Comparative Analysis & Implementation of Foundational Deep Reinforcement Learning Algorithms

Jonathan Samuel
samueljon17@vt.edu

1 Introduction

This research project conducts a comprehensive investigation of three core Deep Reinforcement Learning (DRL) algorithms: Deep Q-Networks (DQN), REINFORCE, and Actor-Critic (A2C). The introductory artificial intelligence course is limited within reinforcement learning. For example, using policy iteration & value iteration assumes the full Markov Decision Process (MDP) framework is known, Q-learning is not scalable for larger, continuous, or high-dimensional spaces and approximate Q-learning relies on handcrafted features that may not properly capture complex, non-linear relationships [6]. This research extends beyond introductory AI coursework, providing practical insights into DRL algorithm development & application.

In contrast, DRL's address these limitations but do face their own challenges. DQN, which utilizes deep neural networks for Q-value approximation, demonstrates proficiency in managing high-dimensional input spaces [3, 4]. However, it's challenged in continuous action spaces & is prone to Q-value overestimation. REINFORCE, a method focused on direct policy optimization via gradient descent, is effective in environments with high-dimensional action spaces but set back by variance in gradient estimates & sample inefficiency [3, 5]. A2C, merging value & policy based, achieves training stability through its dual-network architecture but at the cost of increased complexity in network tuning [3, 5, 4].

By extending beyond the scope of introductory AI coursework, this project aims to provide in-depth technical insights into the development & application of these DRL algorithms. The objective of this project is to develop these algorithms from the ground up, complying to the methodologies outlined in their original research papers. The performance of these algorithms will be evaluated utilizing the OpenAI's Gymnasium platform for experimental validation & PyTorch for algorithmic implementation. All code is provided using the github repository [1].

2 Background

2.1 DQN

Theory suggest Q-learning could store Q-values for all state-action pairs in an extensive lookup table. However, in practice, this approach is computationally impractical for large state & action spaces [2]. To address this, function approximation are employed where a model of features parameterized by θ approximates the Q-values, $Q(s, a; \theta)$. However, when integrated with nonlinear function approximators for Q-value estimation can encounter issues of instability & divergence [1]. This instability largely stems from the compounded errors due to the approximation of Q-values, the temporal correlation between successive samples, & the oscillations in policy updates [1]. Furthermore, the use of bootstrapping in Q-learning, where current estimates are updated based on subsequent estimates, can amplify these errors, leading to divergence in learning [1].

The Deep Q-Network (DQN) [1] introduces two innovative mechanisms to mitigate these challenges, experience replay & a separate target network. Experience replay involves storing transitions (s, a, r, s') in a replay buffer D. Training samples are randomly drawn from D, reducing correlation between consecutive samples and stabilizing training. The use of a separate target network with parameters θ^- to compute target Q-values mitigate the rapid policy oscillations by updating these parameters periodically from the primary network.

The DQN objective is the same as Q-learning, to find an optimal policy π^* maximizing the expected cumulative reward. Likewise, DQN is an active reinforcement learning strategy that updates Q at each time step following equation 1 where α is the learning rate, γ is the discount factor, and r_{t+1} is the reward.

[1]https://github.com/SamuelJon17/drl_foundation

Algorithm 1: Deep Q-learning (DQN) With Experience Replay

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode = 1, $M$ do
    Initialize sequence $s_1 = \{x_1\}$ and prepossessed sequence $\phi_1 = \phi(s_1)$
    for t=1, T do
        With probability $\epsilon$ select a random action $a_t$ otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
        Execute $a_t$ in emulator & observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre-process $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random mini batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        $y_j = \begin{cases} r_j\text{, if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-)\text{, otherwise} \end{cases}$
        Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
        Every C steps reset $\hat{Q} = Q$
    end
end
```

$$Q_{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

The loss function for training the neural network is the mean squared error between predicted & target Q-values 2 where $y^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$, θ^- are the target network parameters, and s' is the next state. The pseudocode for this algorithm is detailed in Algorithm 1.

$$J(\theta) = \mathbb{E}[(y^{DQN} - Q(s, a; \theta))^2] \quad (2)$$

2.2 REINFORCE

Although Q-Learning can achieve an optimal policy, the goal is to learn the state-action value function and infer the policy. Policy gradient directly optimizes the policy function without the need for an intermediary value function. Specifically the policy, $\pi_\theta(a|s)$, is parameterized by θ and denotes the probability of taking action a in state s under policy parameters θ.

The vanilla REINFORCE [6, 3], also referred to as Monte-Carlo policy gradient, operates on the principle of policy gradient, where the objective function, $J(\theta)$, aims to maximize the expected sum of discounted rewards, G_t.

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t] \quad (3)$$

Specifically, the gradient of the objective function with respect to the policy parameters θ is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[G_t \nabla_\theta \log \pi_\theta(a_t|s_t)] \quad (4)$$

The policy parameters θ are updated iteratively using the gradient descent rule where α is the learning rate.

$$\theta_{t+1} \leftarrow \theta_t + \alpha G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \quad (5)$$

Algorithm 2: REINFORCE

```
Initialize $\theta$
for episode = 1, M do
    for t=1, T or $s_t$ is terminal do
        Perform $a_t \sim \pi(a_t|s_t; \theta')$
        Receive reward $r_t$ and new state $s_{t+1}$
    end
    for $t_1, ..., t_{end}$ do
        Estimate the Return
         $G_t = \sum_{k=0}^{end} \gamma^k R_{t+k+1}$
        $\theta \leftarrow \theta + \alpha G_t \nabla_\theta log \pi_\theta(a_t, s_t)$
         $R \leftarrow r_i + \gamma R$
    end
end
```

2.3 A2C

The REINFORCE algorithm is known to exhibit high variance in its gradient estimates due to the erratic nature of trajectories, leading to sub-optimal shifts in policy distribution [4, 5]. To mitigate this, baselines, $b(s_t)$, can be introduced, where the baseline is subtracted from the estimated return, G_t, as shown in equation 4. In this context, the advantage function, $A(s,a) = Q(s,a) - V(s)$, serves as an effective baseline. It quantifies the relative merit of an action compared to the average action at a given state, thus stabilizing the policy update process

The Advantage Actor-Critic (A2C) algorithm is a dual network architecture that integrates the policy gradient approach plus the addition of an advantage baseline (actor) with value function estimation (critic). This integration reduces the variance of policy gradient methods while maintaining their direct policy optimization capability. A2C accomplishes this by employing the critic to estimate the value function, $V(s)$. This estimation guides the policy updates in the actor in a direction that favors actions which perform better than the average .

The policy parameters θ in the actor network are updated similar to that in 5 but subtracting the cumulative discounted reward by their value.

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) \quad (6)$$

The critic updates the value function parameters θ_v using the Temporal Difference (TD) error δ_t, defined in equation 7 where β is the learning rate for the critic, and γ is the discount factor.

$$\delta_t = r_t + \gamma V(s_{t+1}; \theta_v) - V(s_t; \theta_v),$$
$$\theta_{v,t+1} \leftarrow \theta_{v,t} + \beta \delta_t \nabla_{\theta_v} V(s_t; \theta_v) \quad (7)$$

Algorithm 3: Advantage Actor-Critic (A2C)

```
Initialize $\theta$ and $\theta_v$
**for** *episode = 1, M* **do**
    **for** *t=1, T or $s_t$ is terminal* **do**
        Perform $a_t \sim \pi(a_t|s_t; \theta')$
        Receive reward $r_t$ and new state $s_{t+1}$
    **end**
    $R = \begin{cases} 0, \text{for terminal} s_t \\ V(s_t, \theta_v'), \text{for non-terminal } s_t \end{cases}$
    **for** $t_{end}, ..., t_1$ **do**
        $R \leftarrow r_i + \gamma R$
        $d\theta \leftarrow$
          $d\theta + \nabla_{\theta'} log\pi(a_i|s_i; \theta')(R - V(s_i; \theta_v'))$
        $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i; \theta_v'))^2 / \partial\theta_v'$
    **end**
**end**
```

3 Methods

3.1 Environments

The dataset for this research comprises environments from the OpenAI Gym, with a focus on two distinct games, CartPole (CartPole-v1) and MsPacman (MsPacmanNoFrameskip-v4), each offering unique challenges and state representations.

CartPole requires the agent to balance a pole on a moving cart, rewarding the agent for each time step the pole remains upright. The agent has two discrete actions, moving left or right. The observations are comprised of four continuous variables: cart position (± 4.8), velocity ($-\infty, \infty$), pole angle ($\pm 24°$) and angular velocity ($-\infty, \infty$). The episode terminates if the cart position is outside of ± 2.4 range or the pole angle is beyond $\pm 12°$. MsPacman requires the agent to navigates a maze, consuming pellets while evading ghosts, with the reward based on the score accumulated during gameplay. There are 18 discrete actions and the observation space consist of a 210 x 160 pixel image with three layers. The episode is over if MsPacman eats all the pellets or gets eaten by a ghost.

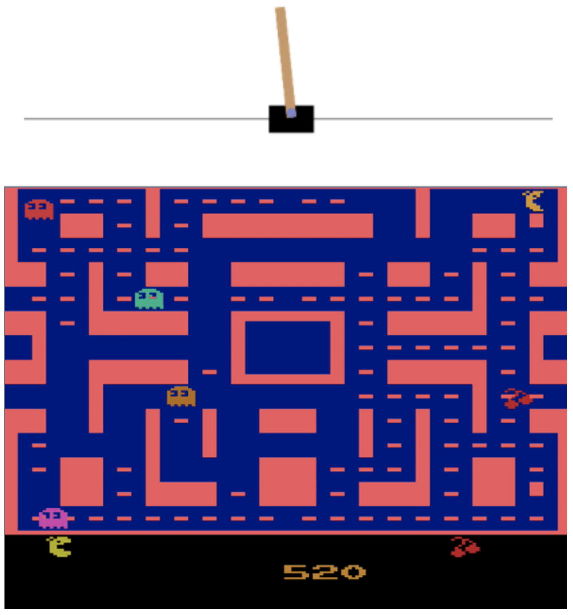

Figure 1: (**Top**) Example of game state in CartPole (**Bottom**) Example of game state in MsPacman

Pre-processing was not conducted on CartPole. For MsPacman, pre-processing followed the DQN paper [1] where observations are resized to 84 by 84 pixels and converted to grayscale, reducing complexity while maintaining motion clarity. Additionally, actions are repeated over four frames to capture motion and stacked, providing a temporal sequence.

3.2 Networks

A baseline agent that used a random policy was used in comparison to the three DRL agents. For CartPole, because the action and observation space are small, it utilizes a simple architecture with two fully connected layers, transforming the input state into a hidden layer of 256 units followed by ReLu activation and an output layer yielding a corresponding state-action value for each possible action. For REINFORCE, an additional softmax activation is applied on the final layer, outputting a probability distribution over all possible actions. The actor-critic architecture consists of two pathways: the actor network maps states to a probability distribution over all possible actions, and the critic network estimates the value of the current state. Both networks share a similar structure mentioned with a hidden layer of 256 units.

MsPacman has a more complex observation space. The model architecture follows the DQN paper [1]. The architecture consists of a convolutional neural network with three layers: the first with 32 filters of size 8x8, the second with 64 filters of size 4x4, and the third with 64 filters of size 3x3, all followed by ReLU activations. This is coupled with two fully connected layers, the first with 512 units and the second corresponding to the number of possible actions, for action-value prediction. Similar to CartPole, for policy gradient methods, a softmax is applied at the final layer.

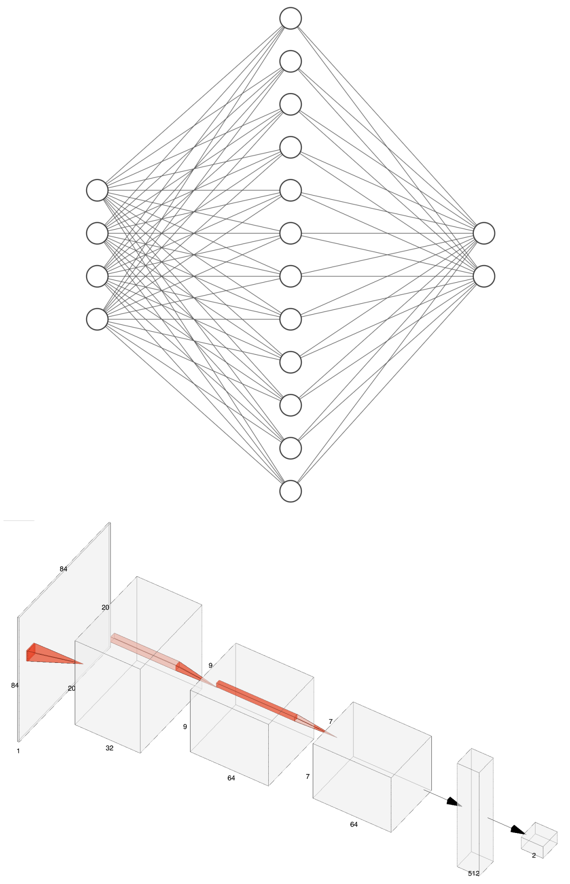

Figure 2: (**Top**) Simple MLP is used for CartPole mapping observation to state-action values or probability distributions over actions (**Bottom**) Three layer CNN followed by two dense layers are used for MsPacman

3.3 Training

Both policy gradients, REINFORCE and A2C, were trained on 2500 episodes (300K time steps) for approximately 5 minutes on CPU's for CartPole. Each episode was capped at 500 time steps which is the max reward threshold for CartPole-v1. The discount factor is 0.99. Both use an episodic training paradigm 2 3 and after each episode, the policy parameters π_θ are updated using backpropagation on the loss, equation 5 6 7, and an Adam optimizer with 2.5E-4 learning rate.

The DQN algorithm was configured for 18K episodes, with a 500-step limit per episode. Training took approximately 10 minutes on CPU's. Training leveraged a replay buffer of 1M experiences and a batch size of 32. The exploration rate decayed from 1 to 0.1 over 500K time steps using a linear scheduler. The discount factor was set at 0.99. Training involves updating the Q-network's parameters by minimizing the mean square error loss, equation 2, using backpropagation. An RMSprop optimizer with learning rate 2.5E-4 and smoothing constant of 0.95 followed the DQN research paper. The target network's parameters were updated after 50K step and updated every 10K steps afterwards.

For MsPacman, due to inefficient computational resources and timelines, I was unable to train on REINFORCE and A2C. However, a DQN agent was trained for 1K episodes (1M time steps), with a 4K step limit per episode. Training took approximately 8 hours on Google Colab's V100 GPU's. The same training configurations as a DQN agent in CartPole apply to MsPacman with the exception of the linear schedule expanding over 1M time steps instead of 500K.

4 Results

In Figure 4, the training outcomes for the REINFORCE and hybrid Actor-Critic (A2C) methods on the CartPole environment are presented, averaged over 100 episodes. Notably, REINFORCE outperformed A2C by achieving an average reward near 450 in the final 500 episodes, indicating a faster convergence to an optimal policy. This could be attributed to differences in policy update mechanisms between the two methods. Visual analysis confirms that both agents effectively learned to balance the pole. It is hypothesized that extended training could elevate A2C's performance to similar levels.

Figure 4 illustrates the training dynamics of a DQN agent on CartPole. The correlation between the linear scheduler (black line) and the average reward over 1,000 episodes (green line) suggests increased rewards correlating with a shift from exploration to exploitation. The red line indicates where training actually started, approximately 2.2K episodes (50K steps), with the target network updated 50 times over the course. Despite a longer training duration, the DQN agent's policy, peaking at an average reward of just over 100 in the last 1,000 episodes, appears suboptimal, characterized by gradual directional movements without effective pole stabilization.

For MsPacman, the assessment is based solely on qualitative metrics. The DQN agent, compared to a random policy agent, showed an average reward increase from 20 to 200. However, across numerous episodes, the DQN agent consistently failed to complete the objective of collecting all pellets, indicating room for further policy refinement either by hyper parameter tuning or increase in training time.

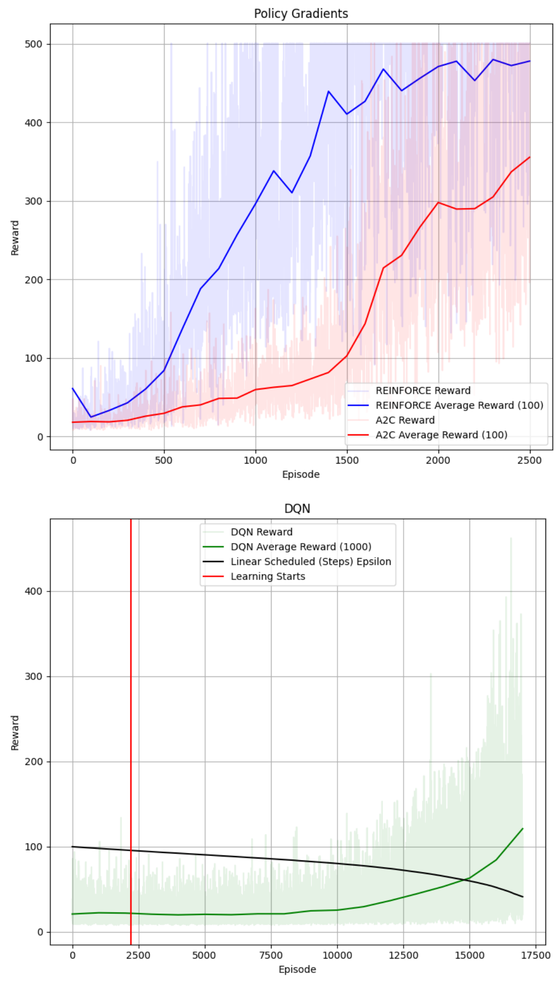

Figure 3: (**Top**) Training results for REINFORCE and A2C agents on CartPole over 2.5K episodes (**Bottom**) Training results for DQN agents on CartPole over 17.5K episodes

5 Conclusion

This research provides a detailed examination of DQN, REINFORCE and A2C algorithms, demonstrating their application and performance in complex reinforcement learning environments. My findings reveal that while REINFORCE quickly converges to an optimal policy in the CartPole environment, it is not without limitations in variance and sample efficiency. A2C, though slower in achieving optimal performance, offers the advantage of stability through its dual-network structure, suggesting potential benefits with further training and tuning.

The DQN model, despite its proficiency in handling high-dimensional input spaces, exhibited limitations in achieving optimal policies in both CartPole and MsPacman environments. This highlights the challenges DQN faces in continuous action spaces and its tendency for Q-value overestimation. The results indicate that while DQN can outperform random policies, it requires further parameter tuning and further training time.

In conclusion, this research underscores the importance of algorithm selection based on the specific requirements and characteristics of the environment. For future work, I'd like to consider more advanced and popular DRL techniques such as Proximal Policy Optimization (PPO) or World Models and integrate these techniques in more complex settings.

6 Contribution

I, Jonathan Samuel, have completed 100% of this project and have not collaborated with anyone else.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[2] S.J. Russell, S. Russell, and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson, 2020.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[4] Lilian Weng. A (long) peek into reinforcement learning. *lilianweng.github.io*, 2018.

[5] Lilian Weng. Policy gradient algorithms. *lilianweng.github.io*, 2018.

[6] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.