Improving Imitation Learning through Efficient Expert Querying

Matthew Hanczor

August 2018
CMU-RI-TR-18-56

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee:
William Whittaker, *Advisor*
David Held
Wen Sun

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Learning from demonstration is an intuitive approach to encoding complex behaviors in autonomous agents. Learners have shown success in challenging tasks like autonomous driving, aerial obstacle avoidance, and information gathering, through observation and mimicry alone. State of the art algorithms like Dataset Aggregation (DAgger) have made significant advances over traditional behavior cloning, demonstrating strong theoretical and empirical results. However, these methods typically impose large sampling burdens on experts which may restrict the type of demonstrators or problems that can be addressed.

In this work we propose a modified version of the DAgger algorithm aimed at reducing expert queries while maintaining learner performance. Randomly initialized policies typically have state distributions unlike those of the final policies, leading to wasted expert labeling especially early in training. By increasing the rate of policy updates we aim to collect more relevant labeled data with respect to the total number of queries. In addition, we implement several supervised active learning approaches as part of our query selection, allowing policy uncertainty to inform expert label queries. We demonstrate our algorithm on a variety of simulated robot manipulator and control tasks.

Contents

List of Figures

Chapter 1

Introduction

As the field of robotics matures, autonomous agents are called on to perform increasingly complex and challenging tasks. Robots provide benefits like increased safety and speed, reduced costs, and higher precision, if properly directed. Today these systems are expected to move beyond highly controlled industrial manufacturing facilities and research labs, and into noisy and dynamic real-world environments. As our demand for more generalized, mult-purpose systems increases, so does the complexity of encoding in these agents the ability to perform complicated tasks in variable environmental conditions.

While programming increasingly intricate behaviors can be at best time consuming or at worst intractable, it is often the case that the desired skills can be easily demonstrated. In most cases a demonstrator is available that can perform a task objective sufficiently well, but would be unsuited for long term use. As an example, humans are often used as experts for many autonomous tasks since a great deal of automation is aimed at collaborating with or replacing human operators. Optimal controllers or task specific heuristics can also act as experts, however at scale they may be too computationally expensive or slow to perform continuously, and an agent that can mimic their behavior with reduced computational complexity is required.

To address these issues, researchers have begun to rely on machine learning to approximate expert demonstrators. By observing examples of the task performed by a demonstrator, or receiving feedback on their own performance, learning agents aim to construct an approximate representation of the demonstrators decision making process. It has been shown repeatedly that it is possible to learn sophisticated behaviors through these expert examples.

One of the major challenges in learning from demonstration is the number of samples required by the learner in order to sufficiently imitate the desired behavior. As task and environmental complexity rises, the required samples can grow exponentially which may limit the types of experts that are practical to use, and the tasks an agent can learn to perform. In real-world robotics, where setting up and performing a task can be time consuming, such a large number of demonstrations may not be feasible to collect.

These challenges beg the question: is it possible to reduce the number of expert samples required for an agent to learn a behavior, while maintaining the level of performance achieved through traditional methods? This paper will aim to show that incrementally updating policies and actively requesting demonstration are significantly more efficient than bulk collection of expert samples. This leads to the main statement of this thesis:

Selectively sampling data for expert labeling, along with incrementally updating the learner policy, can significantly reduce the number of queries required to achieve competency with respect to state-of-the-art imitation learning algorithms.

In the rest of this chapter we provide a brief background on sequential decision making problems and imitation learning. In Chapter 2 we introduce our proposal for improving imitation learning through incremental updates and active learning, along with relevant prior work regarding active learning in non-stationary distributions. In Chapter 3 we cover the implementation of our proposed algorithm, and present results in a number of simulated experiments. Lastly, in Chapter 5 we discuss our conclusions.

1.1 Sequential Decision Making Problems

Autonomous robots require an understanding how a their interaction with the world will affect the state of itself and the environment for some time into the future. Based on its understanding of the environment, the agent (robot/decision maker/learner) should select actions that are likely to be beneficial in accomplishing the task or goal at hand.

This work takes advantage of the Markov Decision Process (MDP) framework [27], which is extensively used in sequential decision making problems, and especially in reinforcement and imitation learning [40]. In short, an MDP is a framework that allows an optimal policy to be computed which maximizes long-term reward over a sequence of decisions. MDPs are defined by the four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$ [1] where:

- $\mathcal{S}$ is the set of all possible states in the system, where a state provides complete information required to select an action.
- $\mathcal{A}$ is the set of actions that the agent can perform, which can be either continuous or discrete.
- $\mathcal{R}$ is the reward function. $R(s, a)$ is the immediate one step reward provided by the environment by performing action a in state s
- $\mathcal{P}$ is the state-transition function of the environment, specifying the probability of a next state dependent on past states and actions.

The defining trait of an MDP is the Markov assumption that any state s_{t+1} is dependent on only s_t and a_t where:

$$p(s_{t+1}|s_t, a_t, \ldots, s_0, a_0) = p(s_{t+1}|s_t, a_t)$$

The goal in any MDP is to construct a policy that maps states to actions, or a distribution of actions, $\pi : \mathcal{S} \rightarrow \mathcal{A}$ in order to maximize the total expected return, or to minimize the total expected cost. An objective function for selecting a policy $\hat{\pi}$ from a policy class Π is defined such that:

$$\hat{\pi} = \underset{\pi \in \Pi}{\operatorname{argmax}} \sum_{t=1}^{T} \mathbb{E}_{s \sim d_\pi^t}[R_\pi(s)] \tag{1.1}$$

where $R_\pi(s) = \mathbb{E}_{a \sim \pi(s)}[R(s, a)]$ and d_π^t is the distribution of states under π at time t such that $d_\pi = \frac{1}{T} \sum_{t=1}^{T} d_\pi^t$ [30].

1.2 Imitation Learning

Imitation learning, or learning from demonstration, is an approach to solving a sequential decision making problem by predicting the actions the demonstrator would

[1]We will also make reference to the cost $\mathcal{C}$ in place of $\mathcal{R}$, where the cost is simply as the negative of the reward $C(s, a) = -R(s, a)$

take at any particular state. Typically optimal or near-optimal experts π^*, who are able to perform the desired task sufficiently well, are used.

As stated in the previous section, the goal of an MDP is to learn $\hat{\pi}$ to maximize the expected reward. However, the learner often does not have access to the true reward and instead only observes the expert's demonstrations. In this case, a surrogate loss function $\ell(s, \pi)$ may be defined to minimize instead. In regression this could be a squared loss, or 0-1 loss for classification, with respect to π^*. Therefore, the new objective becomes:

$$\hat{\pi} = \underset{\pi \in \Pi}{\operatorname{argmin}} \sum_{t=1}^{T} \mathbb{E}_{s \sim d_{\pi}^{t}} [\ell(s, \pi)] \tag{1.2}$$

Behavior Cloning Early work in imitation learning used a strategy known as *Behavior Cloning*, where an expert performs a number of demonstrations, providing the learner with a dataset of state-action pairs $\mathcal{D} = \{(s_i \sim d_{\pi^*}, a_i \sim \pi^*(s_i))\}_{i=1}^{M}$ [2] from which to learn a policy [5]. This is a direct supervised learning approach to imitation learning, where state inputs and labeled action outputs are provided by the expert and used to train the learner. While this approach has been moderately successful, including early self driving vehicles [26], there are significant problems that limit its application in modern practice.

Ross and Bagnell [31] show that the primary issue lies in the fact that during training, states are drawn from a fixed distribution d_{π^*} induced by the expert's policy π^*, however at test time the learner must label states that arise through execution of its own policy $\hat{\pi}$, and therefore its own distribution of states $d_{\hat{\pi}}$. The difference between d_{π^*} and $d_{\hat{\pi}}$ can be significant, since small differences in actions may cause large divergences in the distribution of states over time. They found that if a learner makes a mistake (selects a different action than the expert) with probability ϵ, the total cost grows quadratically on order of $O(\epsilon T^2)$ in a task horizon T compared to traditional supervised learning with cost $O(\epsilon T)$. Intuitively, if the agent makes a mistake it is possible to transition to a state unlike one that the expert would normally encounter. Since there is no demonstration for this new state, the agent is more likely to make another mistake. This can compound over time leading the agent to continually select incorrect actions.

[2]The work in this paper applies to both stochastic and deterministic policies

Forward Training Ross and Bagnell [31] proposed the Forward Training algorithm as a solution to this problem. Instead of having an expert collect states under d_{π^*}, a series of a non-stationary policies are learned where, during training, the expert is queried interactively to label states collected by rolling out, or executing, the learner policy. Using this approach, the authors were able prove that the expected cost under the distribution of states encountered by the learned policy at test time matched the average cost during training. However, the Forward Training algorithm depends on learning a policy for every time step t within the time horizon T which makes it difficult or impossible to implement in long time horizon tasks. In [33], Ross et al. introduce the Dataset Aggregation (DAgger) algorithm as an approach to iteratively train a stationary, no-regret, learner policy.

Algorithm 1: DAGGER

```
Initialize $\mathcal{D} \leftarrow \emptyset$
Initialize $\hat{\pi}_1$ to any policy in $\Pi$
for $i = 1$ to $N$ do
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
    Sample $T$-step trajectories using $\pi_i$
    Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$ and actions given by
      expert
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
    Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
end
Return best $\hat{\pi}_i$ on validation
```

DAgger Algorithm 1 shows the complete DAgger algorithm. Initially, the policy that is executed can be a mix of the learner's and expert's policies controlled by a weighting factor β. This parameter allows the expert to control the action selection in some of the early training episodes, shifting over control to the learner agent as β decays to zero. At each iteration the current policy π_i rolls out a series of trajectories, collecting a set of labeled states $\mathcal{D}_i = \{(s, \pi^*(s))\}$. Once the policy has finished collecting data, the current dataset is aggregated with the data from all previous policies $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$. The next policy π_{i+1} is then trained on the increasingly large dataset.

The authors showed that by using this method the final policy was guaranteed to limit the number of mistakes made by the learner to linear growth in ϵ and T, on par with traditional supervised learning with a stationary data distribution. This finding has led DAgger and its extensions to be applied to solving a variety of challenges like autonomous driving [33] [45] [23], adaptive information gathering [9] [8], UAV control [34], and dependency parsing [39].

As effective and proven as DAgger has become, one of its major limitations is the large sampling burden it imposes onto the task expert. DAgger assumes that at every state, in every sampled trajectory, for every learned policy, the expert is asked to provide an action. While this may be feasible with certain experts that are inexpensive to query, such a large sampling load can limit the practicality of applying DAgger with complex or costly experts. In the next chapter we propose several improvements to expert sampling, in order to achieve performance equivalent to traditional DAgger, with fewer labeled states.

Chapter 2

Expert Sample Reduction

As shown in the previous chapter, imitation learning is a special case of supervised learning where the objective is to accurately predict the action taken by an expert in a state drawn from a distribution induced by the learner's policy. The assumption is that minimizing the difference between the expert and learner policies will improve the learner's performance on the underlying task. In the supervised and imitation learning settings there are two types of data: unlabeled, which in an MDP would be the set of observed states $\mathcal{U} = \{s_j\}_{j=1}^{K}$, and labeled $\mathcal{L} = \{s_i, a_i^{\pi^*}\}_{i=1}^{M}$, the state-action pairs labeled by the expert at state s_i. There is often a real-world cost for collecting each type of data, and typically the collection cost of unlabeled data is significantly less than the cost of labeled data. For example, tens of thousands of photos can be easily scraped off the internet, but annotating each image requires significant time and human intervention. The goal of expert sample reduction is to leverage the cheaply available unlabeled states to reduce the number of labeled state-action pairs required to achieve sufficient performance on the target task. In this chapter we propose two methods for reducing expert sample complexity. First, increasing the rate of policy adjustment per expert samples in order to speed up the transition from irrelevant to relevant state distributions. Second, improve the "quality" of samples selected for labeling by adopting i.i.d. supervised active learning techniques for active imitation learning.

2.1 Incremental Policy Updating

During training of the learner policy $\hat{\pi}$ under DAgger, the algorithm iteratively trains new policies π_i for $i \in 1, \ldots, N$. The initial distribution of states of the randomly initialized π_1, d_{π_1}, is usually very different than d_{π^*}. Because of this, many of the states labeled by the expert in early trajectories may not be relevant or likely to be encountered as training progresses and $d_{\pi_i} \to d_{\pi^*}$.

To reduce expert sampling the agent should ideally move as quickly as possible from its initial distribution of states, to the distribution it will encounter at test time $d_{\hat{\pi}}$, while minimizing the cost of labeling states in early training iterations. We propose that by incrementally updating the policy more frequently with fewer labeled examples, the rate of convergence towards d_{π^*} is faster than with less frequent updates with larger batches of newly collected states, with respect to the total number of labeled states. While a formal theoretical treatment to why this is possible is not provided, there is a compelling empirical basis.

It is often the case that similar states have similar expert actions, where learning the label for one state allows the learner to infer labels for other nearby states. Under a fixed policy, it is likely that the agent will encounter similar states over multiple trajectories, or even within a single trajectory. As the data set grows for a fixed distribution, the utility of adding additional samples decreases. Therefore, moving to new state distributions more frequently, instead of continuing to label in the current distribution, may lead to greater information utility per expert sample.

As states are added to the labeled set $\mathcal{L}$ and the policy is updated, the distribution of states is on average shifted towards d_{π^*}. After each shift in the distribution, the probability of encountering states from d_{π^*} improves, increasing the likelihood of labeling states relevant to the final policy $\hat{\pi}$. This change may have a compounding effect, where labeled states shift the distribution towards $\hat{\pi}$, making future labeled state more relevant, further shifting the distribution. [1]

While a random sampling of states could be used for incremental policy updating, it would be ideal to select states that provided the most utility in terms of improving the current learner's policy. In supervised learning, selecting future examples to be labeled based on the current labeled dataset is known as active learning.

[1] Judah et al. [16] provided a proof of how this similarity of state distributions between the learner and expert changed over iterations using their RAIL algorithm, an approach similar to Forward Training

2.2 Active Learning

Active learning is a strategy used in supervised learning to reduce the number of labeled examples during training while still maintaining the generalization error rate of a passive learner [36]. A canonical example is in the case of a binary linear separator. Imagine a bounded uniform distribution of unlabeled points with possible labels $\{0, 1\}$. Using passive random selection, finding a classifier that correctly labels all points with error ϵ or less would require $O(1/\epsilon)$ samples, assuming realizability (that a perfect linear classifier exists). However, by applying a binary search strategy where the learner is able to reduce the hypothesis space of possible classifiers by roughly half with each sample, only $O(\log_2(1/\epsilon))$ samples are required to achieve the same predictive error. While of course this is a very simple toy problem, the idea of using knowledge gained from past examples to inform future sampling is at the heart of active learning.

Active learning aims to select "informative" examples for labeling, examples that lead to a useful improvement in a learner's understanding of the environment. Active learning is typically grouped into one of three categories:

- **Stream-Based Querying**: In stream-based active learning, samples are sequentially drawn from the underlying distribution and presented to the learner. At every time step the learner receives a single example and decides whether to query the expert or request a new sample. As the learner does not have access to a set of unlabeled examples, sample selection is usually dependent on imposing some threshold of uncertainty such that if the current sample violates the threshold the learner queries the expert, otherwise the sample is discarded.

- **Pool-Based Sampling**: In pool-based sampling, the learner is given a set of unlabeled examples and can select any examples from the set that it would like the expert to label. Typically this selection is done greedily, with samples assigned the highest utility being selected first. This scenario is well studied and due to the ability to select optimal data points has allowed researchers to develop theoretic bounds on the sampling improvement over passively sample selection. Under certain constraints researchers have demonstrated exponential reduction in the number of expert samples required [36].

- **Query Synthesis**: While in the previous scenarios samples were presented to the learner from an external source, an alternative approach is for the learner to propose unlabeled examples based on its internal belief. In this case the learner does not need to look for samples that it is highly uncertain about, but instead can come up with and propose its own queries directly to the expert. In practice however, these techniques generate samples that are either unlikely to occur or are unable to be reliably labeled by an expert. For example, Baum and Lang [2] used membership query synthesis in an optical numerical character recognition task, where a network was trained on self-proposed data using human oracles. They found that many of the images generated to be labeled were either combinations of digits or completely unrecognizable as characters.

In this work we will focus on pool-based sampling for active imitation learning, with the learner collecting a dataset of unlabeled states from which a subset can be selected for expert labeling.

There are many objectives that can be used to guide active sample selection [10] [36]. An active learner could chose samples that have high uncertainty, or ones that the learner has low prediction confidence about [22]. Unlabeled examples could be selected in order to reduce the hypothesis space that is consistent with the current gathered data [37] [11]. Alternatively, the learner could use the structure of the unlabeled data in combination with another metric to guide sample selection. Intuitively it would make sense to select samples that directly reduce the learner's future predictive error in expectation as this is the overall objective [35]. However this approach in practice normally requires retraining the current policy for each unlabeled data point in order to estimate the utility of labeling. This is computationally intractable with large and changing datasets.

We present several practical active learning strategies in Section 3.1 and compare their relative effectiveness on imitation learning tasks in Section 3.2.

2.3 Active Imitation Learning

The aim of active imitation learning is to utilize the research and algorithms for i.i.d. supervised active learning in imitation learning tasks. Due to differences in imitation learning however, it may not always be possible to apply active learning. An interactive expert is of course required, since the learner needs to propose a query to the expert and receive at least a noisy label in return.

One of the significant challenges in using active learning in an MDP is that most active learning strategies assume that unlabeled data is drawn from a stationary distribution, where collecting unlabeled data is not affected by the data that has already been labeled. As discussed in Section 2.1, during training there can be large shifts in the distribution of states after updating the learner's policy, changing the expected future states. Additionally, unlabeled states are not independently collected in an MDP as each are typically sampled by rolling out a trajectory with π_i.

One option is to generate a set of states drawn independently from the distribution d_{π_i}. To do so, a time step is randomly sampled $t \in \{1 \dots T\}$ and the policy is rolled out until t. The state is appended to a set of unlabeled states $\mathcal{U}_{\pi_i}$, and this is repeated M times such that $\mathcal{U}_{\pi_i} = \{s^1_{\pi_i} \dots s^M_{\pi_i}\}$. $\mathcal{U}_{\pi_i}$ is now a set of states independently sampled from d_{π^i}, from which an active learner can select examples for labeling.

While this may be applicable in the infinite sample case detailed in [33], it is not feasible in practice. At every iteration of DAgger multiple trajectories would have to be run to collect a single data point per trajectory, and then from that set of independently selected points, a subset would be chosen by the active learner for labeling. Once the policy is updated, the process would need to be repeated. In the finite sample case, for M samples to be collected $O(MT)$ time steps are needed, and only some subset of those M samples are actually labeled by the active learner.

We propose rolling out trajectories under the current policy, with all states appended to $\mathcal{U}_{\pi^i}$. The active learner can select states from this dataset, leading to a substantial reduction in the number of trajectory roll-outs needed per each policy. This approach is connected to the DAgger finite sample case, where instead of observing the true distribution of trajectories, the algorithm is trained on only a small sample of trajectories at each iteration. Once $\mathcal{U}_{\pi^i}$ has been collected, the utility of each state can be determined in hindsight. B samples can then be selected for labeling, where B is the labeling budget, the number of queries that can be proposed per policy update. This budget can change during the course of training, which allows for more states to be collected per episode later on in training as the agent encounters more relevant states.

The modified DAgger algorithm, Query-Efficient Dataset Aggregation (QE-DAgger), is shown in Algorithm 2. Instead of interactively labeling all states in $\mathcal{U}_i$, as per DAgger, samples are added to $\mathcal{L}_i$ using a selective labeling approach $S(\mathcal{U}_i, \pi_i)$, dependent on $\mathcal{U}_i$ and the current policy π_i. In Chapter 3 we apply this algorithm to a number of simulated environments and compare the results with traditional DAgger.

Algorithm 2: QE-DAgger

```
Initialize $\mathcal{L} \leftarrow \emptyset$
Initialize $\hat{\pi}_1$ to any policy in $\Pi$
for $i = 1$ to $N$ do
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
    Collect unlabeled dataset $\mathcal{U}_i$ by sampling $T$-step trajectories using $\pi_i$
    Label $B$ data points $\mathcal{L}_i = \{(s, \pi^*(s))\}$ with selective labeling method $S(\mathcal{U}_i, \pi_i)$
    Aggregate datasets: $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_i$
    Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{L}$
end
Return best $\hat{\pi}_i$ on validation
```

2.4 Related Work

The most relevant research to our work is by Judah et al. [16], who developed a theoretical approach to using active learning in the context of the Forward Training algorithm [31]. The authors showed that, due to the reduction [31] of imitation to i.i.d. passive learning, active learning techniques could be immediately applied to learn a non-stationary policy $\hat{\pi}_t$ by sampling states from multiple executions of the previous non-stationary policy $\hat{\pi}^{t-1} = (\hat{\pi}_1, \ldots, \hat{\pi}_{t-1})$ up to the current time step t. They demonstrated a practical stationary policy modification to Forward Training, called RAIL (Reduction-based Active Imitation Learning), and showed an improvement over passive sample selection by using a density-weighted query-by-committee [21] active learner. As a contrast, in our work we aim to improve the stationary policies learned by the DAgger algorithm, which provides more robust no-regret guarantees, and employ more sophisticated classes of policy and uncertainty representation compared to the linear policies learned in the authors' experiments.

In [19], the authors address the challenge of imitation learning with high-dimensional image data as a representation of the state. They use a stream-based active learning approach where states are queried if they are deemed "risky". This evaluation is performed by using a one-class Support Vector Machine to determine states that are close to the decision boundary and therefore likely to be misclassified. In addition, they employ a notion of novelty to define regions of the state space where past learned classifiers have performed poorly in order to encourage sampling in those regions.

While the primary goal of our work is querying states with the intent of reducing overall expert sampling, other work has focused on querying experts to promote safe exploration and learning. Zhang et al. [45] applied the DAgger framework to autonomous driving, where they considered the importance of ensuring the safety of the agent during training. In order to maintain safe exploration they used a secondary safety classifier in addition to the learner policy network to predict if, given the current learner policy and state, the learner is likely to deviate from the reference expert policy. In the case the policy is unlikely to deviate from the expert as predicted by the safety classifier, then the learner policy maintains control of the vehicle, otherwise control is handed over to the expert. The authors demonstrated that the safety classifier significantly reduced the number of collisions that occurred during training, though they did not compare performance in relation to the number of queries posed to the expert.

Similarly, Menda et al. [23] also used a stream-based active learning approach to query an expert in order to maintain safe exploration of an agent. In this work the authors used an approximation to a Bayesian network presented by [14] in order to gauge uncertainty in the policy at each state. Again, a predefined threshold was used to determine when the learner's policy was sufficiently uncertain, and control shifted to the expert reference policy if necessary. The results demonstrated that while their approach improved safety, as defined by the authors, performance and episodic reward for the task did not improve over traditional DAgger given the same number of training epochs.

Chapter 3

Implementation

In the last chapter we proposed a modification to the DAgger algorithm whereby policies are updated more frequently with respect to the number of samples collected, and states can be selectively chosen based on the current policy's confidence at that state. In this chapter we lay out several practical implementations of the QE-DAgger algorithm for discrete and continuous action space tasks using deep neural networks to learn viable policies. Deep learning methods are state of the art, having shown the ability to outperform human level performance on a number of challenging problems in imitation and reinforcement learning [24] [41] [39] [38].

3.1 Query Selection

3.1.1 Uniform Sampling

To test the benefits of incremental policy updating without the effects of active learning, QE-DAgger is implemented with uniformly random state selection for labeling. After $\mathcal{U}_i$ is collected under the current policy π_i, B states are randomly sampled without replacement to add to $\mathcal{L}$. This selection method will demonstrate how incremental updating improves on classic DAgger, as well as serve as a baseline comparison for the active learning sampling methods.

3.1.2 Query-By-Committee

In machine learning, a hypothesis space $\mathcal{H}$ is the set of all possible hypotheses under consideration, i.e. all possible weight parameter values in a neural network. One popular active learning strategy is to propose multiple hypotheses from a version space $\mathcal{V} \subseteq \mathcal{H}$, a subset of hypotheses that are consistent with the current dataset. By comparing multiple consistent hypotheses on unlabeled data, it is likely that there will be some disagreement between hypotheses, with greater disagreement implying greater uncertainty. Query-by-committee (QBC), is an active learning strategy that uses a collection of hypotheses in $\mathcal{H}$ to decide which unlabeled states would best reduce $\mathcal{V}$ [36].

Of course in many real-world problems there may not exist a hypothesis that is consistent with the complete labeled dataset $\mathcal{L}$, and for complex, or infinite hypothesis spaces in the case of deep neural networks, finding the extremes of $\mathcal{V}$ can be intractable. Instead, a Bayesian approach of sampling from a posterior distribution of hypotheses in the parameter space, $P(\theta|\mathcal{L})$, is used [36] [21].

One way to sample a hypothesis from a neural network is through the use of dropout at test time [13] [14]. Dropout is a commonly used technique during training of deep neural networks, whereby a random binary mask is applied to each weight layer in a network, sampled from a Bernoulli distribution with a dropout rate parameter $0 \leq d \leq 1$. By using dropout for inference, and sending multiple copies of the state in question through the network, a group of randomly selected possible models are generated to produce predictions. Other approaches use multiple classifiers with different random initializations such that committee members learn different local minima for predictions [20]. The members are updated independently on the labeled dataset. Using a dropout-based approach has the advantage that a single trained network can be used to construct a committee instead of needing to train multiple networks in parallel.

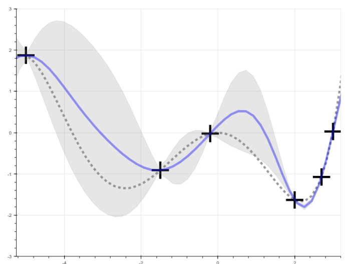
To quantify the level of disagreement per state, Jensen-Shannon divergence [22], an information theoretic measure based on Kullback-Leibler (KL) divergence [18], is used.

$$JS(p_1, \ldots, p_n) = H(\sum_{i=1}^{n} w_i p_i) - \sum_{i=1}^{n} w_i H(p_i) \tag{3.1}$$

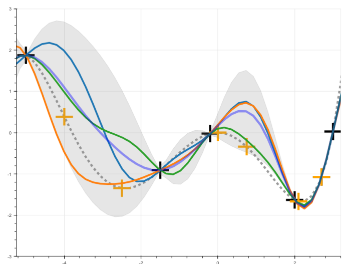
Where $w_i = 1/n$ so as to equally weight each committee member, and $H(p)$ is the Shannon entropy for a distribution p:

$$H(p) = -\sum_{j=1}^{m} p(x_i) \log_2(p(x_i)) \tag{3.2}$$

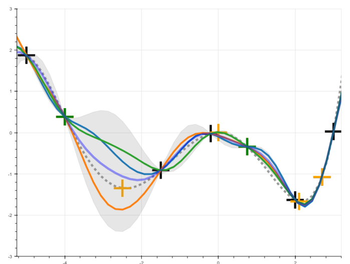
for a hypothesis with m discrete actions.



(a)



(b)



(c)

Figure 3.1: (a) A gaussian process with initial labeled data (black +) mean predicted function (blue line) and two standard deviations of prediction variance (gray shaded region). The true function is represented by a dashed light gray line. (b) An unlabeled batch of data points are collected (orange +) and parameters are sampled to estimate uncertainty at each point. (c) Labeling data with high uncertainty (green +) reduces prediction error and variance efficiently

JS-divergence allows for comparison of class membership probabilities of the entire committee. In contrast, vote entropy is another commonly used QBC metric [16] [11] [36], but it does not directly compare prediction probabilities of the members, and is less useful when the expert's policy is stochastic.

3.1.3 Uncertainty Sampling

Another active learning approach is to choose samples based on how uncertain the current learner is. Ideally, as samples with higher uncertainty are labeled, the average uncertainty over all states decreases.

Gaussian processes are a powerful Bayesian tool used to model uncertainties over functions [29]. Figure 3.1 shows how a Gaussian process is able to model uncertainty by sampling functions from the posterior distribution of parameters. In sections of the function where there are few labeled states, the uncertainty over the true underlying function from which the data is drawn is uncertain. In Figure 3.1b, samples of possible functions vary dramatically in sparsely sampled regions of the input space, while still representing labeled data with high accuracy and certainty.

However, Gaussian processes become computationally intractable as dataset size increases, since they require a matrix inversion the size of the dataset, with time complexity $O(N^3)$, and they may not have the expressiveness of a modern deep neural network. Unfortunately, accurately estimating uncertainty in deep networks is difficult and not well understood. Recently though, work by Gal and Ghahramani [14] demonstrated how dropout applied in a neural network with arbitrary length and non-linearities is mathematically equivalent to an approximation of a deep Gaussian process model. Essentially what they were able to show was that forward passes through the network are samples from the approximate predictive posterior, and by averaging multiple forward passes (Monte-Carlo integration) an estimate of the model's uncertainty could be approximated.

As an extension to this work, Gal et al. proposed a method for adjusting the dropout rate hyper-parameter as a function of the dataset size $|\mathcal{L}|$, called Concrete Dropout [15]. As $|\mathcal{L}|$ increases, model uncertainty along with the appropriate dropout parameter should decrease. Automatic tuning of dropout probability makes dropout sampling feasible for tasks with changing dataset sizes, instead of hand tuning the dropout parameter as more data is collected. For this reason, Concrete Dropout is used for estimating the uncertainty of the learner's policies at specific states in continuous action space tasks.

The learner's concrete dropout network has two output layers, the policy and the predicted log variance of the policy. A loss function is implemented that balances improving the learner's prediction to match the expert, while reducing the predicted log variance.

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V(x^i,\theta)}(y^i - f(x^i,\theta)^2 + \log(V(x^i,\theta)) + \lambda\left\|\theta\right\|^2 \tag{3.3}$$

This loss also includes a regularization term. The learner can achieve a small loss so long as the predictive mean error and variance $V(x^i,\theta)$ is low, however if the network outputs a small variance for an action prediction that is far from the expert's action the loss will be substantial.

3.1.4 Density Weighting

The two active learning approaches detailed above can potentially be fairly myopic as neither considers the structure of the data distribution from which they are selecting samples, only considering the information content of the individual states. It is possible to hedge selection of samples based on the distribution of collected data using density weighting [36]. Typically, an active learner should avoid spending time

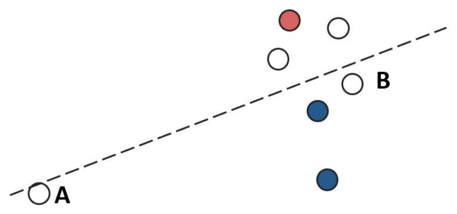


Figure 3.2: Using uncertainty selection in the case of a binary linear classifier, the active learner would select unlabeled point A as it is closer to the decision boundary (dashed line) than all other unlabeled points. Point A is an outlier though, unlikely to occur and gives little information about other states. Using density weighting, point B would instead be labeled.

labeling states in outlying regions of the state distribution and focus more on states that are likely to occur (Figure 3.2). Combining a metric for similarity, with an active learning selection metric, changes the selection preference to states that the learner is both uncertain about but likely to encounter based on past trajectories. Formally:

$$\hat{u} = \underset{u \in \mathcal{U}}{\operatorname{argmax}} \, \phi(u) \Big(\frac{1}{|\mathcal{U}|} \sum_{u' \in \mathcal{U}} \operatorname{sim}(u, u') \Big)^{\gamma} \tag{3.4}$$

where $\phi(u)$ is the utility assigned to unlabeled state u by active learner ϕ. In the experiments below Euclidean distance is used as the density weighting metric $\operatorname{sim}(u, u')$, and the weighting parameter is fixed at $\gamma = 1$.

3.2 Experiment Setup

To evaluate whether Query-Efficient DAgger is able to reduce imitation learning expert querying in practice, experiments were performed in five simulated environments, comprising both discrete and continuous action spaces. These environments are part of the OpenAI Gym collection [6].

3.2.1 Robot Manipulation

The benefits of QE-DAgger are first demonstrated on three continuous action space robotic manipulator tasks. The manipulator is a simulated Fetch robot, a 7-DoF robot arm with a two fingered paddle gripper as the end effector (Figure 3.3). In each

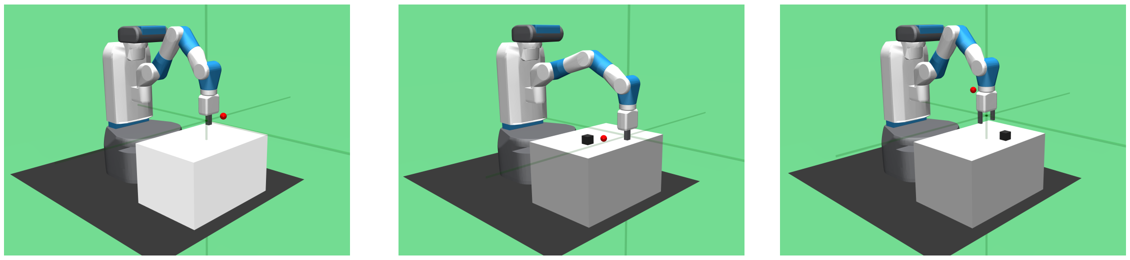

(a) FetchReach-v0 (b) FetchPush-v0 (c) FetchPickAndPlace-v0

Figure 3.3: Fetch Robot Environments

task the aim is to move either the end effector or an object to a desired goal location in $\mathbb{R}^3$. At every time step the agent receives a reward of -1 if the object is not at the goal location, and 0 otherwise. Three scenarios are tested:

- In **FetchReach-v0**, the goal is to move the Fetch gripper to a target position, with an observation space size $\mathbb{R}^{10}$.
- The **FetchPush-v0** task requires a randomly initialized box to be moved to a target location on the table. The end effector is locked closed in this task to ensure the learned behavior involves pushing the block to the goal. The observation space is size $\mathbb{R}^{25}$.
- **FetchPickAndPlace-v0** also involves a box, but in this task the goal is to grasp the box with the end effector and move it to the target location, which can be either on the table or in the air above the table. The observation space is size $\mathbb{R}^{25}$.

The expert policies for each task are trained Actor-Critic networks using DDPG and Hindsight Experience Replay [1], with implementations available from the OpenAI Baselines repository [12]. Expert baselines for each environment are included in the performance plots below. A slight modification was made to these environments during training of the learner policies. These tasks normally have a fixed 50 time step horizon regardless of whether or not the task is achieved, which is beneficial for certain parallel computing training methods. Instead, the episode ends if the object is within the goal location for five consecutive time steps, or if 50 total time steps have passed, which helps to speed up training.

The learner policy neural networks comprises of 3 hidden layers with 256 units each and rectified linear unit (ReLU) non-linearities. The output layer uses a hyperbolic

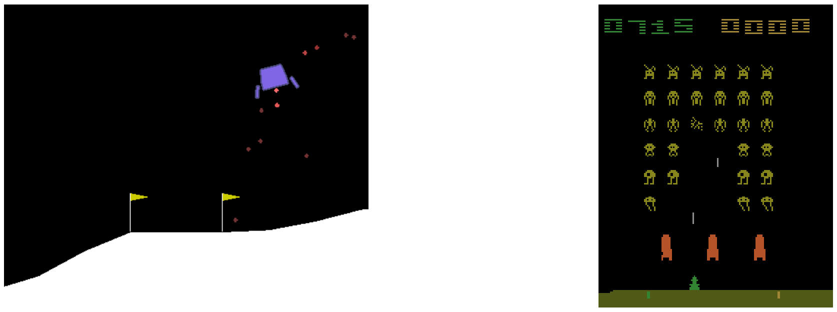

(a) LunarLander-v2 (b) SpaceInvaders-v0

Figure 3.4: OpenAI Gym Environments

tangent (TanH) nonlinearity to bound the action space within $[-1, 1]^4$, where the actions specify the desired movement of the end effector. This policy network closely matches the expert's policy network from [12], so that the learner has the ability to approximate the expert as closely as possible without increasing the number of trainable parameters.

For all of the Fetch tasks a single expert-led demonstration is performed, $\beta_i = \mathbb{I}(i = 1)$, to initialize the learner policy. The states labeled in the initial episode are included towards the count of total queried states. The querying budget $B = 1$ for all Fetch tasks.

In these environments, traditional DAgger is compared to QE-DAgger using uniform sampling, uncertainty selection, and density weighted uncertainty selection.

3.2.2 Game Playing

We also test our approach on two discrete action space environments, shown in Figure 3.4

LunarLander-v2 - The goal of Lunar Lander is to pilot a two legged space craft from a randomly selected initial starting pose and velocity to a resting position within a fixed target area. The agent's observation space is in $\mathbb{R}^4$. The location of the target landing pad is constant at $(0, 0)$, and the agent receives higher reward for becoming stationary close to the target without descending too rapidly and crashing into the terrain below. The agent can perform one of four discrete actions at each time step: do nothing, fire the left orientation engine, fire the right orientation engine, and fire the main body engine.

The expert for this task is a heuristic provided by OpenAI [6]. The learner policy network has 3 hidden layers with 16 units each and ReLU non-linearities. The output is a softmax of probabilities the size of the action space, and actions are chosen greedily with respect to the output distribution. Again, the expert query budget $B = 1$.

SpaceInvadersNoFrameskip-v0 - This is a port of the classic Atari game Space Invaders where the point of the game is avoid attacks from enemy spacecraft while returning fire and knocking out all enemy ships before they are able to descend to the bottom of the screen. The agent is given three lives and the goal is to accrue as many points as possible before the lives are spent.

The observation space of this environment is a sequence of four, 84x84-pixel preprocessed images, with the action space being one of six discrete actions. The details of the observation space preprocessing can be found in the original work of learning to play Atari environments with Deep Q-Networks by [24].

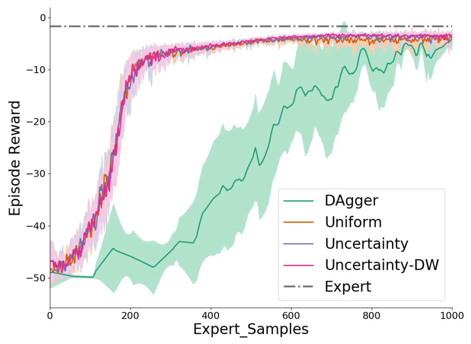
The expert policy is a convolutional neural network trained using a Proximal Policy Optimization baseline available through [12]. The expert was trained for 100 million time steps and achieves an average score of 2486 points. The learner policy network uses the same form of the convolutional layers as the expert's network, but with the inclusion of three additional hidden layers with 256 units before the softmax output layer from which actions are selected greedliy. Due to the long episode length the querying budget is fixed at $B = 30$.

3.3 Results

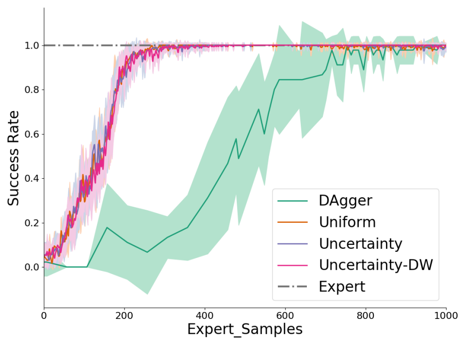
The results from the experiments in each environment are presented below. In order to ensure proper comparison between classic DAgger and our proposed approaches we maintained consistent hyper-parameters (learning rates, dropout, weight regularization, training epochs, etc.) within a task and where applicable.

Fetch Reach

Figure 3.5 shows the episode reward and task success rate for the FetchReach environment. The three QE-DAgger implementations, uniform random selection, dropout uncertainty, and density weighted dropout uncertanty, achieved 90% task success after only 200 expert labels compared to DAgger which did not reach 90% success until 650 labeled examples were collected. Additionally, QE-DAgger learners matched the



(a) Episode Reward

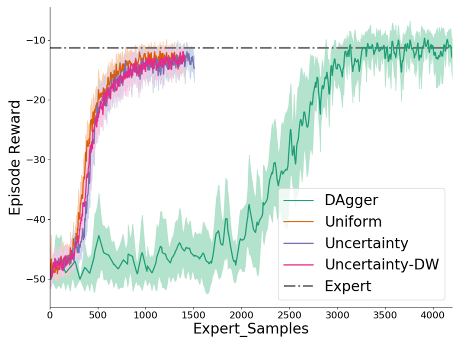


(b) Task Success

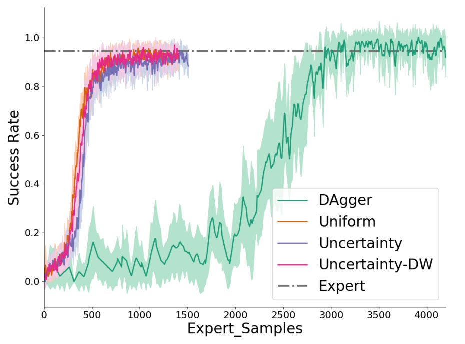
Figure 3.5: Episode reward (a) and task success rate (b) for the Fetch Reach environment.

expert performance of 100% success with only one third of the number of labeled examples compared to DAgger. With respect to one another, the QE-DAgger variations showed almost no differences in performance during training.

Fetch Pick-and-Place



(a) Episode Reward

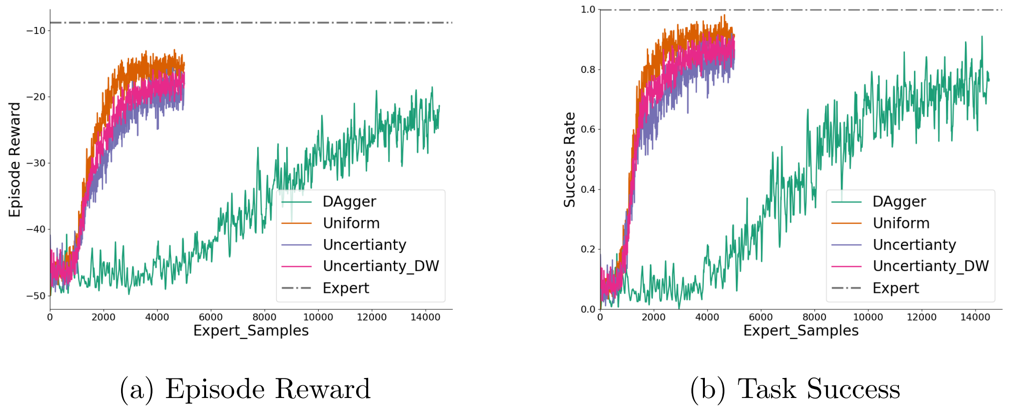


(b) Task Success

Figure 3.6: Episode reward (a) and task success rate (b) for the Fetch Pick-and-Place environment.

In the Fetch Pick-and-Place task, a similar result to the Fetch Reach environment can be seen, where there is a significant reduction in the number of expert queries required to reach expert level performance (Figure 3.6). In addition, the three QE-DAgger variations once again are nearly identical in the number of expert samples required for competency. Here QE-DAgger matches the expert success rate after around 1000 labeled examples, while DAgger requires three times as many queries.

Fetch Push



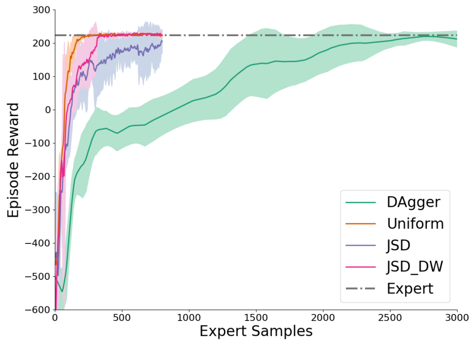
(a) Episode Reward (b) Task Success

Figure 3.7: Episode reward (a) and task success rate (b) for the Fetch Push environment.

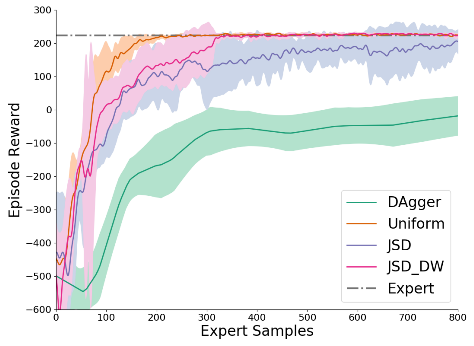
In the final Fetch environment, Fetch Push, all three QE-Dagger variations again showed a significant learning advantage over DAgger, with each averaging over 80% success after 4000 expert samples, compared to DAgger which after 15000 samples was unable to achieve an average success rate of 80% (Figure 3.7). In this experiment, uniformly random expert querying seemed to outperform uncertainty selection, both density weighted and unweighted. Random selection leveled out at a higher episode reward than both uncertainty methods, but none of the tested methods were able to match expert performance. This could be due to the difficulty of the environment of the task in comparison to the other Fetch tasks, or perhaps given more training time the learners would be able to approach expert level performance.

Lunar Lander

The Lunar Lander heuristic averages a total reward of 224 which exceeds the ”solved” episodic reward threshold of 200 points. It is clear from the plots in Figure 3.8 that the



(a)

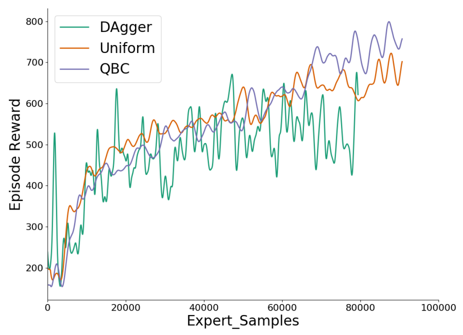


(b)

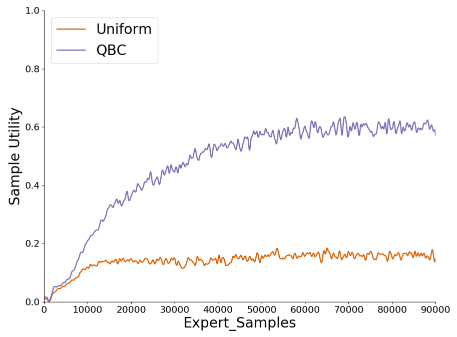
Figure 3.8: Episodic reward for Lunar Lander, plot (b) is a zoomed in version of (a) to show detail.

QE-DAgger approaches required far fewer samples to cross the 200 point threshold. Random sample selection, with policy updates after each sample, reached expert-level performance after only 200 queries. Jensen-Shannon divergence had an average return of 195 points after 800 expert samples, while adding density weighting led to expert-level performance after 320 samples. This is in comparison to traditional DAgger which required 2800 expert samples to match the expert's average reward.

Space Invaders



(a) Episode Reward



(b) Average Sample Utility

Figure 3.9: 3.9a Space Invaders (a) smoothed reward per episode and (b) average selected sample utility per episode.

The Space Invaders expert has an average episodic reward of 2486 points compared to human level performance of 1652 as measured by [24]. A single experiment each of random sample selection, QBC with JS-divergence, and DAgger were ran due to the long episode length and training time for this task. Figure 3.9 shows the episode reward and query utility as measured by JS-divergence during training. While all three methods perform similarly well early, as training progresses random querying and QBC selection trend noticeably upward in comparison to DAgger selection which seems to level out. After ninety thousand expert queries the DAgger learner has an average reward of around 500, compared to 650 for the random learner and 750 for the QBC learner. The query utility was also measured for the QBC and random agents, and there is a clear separation between the JS-divergence of selected examples, as would be expected since the QBC agent is explicitly selecting unlabeled states with high utility. Random selection levels out early on with an average selected utility of 0.1, while QBC converges to a utility of 0.6 much later on. It is possible that this leveling off of utility could lead to better performance for the QBC learner, and Figure 3.9a does show QBC outperforming uniform random selection as the expert samples increase, but this requires additional training and experimentation to answer.

One of the major challenges in this environment is the possibility that mistakes can be fatal. By selecting the wrong action with an enemy laser approaching, the episode can end without the opportunity for the learner to make a corrective action. This makes it difficult to learn since the agent needs to perform perfectly in the presence of dangerous states, which can require a large number of labeled examples.

Chapter 4

Conclusion

As we develop more intelligent and human-like robots, the challenge of encoding complex functionality that generalizes to real-world environments increases. Humans excel at learning new skills rapidly by watching and interacting with an instructor, an ability researchers are trying to instill in autonomous agents. By reducing the number of demonstrations required to perform a task, more intricate and productive behaviors to be learned. The primary conclusion of this work is:

Incrementally updating a learner's policy, with respect to the number of expert queries, significantly reduces the total number of labeled states required to learn a competent policy.

The algorithm proposed in this work, Query Efficient Dataset Aggregation, required fewer labeled states to achieve the same performance compared to the state-of-the-art DAgger algorithm in all experiments. This held true in both continuous and discrete action space environments, in long and short episodic tasks. Additionally, active learning methods applied to non-i.i.d. imitation learning problems are not guaranteed to outperform passive sample selection using QE-DAgger.

4.1 Perspectives

Incrementally updating the learner's policy with passive or active sample selection significantly reduced the total number of expert queries. Early policies often do not collect states relevant to the final policy, so by sparsely labeling states each episode, greater labeling utility is achieved in the form of increased episodic reward per query for each task. Randomly selecting states, even just a single state in an episode, resulted in an order of magnitude reduction in labeled states required to achieve expert level performance in one environment, and in other tasks led to successful policies using thousands of fewer expert queries.

While active sample selection also reduced overall expert queries and in some cases outperformed passive selection, in the experiments presented here there was not a clear benefit to the additional complexity of implementing these approaches. Part of the reason for this is that understanding and measuring uncertainty in deep networks is a challenging task, though very much an area of active research [4] [3]. While dropout techniques have demonstrated quality results for some tasks [14] [15], there are legitimate criticisms regarding their efficacy and accuracy [25]. However, as researchers gain more understanding about the fairly opaque box that is deep learning, improved methods for estimating model uncertainty are likely to be developed. With better uncertainty estimates, learners will be able to select more truly informative unlabeled data points, and active imitation learning may become unequivocally more efficient than random sample selection.

In this work the assumption was made that unlabeled state collection is much less expensive than expert labeling, and while this is often the case, the trade-off between unlabeled data collection and expert querying cost should be balanced in practice. Adjusting the query budget should be problem specific; in scenarios where setup and run time of a task is negligible, a small query budget can be used. The small query budget compared to the task horizon was demonstrated here to highlight the benefits of rapid policy updates per expert sample. Alternatively, in environments where setup and policy roll-out is expensive with respect to expert labeling, using a larger query budget or full-fledged DAgger may be more appropriate. Scaling the budget could be another technique for striking unlabeled state collection vs labeling cost balance. Early on, a smaller budget may be used to push the agent towards relevant state distributions, but as training progresses a larger budget might allow for more relevant states to be collected per episode.

As shown in the Space Invaders task, QE-DAgger may take a long time to learn to perform tasks in environments where states can have very large costs for making an incorrect action. This insensitivity to the cost of performing one action versus another is inherited from the original DAgger algorithm, the effects of which were noted by Ross and Bagnell in [32]. In the next section, we propose future work to address this shortcoming.

4.2 Future Work

We believe that our work is a small step forward for the field of imitation learning, with many open areas of research regarding improving expert querying still remaining. While learning directly from demonstration is a laudable goal on its own, imitation learning is also gaining increasing recognition from other areas of machine learning. Recent work has taken advantage of imitation learning to augment reinforcement learning, sometimes referred to as apprenticeship learning [17]. New research has shown that initially learning from demonstrations substantially improves learning speed and performance [44] [46] [28] compared to randomly initialized reinforcement learning. By addressing the current drawbacks in learning from demonstration, it may continue to augment other areas of machine learning research.

Imitation learning typically learns a policy directly from expert actions, however it is possible that a cost-to-go-oracle may be available in addition to expert demonstrations [39] [32]. Wen et al. [39] demonstrated that when using a sub-optimal cost-to-go oracle, the learner was guaranteed to outperform the oracle. This is opposed to the direct action imitation case shown here, where the learned policy is limited by the expert's performance. Applying active learning and uncertainty to the learner's cost-to-go belief could potentially improve queries to the oracle and lead to better policy performance over direct action imitation.

In pool based active learning approaches, only states encountered under the current policy were considered for query selection. As QE-DAgger makes many incremental policy updates instead of large batch updates, recent policies are more likely to have similar state distributions. Pooling states from recent past trajectories might allow for better query selection without requiring additional roll-outs under the current policy.

Stream-based querying, as mentioned in Section 2.2, is a natural active learning paradigm for sequential decision makers. At each time step the agent is presented a state, and in the stream-based paradigm the agent would immediately decided whether or not to query an expert instead of collecting a set of states and asking for labels in hindsight. This has application in task where the states may be too complex to store or convey to the expert, while also allow for real-time querying and feedback, an idea explored by Chernova et al. [7].

While research in deep network uncertainty is still in its nascent stages, there are a number of probabilistic programming frameworks that allow for accurately modeling deep Bayesian networks which have been introduced to the public recently [42] [43]. Past criticisms of probabilistic programming notes that these methods are typically slower and do not scale as well as deep networks, but at least for smaller problems it would be worthwhile to see how these approaches compare to Bayesian network approximations like concrete dropout.

Bibliography

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017.

[2] Eric B Baum and Kenneth Lang. Query learning can work poorly when a human oracle is used. In *International joint conference on neural networks*, volume 8, page 8, 1992.

[3] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

[4] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[5] Ivan Bratko, Tanja Urbančič, and Claude Sammut. Behavioural cloning: phenomena, results and problems. *IFAC Proceedings Volumes*, 28(21):143–149, 1995.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[7] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25, 2009.

[8] Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, and Debadeepta Dey. Learning to gather information via imitation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 908–915. IEEE, 2017.

[9] Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Adaptive information gathering via imitation learning. *arXiv preprint arXiv:1705.07834*, 2017.

[10] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

[11] Ido Dagan and Sean P Engelson. Committee-based sampling for training probabilistic classifiers. In *Machine Learning Proceedings 1995*, pages 150–157. Elsevier, 1995.

[12] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[13] Melanie Ducoffe and Frederic Precioso. Qbdc: query by dropout committee for training deep supervised architecture. *arXiv preprint arXiv:1511.06412*, 2015.

[14] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[15] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017.

[16] Kshitij Judah, Alan Fern, and Thomas G Dietterich. Active imitation learning via reduction to iid active learning. In *UAI*, pages 428–437, 2012.

[17] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[18] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[19] Michael Laskey, Sam Staszak, Wesley Yu-Shu Hsieh, Jeffrey Mahler, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 462–469. IEEE, 2016.

[20] Ofer Matan. On-site learning. *Submitted for publication*, 1995.

[21] Andrew Kachites McCallumzy and Kamal Nigamy. Employing em and pool-based active learning for text classification. In *Proc. International Conference on Machine Learning (ICML)*, pages 359–367. Citeseer, 1998.

[22] Prem Melville, Stewart M Yang, Maytal Saar-Tsechansky, and Raymond Mooney. Active learning for probability estimation using jensen-shannon divergence. In *European conference on machine learning*, pages 268–279. Springer, 2005.

[23] Kunal Menda, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Dropoutdagger: A bayesian approach to safe imitation learning. *arXiv preprint arXiv:1709.06166*, 2017.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[25] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *arXiv preprint arXiv:1806.03335*, 2018.

[26] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[27] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[28] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[29] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[30] Stephane Ross. Interactive learning for sequential decisions and predictions. 2013.

[31] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

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

[33] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[34] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. *arXiv preprint arXiv:1211.1690*, 2012.

[35] Nicholas Roy and Andrew McCallum. Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*, pages 441–448, 2001.

[36] Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[37] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294. ACM, 1992.

[38] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[39] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030*, 2017.

[40] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction.* MIT press, 1998.

[41] Gerald Tesauro. Td-gammon: A self-teaching backgammon program. In *Applications of Neural Networks*, pages 267–285. Springer, 1995.

[42] Dustin Tran, Alp Kucukelbir, Adji B. Dieng, Maja Rudolph, Dawen Liang, and David M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

[43] AI Uber. Labs. 2017. pyro, a deep probabilistic programming language.(2017).

[44] Matej Vecerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin A Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR, abs/1707.08817*, 2017.

[45] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end simulated driving. In *AAAI*, pages 2891–2897, 2017.

[46] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.