Imitation Learning On Atari Games Using Generative Adversarial Imitation Learning

Usman Anwar
MSDS19001
Information Technology University, Lahore.
usman.anwar@itu.edu.pk

Abstract

Specification of a reward function which aligns with the intentions of human users is a difficult task in reinforcement learning. To circumvent this issue, various methods have been proposed in the literature with the objective of implicitly inferring the reward function from the examples of expert behaviour. However, these methods often suffer from drawbacks such as lack of robustness, difficulty in optimization and high computational burden. Generative Adversarial Imitation Learning (GAIL) subverts these issues by posing the task of learning from demonstrations as an adversarial game between a generator policy network (which learns to imitate expert loss) and a discriminator network (which learns to differentiate between the samples from expert policy and generator policy and hence implicitly capture the reward function of expert). We use GAIL to learn to play two Atari games, Breakout and Pong. Our results are competitive with the state of the art. Further, we use gradient based class activation mapping to interpret the actions chosen by the policy network.

1. Introduction

Reinforcement learning (RL) is a framework for learning policies for sequential decision making in complex, potentially stochastic, environments. There are two chief methods to learn the policy for a given system, characterized as a Markov Decision Process (MDP). In first method, which we refer to as forward reinforcement learning, a reward function is formed and expected reward is maximized. In second method, called imitation learning, while no reward function is provided, samples are available from expert policy and the objective then is to infer a policy which *imitates* expert policy.

While forward reinforcement learning works well when a reward function is easy to get by; in real life problems, such as self driving cars and interactive robotics, engineering a reward function which accurately captures all the desirable aspects of agent's actions and is easy to optimize is generally very hard. However, for such problems, collecting demonstrations from an expert is often quite feasible and can be done cheaply. This makes imitation learning an attractive method for such problems.

Atari games are a challenging environment for testing fidelity of RL algorithms. In recent years, tremendous gains have been made on this task, in particular by DeepMind, finally culminating in Agent57 [2] which achieved superhuman performance on all Atari games. Agent57 trains multitude of policies on the spectrum hedged by fully exploitative policy and fully exploratory policy and uses curiosity driven exploration. However, sample efficiency remains a concern as Agent57 needed as many as 90 Billion frames of Atari games to achieve this result; a unrealistically large number of samples.

Interpretability is currently an active research area and has profound implications for real world deployment of reinforcement learning based applications. There are largely two ways to make a data driven application interpretable. First class of methods, for example, attention mechanisms, bake the interpretability in the algorithm. Second class of methods, such as sensitivity analysis, does not tinker the learning algorithm but instead looks to explain the algorithm directly at time of evaluation. While the native work in interpretable reinforcement learning generally focuses on the first class of methods, we note that in case of discrete action space, as in Atari, policy network can be viewed as a classification network selecting a particular action. We leverage this view to apply the interpretability methods developed for classification purposes on our task.

2. Related Work

The most primitive form of imitation learning is behaviour cloning [8] in which problem is essentially treated as a supervised task. Behaviour cloning has a fatal flaw that it does not account for the fact that it is not individual actions but the set of actions taken successively in demonstrations which have high reward. Hence, policies learned by behaviour cloning often do not generalize well and drift error is quite prominent in such policies.

As an alternative, [1] propose learning a reward function from the expert demonstrations first and then training a policy under the learned reward function. However, due to the fact that the number of demonstrations is finite, there is inherent ambiguity in learning a reward function as a policy may be optimal for many different reward functions. [15] propose to eliminate this ambiguity by learning a reward function such that the stochastic policy learned under that reward function has maximum entropy. While the formulation of [15] guarantees learning of maximum entropy policy under linear reward function only, subsequent works have shown that in practice explicitly maximizing the entropy of policy via dual gradient descent also works well for example [13] uses neural network as a function approximator and observes better results than linear maximum entropy algorithm due to the ability of neural network to express reward function as complex non linear functions. In [5], neural network is used to represent both the policy network and cost function and near optimal trajectories can be recovered from the expert's samples.

However, despite the progress on low dimensional continuous tasks, imitation learning on Atari is still a far cry from being characterized as a success. This is primarily due to the fact that learning from raw pixels creates an additional layer of complexity. Further, state space of Atari is quite large meaning that there is a high chance that an agent will drift away from expert distribution. When this happens, agent's performance sharply deteriorates due to covariate shift. To avoid this, it is essential to enable the agent to be able to return back to states observed in demonstrations if it drifts into states that are not seen in expert data. [3] proposes to train an ensemble of imitation learning agents by using a loss function which combines behaviour cloning loss with an uncertainty loss which characterizes the variance within the ensemble. The intuition is that policies within the ensemble will generally agree on states seen in demonstration data (and hence incurring low cost) but will vary greatly far away from expert distribution (and incurring high cost). This encourages the agent to stay within expert distribution and to quickly return to expert distribution if it moves away from it. Soft Q Imitation Learning (SQIL)[9] works on the similar principle where they design a sparse reward function which gives agent a reward of $+1$ if it performs the demonstrated action in given state and a reward of 0 otherwise. They use Q-learning to learn a policy. Design of experience replay is very critical in Q-learning [14]. SQIL keeps half the demonstrated experiences in the replay buffer and half the experiences collected by the agent. As the agent gets better with time, the collected experiences begin to match closely with the demonstrated experiences. This has the effect of decaying of effective reward to zero.

However, despite the empirical success, all the algorithms mentioned here lack the theoretical guarantees of [6] that in regime of infinite data, learned agent exactly matches the occupancy measure of the expert. Further, all these algorithms, including GAIL, lack robustness to distribution shift. One interesting approach for tacking the behaviour shift is [11] which proposes using expert data to recover constraints which complement given nominal reward function.

There is also rich literature on learning from demonstrations which are not optimal, however, we omit the literature review of such works as they are beyond the scope of this project.

3. Background

Reinforcement learning models the world as a Markov Decision Process (MDP) in which agent has to take a series of actions depending on the current state of the world such that the minimum cost is incurred on average while agent performs the given task.

Formally, a MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C}, \gamma)$ consisting of a state space $\mathcal{S}$, action space $\mathcal{A}$, a dynamics model $\mathcal{P}$ which is generally unknown, a cost function[1] and a discount factor $0 < \gamma \leq 1$.

The objective is to minimize the cost function over all possible trajectories.

$$\min_{\pi} \mathbb{E}_{\tau \sim \pi}[C(\tau)] \tag{1}$$

For a particular policy π, With each state, we can associate a value function $V^{\pi}(s)$ which is the expected value of cost that will be incurred if from that state onward policy π ought to be followed. In similar fashion, we can define a state-action value function $Q^{\pi}(s, a)$ which measures the expected cost to be incurred by taking actions a in state s and following the policy π from thereon.

A popular class of methods called value iteration, or policy iteration, are based on the idea that value function of a state must be equal to the expected value of value function of its neighbouring states. Value iteration based methods have guaranteed convergence in tabular case (i.e. discrete state space and discrete action space with known dynamics) but in continuous case these guarantees are lost due to the fact that we have to use function approximation to represent these values. Hence, in continuous cases, or even high dimensional discrete space, a more attractive alternative is to attempt to directly optimize the objective in 1. This is possible to do due to a result known as *Policy Gradient Theorem*.

[1]Cost function and reward function are often used interchangeably in reinforcement learning literature to refer to the same idea of a function which outputs a scalar number showing whether action is good or bad. Cost function is minimized while reward function is maximized.

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[C(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{T} \nabla_\theta \pi_\theta(a_t|s_t) C(\tau)] \quad (2)$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{T} \nabla_\theta \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma C(s_{t'}, a_{t'})] \quad (3)$$

This gradient estimate has a huge amount of variance due to it being a monte carlo estimate but is unbiased. One easy way to reduce is to include value function $V(s)$ as a baseline. The difference thus attained is called 'advantage' A_t. This has the effect of amplifying the gradients when agent's estimate of expected return mismatches the actual return received from the environment.

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[C(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{T} \nabla_\theta \pi_\theta(a_t|s_t)$$

$$\sum_{t'=t}^{T} \gamma(C(s_{t'}, a_{t'}) - V(s_{t'}))] \quad (4)$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=1}^{T} \nabla_\theta \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma A_{t'})]$$

In our experiments in this project, we use an advanced policy gradient method called Proximal Policy Optimization (PPO). It is well known in the policy gradient literature that if the update step is small then policy continuously improves. However, this results in a painfully slow convergence. Hence, there is incentive for designing algorithms which take the largest possible step without losing guarantees on policy improvement. These methods often constrain the KL divergence between the current policy and updated policy. The most popular method in this class is called Trust Region Policy Optimization (TRPO).

$$\min_{\pi_\theta} \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t \right] \quad (5)$$
$$s.t. \text{KL}\left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right] < \epsilon$$

This objective is time consuming to optimize as it requires computation of the inverse of Fisher Information Matrix characterizing the KL divergence between old and new policy. However, [10] notes that following empirically designed objective does the job equally well and gives similar or superior results to TRPO.

$$\mathcal{L}_{PPO} = \mathbb{E}_t\left[\min(r_t \cdot A_t, \texttt{clip}(r_t, 1-\epsilon, 1+\epsilon) \cdot A_t)\right] \quad (6)$$

Imitation Learning: A central problem in reinforcement learning is specification of a reward function. Because of the complex nature of interactions that occur between an agent and an environment, it is easy to design a reward function with 'holes' or mis-specifications which may result in optimal agent doing things such as compromising safety. To avoid the problem of mis-specification of reward function, a popular approach is to use expert demonstrations such as from human or a first principle based controller to specify optimal behaviour and directly learn to imitate the expert policy. While there is inherent mathematical ambiguity in imitation learning framework due to ill posed nature of the problem; by and large, most approaches try to learn a policy which performs close to expert policy according to some metric M.

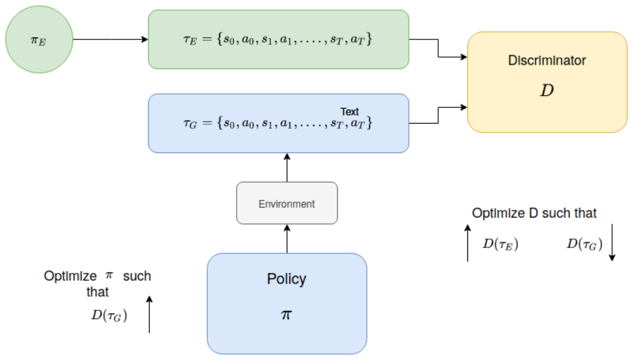

Figure 1. The network is optimized via a multi scale optimization procedure in which first discriminator is updated and then using that discriminator as a proxy to cost function, policy network is optimized.

$$\min_{\pi_\theta} E_{\tau\sim\pi_E}[M(\tau)] - E_{\tau\sim\pi_\theta}[M(\tau)] \quad (7)$$

The most primitive approach of imitation learning, called behavioural cloning, treats it as a supervised learning problem and trains an agent to match the expert's action in a given state. However, expert demonstrations do not cover the entire state space. This results in areas which are 'blind spot' for the agent. An easy fix for this approach is to allow the agent to query human about the optimal action in an arbitrary state; an approach called Data Aggregation (DAGGER). However, humans often are not aware of optimal action on per state basis rather act optimally over the whole trajectory. Further, a common problem that is faced by imitation learning is that expert demonstrations are often only near optimal and not exactly optimal. This supervised learning based approach is further limited by the fact that it views a sequential decision making process as an instance decision making process and loses to leverage any temporal structure present in the demonstrations.

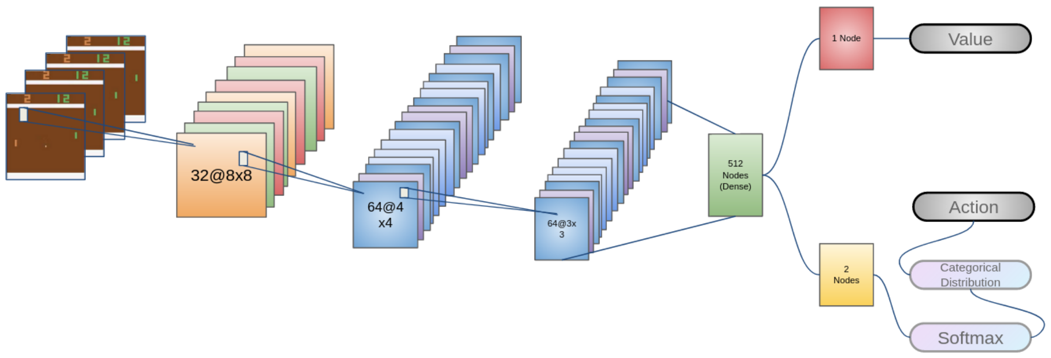

Figure 2. Policy network for Pong consists of three layers of convolution filters. First layer contains 32 8×8 filters. Second layer consists of 64 4×4 filters. Third layer also contains 64 filters but of dimensions 3×3. After the convolution layers, activations are flattened and passed through a dense layer of 512 nodes. From here activations are fed separately to predict state value $V(s)$ and the action a.

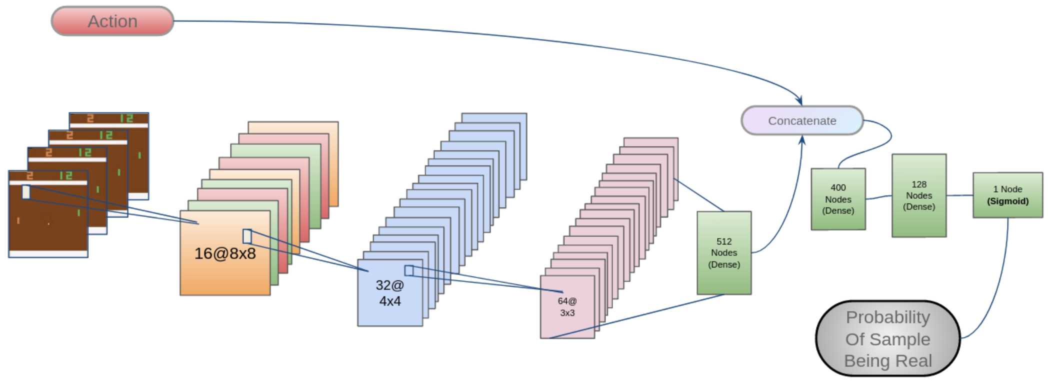

Figure 3. Discriminator has similar structure of convolution layers with half the number of filters in each corresponding layer. We concatenate action with the output from dense layer containing 512 nodes which is then passed through a series of dense layers to finally produce a number in range $[0, 1]$.

4. Data

Atari games consist of frames of dimensions $210 \times 160 \times 128$ where 128 is the range of color palette. In order to minimize computation, we instead convert frames into RGB format and then further use max pooling over channel dimension. Further, consecutive frames typically have a great amount of redundancy, hence, we leverage this redundancy and only sample every $4th$ frame. Frames are downsampled to 84×84 to further minimize computation burden. In games where multiple lives are available, we consider an end of episode only when an agent has exhausted all of its life. This ensures that all states remain reachable in principle. To ensure that markov assumption holds true, we consider a state as a sequence of set number of observations. In games, where start of game requires an action from the player, we sample the initial state by initiating game by taking some random action.

In order to train an imitation learning algorithm, we require data from expert policy. This expert can either be a human or some other algorithm driven code which achieves a level of performance identical to or superior to an expert human. For the experiments in this project, we use expert trained via Proximal Policy Optimization algorithm available in open source RL Zoo github repository. We treat the number of expert trajectories needed as a hyperparameter.

Further the data for training the policy network in GAIL is acquired from Atari emulator and same preprocessing step as mentioned in start of section are followed.

5. Methods

Formally, we consider a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, P, c, \gamma)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is set of actions, P is a transition model mapping state action pairs to next transitioned state and $c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a cost function which returns a scalar corresponding to cost of a particular action. Note that contemporary literature also refers to negative of cost function as a reward function and the two terms are largely interchangeable.

We assume that there exists an expert with full knowledge of the MDP, including the reward function, and has an optimal policy π_E from which samples can be drawn. We further assume that expert can only be queried once and the samples from expert policy, called demonstrations, are then stored in the form of a dataset of demonstrations $\mathcal{D}$. Given the dataset $\mathcal{D}$ and the MDP minus the cost function, the objective is to recover a policy π which closely *imitates* the policy π_E.

Algorithm 1 GAIL

```
Require: Expert trajectories $\tau_E \sim \pi_E$, initial policy network parameters $\theta_0$ and discriminator parameters $w_0$.
  for $i = 1, 2, 3, ....$ do
    Sample trajectories from the current policy network $\tau_i \sim \pi_{\theta_i}$.
    Update the discriminator parameters from $w_i$ to $w_{i+1}$ with the gradient

    $\mathbb{E}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \mathbb{E}_{\tau_E}[\nabla_w \log(1 - D_w(s,a)]$

    Update the parameters of policy network with procedure PPO_Update() using $\tau_i$ as trajectories, $\theta_{i-1}$ as the old parameters & value network parameters and $\log(D_w)$ as the cost function.
  end for
```

Algorithm 2 PPO Update

```
Require: N trajectories $\tau \sim \pi_{\theta_{old}}$, old policy network parameters $\theta_{old}$, Value Network V and discriminator parameters $w_0$.

  Set $\theta = \theta_{old}$
  for each trajectory do
  T = LengthOfCurrentTrajectory
  advantage = []
  ratio = []
  for t = T, ..., 1 do
    $c_t = \log(D_w(s_t, a_t))$ # cost
    $A_t = c_t + V(s_t) - V(s_{t+1})$
    $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_old}(a_t|s_t)}$
    advantage.append($A_t$)
    ratios.append($r_t$)
  end for
  loss = $\min$(ratios $*$ advantage, clip(ratios, $1-\epsilon, 1+\epsilon$) $*$ advantage)
  gradients ← loss.backward()
  $\theta = \theta$ + learning_rate*gradients
  endfor
```

We propose to use Generative Adversarial Imitation Learning (GAIL) [6] for this task. GAIL relies on the insight that traditional imitation learning methods, such as apprenticeship learning, learn a cost function $c(s, a)$ and then learn a policy $\pi(\cdot|s)$ which is used to evaluate this cost function. Based on this insight, they propose the following loss function.

$$\min_{\pi} \max_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_\pi[\log(D(s,a)] - \lambda\mathcal{H}(\pi) + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] \tag{8}$$

D is a discriminator which tries to distinguish between state action pairs from the trajectories induced by policy pi_E and π and $\mathcal{H}(\pi) = E_\pi[\log(\pi(\cdot|s))]$ is the causal entropy.

Intuitively, this loss function tries to match the distribution of state action visitation of expert policy π_E and the learned policy π under some metric function. If the metric function is assumed to be the Jensen Shannon (JS) divergence, then optimizing this loss function corresponds to the minimization of JS divergence between the distribution of state actions of π_E and π.

Unlike the sampling process in traditional GAN, samples from the generator in equation 8 can not be backpropogated through as the sampling process involves performing a roll-out through a non-differentiable simulator. Hence, optimization of GAIL objective is done in two steps. In first steps, objective in 8 is maximized with respect to discriminator parameters. In second step, generator's parameters i.e. policy network is updated by using a policy gradient method; for the puporses of our experiments, we used Proximal Policy Optimization (PPO) [10].

PPO belongs to the class of trust region policy gradient methods. Without providing the details, we note that PPO suggests optimizing the following loss function.

$$\mathcal{L}_{PPO} = \mathbb{E}_t\left[\min(r_t \cdot A_t, \texttt{clip}(r_t, 1-\epsilon, 1+\epsilon) \cdot A_t)\right] \tag{9}$$

where

$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{od}}(a_t|s_t)}, A_t = c_t + V(s_{t+1}) - V(s_t) \tag{10}$$

.

In the above equations, $\pi_{\theta_{old}}$ is the policy under which samples where collected (see figure 4). Intuitively, r_t measure how much the current policy differs from the sampling policy; if r_t is small, then it means that we are in vicinity of the sampling policy or in mathematical language inside a trust region. If r_t is large, we artificially clamp it to lie inside an ϵ-cube centered on old policy in the measure space of the policies. In other words, we reduce the effects of samples from outside the trust region on the update. A_t as

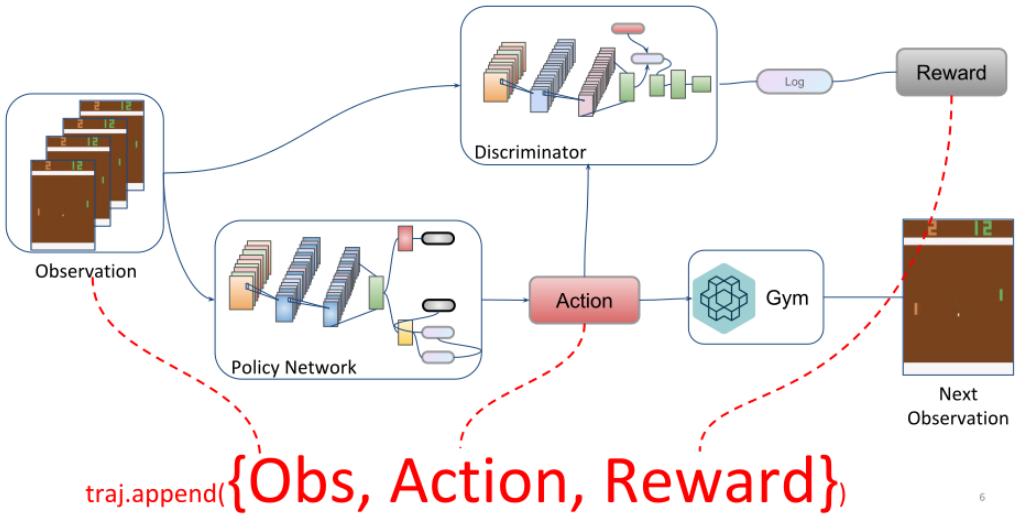

Figure 4. To collect generated samples, we iteratively pass observation through the policy network and choose an action. Both the observation and action are passed to the discriminator and we take log of output of discriminator to get a cost (reward) for the given observation and action. At each step, we append the observation, action and cost/reward received to an array. Action is then passed to the environment which returns the next observation. The process is repeated till a time limit is reached or environment returns a 'done' signal denoting end of episode.

noted in the background section, computes the advantage of the action a_t taken in the state s_t over all the possible other actions in that state.

Experiments in the original paper [6] validated this method only for continuous action spaces with sensory inputs. Our objective here is to apply this method to control discrete action space using high dimensional visual data as input. We have chosen Atari games as environment of our choice due to rich variety of tasks present in it and an easy to use interface available through OpenAI gym software [4].

Network Structure: Atari is a significantly harder testbed as compared to control tasks due to the fact that it constitutes a partially observed Markov Decision Process (MDP). Further, the observations in Atari are image frames and not sensor readings, this essentially means that meaningful features have to be first extracted as well by the agent. In order to tackle this issue, we benefit from the convolutional neural network design proposed by [7]. This network receives a sequence of four prepossessed frames concatenated across the channel dimension. In the first step, 32 8×8 filters are applied to the input with 4×4 strides. At the second layer, 62 4×4 filters are applied with stride of 2. At the third layer, input is convolved with 64 3×3 filters with stride of 1. The output from third convolution layer is flattened and fed to full connected layer with 512 hidden nodes.The output from this hidden layer is then fed separately to a policy network and a value network. The policy network for the pong is shown in figure 2.

For discriminator, we try several different choices of network architecture. We experimented with sharing the convolutional part of architecture between the discriminator and policy network, however, we observed that this fails to work well. Primarily due the fact that discriminator and policy network are looking for different kind of features. We discovered that the best performing choice of discriminator used three convolutional layers as well but contained a smaller number of filters. The final architecture used for discriminator is shown in figure 3.

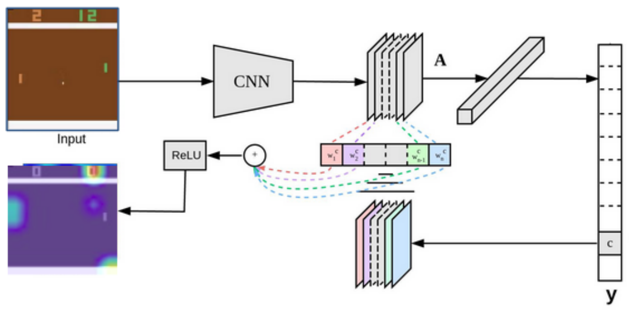

Figure 5. We use gradient based class activation mapping to visually identify the areas within the most recent frame which resulted in higher values of logit for the chosen action.

Grad-CAM: Grad-CAM [12] is a method for identifying what regions in an image contribute highly to the highest

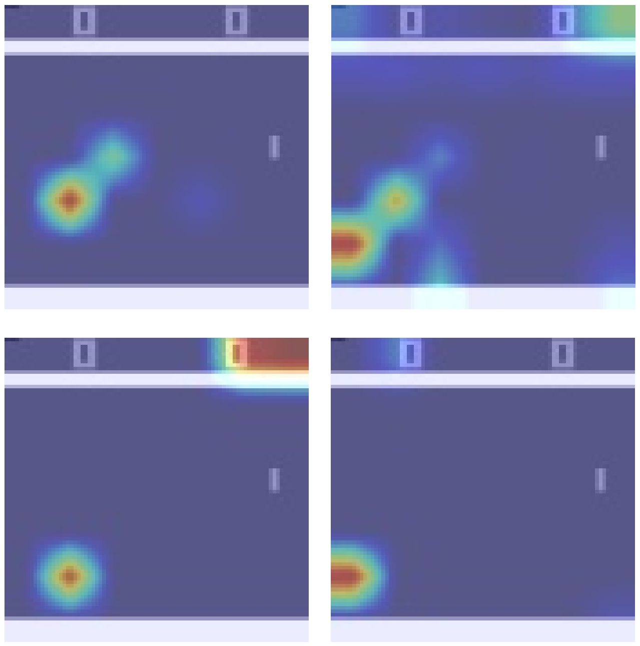

Figure 6. Visualization of what regions in the input frames are causing high score for the action chosen by the agent in the Pong environment. Note that the agent focus is mostly on the ball or on opposite player's stick or on the score.

	Pong	Breakout
GAIL	21	390
Behaviour Cloning	21	150

Table 1. Median reward achieved by GAIL compared with behaviour cloning in our experiments.

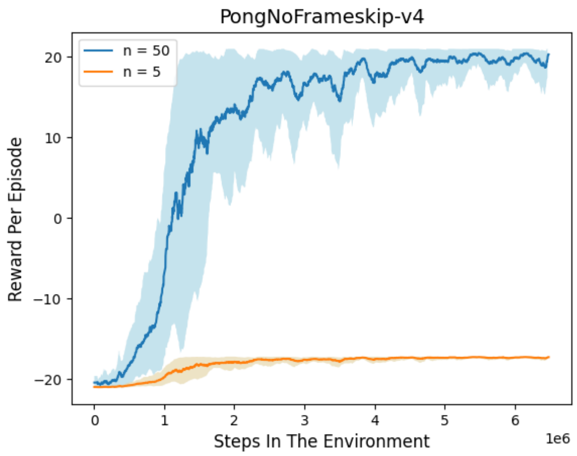

Figure 7. Reward per episode for the Pong game.

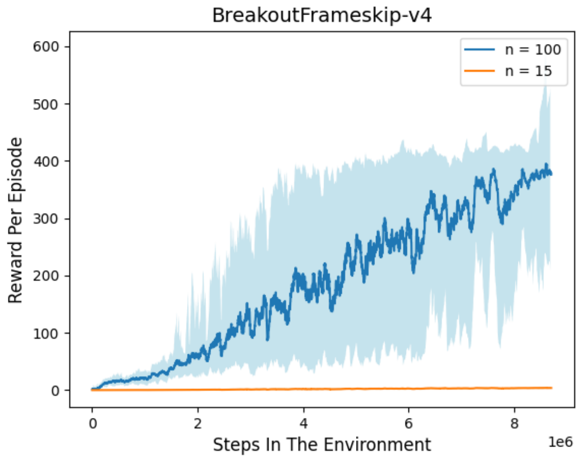

Figure 8. Reward per episode for the Breakout game.

logits class and is used to interpret classifier networks. Here we use this technique to interpret the actions chosen by the policy network.

6. Experiments and Results

We restrict our experiments to two environments of Atari i.e. Pong and Breakout due to limited computational budget. Figures 7 and 8 show the average reward attained by the agent as a function of number of timesteps taken in the environment. We report our results in the table 1. For behaviour cloning, we use results reported in literature [3].

Further, we attempt to visualize what the agent is *doing* by treating the policy network as a classifier and using grad-CAM algorithm. Results of these visualizations are shown in figure 6 for the game of pong.

7. Discussion and Perspective

While GAIL is sample efficient in terms of requiring samples labelled with reward, we found that it is quite inefficient in terms of environment interactions required to achieve competitive reward. For example, for the Breakout game, the policy network of GAIL required about 3 times as much interaction with the environment as the Proximal Policy Optimization (PPO) algorithm to achieve the same level of performance. However, we do note that on simpler environments, such as CartPole, GAIL only required quarter of the interactions as PPO. The inefficacy of GAIL on Atari can be attributed to the fact that it also includes the process of extracting features from the frames and lack of true reward function ultimately makes this process difficult.

Further, we note that training of GAIL is extremely brittle. Training of the reinforcement learning in general is quite difficult and the fact that GAIL requires multi scale optimization of an adversarial game does not help the cause either.

Limited computational resources and the above two factors restricted us to only run our algorithm on two Atari games.

8. Conclusion

We show that despite the fact that GAIL achieves competitive results on Atari domain, it requires an extremely large amount of interactions with the environment.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004. 2

[2] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. *arXiv preprint arXiv:2003.13350*, 2020. 1

[3] Kianté Brantley, Wen Sun, and Mikael Henaff. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*, 2020. 2, 7

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

[5] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016. 2

[6] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016. 2, 5, 6

[7] 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):529–533, 2015. 6

[8] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991. 1

[9] Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: Imitation learning via reinforcement learning with sparse rewards. In *International Conference on Learning Representations*, 2020. 2

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3, 5

[11] Dexter RR Scobee and S Shankar Sastry. Maximum likelihood constraint inference for inverse reinforcement learning. 2020. 2

[12] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019. 6

[13] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015. 2

[14] Shangtong Zhang and Richard S. Sutton. A deeper look at experience replay, 2017. 2

[15] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 2