Play and Learn: Deep Reinforcement Learning

Adithya Bhat
2nd Yr CS Grad

Anshu Raina
1st Yr CS Grad

Arjun Singhvi
2nd Yr CS Grad

Ayon Sen
3rd Yr CS Grad

Abstract

Deep Neural Networks (DNNs) and Reinforcement Learning (RL) have been gaining a lot of attention over the past few years. Over the course of this project, we explore the domain of Deep Reinforcement Learning. The main objective of this project is to gain experience and evaluate the efficacy of using DNNs in conjunction with RL. They key takeaways being - the RL input and reward formulation significantly impacts the performance, the policy gradient algorithm outperforms the Deep Q Network algorithm and LSTM does help in improving performance of games that have partially visible state. Lastly, but more importantly, since we write all of the code from scratch, we have gained a hands-on understanding of the domain we set out to explore.

1 Introduction

In this project, we explore the conjunction of Deep Neural Networks and Reinforcement Learning (RL) to build agents that learn to play games. For the game environment, we make use of the OpenAI Gym Python environment [1]. Then, we re-implement various RL algorithms from scratch in Keras, a Python library. On the neural network side of things, we experiment with Recurrent Neural Networks and various network structures. From the RL perspective, we experiment with changing the state observation and the reward schemes.

The rest of the paper is organized as follows. We first introduce the related terms in Section 2. The exact problems we tackle are discussed in Section 3. Our experimental results and findings are provided in Section 4. In Section 6 we go through the related work in the field of RL. Finally, we discuss the future directions of this project in Section 7 and briefly conclude in Section 8. Individual contributions are stated in Section 5.

2 Background

In this section we provide a primer on a number of concepts so that the the rest of the paper can be appreciated even by readers who have not taken the course.

2.1 Deep Neural Networks

Similar to conventional neural networks, DNNs are composed of three different layers - input layer, hidden layer(s) and output layer. However, the difference being that in

DNNs, the input passes through multiple hidden layers. In other words, DNNs are neural networks that have multiple hidden layers. Intuitively, one can imagine each hidden layer to learn more complex features (by being trained by the features learnt by the previous hidden layer) in comparison to the previous layer. There has been a lot of success in this field in the past few years due to the advent of GPUs, machines with more computation power as well as open-source libraries that provide efficient implementations of vectorized operations, which are the basic operations used while training DNNs.

2.2 Reinforcement Learning

Reinforcement Learning is a subfield of Machine Learning, wherein the goal of a software agent is to take actions in an environment in order to maximize some notion of reward. To be more specific, as seen in Figure 1, a RL model consists of two main entities - agent and environment. At each timestep, the agent performs actions in the environment. This action leads to the environment moving onto a new state and may lead to a reward which is passed to the agent. Next, the agent performs its next action (with the goal of maximizing the cumulative reward) based on the information it received from the environment and this cycle continues.

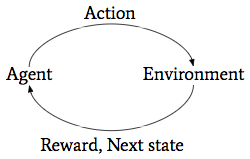


Figure 1: Reinforcement learning loop

It is important to note that RL differs from both supervised and unsupervised learning. In supervised learning one has labeled data and in unsupervised learning one has no labels at all. On the other hand, in reinforcement learning the rewards can be considered to be analogous to the labels and are therefore sparse and time-delayed.

2.3 Deep Reinforcement Learning

As the name suggests, in Deep Reinforcement Learning (DRL), we use deep learning techniques to carry out the task of RL. We choose two DRL algorithms that have been proposed in prior work, which we discuss next.

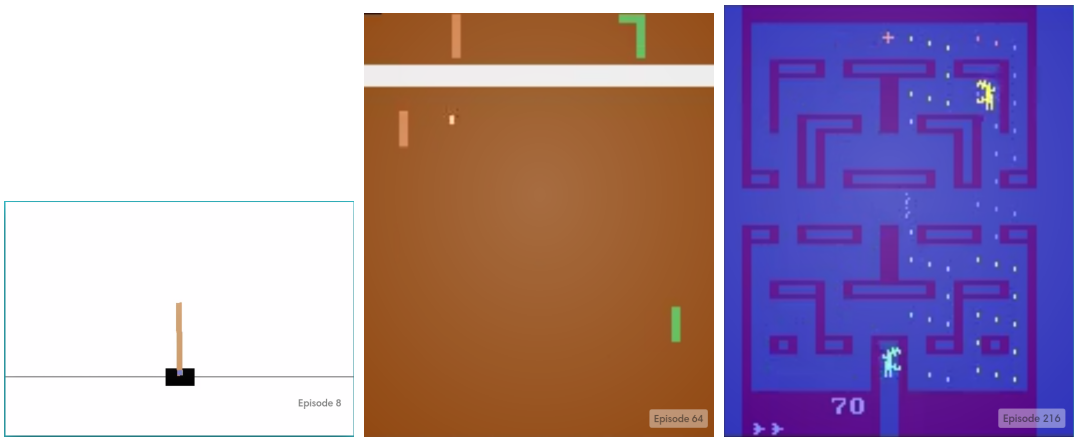
2.3.1 Deep Q Network

Traditionally, Q-Learning has been used in RL. In Q-learning, we define a function that represents the maximum reward that can be achieved when we perform a particular action in a particular state of the environment. The core idea of the Deep Q Network (DQN) algorithm proposed by DeepMind [2] is to represent the Q-function as a DNN. Essentially, the DQN algorithm is a variant of the Q-Learning algorithm. The pseudocode of the algorithm can be found in [2].

DQN introduces the notion of *Experience Replay*. During a gameplay, all the interactions with the environment are stored in a replay memory. While training the deep neural network, random mini batches of the interactions are picked instead of the most recent one. This allows the neural network to be more robust and prevents it from getting stuck at a local minima.

2.3.2 Policy Gradient

Another method for RL that has been garnering a lot of attention during the last few



(a) Cartpole (b) Pong (c) Alien

Figure 2: Screenshots of different games

years is policy gradient [3]. Instead of optimizing the expected reward, policy gradient optimizes policies with respect to expected return. Here for each possible input we learn the best possible action. Let $\theta \in \mathbb{R}^K$ be the policy parameters. Policy gradient optimizes the following expected return

$$J(\theta) = \mathbb{E}\left(\sum_{k=0}^{H} a_k r_k\right) \tag{1}$$

Here a_k denotes the weighting factor and r_k is the reward received at the k-th step. We want to see how a neural network employing policy gradient performs with respect to a DQN.

3 Problem Definition

In this section, we discuss the problem that we are trying to tackle. Our goal is to learn different strategies of performing DRL. To do so we look at three different games namely Cartpole, Pong and Alien. In what follows, we discuss the input and outputs of these games and how the challenges are different for each of them.

3.1 Cartpole

We start our exploration of RL algorithms using a simple environment. For this, we select the Cartpole game. In the Cartpole problem, also known as the inverted pendulum problem, the objective is to maximize the number of timesteps for which the pole remains balanced on the cart, while also ensuring that the cart does not move too far from the center. At each timestep, the agent must choose to push the cart left or right. The state observed by the agent consists of the cart's position, its velocity, the angle the pole makes with the normal and the velocity of the pole's tip. Figure 3 shows the network structure.

3.2 Pong

Our main goal is to figure out how RL works for different games. In most cases different games will have different types of states. Thus generalizing the concept as a whole is

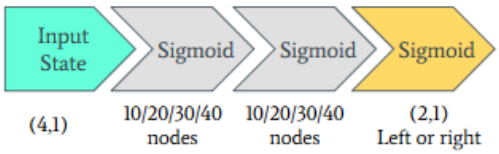
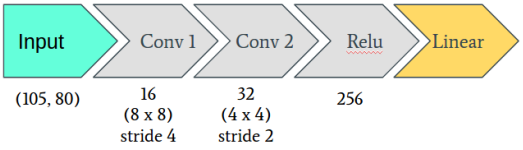


Figure 3: Neural network structure for Cart-pole

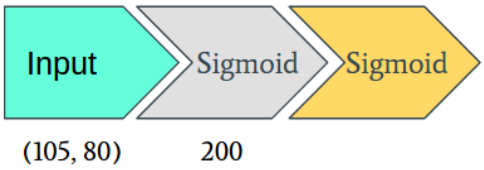
not simple. To work around this, each state in a game can also be viewed as an image. Then the input to the neural network would be the pixel values of that image. This concept can be used for a vast array of games.

To explore this idea we choose the game Pong (Figure 2b). In Pong, the goal is to play a game of virtual ping pong against a computer controlled agent. Whichever agent scores 21 points first wins the game. The input for pong is a $210 \times 160 \times 3$ image. There are 6 possible actions listed in the game. A reward of $+1$ is provided if the computer agent misses the ball. If our agent misses the ball then a reward of -1 is provided. In all other cases the reward is 0. The total reward that can be gained from an episode ranges between $[-21, 21]$.

As mentioned previously, we also wanted to see the impact of using policy gradient algorithm compared to a DQN. We do so by using Pong. Figure 4 shows the neural network structures that we have used to perform the comparison between these two strategies. In both cases we subsample the input image to a gray scale image of size 105×80. For the DQN, we used two convolution layers, followed by a fully connected layer with rectifier linear units (relu) and a linear output layer. For the policy gradient neural network we used a much simpler structure. It has only one hidden layer with 200 units with sigmoid activation function. The output layer also uses sigmoid activation function.



(a) DQN structure for Pong



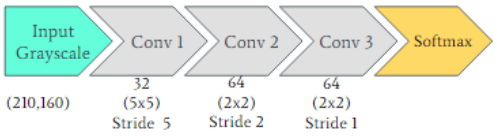
(b) Policy gradient neural network structure for pong

Figure 4: Neural network structures for Pong

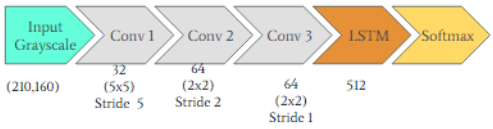
3.3 Alien

For some games, the game world states are not fully visible at all times e.g., there are some games where one portion of the screen is obscure at any given point of time. One such game is Alien. Alien is a maze video game and essentially a clone of Pac-man. The objective of this game is to run through the hallways of your space ship and crush all the alien eggs which have been placed there. The player can also use a flamethrower to kill the adult aliens and snatch up as many prizes as possible. This game has flickering screen and hence the environment is partially observed. The input image size for Alien is $210 \times 160 \times 3$.

Since the input is an image, we first start with using Convolutional Neural Network (CNN) to determine the action of agent. Figure 5a represents the architecture of neural network. We first convert the input image into a gray scale image of 210×160 and then pass it on through the convolutional layers. We also try another structure where we added one recurrent Long Short Term Mem-



(a) DQN neural network structure for Alien using CNN



(b) DQN structure for Alien using CNN and LSTM

Figure 5: Neural network structures for alien

ory (LSTM) layer [4] after the convolutional layers. Figure 5b shows this architecture. The LSTM layer has 512 hidden units. The structure of the network is similar to the structure given in [5]. We believe that the LSTM layer should be helpful for this game because it would make neural network successfully integrate information through time and thus prevent it from taking incorrect decisions based on just the present frame.

4 Experiments and Results

We present our experimental findings in this section. For all our experiments we use Python OpenAI Gym [1] framework. It provides a convenient way of interacting with the game while abstracting away the complexities. At each time step, the environment provides us with an input state (image or parameters for that timestep). We can select from a list of possible actions. Performing that action gives us a reward and also the next input state. Each game ends after a certain criterion is fulfilled. This is known at the end of an episode.

As mentioned previously in Section 3, we try out three games namely Cartpole, Pong and Alien. In each game the goal is to maximize the score that can be achieved for one particular episode. We base the evaluation of the neural networks on this criterion as well. In the following subsection we discuss the methodology. Then we present our results for these three games.

4.1 Methodology

We believe that re-implementing the learning agents would lead to better understanding. For this purpose we used primitives provided by the Keras library in Python. Experiments were run on a combination of CloudLab and Google Cloud. CloudLab provides powerful CPUs with considerable RAM, while Google-Cloud is used for the NVidia K80 GPU, for running experiments with image input.

4.1.1 Randomness in Measurement

There are multiple sources of randomness in the experimentation environment. Once a game reaches a certain state, the underlying physics/game engine provides a consistent experience. However, the start state of each episode is not fixed, making it hard to chart fine grained improvement over time. Secondly, the learning algorithm itself often has a random exploration feature to it. Thirdly, the neural networks are initialized to random weights. Fourthly, a random sample of the past states is used to train the network. While setting the random seed in python was able to solve two of these, it did not resolve the variability of the start state, and as a result, the training sample. Thus, we decided to allow randomness, and ran the experiments duplicate or triplicate. We restrict

our discussion of results to those that we can consistently replicate.

4.1.2 Metrics

In simple supervised learning, the error almost always decreases with more training epochs. However, in reinforcement learning, the same does not hold true for the rewards. In addition to the the randomness discussed above, reinforcement learning algorithms like Q-Learning involve function approximation on a sample of data, which has implications on incremental performance. Thus, looking at individual successive episode scores does not always provide a meaningful picture. Hence, we use the following metric.

Moving Average

At each episode of the data, we compute the simple average of the previous 100 episode scores. This helps to analyze the trend of the scores. The disadvantage of this metric is that individual points of data are lost, only the aggregated values are represented.

4.2 Cartpole

In the Cartpole environment, each episode lasts for 200 timesteps, and the score is the number of timesteps the pole remains upright. We experiment with various network structures, try modifying the reward structure, and test if training for more episodes/timesteps or with more distributed data affects the performance. In addition, we also modify the environment to change the 200 timestep limit to better evaluate real life performance. Thus, there are training scores limited to 200, while evaluation scores are can reach up to 1000.

4.2.1 Effects of Varying the Network Structure

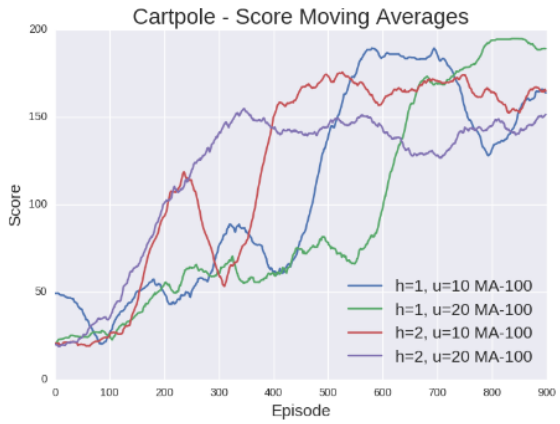
As seen in the Figure 6, we find that a simple network with one input, one hidden and one output layers, and 20 hidden nodes is sufficient to approximate the Q-function. The more complicated network structures probably need more training to improve performance.

4.2.2 Effects of Varying the Reward Mechanism

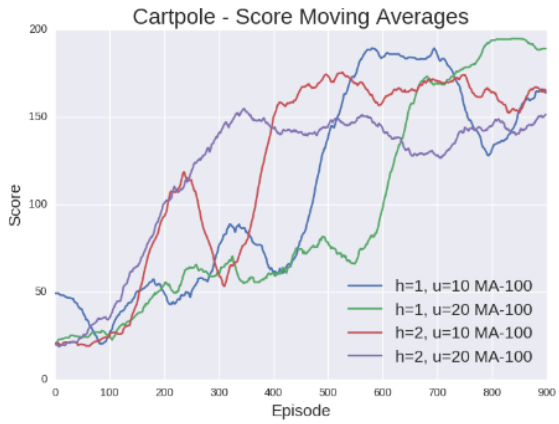
In addition to the default reward mechanism, we experiment with what we call positive enhancement and negative enhancement. In each of these, we positively or negatively bias the score depending on whether the pole is upright or not at the end of the training episode, by a weight of 10 timesteps. We find that positive enhancement performs worse than negative and the default reward structure. This is not surprising since we do not know how long the pole will actually stay upright, which could be orders of magnitude more than the 10 timesteps we add. This would result making the training data for the Q-function approximation to be noisy. While the default mechanism and the Negative Enhancement perform comparably in the training episodes, as seen in Figure 7a, in the evaluation episodes we find that Negative Enhancement performs significantly better, as shown in Figure 7b.

4.2.3 Effects of Training with more data (More Episodes)

This experiment highlights the difference between reinforcement learning and regular supervised learning. Comparing the performance of models that are trained for 1000

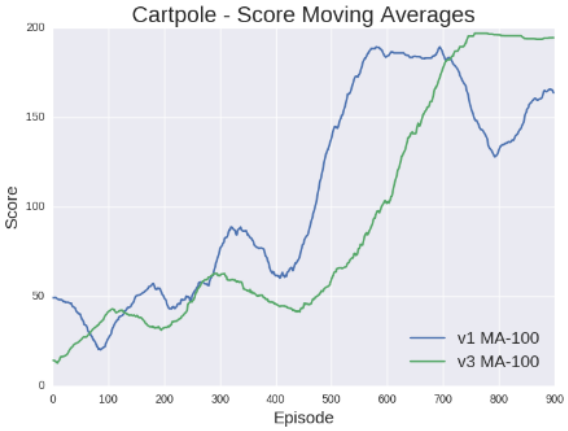


(a) 1 hidden layer vs 2 hidden layers

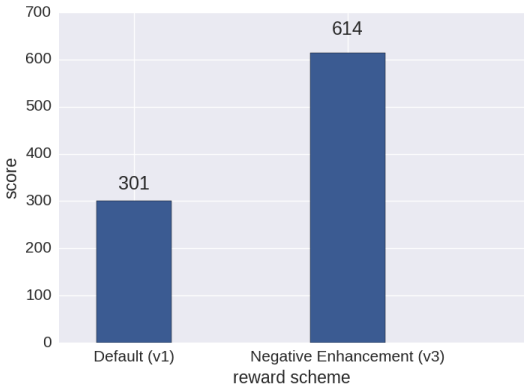


(b) 1 hidden layer with various number of nodes

Figure 6: Various network structures for Cartpole - training scores



(a) Training scores



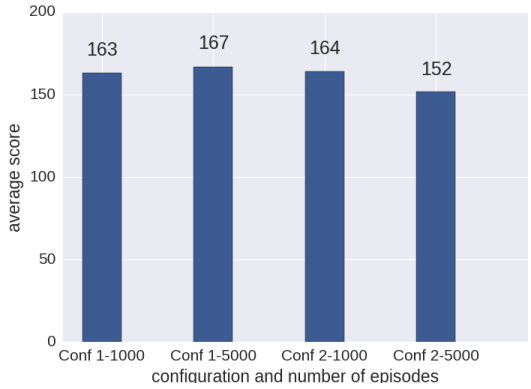
(b) Average evaluation score (Rounded to Nearest Integer)

Figure 7: Reward Schemes - Default(v1) and Negative Reinforcement(v3) - Cartpole training scores schemes

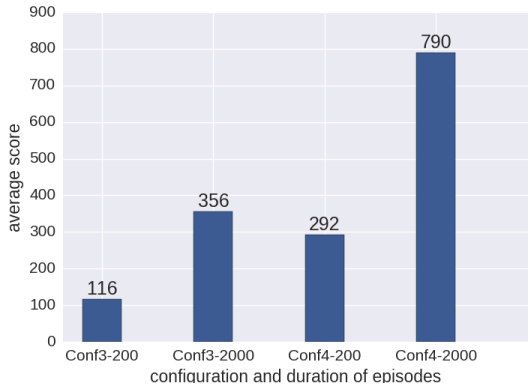
and 5000 episodes, all else remaining constant, the performance of the two models is almost identical as seen in Figure 8a.

4.2.4 Effects of Training with better data (Longer Episodes)

We observe that given that the start state varies in a small range of possible values, in the 200 timestep episode length, a vast majority of the visited states are near the center of the state space. This impacts the sample



(a) Training with more episodes



(b) Training with longer episodes

Figure 8: Effects of training with more and better data (average score rounded to nearest integer)

of the states that are used to train the neural network too. In other words, the training data is biased with insufficient data on state-action pairs near the corners. This results in the agent learning how to keep the pole upright, but it fails to learn to keep the cart inside the arena. We decided to experiment on this by modifying the training episode duration to 2000 timesteps. As seen in the Figure 8b, this resulted in much better performance.

A key point to note here, is that since the sample size for each training step is the same, and the number of episodes/training steps is the same, the quantity of training is exactly the same. Only the distribution of the training examples, i.e., the quality of the training data, varies.

4.3 Pong

The neural network structure used for Pong is already discussed in Section 3. As mentioned, our goal is to compare a DQN against a policy gradient neural network. Also note that the input images of Pong do not provide all necessary information of a state. For example, just by looking at an image we cannot decide which way the ball is moving. One common strategy to counter this is to take the difference between two consecutive images and use that as the state. We also wanted to see how using this strategy works against taking just the input image as a state. To do so we use the difference between consecutive images as a state for both the DQN and the policy gradient (PG1) neural network. We also train a separate policy gradient neural network with the same structure where the input state was just the image and not the difference between two consecutive states (PG2). The learning rate was set to 0.001 in all cases. We present our findings in this section.

We trained all three neural networks for 12000 episodes. We used the average reward as our metric for measuring which network performs the best. The results are shown in Figure 9.

As can be seen from the figure, the average reward improves a lot more for the policy gradient neural networks. This clearly suggests that the policy gradient neural network

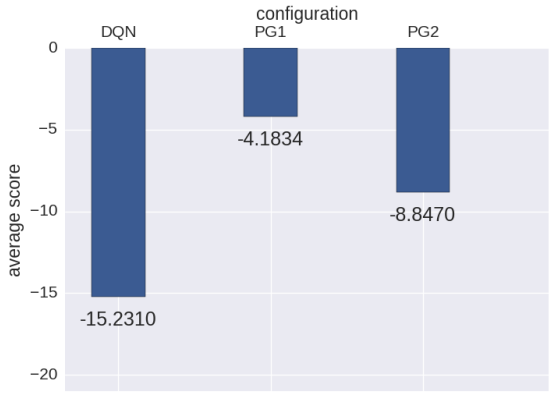


Figure 9: Training results for Pong after training for 12000 episodes

learns much more quickly than a traditional DQN even when convolution layers are not being used. Also note that PG1 performs better than PG2, which suggests that using the difference between two consecutive images as the state does provide more useful information to the neural network.

4.4 Alien

As discussed earlier in section 3, Alien has a flickering game screen which makes its environment partially observed and the input to the neural network is an image. In this section, we analyze the performance benefits of adding an LSTM layer on the top of convolution layers in the neural network. We report our scores on both configurations (with and without an LSTM layer) and then perform a comparison between the two.

4.4.1 Effects of adding a LSTM layer

The structure of the neural networks for Alien is given in Figure 5. Note that in Figure 5b all the parameters including the number of convolution layers and the filter size for each layer remain the same as Figure 5a. Only difference is the introduction of an LSTM layer.

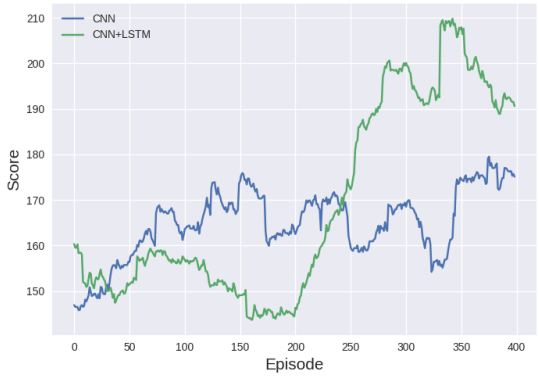


Figure 10: Comparison of the average scores in Alien with and without an LSTM layer

Figure 10 presents the comparison of moving averages of best 500 episodes with and without LSTM. The peak average score which we get when we use only CNN is 180. But the peak average jumps to 210 after adding an LSTM layer. There is 17% improvement in the peak average score as compared to using only CNN. The reason can be attributed to the LSTM layer being able to keep track of the past history and thus resulting in more correct decisions. So, LSTM layer does help in the games that have flickering frames. The score which we got is one of the top scores for Alien-v0 in Open AI Gym. The algorithm can be found at [6].

5 Individual Contribution

Everyone in the team was new to OpenAI. So, all of us started with familiarizing ourselves with the OpenAI Gym by looking at the Cartpole environment. Adithya took lead on setting up the different versions of the Cartpole environment. Ayon took lead on the

implementing the DNN for the Pong enviroment. Anshu took lead on implementing the DNN for the Alien environment. Arjun took lead in setting up the infrastructure.

The algorithms which we implemented were detailed, and though each one had a lead person, all the others were involved in debugging and thus learnt what the other person was doing. All of us were involved in collecting and interpreting the results as well as worked on the report.

6 Related Work

DeepMind's paper [2] is the first one to investigate the impact of using DNNs in the context of reinforcement learning. The DQN algorithm was proposed in [2] and its effectiveness was evaluated to seven Atari games. However, all the games chosen by them used images as an input. As a part of our work, in addition to evaluating the DQN algorithm when images are used as input, we even tested its efficacy when that was not the case. The DNN structure mentioned in [2] was used as a reference point for our experiments.

For the partially observed states, there have been efforts to add recurrency to the neural network [5]. This paper provides recurrency as an alternative to stacking of frames and feeding it to the Deep Q network.

7 Future Work

Our goal for this project is to learn how the different RL algorithms work in conjunction with deep neural networks. Even though we try out several problems for this purpose, we are not able to run all of them to convergence due to time and processing power limitations. As a future task we would like to see the highest scores we can get for each of these tasks. We would also like to vary the different hyperparameters of the neural networks e.g., number of hidden layers, activation functions used in hidden layers etc. and see how that impacts the individual tasks. That might give us some insight on which structures are best for a particular type of task. Visualizing the layer activations, and using pre-trained models for a different task, such as image classification, to initialize our model would also be an interesting approach.

8 Conclusion

For this project, we try out reinforcement learning in the setting of deep neural networks. We use multiple games from the OpenAI Gym as the problems to tackle. Moreover, we analyze our results using different neural network structures and RL strategies. Our experiments suggest that multiple possible neural network structures may perform equivalently for reinforcement learning. The policy gradient strategy performs better than Deep-Q learning. Furthermore, for image inputs it is better to take as input the difference between two images instead of using just the image for a given time as the former provides more information. A higher level take-away is that if the design of inputs and reward system is appropriate, it significantly enhances the performance of our agents. Finally, for the games that have partially visible state, adding a recurrent neural network(LSTM) on the top of convolution layers helps to increase the performance.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[3] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *arXiv preprint arXiv:1507.06527*, 2015.

[6] "Our algorithm for alien." [Online]. Available: https://gym.openai.com/evaluations/eval_yzRouTxgS1auUoWQlQvQA