Abstract

BotStrapping- A multi-expert deep reinforcement learning approach

December 12, 2018

In this work, we make use of demonstration data from multiple experts to pre-train the agent so that it can perform well in the task from the start of learning, and then continue improving from its own self-generated data. Enabling learning in this framework opens up the possibility of applying RL to many real world problems where we have multiple experts' data available but accurate simulators dont exist.

Our method is closest to the recent approach - **Deep Q-Learning From Demonstrations (DQfD)** which combines demonstrations with reinforcement learning. DQfD improves learning speed on Atari, including a margin loss which encourages the expert actions to have higher Q-values than all other actions. This loss can make improving upon the demonstrator policy impossible which is not the case for our method. Prior work has previously explored improving beyond the demonstrator policy but by using only one expert's data. While our approach is a **multi-expert approach** where we are trying to see whether the agent is able to **explore quickly** by building upon the knowledge gained from the experts data since all the experts have been trained using a different technique and are expected to have a different policy to perform exploration and exploitation. While previous work focuses on **beating the experts it learned from**, we show that we can extend the state of the art in RL with demonstrations by introducing new methods to incorporate demonstrations from multiple experts and quickly converges along with defeating those individual experts.

We evaluated the multi-expert DQfD on **Cart-Pole-v1** setup. For our experiments, we evaluated four different algorithms: **Full DQfD from 2 experts (DDQN and Actor-Critic), DDQN without any demonstration, DQfD from one expert data (DDQN Expert) and DQfD from one expert data (Actor-Critic Expert)**. We first trained two agents, one with DDQN with experience replay buffer and target network and another with Actor Critic setup. Both these experts were trained till their performance reached a decent level but not the best score of **500**. The **AC expert** was stopped when it started to attain a **mean score** of **185.7** with a **SD** of **41.1** whereas the **DDQN expert** was stopped when it started to attain a **mean score** of **336.1** and a **SD** of **74.7** on the Cart-Pole-v1 task. Once we had these two experts at our disposal, we started the training of our agent using DQN but the main difference was that the agent was pre-trained on the roll-outs of these two agents before actually interacting with the environment. We had to be careful about the number of roll-outs we were stacking together of those two experts. If they are not approximately similar in count then our agent will tend to adapt to the policy of the expert with more number of roll-outs present in the buffer memory.

From the results shown in the paper below we can clearly see that the **DQfD with multi-expert outperforms the vanilla DDQN by a fair margin**. Also one thing to notice here is that the **vanilla DDQN model is never able to achieve the highest score** of **500** *(not even once)* whereas our **DQfD with two experts(AC and DDQN)** where both these experts had an average score of **150-300**, is able to achieve a **max score of 500**. Which means it not only beats the vanilla DDQN by a huge margin but it also **outperforms those individual experts** fairly well.

Another key finding was that **DQfD is able to learn faster when it is pre-trained on data from several experts as compared to when its shown data from just one expert**. In Figure 2 it shows that our DQfD from multi-experts achieves a score of **300-350** pretty soon and then keeps on gradually increasing and stabilizes around **450-500**. The single expert trained DQfD also eventually reaches that range but it takes it more time to build upon the knowledge gained from single expert.

The same thing can be observed in Figure 4 where we compare DQfD on one expert (DQN) v/s DQfD from multi-expert. Even here we can observe the same behavior although **not as evident** as in Figure 2 **because our DQfD from DQN expert had an average score of 350 which means it was a really good expert** in itself as compared to the Actor-Critic expert.

BotStrapping: A Multi-expert Deep Reinforcement Learning Approach

Vikramank Singh
vikramank@berkeley.edu

Maksim Ivanov
m0597@berkeley.edu

Abstract

Deep reinforcement learning (RL) has achieved several high profile successes in difficult decision-making problems. However, these algorithms typically require a huge amount of data before they reach reasonable performance. Performing Imitation learning provides a simple and cheap way to perform supervised learning on control tasks. However, it is well known that the generalization ability of this technique is highly constrained by number of samples and is vulnerable to covariate shifts. This paper attempts to leverage multi-experts' experience and use reinforcement learning to train a learning agent with the goal of agent outperforming the individual experts that it learned from.

Introduction

Over the past few years, there have been a number of successes in learning policies for sequential decision-making problems and control. Notable examples include deep model-free Q-learning for general Atari game-playing (Mnih et al. 2015), end-to-end policy search for control of robot motors (Levine et al. 2016), model predictive control with embeddings (Watter et al. 2015), and strategic policies that combined with search led to defeating a top human expert at the game of Go (Silver et al. 2016). An important part of the success of these approaches has been to leverage the recent contributions to scalability and performance of deep learning (LeCun, Bengio, and Hinton 2015). The approach taken in (Mnih et al. 2015) builds a data set of previous experience using batch RL to train large convolutional neural networks in a supervised fashion from this data. By sampling from this data set rather than from current experience, the correlation in values from state distribution bias is mitigated, leading to good (in many cases, super-human) control policies.

It still remains difficult to apply these algorithms to real world settings such as data centers, autonomous vehicles (Hester and Stone 2013), helicopters (Abbeel et al. 2007), or recommendation systems (Shani, Heckerman, and Brafman 2005). Typically these algorithms learn good control policies only after many millions of steps of very poor performance in simulation. This situation is acceptable when there is a perfectly accurate simulator; however, many real world problems do not come with such a simulator. Instead, in these situations, the agent must learn in the real domain with real consequences for its actions, which requires that the agent have good online performance from the start of learning. While accurate simulators are difficult to find, most of these problems have data of the system operating under a previous controller (either human or machine) that performs reasonably well but are not best at that given task.

In this work, we make use of this demonstration data from multiple experts to pre-train the agent so that it can perform well in the task from the start of learning, and then continue improving from its own self-generated data. Enabling learning in this framework opens up the possibility of applying RL to many real world problems where we have multiple experts' data available but accurate simulators don't exist.

Background

We adopt the standard Markov Decision Process (MDP) formalism for this work (Sutton and Barto 1998). An MDP is defined by a tuple which consists of a set of states S, a set of actions A, a reward function R(s, a), a transition function $T(s, a, s) = P(s|s, a)$, and a discount factor γ. In each state s ϵ S, the agent takes an action a ϵ A. Upon taking this action, the agent receives a reward R(s, a) and reaches a new state s, determined from the probability distribution $P(s|s, a)$. A policy π specifies for each state which action the agent will take. The goal of the agent is to find the policy π mapping states to actions that maximizes the expected discounted total reward over the agentâĂŹs lifetime. The value $Q_\pi(s, a)$of a given state-action pair (s, a) is an estimate of the expected future reward that can be obtained from (s, a) when following policy π. The optimal value function $Q_*(s, a)$ provides maximal values in all states and is determined by solving the Bellman equation:

$$Q_*(s, a) = E[R(s, a)+\gamma \sum P(s'/s, a) max_{a'} Q_*(s', a')]$$

The optimal policy π is then $(s) = argmax_{a\epsilon A} Q(s, a)$. DQN (Mnih et al. 2015) approximates the value function $Q(s, a)$ with a deep neural network that outputs a set of action values $Q(s, ;)$ for a given state input s, where θ are the parameters of the network. There are two key components of DQN that make this work. First, it uses a separate target

network that is copied every τ steps from the regular network so that the target Q-values are more stable. Second, the agent adds all of its experiences to a replay buffer D^{replay}, which is then sampled uniformly to perform updates on the network.

The double Q-learning update (van Hasselt, Guez, and Silver 2016) uses the current network to calculate the argmax over next state values and the target network for the value of that action. The double DQN loss is -

$$J_{DQ}(Q) = (R(s,a) + \gamma Q(s_{t+1}, a_{t+1}^{max}; \theta') - Q(s,a;\theta))^2$$

where θ' are the parameters of the target network, and $a_{t+1}^{max} = argmax_a Q(st+1, a; \theta)$. Separating the value functions used for these two variables reduces the upward bias that is created with regular Q-learning updates.

Prioritized experience replay (Schaul et al. 2016) modifies the DQN agent to sample more important transitions from its replay buffer more frequently. The probability of sampling a particular transition i is proportional to its priority, $P(i) = p_i^a / \sum_k p_k^a$, , where the priority $pi = \delta_i + \epsilon$, and δ_i is the last TD error calculated for this transition and ϵ is a small positive constant to ensure all transitions are sampled with some probability. To account for the change in the distribution, updates to the network are weighted with importance sampling weight, $w_i = (1/N.1/P_i)^\beta$ where N is the size of the replay buffer and β controls the amount of importance sampling with no importance sampling when $\beta = 0$ and full importance sampling when β = 1. β is annealed linearly from β_0 to 1.

Related Work

Learning methods for decision making problems such as robotics largely divide into two classes: imitation learning and reinforcement learning (RL). In imitation learning (also called learning from demonstrations) the agent receives behavior examples from an expert and attempts to solve a task by copying the expert's behavior. In RL, an agent attempts to maximize expected reward through interaction with the environment. Our work combines aspects of both to solve complex tasks.

In Imitation Learning, the most common form is behavior cloning (BC), which learns a policy through supervised learning on demonstration state action pairs. BC has seen success in autonomous driving [5], [6], quadcopter navigation [7], locomotion [8], [9]. BC struggles outside the manifold of demonstration data. Dataset Aggregation (DAGGER) augments the dataset by interleaving the learned and expert policy to address this problem of accumulating errors [10]. However, DAGGER is difficult to use in practice as it requires access to an expert during all of training, instead of just a set of demonstrations.

Fundamentally, BC approaches are limited because they do not take into account the task or environment. Inverse reinforcement learning (IRL) [11] is another form of imitation learning where a reward function is inferred from the demonstrations. Among other tasks, IRL has been applied to navigation [12], autonomous helicopter flight [13], and manipulation [14]. Since our work assumes knowledge of a reward function, we omit comparisons to IRL approaches.

Reinforcement learning methods have been harder to apply in robotics, but are heavily investigated because of the autonomy they could enable. Through RL, robots have learned to play table tennis [15], swing up a cartpole, and balance a unicycle [16]. A renewal of interest in RL cascaded from success in games [17], [18], especially because of the ability of RL with large function approximators (ie. deep RL) to learn control from raw pixels. Robotics has been more challenging in general but there has been significant progress. Deep RL has been applied to manipulation tasks [19], grasping [20], [21], opening a door [22], and locomotion [23], [24], [25]. However, results have been attained predominantly in simulation per high sample complexity, typically caused by exploration challenges.

Block stacking has been studied from the early days of AI and robotics as a task that encapsulates many difficulties of more complicated tasks we want to solve, including multi-step planning and complex contacts. SHRDLU [26] was one of the pioneering works, but studied block arrangements only in terms of logic and natural language understanding. More recent work on task and motion planning considers both logical and physical aspects of the task [27], [28], [29], but requires domain specific engineering. In this work we study how an agent can learn this task without the need of domain-specific engineering.

One RL method, PILCO [16] has been applied to a simple version of stacking blocks where the task is to place a block on a tower [3]. Methods such as PILCO based on learning forward models naturally have trouble modelling the sharply discontinuous dynamics of contacts; although they can learn to place a block, it is a much harder problem to grasp the block in the first place. One-shot Imitation [4] learns to stack blocks in a way that generalizes to new target configurations, but uses more than 100,000 demonstrations to train the system. A heavily shaped reward can be used to learn to stack a Lego block on another with RL [30]. In contrast, our method can succeed from fully sparse rewards and handle stacking several blocks.

Combining RL and Imitation Learning: Previous work has combined reinforcement learning with demonstrations. Demonstrations have been used to accelerate learning on classical tasks such as cart-pole swing-up and balance [31]. This work initialized policies and (in model-based methods) initialized forward models with demonstrations. Recently, demonstration data has been shown to help in difficult exploration problems in RL (Subramanian, Jr., and Thomaz 2016). There has also been recent interest in this combined imitation and RL problem. For example, the HAT algorithm transfers knowledge directly from human policies (Taylor, Suay, and Chernova 2011). Follow-ups to this work showed how expert advice or demonstrations can be used to shape rewards in the RL problem (Brys et al. 2015; Suay et al. 2016). A different approach is to shape the policy that is used to sample experience (Cederborg et al. 2015), or to use policy iteration from demonstrations (Kim et al. 2013; Chemali and Lezaric 2015).

Our algorithm works in a scenario where rewards are

given by the environment used by the demonstrator. This framework was appropriately called Reinforcement Learning with Expert Demonstrations (RLED) in (Piot, Geist, and Pietquin 2014a) and is also evaluated in (Kim et al. 2013; Chemali and Lezaric 2015). Our setup is similar to (Piot, Geist, and Pietquin 2014a) in that we combine TD and classification losses in a batch algorithm in a model-free setting; ours differs in that our agent is pre-trained on the demonstration data initially and the batch of self-generated data grows over time and is used as experience replay to train deep Q-networks. In addition, a prioritized replay mechanism is used to balance the amount of demonstration data in each mini-batch. (Piot, Geist, and Pietquin 2014b) present interesting results showing that adding a TD loss to the supervised classification loss improves imitation learning even when there are no rewards.

Another work that is similarly motivated to ours is (Schaal 1996). This work is focused on real world learning on robots, and thus is also concerned with on-line performance. Similar to our work, they pre-train the agent with demonstration data before letting it interact with the task. However, they do not use supervised learning to pre-train their algorithm, and are only able to find one case where pre-training helps learning on Cart-Pole.

In one-shot imitation learning (Duan et al. 2017), the agent is provided with an entire demonstration as input in addition to the current state. The demonstration specifies the goal state that is wanted, but from different initial conditions. The agent is trained with target actions from more demonstrations. This setup also uses demonstrations, but requires a distribution of tasks with different initial conditions and goal states, and the agent can never learn to improve upon the demonstrations.

AlphaGo (Silver et al. 2016) takes a similar approach to our work in pre-training from demonstration data before interacting with the real task. AlphaGo first trains a policy network from a dataset of 30 million expert actions, using supervised learning to predict the actions taken by experts. It then uses this as a starting point to apply policy gradient updates during self-play, combined with planning rollouts. Here, we do not have a model available for planning, so we focus on the model-free Q-learning case.

Human Experience Replay (HER) (Hosu and Rebedea 2016) is an algorithm in which the agent samples from a replay buffer that is mixed between agent and demonstration data, similar to our approach. Gains were only slightly better than a random agent, and were surpassed by their alternative approach, Human Checkpoint Replay, which requires the ability to set the state of the environment. While their algorithm is similar in that it samples from both datasets, it does not pre-train the agent or use a supervised loss. Our results show higher scores over a larger variety of games, without requiring full access to the environment. Replay Buffer Spiking (RBS) (Lipton et al. 2016) is another similar approach where the DQN agent's replay buffer is initialized with demonstration data, but they do not pre-train the agent for good initial performance or keep the demonstration data permanently.

The work that relates to ours is a workshop paper presenting Accelerated DQN with Expert Trajectories (ADET) (Lakshminarayanan, Ozair, and Bengio 2016). They are also combining TD and classification losses in a deep Q-learning setup. They use a trained DQN agent to generate their demonstration data, which on most games is better than human data. It also guarantees that the policy used by the demonstrator can be represented by the apprenticeship agent as they are both using the same state input and network architecture. They use a cross-entropy classification loss rather than the large margin loss DQfD uses and they do not pre-train the agent to perform well from its first interactions with the environment.

Our method is closest to the recent approach - Deep Q-Learning From Demonstrations (DQfD) which combines demonstrations with reinforcement learning. DQfD improves learning speed on Atari, including a margin loss which encourages the expert actions to have higher Q-values than all other actions. This loss can make improving upon the demonstrator policy impossible which is not the case for our method. Prior work has previously explored improving beyond the demonstrator policy but by using only one expert's data. While our approach is a multi-expert approach where we are trying to see whether the agent is able to explore quickly by building upon the knowledge gained from the experts data since all the experts have been trained using a different technique and are expected to have a different policy to perform exploration and exploitation. While previous work focuses on beating the expert it learned from, we show that we can extend the state of the art in RL with demonstrations by introducing new methods to incorporate demonstrations from multiple experts and quickly converges along with defeating those individual experts.

DQfD from multiple experts

In many real-world settings of reinforcement learning, we have access to data of the system being operated by its previous controllers, but we do not have access to an accurate simulator of the system. Therefore, we want the agent to learn as much as possible from the demonstration data provided by several experts before running on the real system. The goal of the pre-training phase from multiple experts is to learn to imitate a generalized demonstrator with a value function that satisfies the Bellman equation so that it can be updated with TD updates once the agent starts interacting with the environment. The demonstration data is collected by rolling out the experts and collecting those roll-outs and placing them in the replay buffer. During this pre-training phase, the agent samples mini-batches from the buffer and updates the network by applying four losses: the 1-step double Q-learning loss, an n-step double Q-learning loss, a supervised large margin classification loss, and an L2 regularization loss on the network weights and biases. The supervised loss is used for classification of the demonstrator's actions, while the Q-learning loss ensures that the network satisfies the Bellman equation and can be used as a starting point for TD learning.

The supervised loss is critical for the pre-training to have any effect. Since the demonstration data is necessarily cov-

ering a narrow part of the state space and not taking all possible actions, many state-actions have never been taken and have no data to ground them to realistic values. If we were to pre-train the network with only Q-learning updates towards the max value of the next state, the network would update towards the highest of these ungrounded variables and the network would propagate these values throughout the Q function. We add a large margin classification loss (Piot, Geist, and Pietquin 2014a):

$$J_E(Q) = max_a[Q(s,a) + l(a_E, a)] - Q(s, a_E)$$

Where a_E is the action the expert demonstrator took in state $s, l(a_E, a)$ is a margin function that is 0 when $a = a_E$ and positive otherwise. This loss forces the values of the other actions to be at least a margin lower than the value of the demonstrator's action. Adding this loss grounds the values of the unseen actions to reasonable values, and makes the greedy policy induced by the value function imitate the demonstrator. If the algorithm pre-trained with only this supervised loss, there would be nothing constraining the values between consecutive states and the Q-network would not satisfy the Bellman equation, which is required to improve the policy on-line with TD learning.

Adding n-step returns (with n = 10) helps propagate the values of the expert's trajectory to all the earlier states, leading to better pre-training. The n-step return is:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{n-1} r_{t+n-1} + max_a \gamma^n Q(s_{t+n}, a),$$

which we calculate using the forward view, similar to A3C (Mnih et al. 2016). We also add an L2 regularization loss applied to the weights and biases of the network to help prevent it from over-fitting on the relatively small demonstration dataset. The overall loss used to update the network is a combination of all four losses:

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q)$$

The λ parameters control the weighting between the losses. We examine removing some of these losses in Section.

Once the pre-training phase is complete, the agent starts acting on the system, collecting self-generated data, and adding it to its replay buffer D_{replay}. Data is added to the replay buffer until it is full, and then the agent starts overwriting old data in that buffer. However, the agent never over-writes the demonstration data. For proportional prioritized sampling, different small positive constants, ϵ_a and ϵ_d, are added to the priorities of the agent and demonstration transitions to control the relative sampling of demonstration versus agent data. All the losses are applied to the demonstration data in both phases, while the supervised loss is not applied to self-generated data ($\lambda_2 = 0$).

Overall, Deep Q-learning from Demonstration (DQfD) differs from PDD DQN in six key ways:

1. Demonstration data: DQfD is given a set of demonstration data, which it retains in its replay buffer permanently.
2. Pre-training: DQfD initially trains solely on the demonstration data before starting any interaction with the environment.
3. Supervised losses: In addition to TD losses, a large margin supervised loss is applied that pushes the value of the demonstrator's actions above the other action values (Piot, Geist, and Pietquin 2014a).
4. L2 Regularization losses: The algorithm also adds L2 regularization losses on the network weights to prevent overfitting on the demonstration data.
5. N-step TD losses: The agent updates its Q-network with targets from a mix of 1-step and n-step returns.
6. Demonstration priority bonus: The priorities of demonstration transitions are given a bonus of ϵ_d, to boost the frequency that they are sampled.

Pseudo-code is sketched in Algorithm 1. The behavior policy $\pi^{\epsilon Q_\theta}$ is ϵ-greedy with respect to Q_θ.

Algorithm 1 DQfD from Multiple Experts

Start training your experts but do early stopping to keep them at a mediocre level.
Once trained, collect roll-outs from these experts *(in same ratio)*
Store them in the replay buffer and shuffle
Inputs: D^{replay}: : initialized with above created demonstration data set, θ: weights for initial behavior network (random), θ': weights for target network (random), τ: frequency at which to update target net, k: number of pre-training gradient updates
for steps t ϵ 1,2,3...k **do**
    Sample a mini-batch of n transitions from D^{replay} with prioritization
    Calculate loss J(Q) using target network
    Perform a gradient descent step to update θ
    if t $\mod \tau$=0 **then** $\theta' \leftarrow \theta$
for steps t ϵ 1,2,3.. **do**
    Sample action from behavior policy a $\pi^{\epsilon Q_\theta}$
    Play action a and observe (s',r)
    Store (s, a, r, s') into D^{replay}, overwriting oldest self-generated transition if over capacity
    Sample a mini-batch of n transitions from Dreplay with prioritization
    Calculate loss J(Q) using target network
    Perform a gradient descent step to update θ
    if t $\mod \tau$=0 **then** $\theta' \leftarrow \theta$
    s $\leftarrow$ s'

Experimental Setup

We conducted our experiments in two parts as described below respectively.

Part 1: Behavioral Cloning on multi-expert data

In order to attempt learning from different experts, we firstly need experts that are meaningfully different. The first question we face in our thesis is what does it mean to have "different experts" that solve the same task. For example, if two

experts are trained using the same model started from different random states, they will probably converge to a similar result, and learning from them would not be far removed from using plaid DAGGER with a single expert. Indeed, we are more interested in learning from agents that are similar in some metrics but work through the task using sufficiently different methods.

In an attempt to create such a setting, we decided to first train our own experts with RL but using different models. We used **HalfCheetah-v2** environment in OpenAI Gym to conduct our experiments. First, we trained one expert with Actor-Critic and another expert with Model-Based RL, with both achieving a return of roughly 700 over 20 roll-outs.

	Random Policy	A-C	Model-Based RL
Average Return	-318.4	671.2	705.3
Standard Dev. of Return	44.6	33.4	81.22

Thus we have our two experts with similarly decent performance (though not great, as TRPO experts have achieved 4000+ mean rewards on this task). Initally, we get our baselines by running behavioral cloning on each of these experts individually.

Note that while A-C model achieves a return of 671, the policy it learns is to tip the Half-Cheetah forward on its back and wiggle the legs while upside down to generate momentum and scoot forward. Unsurprisingly, the behavioral clone of this model learns this behavior, achieving an average return of 507 with standard deviation of 31. We suspect that this yields a decent reward because the half-cheetah remains more stable on its back, and it doesn't matter how precise the leg movements are as long as they generate forward momentum.

The MBRL expert doesn't tip forward as the cost function specifically penalizes that, so the behavioral clone doesn't imitate this behavior. However, we notice that the behavioral clone of MBRL expert doesn't perform well, achieving an average return of around 0, with standard deviation of around 3. This can be explained by the fact that since the Half-Cheetah stays upright, it matters much more how its legs are moved, and it is more vulnerable to covariate shift in states than the A-C expert that predictably tips over and stays upside down.

What happens when we try to clone both of these experts at the same time? The clone trained on data combined from two experts stays upring, and achieves an average return of 182 and standard deviation of 22. In terms of reward, this result is better than clonging MBRL experts alone, but worse than cloning the A-C expert by itself. However, as detailed above, A-C expert achieves its reward by "cheating" - tipping over on its back, thus sabotaging the goal of learning how to run. While worse than A-C clone in terms of reward, the new clone is better than MBRL clone in reward and significantly better at achieving the **actual goal of running** than either of the baseline clones.

In conclusion, the combination of two different experts did not yield absolute reward improvement, however we find the overall result promising - analytically, we were able to take an MBRL expert, who is hard to imitate due to covariate shift, augment its data with A-C expert, that gets high reward by cheating the reward function, and get a clone that performs worse in terms of reward than the cheater but better in terms of imitating actual running.

Part 2: DQfD from Experts

We evaluated the multi-expert DQfD on **Cart-Pole-v1** setup provided by OpenAI in the gym environment. Gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no assumptions about the structure of your agent, and is compatible with any numerical computation library, such as TensorFlow or Theano. The gym library is a collection of test problems - environments - that you can use to work out your reinforcement learning algorithms. These environments have a shared interface, allowing you to write general algorithms.

In the Cart-Pole environment a pole is attached by an unactuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

For all of our experiments, we evaluated four different algorithms:

1. Full DQfD from 2 experts (DDQN and Actor-Critic)
2. DDQN without any demonstration
3. DQfD from one expert data (DDQN Expert)
4. DQfD from one expert data (Actor-Critic Expert)

We trained two experts on Cart-Pole-v1; a DQN expert with mean return of 185.7 and a standard deviation of 41.1 and another an Actor Critic expert with mean return of 336.1 and a standard deviation of 74.7

We used informal parameter tuning for all the algorithms and used the same parameters for the all the experiments we ran. Our coarse search over prioritization and n-step return parameters led to the same best parameters for DQfD and DDQN. For the supervised imitation comparison, we performed supervised classification of the demonstratorâĂŹs actions using a cross-entropy loss, with the same network architecture and L2 regularization used by DQfD. The imitation algorithm did not use any TD loss. Imitation learning only learns from the pre-training and not from any additional interactions.

We first trained two agents, one with DDQN with experience replay buffer and target network and another with Actor Critic setup. Both these experts were trained till they performance reached a decent level but not the best score of **500**. The **AC expert** was stopped when it started to attain a mean score of **185.7** with a SD of **41.1** whereas the **DDQN expert** was stopped when it started to attain a mean score of **336.1** and a SD of **74.7** on the Cart-Pole-v1 task. Once we had these two experts at our disposal, we started the training of our agent using DQN but the main difference was that the agent was pre-trained on the roll-outs of these two agents before actually interacting with the environment. We had to be careful about the number of roll-outs we were stacking

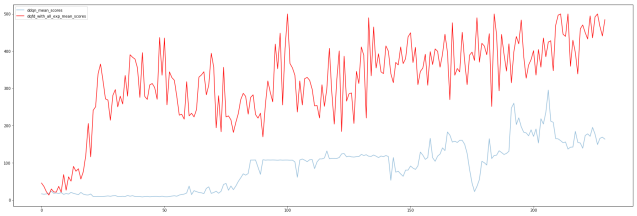

Figure 1: Vanilla DQN v/s DQfD from multi-expert

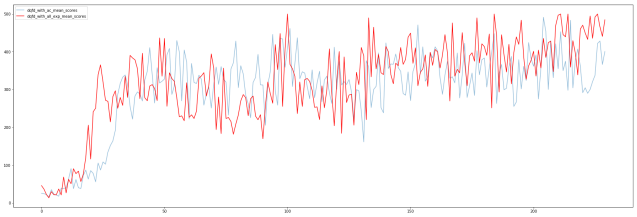

Figure 2: DQfD on AC expert v/s DQfD on multi-expert

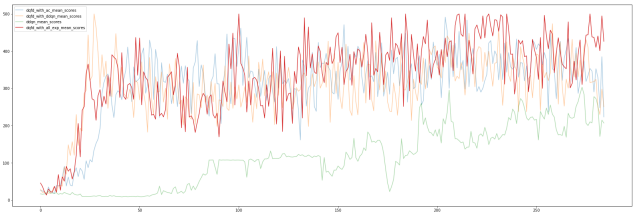

Figure 3: Vanilla DQN v/s DQfD from single and multi-expert

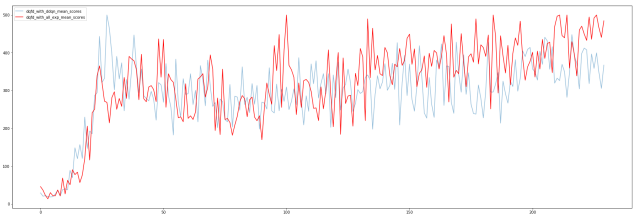

Figure 4: DQfD on DQN expert v/s DQfD on multi-expert

together of those two experts. If they are not approximately similar in count then our agent will tend to adapt to the policy of the expert with more number of roll-outs present in the buffer.

Once the roll-outs from both these agents are collected and stored in the replay buffer of the agent then we start training out agent with DDQN. DQfD learns from a very small dataset compared to other similar work, as AlphaGo (Silver et al. 2016) learns from **30 million** human transitions, and DQN (Mnih et al. 2015) learns from over **200 million** frames. DQfD's smaller demonstration dataset makes it more difficult to learn a good representation without overfitting. In our case, our agent learns just from approximately **5000** roll-outs of expert data during the pre-training phase.

We found that our agent once trained on multiple experts learns to explore quickly as compared to the one which is trained only on DQN without any expert data. It reaches the score of **300** within first 50 steps as compared to the vanilla DQN agent which takes more than **250** steps to touch 300. Also, one more thing to notice here is that our agent, which is trained on multiple experts that had average scores of **150-300**, is able to achieve a score of **500** after **200** steps thus beating all the individual experts it learned from.

Results

First, we show learning curves in Figure 1 for an agent that is trained only using DDQN without any expert demonstrations (in blue) v/s the one that is trained using DQfD from 2 experts data (Actor Critic and DDQN). Clearly the DQfD with multi-expert outperforms the vanilla DDQN by a fair margin. Also one thing to notice here is that the vanilla DDQN model is never able to achieve the highest score of 500 (not even once) whereas our DQfD with two experts (AC and DDQN) where both these experts had an average score of 150-300, is able to achieve a max score of 500. Which means it not only beats the vanilla DDQN by a huge margin but it also outperforms those individual experts fairly well. In Figure 2 we are showing a comparison of DQfD from only one expert i.e Actor Critic v/s DQfD from two experts (AC and DDQN). Here the idea is to observe whether our DQfD performs or learns better if it is trained on expert data from multiple experts v/s only one expert. Here also we can clearly notice that DQfD is able to learn faster when it is pre-trained on data from several experts as compared to when it's shown data from just one expert. In Figure 2 it shows that our DQfD from multi-experts achieves a score of 300-350 pretty soon and then keeps on gradually increasing and stabilizes around 450-500. The single expert trained DQfD also eventually reaches that range but it takes it more time to build upon the knowledge gained from single expert.

The same thing can be observed in Figure 4 where we compare DQfD on one expert (DQN) v/s DQfD from multi-expert. Even here we can observe the same behavior although not as evident as in Figure 2 because our DQfD from DQN expert was able to achieve an average score of 350 which means it was a really good expert in itself as compared to the Actor-Critic expert.

Figure 3 is a comparison of agents trained on single expert v/s multi-experts v/s vanilla DQN. This graphs shows a clear picture of the point we are trying to make. Vanilla DQN is nowhere near to the ones that are trained on DQfD from expert data. However, also among those the one which is trained on multi-expert data is able to perform well than the ones trained on single expert data.

DQfD from multiple experts here achieves these results despite having a very small amount of demonstration data (5,000 per expert) that can be easily generated in just a few minutes of gameplay. DQN and DQfD receive three orders of magnitude more interaction data for RL than demonstration data. DQfD demonstrates the gains that can be achieved by adding just a small amount of demonstration data with the right algorithm. As the related work comparison shows, naively adding (e.g. only pre-training or filling the replay

buffer) this small amount of data to a pure deep RL algorithm does not provide similar benefit and can sometimes be detrimental.

These results may seem obvious given that agent has access to privileged data from multiple experts, but the rewards and demonstrations are mathematically dissimilar training signals, and naive approaches to combining them can have disastrous results. Simply doing supervised learning on the experts' demonstrations is not successful, while DQfD learns to out-perform all these demonstrators. DQfD also outperforms three prior algorithms for incorporating demonstration data into DQN. We argue that the combination of all four losses during pre-training is critical for the agent to learn a coherent representation that is not destroyed by the switch in training signals after pre-training. Even after pre-training, the agent must continue using the expert data.

Discussion

The learning framework that we have presented in this paper is one that is very common in real world problems such as controlling data centers, autonomous vehicles (Hester and Stone 2013), or recommendation systems (Shani, Heckerman, and Brafman 2005). In these problems, typically there is no accurate simulator available, and learning must be performed on the real system with real consequences. However, there is often data available of the system by various already trained mediocre experts. We are presenting a way in which we can leverage techniques like DQfD obtained from multiple experts and learn quickly and efficiently such that it could beat those individual experts.

It first pre-trains solely on demonstration data from multiple experts, using a combination of 1-step TD, n-step TD, supervised, and regularization losses so that it has a reasonable policy that is a good starting point for learning in the task. Once it starts interacting with the task, it continues learning by sampling from both its self generated data as well as the demonstration data. The ratio of both types of data in each mini-batch is automatically controlled by a prioritized-replay mechanism.

Learning from multiple experts is particularly difficult. In most games, imitation learning is unable to perfectly classify the demonstrator's actions even on the demonstration dataset. Different experts may play the games in a way that differs greatly from each other and from a policy that an agent would learn, and may be using information that is not available in the agent's state representation. In future work, we plan to measure these differences between demonstration and agent data to inform approaches that derive more value from the demonstrations. Another future direction is to apply these concepts to domains with continuous actions, where the classification loss becomes a regression loss.

Also, an exciting extension to our approach will be training an agent from multiple experts that are experts but on different tasks. The main question will be whether or not our newly trained agent on the knowledge gathered from these multiple experts is able to generalize well on all those individual tasks. It will be interesting to see how the sharing and cross-functionality of knowledge that agent can learn from one task and apply to another.

References

[Abbeel et al. 2007] Abbeel, P.; Coates, A.; Quigley, M.; and Ng, A. Y. 2007. An application of reinforcement learning to aerobatic helicopter flight. In Advances in *Neural Information Processing Systems (NIPS)*.

[Bellemare et al. 2013] Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. Journal of Artifificial Intelligence Research (JAIR) 47:253âĂŞ279.

[Brys et al. 2015] Brys, T.; Harutyunyan, A.; Suay, H.; Chernova, S.; Taylor, M.; and Now e, A. 2015. Reinforcement learning from demonstration through shaping. In International Joint Conference on Artificial Intelligence (IJCAI).

[Cederborg et al. 2015] Cederborg, T.; Grover, I.; Isbell, C.; and Thomaz, A. 2015. Policy shaping with human teachers. In International Joint Conference on Artificial Intelligence (IJCAI 2015).

[Chemali and Lezaric 2015] Chemali, J., and Lezaric, A. 2015. Direct policy iteration from demonstrations. In International Joint Conference on Artificial Intelligence (IJCAI).

[Duan et al. 2017] Duan, Y.; Andrychowicz, M.; Stadie, B. C.; Ho, J.; Schneider, J.; Sutskever, I.; Abbeel, P.; and Zaremba, W. 2017. One-shot imitation learning. CoRR abs/1703.07326.

[Finn, Levine, and Abbeel 2016] Finn, C.; Levine, S.; and Abbeel, P. 2016. Guided cost learning: Deep inverse optimal control via policy optimization. In International Conference on Machine Learing (ICML).

[Gruslys et al. 2017] Gruslys, A.; Gheshlaghi Azar, M.; Bellemare, M. G.; and Munos, R. 2017. The Reactor: A Sample-Efficient Actor-Critic Architecture. ArXiv e-prints.

[Hester and Stone 2013] Hester, T., and Stone, P. 2013. TEXPLORE: Real-time sample-efficient reinforcement learning for robots. Machine Learning 90(3).

[Ho and Ermon 2016] Ho, J., and Ermon, S. 2016. Generative adversarial imitation learning. In Advances in Neural Information Processing Systems (NIPS).

[Hosu and Rebedea 2016] Hosu, I.-A., and Rebedea, T. 2016. Playing atari games with deep reinforcement learning and human checkpoint replay. In ECAI Workshop on Evaluating General Purpose AI

[Jaderberg et al. 2016] Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2016. Reinforcement learning with unsupervised auxiliary tasks. CoRR abs/1611.05397.

[Kim et al. 2013] Kim, B.; Farahmand, A.; Pineau, J.; and Precup, D. 2013. Learning from limited demonstrations. In Advances in Neural Information Processing Systems (NIPS).

[Lakshminarayanan, Ozair, and Bengio 2016] Lakshminarayanan, A. S.; Ozair, S.; and Bengio, Y. 2016. Reinforcement learning with few expert demonstrations. In NIPS Workshop on Deep Learning for Action and Interaction.

[LeCun, Bengio, and Hinton 2015] LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. Nature 521(7553):436-444.

[Levine et al. 2016] Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-end training of deep visuomotor policies. Journal of Machine Learning (JMLR) 17:1âĂŞ40.

[Lipton et al. 2016] Lipton, Z. C.; Gao, J.; Li, L.; Li, X.; Ahmed, F.; and Deng, L. 2016. Efficient exploration for dialog policy learning with deep BBQ network replay buffer spiking. CoRR abs/1608.05081.

[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. Nature 518(7540):529-533.

[Mnih et al. 2016] Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In International Conference on Machine Learning, 1928-1937.

[Ostrovski et al. 2017] Ostrovski, G.; Bellemare, M. G.; van den Oord, A.; and Munos, R. 2017. Count-based exploration with neural density models. CoRR abs/1703.01310.

[Piot, Geist, and Pietquin 2014a] Piot, B.; Geist, M.; and Pietquin, O. 2014a. Boosted bellman residual minimization handling expert demonstrations. In European Conference on Machine Learning (ECML).

[Piot, Geist, and Pietquin 2014b] Piot, B.; Geist, M.; and Pietquin, O. 2014b. Boosted and Reward-regularized Classification for Apprenticeship Learning. In International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[Ross, Gordon, and Bagnell 2011] Ross, S.; Gordon, G. J.; and Bagnell, J. A. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In International Conference on Artificial Intelligence and Statistics (AISTATS).

[Schaal 1996] Schaal, S. 1996. Learning from demonstration. In Advances in Neural Information Processing Systems (NIPS).

[Schaul et al. 2016] Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized experience replay. In Proceedings of the International Conference on Learning Representations, volume abs/1511.05952.

[Shani, Heckerman, and Brafman 2005] Shani, G.; Heckerman, D.; and Brafman, R. I. 2005. An mdp-based recommender system. Journal of Machine Learning Research 6:1265-1295.

[Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529:484-489.

[Suay et al. 2016] Suay, H. B.; Brys, T.; Taylor, M. E.; and Chernova, S. 2016. Learning from demonstration for shaping through inverse reinforcement learning. In International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[Subramanian, Jr., and Thomaz 2016] Subramanian, K.; Jr., C. L. I.; and Thomaz, A. 2016. Exploration from demonstration for interactive reinforcement learning. In International Conference on Autonomous Agents and Multiagent Systems (AAMAS).

[Sun et al. 2017] Sun, W.; Venkatraman, A.; Gordon, G. J.; Boots, B.; and Bagnell, J. A. 2017. Deeply aggrevated: Differentiable imitation learning for sequential prediction. CoRR abs/1703.01030.

[Sutton and Barto 1998] Sutton, R. S., and Barto, A. G. 1998. Introduction to reinforcement learning. MIT Press.

[Syed and Schapire 2007] Syed, U., and Schapire, R. E. 2007. A game-theoretic approach to apprenticeship learning. In Advances in Neural Information Processing Systems (NIPS).

[Syed, Bowling, and Schapire 2008] Syed, U.; Bowling, M.; and Schapire, R. E. 2008. Apprenticeship learning using linear programming. In International Conference on Machine Learning (ICML).