Analysis Of CNN vs FNN on Various Reinforcement Learning Algorithms

Justin Zhang
Department of Computer Science
Stony Brook University
justin.zhang.1@stonybrook.edu

Abstract

Historically, Convolutional Neural Networks have been proven to be better than any other form of neural network including a Feed Forward Neural Network at learning from images. In this experiment, I will explore and quantify the differences between the use of the two different neural networks.

1. Introduction

A Feed Forward Neural Network (FNN) is a basic neural network structure that consists of an input fully connected layer, hidden fully connected layers, and an output fully connected layer. A Convolutional Neural Network (CNN) is similar, however, it has a series of convolutional layers before the fully connected layers. These layers allow the CNN to learn from properties within an image like a line or circle. With this in mind, I tried to compare the results of a feed forward neural network and a convolutional neural network to see why a CNN is better than a FNN at learning from images in this project. The environment I used was the InvertedPendulumMuJoCoEnv-v0 environment where I extracted the image state and used that as the input to both neural networks. I also made use of REINFORCE, Fitted Q Iteration, and DQN as the learning algorithms. I made sure to keep the implementation of the logic for each algorithm identical between the implementations for the CNN vs FNN.

2. Preprocessing

2.1 Image

For ease of use, the states of the environment need to be preprocessed. The output of the environment is a 240 x 320 x 3 size np array. Training using this size of input would take far too long. The solution that I used was to first grayscale the image so the new shape is 240 x 320 to remove the extra rgb layers. Next I scaled the image down to a 50 x 50 image which is far more efficient to use than the original 240 x 320 x 3 image and ran far faster.

2.2 Action

The action space of the inverted pendulum environment is a continuous space of [-1, 1] which could be difficult to work with in algorithms such as Fitted Q Iteration and DQN. My solution for this is to discretize the action space into 21 distinct actions starting from -1 and incrementing by 0.1 until 1.

2.3 Replay Buffer

The replay buffer class helped with learning in the fitted Q iteration and DQN algorithms. It saved pairs of state, action, reward, and future state (SARS) which could be batched together for learning purposes in the algorithms.

3. Neural Network Architecture

For this experiment, I need two base neural networks, one CNN and one FNN. The FNN is a simple neural network that consists of an input layer of 2500 nodes that represents the flattened preprocessed image state from the environment. The output layer consists of 21 nodes that represent the 21 discretized actions that the neural network can take. The CNN is a convolutional layer that takes an input of one 50 x 50 image state from the environment. The output is a fully connected layer that consists of 21 nodes that also represent the 21 discretized actions discussed prior.

4. Training Logic

One very important thing to mention is that the logic between the two neural networks' implementations of each algorithm is identical except that the input state for the FNN is flattened into a 2500 long tensor. While the cnn is kept as an image

4.1 REINFORCE

For this algorithm, I ran a total of 2000 episodes to gather all the data and batched the data into lists. The neural networks outputted a 21 sized tensor which represented specifically the probabilities of each action being taken so it was necessary that a softmax was done on the output of the final layer. Once the batched threshold size is reached, the loop to optimize the neural network weights is used to back propagate the gradients. The gradient is described in *figure 1* where each trajectory t is batched together into lists which are then reset at the end of the loop for future learning.

$$\nabla_\theta J(\theta) = \sum_i \left(\sum_t \nabla_\theta log \pi_\theta \left(a_t^i | s_t^i \right) \right) \left(\sum_t r\left(s_t^i, a_t^i \right) \right)$$

Figure 1

4.2 Fitted Q Iteration

The policy used here was a decaying epsilon greedy policy where epsilon started off as 1 and decays to 0.02. This allows the algorithm to first explore while training before reinforcing the good habits. Every step taken is stored into the replay buffer which can store up to 10,000 different SARS pairs to batch from. The target neural network weights must be updated every step in the environment to align with the FQI algorithm. After the buffer size reaches a certain threshold, specifically 2 times the batch size which is 32, The loop for backpropagation starts to optimize the weights. The loss function used here is mean squared error to prevent multidimensional loss. The target for this loss calculation is shown in *figure 2* with a gamma of 0.99. The algorithm then loops again for the next iteration/episode where the buffer is not reset.

$$\sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)\left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma max_{a'}\left(s_i', a_i' \right) \right] \right)$$

Figure 2

4.3 DQN

The implementation of the DQN algorithms are very similar to fitted Q iteration. The policy was also a decaying epsilon with the initial epsilon being 1 and decaying to 0.02. Once again this ensures that the agent can explore at first before reinforcing the good behaviors. Every step taken is also stored in its own replay buffer of size 10000 for future batching. Once the batch size reaches the same size threshold of 32, the loss is back propagated to update the weights with the loss function used here also being mean squared error. The target for this loss function is described below in *figure 3* where gamma is 0.99. The only difference between this algorithm and the fitted Q iteration is that the target neural network is updated outside the steps before the state of each episode to align with the DQN algorithm.

$$\sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)\left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma max_{a'}\left(s_i', a_i'\right)\right]\right)$$

Figure 3

4. Results

This section describes the results of each implementation and tries to compare the two neural networks.

4.1 REINFORCE

The graph in *figure 4*, shows the episodes vs rewards while training the CNN. The *figure 5* graph is the FNN episodes vs rewards. It is clear that the FNN is very unstable compared to CNN resulting in poor performance in the average across 10 episode tests where the CNN returned around 21 reward while the FNN returned around 11 reward.

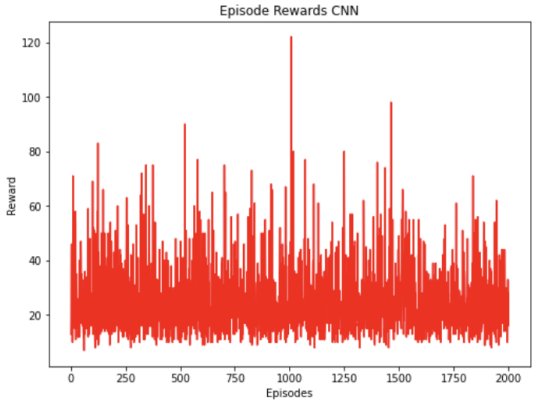

Figure 4

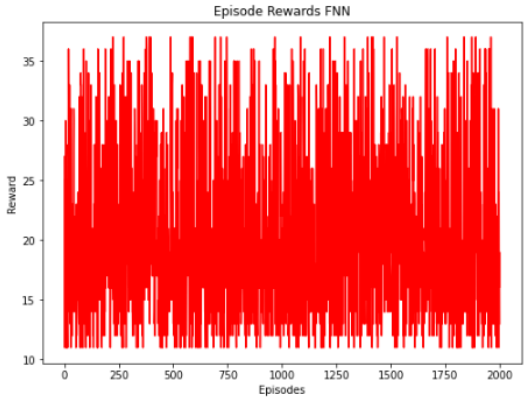

Figure 5

4.2 Fitted Q Iteration

Figure 6 shows the rewards vs episodes of the CNN when trained while *figure 7* shows the rewards vs episodes of the FNN when trained. It is clear that neither of the algorithms worked but we can clearly see that the FNN was very unstable especially around the episode 500 mark and afterwards. The CNN returned on average 12.8 reward across 10 episodes while the FNN returned around 8.8

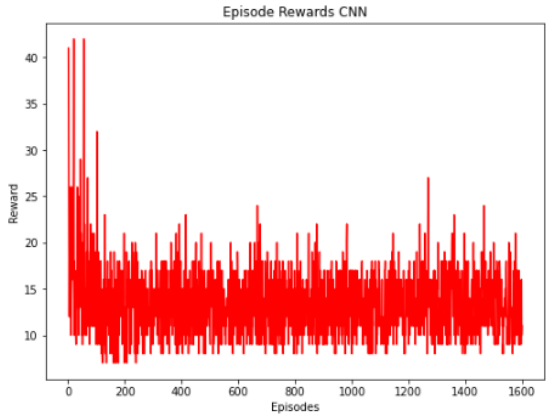

Figure 6

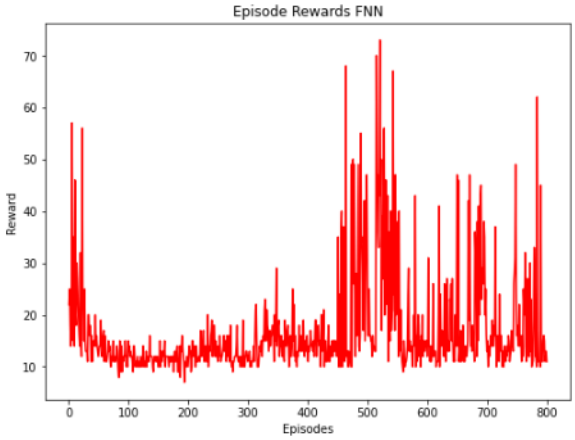

Figure 7

4.3 DQN

The graph in figure 8 depicts the episodes vs reward from the training of the CNN while the graph in figure 9 depicts the episodes vs reward from the training of the FNN. Once again, neither of the algorithms learned properly but it is clear that the FNN was very unstable as the CNN, at the very least, converged around 10. The average return across 10 episodes for the CNN was 11 and the average return for the FNN was also 11.

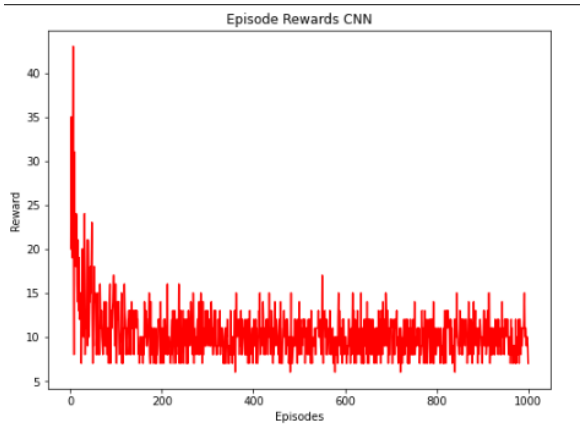

Figure 8

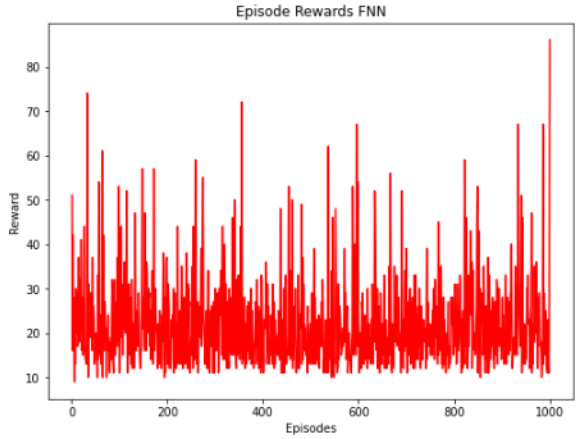

Figure 8

5. Conclusion

Regardless of the outcome of the training of individual neural networks, we can still see the clear difference between using a CNN and FNN. All three algorithms resulted in a more stable CNN network than a FNN network. This is because of the flattening of the state when processing the image. The CNN was able to capture the properties of an image, specifically where the pendulum is. FNNs ran faster than CNNs but at the cost of stability in the final trained neural network. In the end, it is clear that CNNs are better at learning than FNNs when it comes to images.

6. Source Code

https://drive.google.com/drive/folders/1ngUyu5iQ-nIdNW2qv4wUem4h_6faBhbw?usp=sharing