Gimme a Break(out): Project Report

Anuj Jitendra Diwan - 170070005 Arnab Jana - 170100082 Soumya Chatterjee - 170070010

1 Introduction

This project aims to present an overview and comparison of different Reinforcement Learning algorithms as applied to the Atari game Breakout. We also show the results of training the game Pong on Vanilla Policy Gradient for 15 days. We try out the following Reinforcement Learning algorithms on Breakout:

1. Deep Q-learning
2. Advantage Actor-Critic Learning
3. Proximal policy optimization
4. Trust Region Policy Optimization

2 Theoretical explanations for algorithms used

2.1 Deep Q-learning

- In RL, given a state s $\in \mathbb{S}$ and an action a $\in \mathbb{A}$, the Q-function, Q:$\mathbb{S} \times \mathbb{A} \mapsto \mathbb{R}$ is defined as

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s^{'}, a)$$

where r(s,a) is the expected reward from taking an action a at state s and s$^{'}$ is the next state that yields the highest Q-value.

- The Q values are updated as follows

$$Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha[r(s_t, a, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a)]$$

where $r(s_t, a, s_{t+1})$ is the reward associated with a transition from state s$_t$ to state s$_{t+1}$ on action a and γ is the discount-factor.

- In deep Q-learning, a deep Neural Network is used to approximate the Q-function, which given a state as input outputs Q-values for all possible actions.
- Here, α corresponds to exploration and (1-α) corresponds to exploitation. Thus, varying this hyperparameter allows to adjust between exploration and exploitation.
- We also use a greedy-epsilon strategy in which fix a small value epsilon. Given an estimator, the best predicted action is assigned a probability (1-ϵ)+ϵ/k and all other actions are assigned a probability ϵ/k, where k is the total number of actions. This is a very simple method to balance exploration-exploitation trade-off.
- To prevent overfitting to current episodes, we have used Experience Replay, which loads and reuses past transitions to avoid catastrophic forgetting.

2.2 Policy-Gradient based Learning

- Policy-gradient method produces a stochastic policy $\pi_\theta(s,a)$ parametrized by the policy model parameters θ. Thus we would like to optimize for the parameters θ by maximizing a policy score function. A good one would be the average reward per timestep or the expected future reward, both of which are expected to be similar:

$$J_{avR}(\theta) = E_\pi(r) = \sum_{s\in S} d(s) \sum_{a \in A} \pi_\theta(s,a) R_s^a$$

or

$$J(\theta) = E_\pi[R(\tau)]$$

Here $d(s)$ is the state distribution(prob of being in s), and R_s^a is the immediate reward.

- We perform gradient ascent on this function and hence obtain the following policy parameter update:

  $$\Delta\theta = \alpha * \nabla_\theta \log\left(\pi(s,a,\theta)\right) R(\tau)$$

  Here α is the LR.

- This policy parameter update is done at the end of each episode(Monte-Carlo)(to get a good estimate of $R(\tau)$, the future reward. This is slightly problematic because we update at the end of each episode, effectively averaging the goodness of all the actions taken during that episode. We would like to update at each timestep, which is done in Actor-Critic.

2.3 Advantage Actor-Critic Learning

- Advantage actor-critic learning is a hybrid policy-and-value-based optimization scheme.
- Now, instead of using the end-of-episode reward and per-episode parameter update, we predict an estimate of the future reward at each time step and use that in place of the reward in the previous expression, hence allowing us to perform updates at each time step.
- The per-timestep update is now:

  $$\Delta\theta = \alpha * \nabla_\theta * \left(\log \pi\left(S_t, A_t, \theta\right)\right) * Q\left(S_t, A_t\right)$$

  where $Q(S_t, A_t)$ replaces $R(\tau)$. Thus we need someone to produce Q, a good estimate of the reward. This is the Critic, while the person who plays in the environment and uses the policy is the Actor.

- The Critic learns how to rate certain actions at certain states and communicates this to the Actor through q. So we have two different models to optimize(like GANs, except that here they work in unison): $\pi_\theta(s,a)$ and $\hat{q}_w(s,a)$.
- The value function is updated using temporal difference learning. This type of learning uses the immediate reward and a (discounted) difference of the current timestep $\hat{q}$ and the next timestep, using incremental changes to guide the learning.

  $$\Delta w = \beta\left(R(s,a) + \gamma \hat{q}_w\left(s_{t+1}, a_{t+1}\right) - \hat{q}_w\left(s_t, a_t\right)\right) \nabla_w \hat{q}_w\left(s_t, a_t\right)$$

- Both these optimizations are done in unison: at time t, the actor uses the policy to take an action a_t. The critic evaluates this action using $\hat{q}$ using which the actor updates its parameters. The actor then uses the new policy to take an action a_{t+1} and the critic then updates its parameters.

2.4 Proximal Policy Optimization

- In policy-gradient, one may have high variability during training if policy gradients are too high or too low.
- In this case, one can first use the advantage function $A(s,a) = Q(s,a) - V(s)$ instead of the $Q-function$. This function basically rewards a positive advantage i.e. action that does better than average rather than rewarding the absolute Q-function.
- Also, we enforce that the new policy does not change too much from the old one using a clipped optimization function. First, we use the ratio of the confidence under new policy to that under old policy

  $$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}$$

  and try to maximize:

  $$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \operatorname{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right]$$

  Here the first term will try to optimize θ such that $r_t(\theta)$ increases(new policy assigns greater confidence) if advantage is greater than 0 and such that it decreases if it is lesser than 0. The second term clips the maximum and minimum increases and decreases of confidence from the old to the new policy to reduce high variability during training.

2.5 Trust Region Policy Optimization

- Trust Region Policy Optimization is a modification on standard policy gradient that uses trust region based methods to optimize the loss function instead of steepest descent methods.
- Line search is used on the gradient of the score function where the estimate of the optimum point on the line(largest score on the line) is computed using second order derivatives i.e. Hessians.

3 Implementation details

3.1 Deep Q-learning-Breakout

- Our deep neural network model consists of 3 2D convolution layers on the input, the game's frames, with the ReLu activation function. The third convolution layer is connected to a Flatten layer which is connected to a Fully Connected layer. The loss function is mean-squared error and the optimizer used is RMSPropOptimizer.
- We also use Experience Replay and a greedy-epsilon strategy.
- Our code is implemented in Tensorflow.

3.2 Stochastic Policy Gradients - Pong

- Our code is based on Pong from Pixels
- The policy is learnt using a simple two layer neural network with the input being the difference between images of the game from the current time step and the previous. The resultant difference is converted to grayscale and cropped to 80×80 for faster computation. Thus, the input is of size 6400, the hidden layer has 200 neurons and the outputs is the probability of choosing the action UP (2 in Gym Atari environment). The other action DOWN (3) is chosen with probability (1-output).
- Xavier initialization is used for the network weights and it is trained using RMSProp a batch size of 10 epochs.

3.3 A2C PPO and TRPO

- We use the default configuration as in OpenAI baselines
- The model is a simple CNN with three convolutional and one fully connected layer

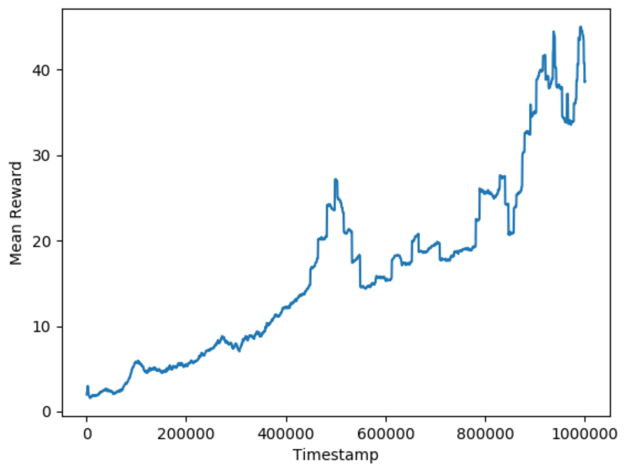
4 Results, Comparisons and Analysis

4.1 Reward comparisons

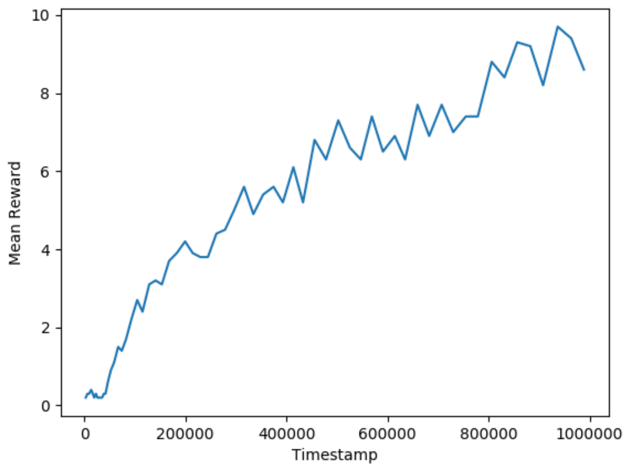
Timesteps	Algorithm	Mean reward
2×10^5	Deep-Q Learning	4.2
	Proximal Policy Optimization	5.37
	Advantage Actor-Critic	1.96
	Trust Region Policy Optimization	0.575
6×10^5	Deep-Q Learning	6.5
	Proximal Policy Optimization	15.43
	Advantage Actor-Critic	2.5
	Trust Region Policy Optimization	1.65
8×10^5	Deep-Q Learning	8.8
	Proximal Policy Optimization	25.68
	Advantage Actor-Critic	2.5
	Trust Region Policy Optimization	2.05
10×10^5	Deep-Q Learning	9.4
	Proximal Policy Optimization	38.64
	Advantage Actor-Critic	2.87
	Trust Region Policy Optimization	1.35

Table 1: Comparing different algorithm performance on Breakout

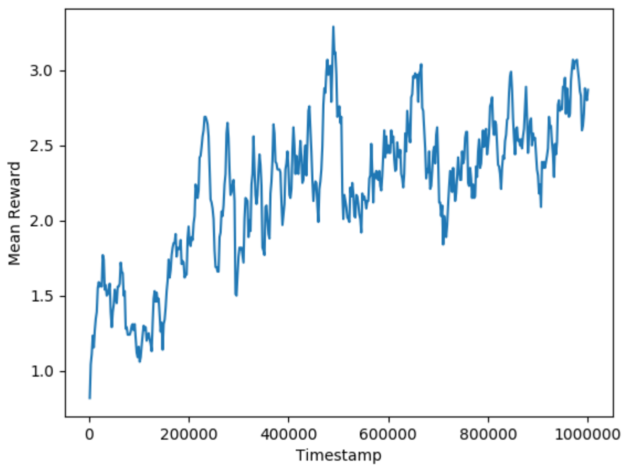
4.2 Training plots



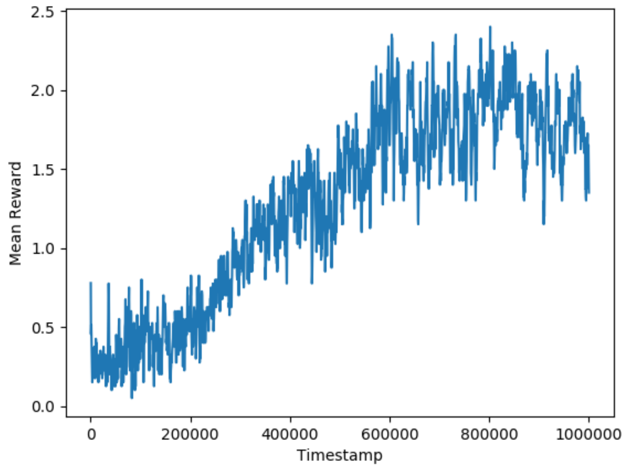
(a) Proximal Policy



(b) Deep-Q



(c) Advantage Actor-Critic



(d) Trust Region Policy Optimization

4.3 Analysis and conclusion

- For breakout, Proximal Policy has,by far, performed the best, followed by Deep-Q, Advantage Actor-Critic, and Trust Region.
- We can clearly see that reducing the variability in training(Proximal Policy) is crucial for Breakout. Other optimizations like Actor-Critic and Trust Region do not offer much improvement.
- This is probably because the Breakout state space is sparsely connected(paddle moves only left or right) and thus large changes in predicted action are detrimental.
- Also we observe that Deep Q-learning does well. Since Deep Q-learning's success is largely dependent on the deep network used, we conclude that the CNN network used was able to predict the Q-value well.
- The advantage actor-critic is learning stochastically probably because the setting is somewhat like a GAN, because of which the actor and critic will keep pulling the reward up and down as and when the actor/critic worsens/improves.(it is not a single smooth gradient descent like in Deep Q-learning).
- A final observation is that the Pong environment being dynamically changing(the opponent who is also playing is part of the environment for the player) and as a result takes much longer to train.

5 References

- Deep Q Learning : https://github.com/dennybritz/reinforcement-learning/blob/master/DQN/dqn.py
- Pong from Pixels : http://karpathy.github.io/2016/05/31/rl/
- OpenAI Baselines : https://github.com/openai/baselines