the number of queries to the expert. The bottom row shows the number of queries as a function of the number of words seen.

using Wiktionary as follows: for word w in our training data. If word w is not in Wiktionary, we use a default label of "X". The label "X" is chosen 90% of the time. For the remaining 10%, the heuristic achieves an accuracy of 0.67 on the training data.

4.2.3 Experimental Results

In Figure 4.2, 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). 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 (Q2). 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.

5 Removing Interactive Feedback

5.1 DRIL

[1] 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.

Our theoretical results show that, subject to realizability and optimization oracle assumptions, our algorithm obtains a $\mathcal{O}(\epsilon\kappa T)$ regret bound, where κ is a measure which quantifies a tradeoff between the concentration of the demonstration data and the diversity of the ensemble outside the demonstration data.

5.1.1 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 $\Pi_E = \{\pi_1, ..., \pi_E\}$ of policies trained on the demonstration data $\mathcal{D}$. We call this the uncertainty cost, which is defined as:

$$C_{\mathrm{U}}(s,a) = \mathrm{Var}_{\pi\sim\Pi_{\mathrm{E}}}(\pi(a|s)) = \sum_{i=1}^{|E|}\Big(\pi_i(a|s) - \sum_{i=1}^{|E|}\pi_i(a|s)/|E|\Big)^2 \tag{5.1}$$

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 $\pi^\star$ to do as well. The total cost which the algorithm optimizes is given by:

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

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. The full algorithm is shown in Algorithm 4.

[1]Paper published at ICLR 2019 [19]

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 $i = 1, ...$ 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 $\mathbb{E}_{s\sim d_\pi, a\sim\pi(\cdot|s)}[C_{\text{U}}(s,a)]$.
10: end for
```

5.1.2 Coverage Coefficient

We will show that, subject to assumptions that the policy class contains an optimal policy and that we are able to optimize costs within ϵ of their global minimum, our algorithm obtains a regret bound which is linear in κT, where κ is a quantity which depends on the environment dynamics, $d_\pi^\star$ and our learned ensemble.

The motivation behind our algorithm is that the policies in the ensemble agree inside the expert's distribution and disagree outside of it. 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 $\alpha(\mathcal{U}) = \max_{\pi\in\Pi} \sup_{s\in\mathcal{U}} \frac{d_\pi(s)}{d_{\pi^\star}(s)}$.*

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.

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

Definition 4. *We define $\kappa = \min_{\mathcal{U}\subseteq\mathcal{S}} \frac{\alpha(\mathcal{U})}{\beta(\mathcal{U})}$.*

We can view κ 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.

5.1.3 Regret Bound

We now establish a relationship between the κ coefficient just defined, the cost our algorithm optimizes, and J_{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 Appendix [19]

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

If κ 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.

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

Here ϵ is the threshold specified in Assumption 2. Combining these two lemmas with the previous result of [85], we get a regret bound which is linear in κT.

Theorem 5.1.1. *Let $\hat{\pi}$ be the result of minimizing J_{alg} using our optimization oracle, and assume that $Q^{\pi^\star}_{T-t+1}(s,a) - Q^{\pi^\star}_{T-t+1}(s,\pi^\star) \leq u$ for all $a \in \mathcal{A}, t \in \{1,2,...,T\}, d^t_\pi(s) > 0$. Then $\hat{\pi}$ satisfies $J_{\text{C}}(\hat{\pi}) \leq J_{\text{C}}(\pi^\star) + 5u\kappa\epsilon T$.*

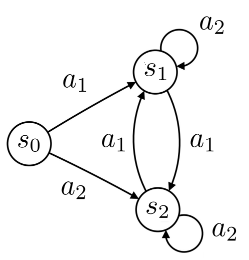


Figure 5.1: Example of a problem where behavioral cloning incurs quadratic regret.

Our bound is an improvement over that of behavior cloning if κ is less than $\mathcal{O}(T)$. Note that DRIL does not require knowledge of κ. The quantity κ is problem-dependent and depends on the environment dynamics, the expert policy and the class of policies available to the learner.

Example 1. *Consider the tabular MDP given in [81] 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 π can be represented as a set of probabilities $\pi(a_1|s)$ for each state $s \in \mathcal{S}$ [2]. 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 $Beta(n_1+1, n_2+1)$ 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 $\pi^\star(a_1|s_0) = 1, \pi^\star(a_1|s_1) = 0, \pi^\star(a_1|s_2) = 1$. Here $d^\star_\pi = (\frac{1}{T}, \frac{T-1}{T}, 0)$. For any π, $d_\pi(s_0) = \frac{1}{T}$ and $d_\pi(s_1) \leq \frac{T-1}{T}$ due to the dynamics of the MDP, so $\frac{d_\pi(s)}{d^\star_\pi(s)} \leq 1$ for $s \in \{s_0, s_1\}$. Writing out $\alpha(\{s_0,s_1\})$, we get: $\alpha(\{s_0,s_1\}) = \max_{\pi\in\Pi} \sup_{s\in\{s_0,s_1\}} \frac{d_\pi(s)}{d^\star_\pi(s)} \leq 1$.*

Furthermore, since s_2 is never visited in the demonstration data, for each policy π_i in the ensemble we have $\pi_i(a_1|s_2), \pi_i(a_2|s_2) \sim Beta(1,1) = Uniform(0,1)$. It follows that $\text{Var}_{\pi\sim\Pi_{\text{E}}}(\pi(a|s_2))$ is approximately equal [3] 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 5.1.1, 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[4].

5.2 Experiments

5.2.1 Atari Environments

We next evaluated our approach on six different Atari environments. We used pretrained PPO [89] agents from the stable baselines repository [45] to generate $N = \{1,3,5,10,15,20\}$ expert

[2] Note that $\pi(a_2|s) = 1 - \pi(a_1|s)$.

[3] Via Hoeffding's inequality, with probability $1-\delta$ the two differ by at most $\mathcal{O}(\sqrt{\log(1/\delta)/|\Pi_{\text{E}}|})$.

[4] 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 ϵ but a regret of ϵT^2.

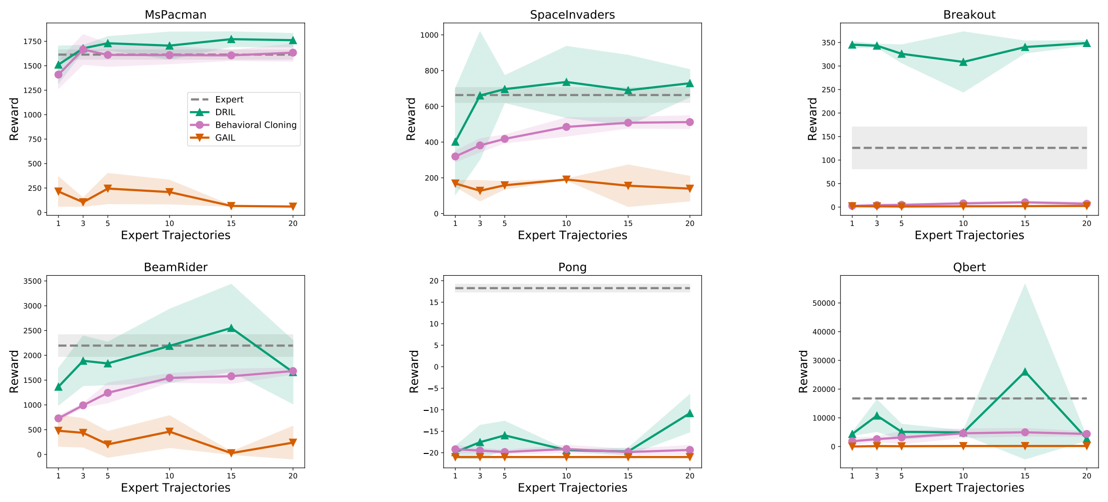


Figure 5.2: Results on Atari environments.

trajectories. We compared against two other methods: standard behavioral cloning (BC) and Generative Adversarial Imitation Learning (GAIL). –Results are shown in Figure 5.2. 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.

5.2.2 Continuous Control

We next report results of running our method on a 6 different continuous control tasks. Results are shown in Figure 5.3. 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.

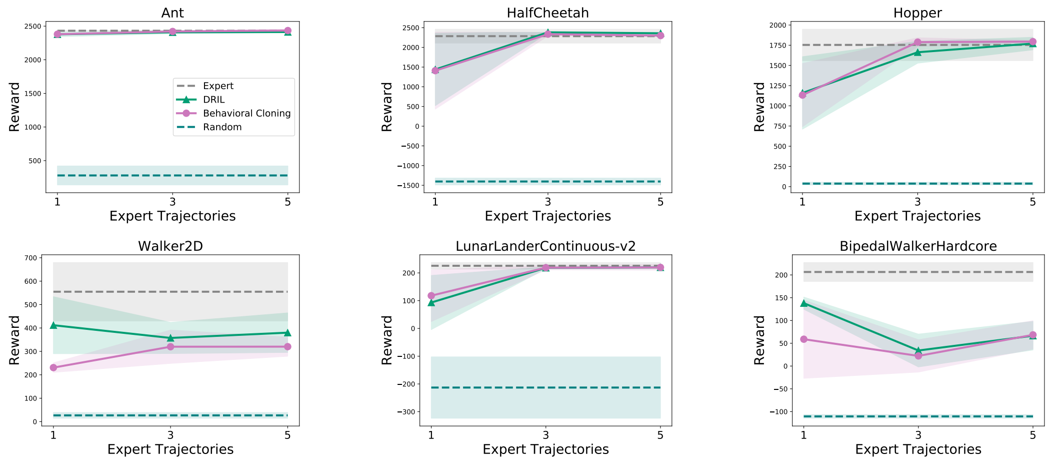


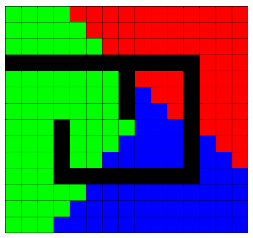
Figure 5.3: Results on continuous control tasks.

6 Demonstration Feedback for Feature Matching

[1] 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 π, and describes how the features we will observe in the future depend on the current state under π. 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.

6.1 Imitation by Feature Matching


(a) f of maze MDP.

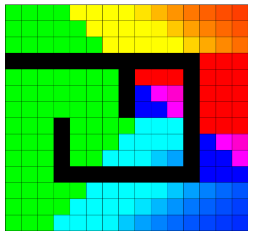

(b) $\phi^{\text{go-left}}$ with $\gamma = 0.75$.

Figure 6.1: Maze environment example.

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* [3] 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 $s_1, a_1, s_2, a_2, s_3, a_3, \ldots$, then the empirical discounted feature vector is

$$f(s_1, a_1) + \gamma f(s_2, a_2) + \gamma^2 f(s_3, a_3) + \ldots$$

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 ϕ^d.

Analogously, for a policy π, 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 ϕ^d and ϕ^π; in this case we have the option of setting $\gamma = 1$.

[1]Paper published at AAAI 2021 [17]

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 π with

$$\phi^\pi = \phi^d$$

For example, suppose our world is a simple maze MDP like Fig. 6.1a. 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.

6.1.1 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, while successor-style representations are often used in model-free RL. However, as Lehnert and Littman [58] 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 $F_a \in \mathbb{R}^{d \cdot k}$ for each action a. In either case, define the successor features to be

$$\phi^\pi(q) = \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} f(q_t, a_t) \,\middle|\, \text{do } q_1 = q \right]$$

Interestingly, the function ϕ^π is *linear* in q. That is, for each π, there exists a matrix $A^\pi \in \mathbb{R}^{d \cdot k}$ such that $\phi^\pi(q) = A^\pi q$. We call A^π the *successor feature matrix* for π; it is related to the parameters of the *Linear Successor Feature Model* of Lehnert and Littman [58].

We can compute A^π recursively by working backward in time (upward from the leaves of a policy tree): for a tree with root action a, the recursion is

$$A^\pi = F_a + \gamma \sum_o A^{\pi(o)} T_{ao}$$

6.2 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 Φ, which is the limit of $\Phi^{(H)}$ as $H \to \infty$.

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 Φq tells us about one policy, and gives us moments of the distribution of future states under that policy. The extreme points of Φq 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. 6.2.1). That way, we can reason about all possible policies instead of just a fixed list. Another benefit of Φ 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 Φ is a complicated object; it can easily become impractical to work with.

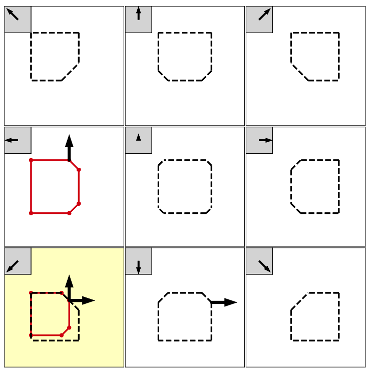


Figure 6.2: Visualization of the successor feature set Φ for a $3 \cdot 3$ gridworld MDP with 2d features. Start state is in yellow. Gray insets show one-step feature vectors, which depend only on the state, not the action. Each subplot shows one projection Φe_j (scale is arbitrary, so no axes are necessary). 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.

6.2.1 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.

The update from $\Phi^{(H-1)}$ to $\Phi^{(H)}$ is a contraction: see the supplementary material online for a proof. So, as $H \to \infty$, $\Phi^{(H)}$ will approach a limit Φ; this set represents the achievable successor feature matrices in the infinite-horizon discounted setting. Φ is a fixed point of the Bellman backup, and therefore satisfies the *stationary* Bellman equations

$$\Phi = \text{conv} \bigcup_a \left[F_a + \gamma \sum_o \Phi T_{ao} \right]$$

6.2.2 Feature Matching and Optimal Planning

Once we have computed the successor feature set, we can return to the feature matching task described in Section 6.1. Knowing Φ makes feature matching easier: for any target vector of discounted feature expectations ϕ^d, we can efficiently either compute a policy that matches ϕ^d or verify that matching ϕ^d is impossible. We detail an algorithm for doing so in Alg. 5.

Fig. 6.2 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: repeat
5:     Choose actions $a_{it}$, vectors $\phi_{it} \in \Phi_{a_{it}} q_t$,
6:     and convex combination weights $p_{it}$ s. t. $\phi_t^d = \sum_{i=1}^{\ell} p_{it} \phi_{it}$.
7:     Choose an index $i$ according to probabilities $p_{it}$, and execute the corresponding action:
       $a_t \leftarrow a_{it}$.
8:     Write the corresponding $\phi_{it}$ as
9:     $\phi_{it} = F_{a_t} q_t + \gamma \sum_o \phi_{ot}$ by choosing $\phi_{ot} \in \Phi T_{a_t o} q_t$ for each $o$.
10:    Receive observation $o_t$, and calculate $p_t = P(o_t \mid q_t, a_t) = u^T T_{a_t o_t} q_t$.
11:    $q_{t+1} \leftarrow T_{a_t o_t} q_t / p_t$
12:    $\phi_{t+1}^d \leftarrow \phi_{o_t t} / p_t$
13:    $t \leftarrow t + 1$
14: until done
```

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:

$$V^*(q) = \max_\pi r^T \phi^\pi q = \max \{ r^T \psi q \mid \psi \in \Phi \}$$

7 Proposed Work

7.1 Multi-Agent Imitation Learning in Games

Most existing successes in multi-agent scenarios revolve around games such as [71] and [94]. Learning multi-agent games such as 'go' from scratch [94] outperforms warm starting the learning process using demonstration data [93]. In sparse reward settings agents no longer have immediate feedback for evaluating actions taken and suffer from the temporal credit assignment problem (i.e. CAP) [67]. In settings where the reward function is sparse training multi-agents from scratch becomes difficult because of exploration.

Multi-Agent RL Framework

Multi-agent RL solves sequential decision-making problems with more than one agent. In a multi-agent setting, the system state, transition, and reward function are affected by the joint actions of all agents. Each agent has its long-term reward to optimize, which is a function of all other agents' behaviors.

7.1.1 Extra-NRPI

Two methods that were proposed to address sparse reward issues in the past are potential base shaping [70] and reward hacking [75] which both suffer from issues around the difficulty of modifying the reward function. The most successful multi-agent reward shaping technique was seen in [71] where they use human preference for shaping; this shaping technique does not generalize to a new domain. I propose a novel imitation learning algorithm called, 'Extra-NRPI' that addresses two challenges in sparse reward multi-agent game settings: first is how to train multi-agent policies using gradient techniques, and second how to use the demonstration beyond mimicking the expert.

Imitation learning in a multi-agent setting is more complex than in a single-agent setting. Multi-agent environments require collecting demonstrations from all agents acting in the environment, making the error problems discussed in the background section for behavior cloning more severe. The DAgger issue sample inefficiencies around querying the expert at every state are amplified because you need to query an expert at after state for every learner. Given these drawbacks of current methods in single-agent, we instead use the demonstration as a start stat distribution (i.e. replay distribution); where instead of mimicking the expert action because we have sparse reward function, we just need to optimize the sparse reward from the states the expert visits.

Extragradient No-Regret Policy Iteration

I propose a multi-agent policy search algorithm this motivated by two ideas: i) we use the expert demonstration as start state distribution ν (i.e similar to [7]) and ii) we formulate multi-agent reinforcement in games using game theory [2]. This formulation allows us to address policy gradient issues in multi-agent games as a variational inequality problem (VI) [41]. Concretely, we plan to use the Approximate Policy Iteration algorithm No-Regret Policy Iteration (NRPI) [83] (extending NRPI from single-player to multi-player) and extragradient using extrapolation, which is a popular technique used to solve VIs. We plan to show that both ideas are needed to solve

Algorithm 6 Extra-NRPI Algorithm

```
Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\boldsymbol{\pi}}_1$ to any policy in $\Pi$.
for $i = 1$ to $N$ do
    Collect $m$ data points as follows:
    for $j = 1$ to $m$ do
        Sample uniformly $t \in \{1, 2, \ldots, T\}$.
        Sample state $s_t$ from exploration distribution $p^{\boldsymbol{\pi}}$.
        Execute some exploration action $\boldsymbol{a}^t$ in current state $s_t$ at time $t$
        Execute $\hat{\boldsymbol{\pi}}_i$ from time $t+1$ to $T$, and observe estimate of cost-to-go $\hat{\boldsymbol{Q}}$ starting at time $t$
    end for
    Get dataset $\mathcal{D}_i = \{(s, \boldsymbol{a}, t, \hat{\boldsymbol{Q}})\}$ of states, actions, time, with current policy's cost-to-go.
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
    Train using extragradient $\hat{\pi}_{i+1}$ on $\mathcal{D}$
end for
Return best $\hat{\boldsymbol{\pi}}_i$ on validation.
```

large state space games where learning from scratch is hard because the reward function is sparse making exploration difficult.

Markov Games

We consider a multi-agent extension of the Markov decision process called Markov games [60], also known as stochastic games [92]. I introduce the formal definition as below.

Definition 5. *A* Markov game *with N agents can be defined by a tuple $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^{(i)}\}_{i \in \mathcal{N}}, \mathcal{P}, \{R^{(i)}\}_{i \in \mathcal{N}}, \rho_0, \gamma)$. Where $\mathcal{N} = \{1, \cdots, N\}$ is the set of $N > 1$ agents, $\mathcal{S}$ is the state space observed by all agents, $\mathcal{A}^{(i)}$ is the action space of agent i, where $i \in \mathcal{N}$. Let $\boldsymbol{\mathcal{A}} : \mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(N)}$, then $\mathcal{P} : \mathcal{S} \times \to \Delta(\mathcal{S})$ is the state transition probability $s \in \mathcal{S}$ to any state $s' \in \mathcal{S}$, $\rho_0 : \mathcal{S} \to [0, 1]$ is the distribution of the initial state s^0, $\gamma \in [0, 1]$ is the discount factor and for any joint action $\mathbf{a} \in \boldsymbol{\mathcal{A}}$; $R^{(i)} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function that determines the immediate reward received by agent i for a transition from (s, a) to s'.*

We use $-i$ to represent the set of agents except i, and **bold** variables without superscript i to denote concatenation of all variables. For example, $\boldsymbol{\pi}$ represents the joint policy and $\mathbf{a}$ represents the joint action. The local polices π^i of the joint policy $\boldsymbol{\pi}$ are usually parameterized by some parameter $w_i \in \Theta_i$, denoted by $\pi^{i,w}$. The joint policy cost-go function $\boldsymbol{Q}^{\boldsymbol{\pi}}(s, \boldsymbol{a})$ denotes the expected future cost of executing action $\boldsymbol{a}$ in state s. The objective of agent i is to maximize its own total expected return. The average reward for agent i under this joint policy is defined as

$$J^{(i)}_{\pi^i, \pi^{-i}}(\theta^{(i)}) := \lim_{T \to \infty} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t R^{(i)}(s_t, a_t, s_{t+1}) \,\middle|\, a_t^{(i)} \sim \pi^{(i)}(\cdot \mid s_t), s_0 = s \right],$$

where $\theta^{(i)}$ is the policy parameters for agent i and $\boldsymbol{\theta}$ is the concatenation of all the agents' parameters.

The solution to the Markov game is different than an MDP because the reward function for each agent depends on the join agent actions. The *optimal* performance of each agent is dependent on its own and other policies as well. A common solution to the Markov game is an ϵ-Nash equilibrium which extends a Nash equilibrium (NE) [68].

Definition 6. *A ϵ-Nash equilibrium of the Markov game $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^{(i)}\}_{i\in\mathcal{N}}, \mathcal{P}, \{R^{(i)}\}_{i\in\mathcal{N}}, \rho_0, \gamma)$ is a joint policy $\boldsymbol{\pi}^* = (\pi^{1,*}, \cdots, \pi^{N,*})$ for any $s \in \mathcal{S}$ and $i \in \mathcal{N}$, such that $\exists \epsilon > 0$:*

$$J^{(i)}_{\pi^{i,*},\pi^{-i,*}}(s) \geq J^{(i)}_{\pi^{i},\pi^{-i,*}}(s) - \epsilon, \qquad \textit{for any} \;\; \pi^i.$$

Note that $\epsilon = 0$ then ϵ-Nash equilibrium is equivalent to a Nash equilibrium. At the Nash equilibrium point π^* none of the agents have the incentive to deviate. That means for any agent $i \in \mathcal{N}$, the policy $\pi^{i,*}$ is the *best-response* of $\pi^{-i,*}$. Formally,

Definition 7. *Given a joint policy $\boldsymbol{\pi}^* = (\pi^{1,*}, \cdots, \pi^{N,*})$, the best response to π^{-i} is the policy $\hat{\pi}^i$ that maximizes $J^{(i)}_{\pi^i,\pi^{-i}}$:*

$$\hat{\pi}^i = \operatorname*{argmax}_{\pi^i} J^{(i)}_{\pi^i,\pi^{-i}}$$

Approximate Policy Iteration

Approximate policy iteration (API) techniques have been important for being able to solve more realistic reinforcement learning problems (e.g. TRPO [88], PPO [90], CPI [50]). In PSDP [7], we assume we are given a sequence of base distributions $\mu_0, \mu_1, \ldots, \mu_{T-1}$ over the states. These μ_T tells us how often a good policy visits each state a time t. The PSDP algorithm works by finding π_{T-1}, then $\pi_{T-2}, \ldots$ to π_0 where each policy is chosen from a stationary policy class Π. In general, if $\mu_0, \mu_1, \ldots, \mu_{T-1}$ provides a good distribution over the state space, then PSDP will return a good policy. There has been a lot of recent work in the literate that are PSDP style variants ([47, 54, 59, 80, 86, 99]

The variant that we build on is No-Regret Policy Iteration (NRPI) [83] which generalizes PSDP. NRPI learns a stationary policy instead of a non-stationary policy that is learned in PSDP. NRPI non-stationary policy is more practical because the horizon in some setting can be very large. NRPI, similar to PSDP, assumes access to a state exploration distribution ν_t for all time t in $1, 2, \ldots, T$ and this state distribution should be close to the near-optimal policy in the policy class Π (i.e. ν_t could be the state distribution from the expert policy $\nu_t = d^t_{\pi^*}$).

In our multi-agent game setting the ν_t is $p^{\boldsymbol{\pi}}$, the joint state distribution for all policies. Essentially, we replay all the set of states and actions taken in the game up to some point and then take an exploration action with all the players similar to what is done in normal NRPI. Naively optimizing the multi-agent policies to finding a Nash Equilibrium using gradient descent ascent will lead to the policies oscillating or diverging[66]. To combat this issue we propose to optimize the policies using extragradient using extrapolation.

Variational Inequality Problem

Assuming that we have access to a simultaneous gradient of the game,
$F \triangleq (\nabla^0_\theta J^{(0)}, \nabla^i_\theta J^{(1)}, \ldots, \nabla^N_\theta J^{(N)})$; The variational inequality (VI) problem can be defined as finding

$$\theta^* \in \Theta \quad \text{such that} \quad F(\theta^*)^\top (\theta - \theta^*) \geq 0 \quad \forall \theta \in \Theta, \tag{7.1}$$

When the gradient of a parameter θ^i includes the θ^{-i}. To do gradient properly, a simultaneous gradient (i.e. lock step gradient) has to be implemented. That is you compute the derivate for all parameters and update them all at once, instead of updating them one at a time. Simultaneous gradient descent may converge slow and away from the Nash. The extra-gradient method [53] provides better guarantees than simultaneous gradient [69] and has shown to work well in practice

[37?]. The algorithm consists of at each iteration τ, computing the gradient at an (extrapolated) point different from the current point from which the update is performed:

$$\begin{aligned} \text{(extrapolation)} \quad & \theta_{\tau+1/2} = proj_{\Theta}[\theta_\tau - \gamma_\tau F(\theta_\tau)], \\ \text{(update)} \quad & \theta_{\tau+1} = proj_{\Theta}[\theta_\tau - \gamma_\tau F(\theta_{\tau+1/2})], \end{aligned} \tag{7.2}$$

where $proj_{\Theta}[\cdot]$ is a projection onto the constraint set; if constraints exist.

The final algorithm that combines extragradient with NRPI can be seen in Algorithm 6. The algorithm works by first sampling a time step t and using it to sample a state from joint policy $\boldsymbol{\pi}$ state distribution $p^{\boldsymbol{\pi}}$. After rolling-in into s_t, we execute an exploration action $\boldsymbol{a}$ and then follow our current joint policy $\hat{\boldsymbol{\pi}}$ until the end of the episode and observe some cost. We aggregate all of this information and train a new joint policy using extragradinet as our optimizer. The algorithm is very similar to the original NRPI except for that we use extragradient for updating the policies instead of a cost-sensitive classifier.

Using demonstration data in a multi-agent setting is difficult, even for popular techniques in single-agent imitation learning. Given that we have access to a sparse reward function, we use the demonstration data as a proxy for good state visitation by creating a start state distribution from the demonstration data.

7.1.2 Experiments

We plan to perform experiments on 6 sparse-reward multi-agent environments and compare them to two baselines: Imitation Learning technique Behavior Cloning; and two Multi-agent policy gradient technique MADDPG [63] and SEAC [25].

Environments

Predator-Prey (PP): We first consider an altered version of the Predator-Prey environment [63] proposed by [25]. The sparse-reward Predator-Prey environment has three predator agents that must catch prey by coordinating and approaching it simultaneously. When two predators are adjacent to the prey, then they each receive a reward of one. Agents are penalized for leaving the bounds of the map but otherwise receive zero rewards.

Level-Based Foraging (LBF): LBF [25] is a mixed cooperative-competitive game that focuses on coordination. Agents navigate a grid world and collect foods by cooperating with other agents if required. The agents are rewarded for a fraction of items collected in every episode.

Multi-Robot Warehouse (RWARE): RWARE [25] is an environment where robots move items around a warehouse. The environment requires agents to moved requested shelves to the goal posts and back to an empty location. The authors [25] use total returns given by the number of deliveries over an episode of 500 timesteps as the reward signal.

Pommerman[36, 79] Pommerman has been noticed as being difficult because of exploration issues. In particular, the reward is obtained at the end of the episode, but often a winning reward is due to the opponent's suicide (i.e. not due to an agent's actions).

Atari[12, 100] We propose to first train experts using MADDPG [63] to obtain experts and then alter the rewards in Atari games to be sparse. We plan to alter the reward so that agents receive bandit feedback of $+1$ if they win and -1 if they loose.

Conclusion: At the end of this project we will have potentially developed am imitation learning technique that could scale to large hard to explore video games. Video games are one of the most promising domains to do real-world reinforcement learning on, so being able to have an algorithm that could potentially scale to large hard exploration problems is important.

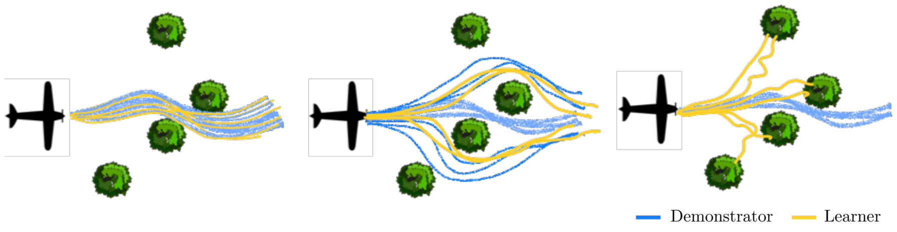
7.2 An Invitation to Imitation[8]: Part 2 (Benchmarking IL)

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. In normal supervised learning, the availability of datasets has allowed people to accurately track progress and has been a very important component of success. Because imitation learning in its simplest form reduces to normal supervised learning – holding practitioners in the field to compare against each other using the same datasets and environments becomes vital for monitoring progress. This proposed work aims to standardize comparison between imitation learning algorithms.

7.2.1 Imitation Learning algorithms Evaluations and Benchmarks

I plan to introduce a novel collection of environments (along with pretrain agents), baseline algorithms and metrics. The collection of environments include all of previous used environments such as Mujoco [101], PyBullet[27], Atari [11] and more — but we emphasize more on the opensource environments. The standardization of imitation learning algorithms through this framework will make research more accessible and address any issues of reproducibility.

7.2.2 Formalizing the Benchmnark Environments



	The Easy Regime	**The Goldilocks Regime**	**The Hard Regime**
	Realizable, Finite density-ratio	Model misspecification, Finite density-ratio	Model misspecification, Infinite density-ratio
BC	Bayes error	$O(TC\epsilon)$	$O(T^2\epsilon)$
DAgger	Bayes error	$O(T\epsilon)$	$O(T\epsilon)$

Figure 7.1: The three regimes for feedback in imitation: Easy regime where the demonstrator is realizable; Goldilocks regime and Hard regime the expert is not realizable. Below the illustrations are the bounds for the best we expected from imitation learning DAgger and behavior cloning.

The simplest form of imitation learning is called 'Behavior Cloning'. 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 [73], was formalized in the work of [81] who proved a quadratic $\mathcal{O}(\epsilon T^2)$ bound on the regret and showed that this bound is tight. The subsequent work of 'DAgger', [85] 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.

Though the theoretical analysis formalized by [81] proved a quadratic $\mathcal{O}(\epsilon T^2)$ bound for Behavior Cloning – in practice this simple technique has been shown to work well in practice for real [16] and simulated environments [6, 19, 46]. There have been many approaches to relax this assumption of needing the ability to query an expert without a significant reduction in performance. There are many problems with more modest demands [97] which justifies the need for algorithms that relax this assumption. [97] introduces three different learning regimes in imitation learning to explain why Behavior Cloning, sometimes works well in practice. The three learning regimes are: The EASY regime, The GOLDILOCKS regime, The HARD regime. We use these regimes to reason with all of the environments that we introduce.

The Easy Regime: Realizable

The expert is realizable ($\pi^* \in \Pi$) in the Easy Regime. In this regime, it is sufficient that if you drive the ϵ classification error down you will be able to recover the expert policy. Most of the standard benchmarks used in imitation learning today fall into this regime. Essentially, you keep collecting as much data as possible and keep increasing your policy class. It has been shown that naive behavior cloning works well in these settings [81, 97].

The Hard Regime: Model Misspecification, Infinite Density-ratio

The expert is not realizable ($\pi^* \notin \Pi$) in the Hard Regime and does not visit every state. In this regime driving the ϵ classification error down is hard because there are limits on the classification error because the expert is not realizable and does not cover the entire state space. In particular, these limits leadto a compounding error of $O(T^2\epsilon)$ as pointed out by [81, 97]. Essentially, because the expert never visits every state, if our learner visits a state not covered by the expert, there is no way to recover what to do. The only way to address issues in this regime is to interact with the expert [85].

The Goldilocks Regime: Model Misspecification, Finite Density-ratio

The Goldilocks Regime relaxes the need for interactive experts by assuming the demonstrator visits all states with some small probability, but still keeping the assumption that the expert is not realizable $\pi^* \notin \Pi$. By assuming that the demonstrator has a non-zero probability of visiting every state we can bound the learner demonstrator density ratio $||\rho_{\hat{\pi}}/\rho_{\pi^*}||\infty \leq C$. Note that because the expert is not realizable there will exist some classification error. [97] showed that behavior cloning in this regime provides a sub-optimal performance of $O(TC\epsilon)$. Essentially, the demonstration data provides good exploration to learn a good recovery policy.

Tasks

Below are the set of tasks that we will consider in our benchmark paper. We plan to classify each task as fitting into the Easy, Hard, or Goldilocks regime. In particular, unlike previous work, we consider algorithmic and human demonstration data in order to verify that algorithms are not overfitting to the demonstration from certain types of demonstrators.

Continuous Control Environments For continuous control environments we consider both human demonstration data and algorithm demonstration data. For the algorithmic demonstration data we follow standard paradigms of pretraining a reinforcement learning algorithm on Mujoco [101] and PyBullet [27]; then gather demonstration from the pretrained model. For human demonstration, we use the Simitate environments [1], where human demonstrations are collected using a Microsoft Kinect and then transformed into a PyBullet [27] environment.

Pixel Based Environments For pixel environments we also consider both human demonstration and algorithmic demonstration data. For the algorithmic demonstration we pretrained experts using reinforcement learning in Atari [11]; then collected demonstration data from these experts. For the human demonstration data, we use data collected from [51] for Atari. [51] also collected data for Doom and Modern Videos games and consider these environments as well. [109] introduces several pixel-based reinforcement learning environments and we consider these as well.

Natural Language Processing Environments For natural language processing environments we consider traditional structure prediction setting POS and NER [28]. We also optical character recognition (OCR) and the Spelling data from [56]; keyphrase extraction [18], semantic parsing [104], word-level quality estimation [65] and phrase generation [33].

7.2.3 Formalizing the Evaluation Metrics

The benchmark will provide trained experts for environments where it is possible to train a reinforcement learning agent from scratch. We will also provide a collected set of demonstrations from each of the experts that were trained. Below are the metrics we will use to evaluate imitation learning algorithm in this benchmark:

Rewards For all environments because we have access to a simulator in imitation learning we can evaluate the policy performance after training and tuning the model is complete in the simulator. This metric is commonly used by most imitation learning techniques.

Expert Mismatch Because some environments have access to a trained expert policy, we can compute on-policy expert mismatch $\mathbb{E}_{s\sim\rho_\pi}\left[\mathbb{1}_{\pi(s)\neq\pi^*(s)}\right]$ [97]. This metric is equivalent to have a reward function that is 1 if the $\hat{\pi}$ and π^* take the same action in a given state and zero otherwise (Note that DAgger optimizes this exactly).

7.2.4 Formalizing the Baselines

The standard baseline that all imitation learning normally (− and should) compare against is Behavior Cloning. But two additional baselines that have been forgotten about in the literature that are much simpler than Behavior cloning. Both of these baselines can be classified as Instance-based learning techniques (i.e Lazy Learners). Lazy learners compare new problem instances with instances seen in the training data, which has been collected from some demonstrators. The two lazy learners that have been proposed in the past in imitation learning and that we are adding as standard baselines are 'Locally Weighted Learning'[4, 48] and k-nearest neighbors [74, 95]. As a sanity check that Behavior Cloning is performing as expected because it is a strong baseline in certain regimes, comparing it to simpler imitation learning techniques becomes vital (i.e. Lazy Learners). Also, running lazy learners in certain domains in the Easy regime can provide practitioners with a sense of how easy particular domains are. Below are the standard baselines that we plan to incorporate into the benchmark.

Behavior Cloning [73] trains a policy π using demonstrations from the expert to classify the expert actions from the expert states in the demonstration data:

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

where ℓ^{cs} is a classification loss. Besides the introduction of the formal regret bound there has been no study or suggestions on how to train a behavior cloning model given the advancements in normal supervised learning. Given that Behavior Cloning is such a strong baseline we plan to study how different traditional supervised learning regularizations effects the model performance.

k-nearest neighbors [74, 95] is a nonparametric Instance-Based learning algorithm that leverages demonstration from the expert for predictions (i.e inference). The demonstration data contains state-action pairs $(s_i, a_i) \in \mathcal{D}_{\text{exp}}$. During inference time out $\hat{\pi}$ see a s_t and we first query using a distance metric (e.g. Euclidean distance) to find the k closet states to s_t. The resulting k closest states is denoted by $\mathcal{D}^k_{\text{exp}} = \{(s_0, a_0), (s_1, a_1), \dots, (s_k, a_k)\}$ where $|\mathcal{D}^k_{\text{exp}}| < |\mathcal{D}_{\text{exp}}|$. We can define the k-nearest neighbors $\hat{\pi}$ prediction 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:

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

where δ is the Kronecker Delta function. There has been recent work in the area of reinforcement learning to suggest that nonparametric models work in some of Easy regime environments [64].

Locally Weighted Learning [4, 48] is a nonparametric Instance-Based learning algorithm that leverages demonstration from the expert as well. The simplest form of Locally Weighted Learning is, distance weighted averaging – which is analogous to k-nearest neighbors – where now the estimate of the error can be denoted as:

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

In Locally Weighted Learning we can either weight the data directly or weight the error criterion 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 $K(d) = e^{-d^d}$ where d is some distance metric (e.g. Euclidean distance). Then the weights can be used in a weighted average to compute a $\hat{\pi}$ actions:

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

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

$$\varepsilon(s_t) = \sum_{i=0}^{k}\left[(\hat{\pi}(\cdot|s_t) - a_i)^2 K(d(s_i, s_t))\right]$$

where the goal is to find the best estimate $\hat{\pi}(\cdot|s_t)$ that will minimize the cost $\varepsilon(s_t)$ (e.g. we could use regression) This is similar to work done by [15].

Conclusion: After this work, we will have a benchmark of all the most recent imitation learning algorithms on several domains with demonstration data from algorithms and humans. This benchmark will allow us to validate the progress we have made and will allow us to thoroughly understand the pros and cons of recent methods.

8 Conclusion and Timeline

In this proposal, we show that traditional learning through interaction using reinforcement learning requires a lot of interactions to learn. Imitation learning has been proposed as a way to use expert feedback to low the sample complexity of learning. Despite the field being similar in spirit to supervised learning, there has been no proposed standard benchmark for comparing algorithms. This is important because it allows us to know what ideas work and how close we are to being able to run learning algorithms on real-world systems. Therefore we propose to benchmark the recent techniques in imitation learning. The second thing we propose to do this develop an imitation learning technique for multi-agent games because, unlike self-driving cars, the cost of mistakes in game simulations is very low. Developing techniques in these domains moves us close to deploying agents that learn through interaction on real-world systems.

8.1 Timeline

The timeline for my proposed work is as follows:

- 2021-04: Proposal
- 2021-05: Multi-Agent Imitation Learning in Games
- 2022-10: An Invitation to Imitation[8]: Part 2 (Benchmarking IL)
- 2022-02: Thesis
- 2022-04: Defense

Bibliography

[1]

[2] Theory of games and economic behavior.

[3] 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.

[4] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. 1997. Locally weighted learning. *Artif. Intell. Rev.*, 11(1–5):11–73.

[5] Les E Atlas, David A Cohn, and Richard E Ladner. 1990. Training connectionist networks with queries and selective sampling. In *NeurIPS*.

[6] Drew Bagnell. 2016. Feedback in machine learning. In *Workshop on Safety and control for artificial intelligence, CMU*.

[7] J. Andrew Bagnell, Sham Kakade, Andrew Ng, and Jeff Schneider. 2003. Policy search by dynamic programming. In *NeurIPS*.

[8] J. Andrew (Drew) Bagnell. 2015. An invitation to imitation. Technical Report CMU-RI-TR-15-08, Carnegie Mellon University, Pittsburgh, PA.

[9] 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.

[10] 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.

[11] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[12] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[13] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *NeurIPS*.

[14] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[15] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. 2016. Model-free episodic control.

[16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

[17] Kianté Brantley, Soroush Mehri, and Geoffrey J Gordon. 2021. Successor feature sets: Generalizing successor representations across policies. *arXiv preprint arXiv:2103.02650*.

[18] Kianté Brantley, Amr Sharaf, and Hal Daumé III. 2020. Active imitation learning with noisy guidance. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2093–2105, Online. Association for Computational Linguistics.

[19] Kianté Brantley, Wen Sun, and Mikael Henaff. 2019. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*.

[20] 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.

[21] 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*.

[22] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. 2006. Worst-case analysis ofselective sampling for linear classification. *JMLR*.

[23] 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*.

[24] David Chiang. 2012. Hope and fear for discriminative training of statistical translation models. *JMLR*, 13.

[25] Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. 2020. Shared experience actor-critic for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.07169*.

[26] Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL*.

[27] Erwin Coumans and Yunfei Bai. 2017. Pybullet, a python module for physics simulation in robotics, games and machine learning.

[28] Hal Daumé, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *CoRR*, abs/0907.0786.

[29] Hal Daumé, III. 2009. Unsupervised search-based structured prediction. In *International Conference on Machine Learning*, Montreal, Canada.

[30] Hal Daumé, III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning Journal*.

[31] Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.

[32] Peter Dayan. 1993. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624.

[33] Wanyu Du and Yangfeng Ji. 2019. An empirical comparison on imitation learning and reinforcement learning for paraphrase generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6012–6018, Hong Kong, China. Association for Computational Linguistics.

[34] Corina Florescu and Cornelia Caragea. 2017. PositionRank: An unsupervised approach to keyphrase extraction from scholarly documents. In *ACL*.

[35] Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George E Dahl. 2018. The importance of generation order in language modeling. *arXiv preprint arXiv:1808.07910*.

[36] Chao Gao, Bilal Kartal, Pablo Hernandez-Leal, and Matthew E Taylor. 2019. On hard exploration for reinforcement learning: A case study in pommerman. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 24–30.

[37] Gauthier Gidel, Hugo Berard, Gaëtan Vignoud, Pascal Vincent, and Simon Lacoste-Julien. 2018. A variational inequality perspective on generative adversarial networks. *arXiv preprint arXiv:1802.10551*.

[38] Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *NAACL*.

[39] Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé, III. 2014. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *EMNLP*.

[40] Aria Haghighi and Dan Klein. 2006. Prototype-driven learning for sequence models.

[41] P. T. Harker and J.-S. Pang. 1990. Finite-dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms and applications. *Math. Program.*, 48(2):161–220.

[42] 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.

[43] He He, Jason Eisner, and Hal Daume. 2012. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*, pages 3149–3157.

[44] David P. Helmbold, Nicholas Littlestone, and Philip M. Long. 2000. Apple tasting. *Information and Computation*.

[45] 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`.

[46] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573.

[47] Ionel-Alexandru Hosu and Traian Rebedea. 2016. Playing atari games with deep reinforcement learning and human checkpoint replay. *arXiv preprint arXiv:1607.05077*.

[48] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. 2002. Learning attractor landscapes for learning motor primitives. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, page 1547–1554, Cambridge, MA, USA. MIT Press.

[49] 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.

[50] Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer.

[51] Anssi Kanervisto, Joonas Pussinen, and Ville Hautamäki. 2020. Benchmarking end-to-end behavioural cloning on video games.

[52] 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.

[53] Galina M Korpelevich. 1976. The extragradient method for finding saddle points and other problems. *Matecon*, 12:747–756.

[54] H. W. Kuhn. 2016. *9. A SIMPLIFIED TWO-PERSON POKER*, pages 97–104. Princeton University Press.

[55] 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.

[56] Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. 2018. SeaRNN: Training RNNs with global-local losses. In *ICLR*.

[57] Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. 2018. SEARNN: Training RNNs with global-local losses. In *ICLR*.

[58] Lucas Lehnert and Michael L. Littman. 2019. Successor features combine elements of model-free and model-based reinforcement learning. Technical Report arXiv:1901.11437.

[59] Sergey Levine and Vladlen Koltun. 2013. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR.

[60] Michael L Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier.

[61] Michael L Littman, Richard S Sutton, and Satinder P Singh. 2001. Predictive representations of state. In *NIPS*, volume 14, page 30.

[62] Chi-kiu Lo. 2018. YiSi: A semantic machine translation evaluation metric for evaluating languages with different levels of available resources.

[63] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*.

[64] Elman Mansimov and Kyunghyun Cho. 2018. Simple nearest neighbor policy method for continuous control tasks.

[65] André F. T. Martins and Julia Kreutzer. 2017. Learning what's easy: Fully differentiable neural easy-first taggers. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 349–362, Copenhagen, Denmark. Association for Computational Linguistics.

[66] Panayotis Mertikopoulos, Christos Papadimitriou, and Georgios Piliouras. 2018. Cycles in adversarial regularized learning. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2703–2717. SIAM.

[67] Marvin Minsky. 1961. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30.

[68] John Nash. 1951. Non-cooperative games. *Annals of mathematics*, pages 286–295.

[69] Arkadi Nemirovski. 2004. Prox-method with rate of convergence o (1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251.

[70] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287.

[71] OpenAI. Openai five. `https://blog.openai.com/openai-five/`.

[72] 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.

[73] Dean A. Pomerleau. 1989. Alvinn: 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.

[74] 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.

[75] Jette Randløv and Preben Alstrøm. 1998. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 463–471, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[76] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

[77] Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *EMNLP*.

[78] Larry Rendell. 1986. A general framework for induction and a study of selective induction. *Machine Learning Journal*.

[79] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. 2018. Pommerman: A multi-agent playground. *CoRR*, abs/1809.07124.

[80] Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alexander Peysakhovich, Kyunghyun Cho, and Joan Bruna. 2018. Backplay:" man muss immer umkehren". *arXiv preprint arXiv:1807.06919*.

[81] 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.

[82] Stephane Ross and J Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*.

[83] Stéphane Ross and J. Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. ArXiv:1406.5979.

[84] 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*.

[85] 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.

[86] Tim Salimans and Richard Chen. 2018. Learning montezuma's revenge from a single demonstration. *CoRR*, abs/1812.03381.

[87] Allen Schmaltz, 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.

[88] 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.

[89] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

[90] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[91] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

[92] L. S. Shapley. 1953. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100.

[93] D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

[94] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[95] B. W. Silverman and M. C. Jones. 1989. E. fix and j.l. hodges (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on fix and hodges (1951). *International Statistical Review / Revue Internationale de Statistique*, 57(3):233–238.

[96] Satinder Singh, Michael James, and Matthew Rudary. 2012. Predictive state representations: A new theory for modeling dynamical systems. *arXiv preprint arXiv:1207.4167*.

[97] Jonathan Spencer, Sanjiban Choudhury, Arun Venkatraman, Brian Ziebart, and J. Andrew Bagnell. 2021. Feedback in imitation learning: The three regimes of covariate shift.

[98] Veselin Stoyanov and Jason Eisner. 2012. Easy-first coreference resolution. *Proceedings of COLING 2012*, pages 2519–2534.

[99] Arash Tavakoli, Vitaly Levdik, Riashat Islam, and Petar Kormushev. 2018. Prioritizing starting states for reinforcement learning. *CoRR*, abs/1811.11298.

[100] Justin K Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Luis Santos, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, Caroline Horsch, and Praveen Ravi. 2020. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*.

[101] 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.

[102] 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.

[103] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

[104] 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.

[105] Sean Welleck, Kianté Brantley, Hal Daumé, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. In *ICML*.

[106] 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.

[107] Yorick Wilks. 2008. *Machine translation: its scope and limits*. Springer Science & Business Media.

[108] Torsten Zesch, Christof Müller, and Iryna Gurevych. 2008. Extracting lexical semantic knowledge from Wikipedia and Wiktionary. In *LREC*.

[109] Amy Zhang, Yuxin Wu, and Joelle Pineau. 2018. Natural environment benchmarks for reinforcement learning.

[110] Chicheng Zhang and Kamalika Chaudhuri. 2015. Active learning from weak and strong labelers. In *NeurIPS*.

[111] 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.

Reading List:

Imitation Learning

1. 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.
2. Stephane Ross and J Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. arXiv preprint arXiv:1406.5979.
3. 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.
4. Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé, III, and John Langford. Learning to search better than your teacher. In Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, ICML, pages 2058–2066. JMLR.org, 2015.
5. Ziebart, Brian D., et al. "Maximum entropy inverse reinforcement learning." Aaai. Vol. 8. 2008.
6. Hester, Todd, et al. "Deep q-learning from demonstrations." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.
7. Venkatraman, Arun, Martial Hebert, and J. Bagnell. "Improving multi-step prediction of learned time series models." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, no. 1. 2015.
8. Daumé, Hal, John Langford, and Daniel Marcu. "Search-based structured prediction." Machine learning 75.3 (2009): 297-325.
9. He, He, Jason Eisner, and Hal Daumé. "Imitation learning by coaching." Advances in Neural Information Processing Systems 25 (2012): 3149-3157.
10. Dean A. Pomerleau. 1989. Alvinn: 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.

Reinforcement Learning

1. Ng, Andrew Y., Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping." Icml. Vol. 99. 1999.
2. Randløv, Jette, and Preben Alstrøm. "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping." ICML. Vol. 98. 1998.
3. Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." NIPs. Vol. 99. 1999.
4. Kakade, Sham, and John Langford. "Approximately optimal approximate reinforcement learning." In Proc. 19th International Conference on Machine Learning. 2002.
5. Schulman, John, et al. "Trust region policy optimization." International conference on machine learning. PMLR, 2015.
6. Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
7. Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.
8. Henderson, Peter, et al. "Deep reinforcement learning that matters." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.
9. Peter Dayan. Improving generalization for temporal difference learning: The successor repre- sentation. Neural Computation, 5(4):613–624, 1993.

10. Sutton, Richard S. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming." Machine learning proceedings 1990. Morgan Kaufmann, 1990. 216-224.

Sequence Prediction (e.g. Generation and Classification)

1. Bahl, L. R., Jelinek, F., and Mercer, R. L. A maximum likelihood approach to continuous speech recognition. IEEE transactions on pattern analysis and machine intelligence, 5(2):179–190, 1983.
2. Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. Sequence level training with recurrent neural networks. arXiv preprint arXiv:1511.06732, 2015.
3. Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).
4. Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
5. Vaswani, Ashish, et al. "Attention is all you need." arXiv preprint arXiv:1706.03762 (2017).
6. Collins, Michael, and Brian Roark. "Incremental parsing with the perceptron algorithm." Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04). 2004.
7. Hal Daumé, III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. Machine Learning Journal
8. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
9. Ratnaparkhi, Adwait. "A maximum entropy model for part-of-speech tagging." Conference on empirical methods in natural language processing. 1996.
10. Bengio, Samy, et al. "Scheduled sampling for sequence prediction with recurrent neural networks." arXiv preprint arXiv:1506.03099 (2015).