An Exploration of Deep Reinforcement Learning Algorithms

Edwin Zhang
Department of Computer Science
University of Waterloo
eazhang@uwaterloo.ca

Ajay Patel
Department of Computer Science
University of Waterloo
a346pate@uwaterloo.ca

1 Introduction

How exactly do organisms decide how to behave and interact with the world? One of the fundamental questions in neuroscience concerns the decision-making processes by which these organisms decide on actions within a given environment, and their responses to reward and punishment.

In behavioural psychology, this question has been explored in detail through the paradigms of classical and instrumental conditioning, where stimuli are applied before and after actions, respectively, to change the behaviour of an organism. Through instrumental conditioning, organisms learn stimulus to response associations, also known as the *law of effect*. That is, given an environment, if an organism chooses an action with a positive outcome, then the association between the stimulus and response is strengthened through neuromodulators such as dopamine. If the outcome is negative, then the connection is weakened. Neuroscience research has suggested that dopamine reward systems provide the basal ganglia target structures with phasic activation signals that convey a reward prediction error to directly influence action selection and learning [9].

A similar process of decision making arises in the field of reinforcement learning (RL), where agents interact with an environment by choosing actions based on their current state or observations. After each action, the environment provides the agent with a reward signal or return, indicative of how good the action was. Like humans and other biological organisms, RL agents attempt to maximize long-term rewards by learning how to select the optimal action at any given state. Recent success in various RL tasks has been demonstrated by both non-biological and biologically-inspired models such as Spaun from Eliasmith et al. [3], however, our efforts will focus on deep reinforcement learning.

With the advent of deep learning, modern RL algorithms incorporate deep neural networks as function approximators, which learn to choose optimal actions within large environments by minimizing predetermined loss functions. Although state-of-the-art deep RL models do not directly model biological function in the brain, modern RL research still draws on many aspects of neuroscience and biology.

This project aims to explore and compare the learning performance of three state-of-the-art deep RL algorithms - Deep Q-Learning (DQN) [5, 6], Advantage Actor-Critic (A2C) [4], and Proximal Policy Optimization (PPO) [8] - to see how they perform on several classic control tasks in OpenAI Gym [2]. We train and compare the performance of these agents by measuring average reward in three environments, namely, `CartPole-v1`, `Acrobot-v1`, and `MountainCar-v0`.

Based on results from the respective papers, we expect PPO to be the most stable algorithm during training and that it will be able to converge faster than both A2C or DQN on the given tasks. We expect all three algorithms to perform well in `CartPole-v1` since this environment provides dense rewards. However, due to the sparse rewards in `Acrobot-v1` and `MountainCar-v0`, we expect some, or all of the algorithms to struggle in finding a solution.

2 Methods

Reinforcement learning has two main classes of algorithms: model-free and model-based. A model can be defined as a function which predicts state transitions based on actions and their rewards. In a model-based algorithm, an agent is provided with a model, which allows it to plan ahead by observing outcomes for all possible choices. However, models are typically not available in environments and are difficult to provide, in which case, model-free algorithms are needed. Model-free algorithms learn to build their own model of an environment based on observations and actions. By comparison, these algorithms are normally easier to implement, but suffer from sample inefficiency. All three algorithms implemented in this paper are model-free, and can be further divided into Q-learning and policy optimization algorithms, which are explained in more detail below.

2.1 DQN

Q-learning algorithms learn an optimal action-value approximator $Q_\theta(s, a)$, which outputs the expected return given the state s and action a. Actions are then chosen based on the highest expected return. They optimize the parameters θ to maximize their future expected return.

DQN is a Q-learning algorithm that uses deep learning models as action-value approximators, and was one of the first major breakthroughs in deep RL [5]. In DQN, we consider sequences of states and actions, and learn optimal strategies based on these sequences. A sequence or trajectory τ can be defined as

$$\tau = (s_0, a_0, s_1, a_1, ...)$$

Additionally, the discounted sum of future rewards at time t for a fixed number of steps T, reward r, and discount factor γ can be defined as

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$$

This represents the sum of all rewards obtained by the agent, but discounted based how far in the future they are obtained. Lastly, we define the optimal action-value function $Q^*(s, a)$ as the maximum expected return achievable by following the optimal policy π, after some sequence s and then taking some action a

$$Q^*(s, a) = \max_\pi \mathbb{E}_{\tau\sim\pi} \left[R(\tau)|s_0 = s, a_0 = a\right]$$

The optimal action-value function obeys the *Bellman equation* which is based on the following intuition: if the optimal value $Q^*\left(s', a'\right)$ of the sequence $\boldsymbol{s'}$ at the next time-step is known for all possible actions a', then the optimal strategy is to select the action a' maximising the expected value of $r(s, a) + \gamma Q^*\left(s', a'\right)$. DQN estimates the action value function $Q_\theta(s, a) \approx Q^*(s, a)$ using a deep neural network which is trained by minimising the loss $L(\theta)$ with stochastic gradient descent

$$\nabla_\theta L(\theta) \sim \nabla_\theta \mathbb{E}_{s'\sim P(s'|s,a)} \left[\left(y - Q_\theta\left(s, a\right)\right)^2\right]$$

where y is the target optimal action-value estimate.

Mnih et al. also add two major improvements to DQN over traditional Q-learning: experience replay memory and the usage of a target Q-network for generating the target value when computing loss. With experience replay memory, we store a transition containing the state, action, reward, and next state at each time step in a data set of fixed-size. For each iteration of the algorithm, we sample a random batch of transitions and apply the Q-learning update for each. This allows each transition to potentially be used for multiple updates and helps prevent the agent from getting stuck in a poor local minimum. It also allows for greater sample efficiency when compared to learning from consecutive samples, because consecutive samples can be highly correlated and lead to inefficient learning.

The second improvement of using a target Q-network allows DQN to be more stable when training. In standard Q-learning, an update that changes $Q(s_t, a_t)$ can also significantly change $Q(s_{t+1}, a_{t+1})$

when computing the target value y in the loss, which leads to divergence or oscillations in the policy. By using a second target Q-network to predict the target value, this adds a delay between an update to the main Q-network and when the update affects y, reducing oscillations. In practice, the target network is updated every C steps by copying the weights of the main network.

2.2 A2C

Policy optimization algorithms such as A2C and PPO learn to represent a policy $\pi_\theta(a|s)$, which unlike Q-learning, directly outputs the probability distribution of a given action a based on the state s. These algorithms optimize parameters θ by maximizing some predefined performance objective.

Actor-critic algorithms are a general approach to implementing policy optimization algorithms and train two networks, the actor and the critic. The actor outputs the policy, while the critic outputs an estimate of the state-value $V_\theta(s)$, which is the expected return from a given state. The critic is used to judge the actor's policy and compute update gradients for the actor network to improve future policies. In actor-critic methods, this is performed by an advantage function $A(s, a)$ defined as

$$A(s_t, a_t) = Q_\theta(s_t, a_t) - V(s_t)$$

In other words, the advantage computes the difference between the expected return from taking a specific action a_t compared to the average action at a given state.

Furthermore, $Q_\theta(s_t, a_t) = \mathbb{E}\left[r_{t+1} + \gamma V(s_{t+1})\right]$, so we can rewrite the advantage function as

$$A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

which is also known as the temporal difference (TD) residual.

A2C is an actor-critic algorithm that optimizes for the advantage function defined above. At each iteration, the gradients are computed according to the objective function $J(\theta)$ and updated via stochastic gradient descent

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)]$$

2.3 PPO

PPO is a policy optimization algorithm that is commonly implemented as an actor-critic algorithm, and therefore behaves similarly to A2C. The main difference is that PPO optimizes for a *surrogate objective function* based on its old and current policies, rather than just the advantage function. Using this surrogate objective, the algorithm attempts to find the largest possible improvement step on a given policy, without causing accidental performance collapse by stepping too far. We implement the most common variant of PPO, also known as PPO-clip. This algorithm relies on specialized clipping in the objective function to remove incentives for a new policy to deviate far away from the old policy. Let π_θ denote a policy with parameters θ.

Using the current policy parameters θ and old policy parameters θ_{old}, PPO-clip updates policies via the surrogate objective $L(\theta)$ using stochastic gradient descent

$$\nabla_\theta L(\theta) \sim \nabla_\theta \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A(s_t, a_t), clip(\epsilon, A(s_t, a_t))\right)$$

where

$$clip(\epsilon, A(s_t, a_t)) = \begin{cases} (1+\epsilon)A(s_t, a_t) & A(s_t, a_t) \geq 0 \\ (1-\epsilon)A(s_t, a_t) & A(s_t, a_t) < 0 \end{cases}$$

The parameter ϵ, which is also known as the clip-ratio, roughly tells us how far away the new and old policies are allowed to diverge. The advantage function $A(s_t, a_t)$ can be the same as the advantage

function used in A2C or some other advantage function. We implement Generalized Advantage Estimation (GAE) [7], which was used in the original PPO paper and has shown better empirical results compared to other advantage functions. GAE reduces variance in advantage calculations by using a discounted sum of TD residuals δ_t

$$\delta_t^V = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

where the GAE is computed using discount factor γ and smoothing parameter λ

$$A_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_t^V$$

3 Experiment Setup

We perform experiments on three classic control tasks in OpenAI Gym: `CartPole-v1`, `Acrobot-v1`, and `MountainCar-v0` [2]. All three algorithms are implemented using TensorFlow [1], and mirror the algorithms described in the original papers as much as possible. For all experiments, we perform stochastic gradient descent using the Adam optimizer with a learning rate of 0.001. Note that we only trained our agents in environments with discrete action spaces, since DQN is unable to run in continuous action spaces, while A2C and PPO would require changes to policy outputs. In terms of hardware, the experiments and environments we use are not very demanding, so mid-range laptop CPUs were used for training.

For DQN, we implement experience replay memory and use a target Q-network to compute loss. In the original paper, DQN learned to solve several Atari environments by directly learning from input observation images using a convolutional neural network as the approximator for $Q_\theta(s, a)$. Compared to Atari input images, the observations provided by our chosen environments are much smaller, which means that we can use a much simpler Q-network.

Both A2C and PPO are implemented as actor-critic methods, which use two separate models for the actor and critic networks. Both networks have the same number of hidden layers and units. PPO implements GAE as an advantage function while A2C uses the standard advantage function. As a small improvement, we also scale down rewards by a factor of 0.01 for both algorithms. The intuition behind this is that the agent learns to optimize the frequency of rewards rather than the total sum, leading to better long term performance [10].

Specific hyperparameter configurations for the three models on each environment are included in configuration files with the code. Due to time limitations, we were unable to perform large parameter sweeps and mainly used hyperparameters given in the papers. Therefore these models are not as finely tuned as those presented in academic literature. However, we did find that DQN performed much better as we increased the number of hidden units in each layer whereas PPO and A2C actually converged faster on smaller layers. This is likely due to greater sample inefficiency in PPO and A2C compared to the replay memory batch updates in DQN.

4 Results

Here, we show results for all three algorithms trained on our three environments. We plot the exponential moving average of reward per episode when training, where one episode is defined as the entire sequence of actions from a start state to a terminal state. For each episode, we also plot the actual reward of the agent with lower opacity. In figure 1, we see that all three agents are able to successfully solve both `CartPole-v1` and `Acrobot-v1`, while only DQN is capable of solving `MountainCar-v0`.

4.1 Dense Reward Environments

In `CartPole-v1`, the agent's goal is to balance a pole on top of a moving cart by moving left and right. At each step, the agent receives a reward of +1 if the pole is upright and -1 otherwise. This means

that the environment has a fairly dense reward system, where the agent gets immediate feedback at every time step. It is expected that agents have an easier time solving environments with dense as opposed to sparse rewards due to the consistent learning signal. This is consistent with our results since all three agents learned to successfully solve the environment. As expected, PPO outperformed A2C, but DQN appeared to be the most stable and converged the fastest.

4.2 Sparse Reward Environments

By comparison, both `Acrobot-v1` and `MountainCar-v0` have fairly sparse reward signals. In `Acrobot-v1`, the agent attempts to swing a system consisting of 2 joints and 2 links up to a given height. The agent receives a reward of -1 at every step until it reaches the height threshold, where it receives a reward of 0. In `MountainCar-v0`, the agent's goal is to make it to the top of a mountain by driving back and forth to build up momentum. The agent receives a reward of -1 at each step unless it reaches the top, where it receives a reward of +100. The agents performed surprisingly well on `Acrobot-v1`, but only DQN was able to solve `MountainCar-v0`.

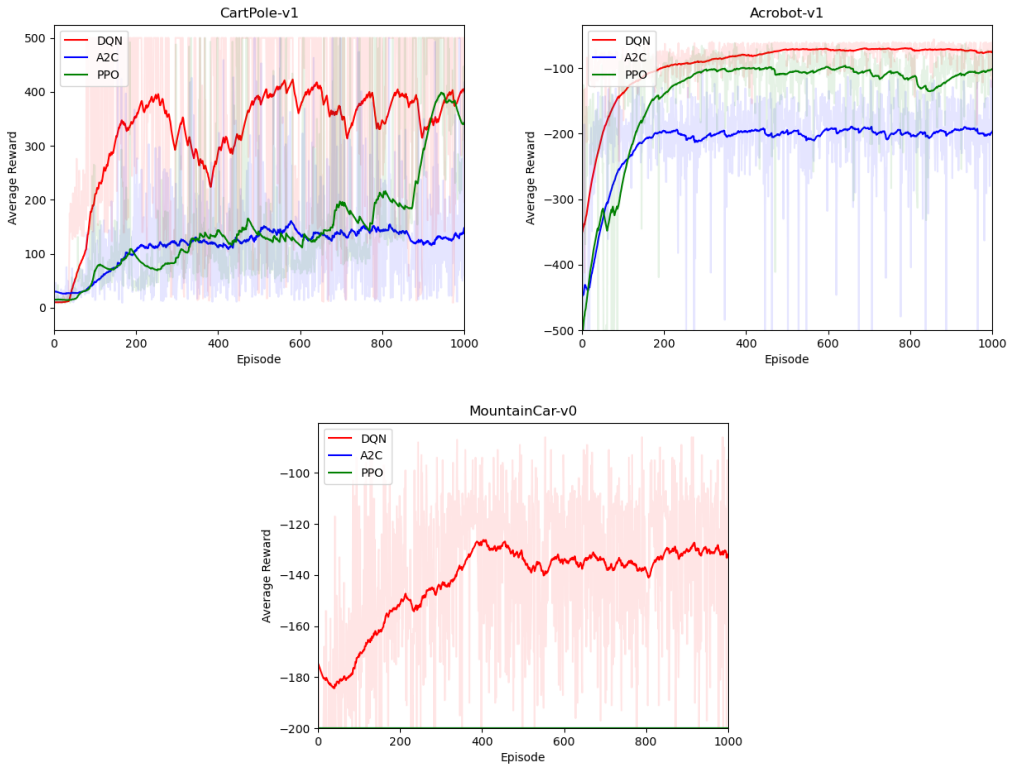

Figure 1: Average exponential reward per episode plotted alongside true reward per episode in `CartPole-v1`, `Acrobot-v1`, and `MountainCar-v0`.

The results above are taken from the best training runs of each agent. Overall, DQN turned out to be very easy to train and consistently produced good results while A2C and PPO suffered from frequent convergence to poor local minima. We also find that although the average reward of the algorithms may converge, there is also a significant amount of variance and oscillation across individual episodes. Based on our plots of the true reward per episode, PPO exhibited the smallest oscillations even though it may have not performed the best.

Due to sensitivity to small changes and the effects of randomness on different training runs, it was difficult to tune hyperparameters so that the agents consistently produced better results. However, we did find that in DQN, a higher value of ϵ, or the probability of taking a random action, helped a lot in `MountainCar-v0`. After a certain number of episodes, it seems like DQN eventually reached the top of the mountain by random chance. Once this happened, the agent was able to update itself based on experience replay and converge to a solution. A2C and PPO don't have similar exploration mechanisms, which could explain why they performed so poorly.

5 Discussion

Despite recent advancements in deep reinforcement learning, we find that many of these algorithms are still quite difficult to train and suffer from problems relating to instability and sparse rewards. This was particularly evident with the policy optimization methods, which tended to be very inconsistent across multiple runs.

Interestingly, DQN converged quickly and outperformed both of the policy optimization methods on all three environments. Although there were some oscillations in reward values, especially in `CartPole-v0`, DQN was overall very stable during training. Part of the reason for this might be due to the experience replay mechanism, which allows the agent to train itself based on past experiences and maximize sample efficiency. However, one other reason for this could be the lack of hyperparameter tuning, which could be holding back A2C and PPO from optimal performance. DQN also performs more updates per step than either one of the other algorithms, which can explain its fast convergence.

Next, we can compare the performance of A2C to PPO, where as expected, PPO performed the best and converged to higher reward values. This can be attributed to two things: using GAE as an improved advantage function to reduce variance and the surrogate objective function, which allows for more stability when training. This reduced variance and increased stability is clearly evident in the much smaller reward value oscillations for each episode.

From our results, we also saw that all three agents performed well on one sparse environment (`Acrobot-v1`) but not on the other (`MountainCar-v0`). This is most likely due to `Acrobot-v1` having an easier goal to reach, which means that the agents would not need to have as good of an exploration policy. As to why A2C and PPO were unable to solve `MountainCar-v0`, it could be due to the idea that both of these algorithms are on-policy algorithms, where each update only uses data collected while using the most recent policy. In a sparse reward environment, it is difficult if not impossible to improve the policy based on only negative feedback. DQN on the other hand is off-policy and is therefore able to update itself based on data collected at any point during training. This, in combination with some level of random exploration, allows DQN to perform well.

For future improvements to the project, one idea would be to perform better hyperparameter tuning and run experiments based on fixed random seeds so that we can eliminate some of the inherent randomness in benchmarking these algorithms. In addition, we could attempt harder or more demanding environments such as the Atari environments in OpenAI Gym. It would be valuable to try a wider variety of dense and sparse reward environments in order to build a larger experiment sample size. Lastly, it could be interesting to try newer methods such as the Soft Actor-Critic (SAC) or Deep Deterministic Policy Gradient (DDPG) algorithms and examine the impact of these novel techniques in deep RL.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *ArXiv*, abs/1606.01540, 2016.

[3] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, 2012.

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *ArXiv*, abs/1602.01783, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

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

[7] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

[9] W. Schultz. Predictive reward signal of dopamine neurons. *Journal of neurophysiology*, 80 1:1–27, 1998.

[10] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver. Learning values across many orders of magnitude. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4294–4302, Red Hook, NY, USA, 2016. Curran Associates Inc.