Q-Learning for the Cart-Pole Problem

Joe Shymanski

April 28, 2022

Abstract

This project seeks to investigate Q-learning using the cart-pole problem. Double Q-learning will also be implemented as a potential improvement to the pitfalls of the former model. All results are displayed graphically in terms of game steps per game and compared against each other to obtain a best solution.

1 Introduction

Reinforcement learning (RL) is has numerous practical applications. Often, it is beneficial to have a RL algorithm which can model any environment effectively. For this project, Q-learning will fulfill this environment-agnostic approach. This algorithm has its shortfalls, which double Q-learning can help fix.

2 Problem

The cart-pole problem, easily implemented in Python using OpenAI Gym [1], has a four-dimensional input space and a one-dimensional output space. The environment consists of a cart with a vertical pole attached to its center at a pivot point. At each step, the cart must move either right or left on its one-dimensional track. The goal is to balance the pole for as long as possible while staying within a certain distance from the center.

3 Algorithms

The Q-learning algorithm is used for RL. At any given state, it seeks to choose the action which yields the highest quality value according to the immediate reward as well as the rewards of the best future actions. Q-learning can sometimes have an issue with overestimation, so alternative methods have been created to fix this.

The other algorithm investigated in this project is double Q-learning. This is an extension of Q-learning with a slight change: there are now two Q functions for determining the quality of a state and action (Q_A and Q_B). The update rule for Q_A replaces the term $\max_a Q(s_{t+1}, a)$ with the term $Q_B(s_{t+1}, \operatorname{argmax}_a Q_A(s_{t+1}, a))$. Essentially, the action that would be chosen by policy A is fed into the value estimator of B and vice versa. This solves the problem of overestimation that traditional Q-learning faces.

4 Results

The default parameters for the Q-learning algorithm are as follows: two hidden, dense neural network layers, each with 64 neurons; LeakyReLU with default parameters as the hidden layer activation function; a learning rate which starts out as 0.01 and decays according to an exponential schedule with the decay rate set to 0.9; a discount factor of 0.99; and an epsilon value which decays from 0.3 to 0.05 at a rate of 0.99, which is used to determine whether or not to take a random action. The performance of this model is shown in 1.

The first test on traditional Q-learning was to alter the architecture of the Q-learning network to see if any improvements were possible from the default. Figure 2 shows these different performances. Then the rate of decay for the learning rate was altered, which is shown in Figure 3. Finally, alternative activation functions were investigated, specifically ReLU and tanh. Figure 4 demonstrates these results.

After obtaining and interpreting these graphs, the best parameters were then used to test the double Q-learning algorithm. These results can be found in Figure 5.

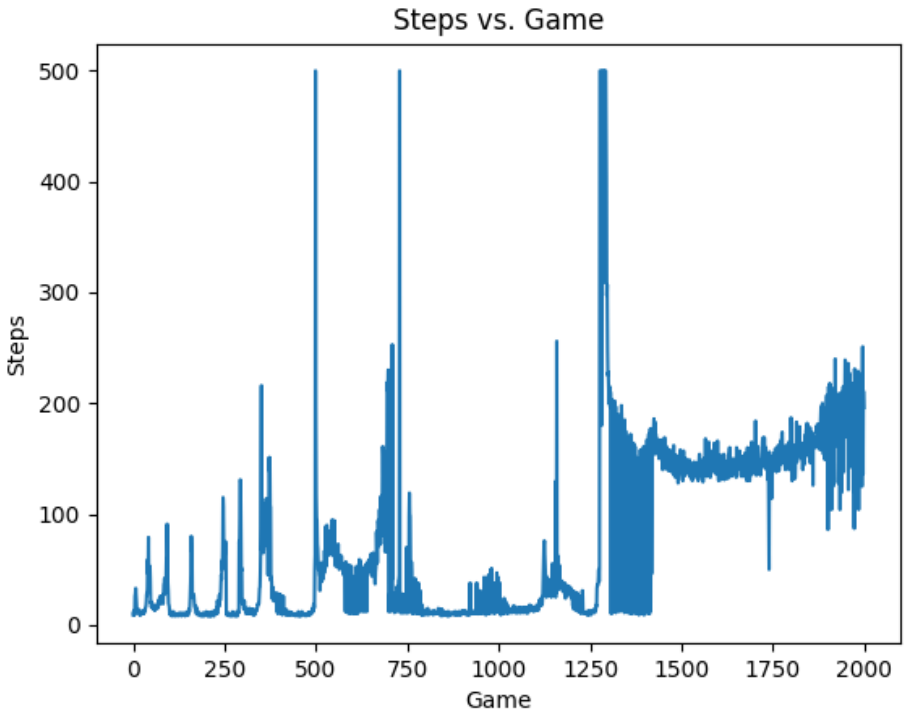
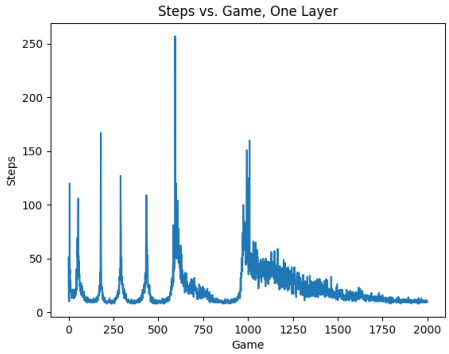
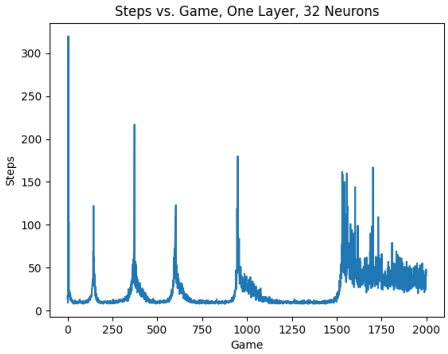


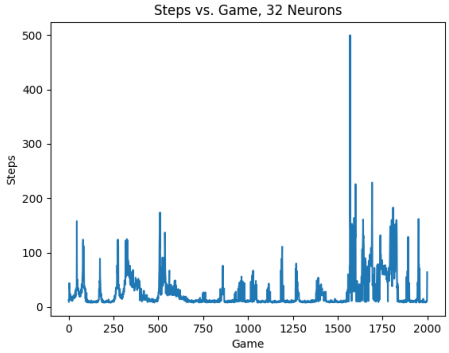
Figure 1: Default parameters



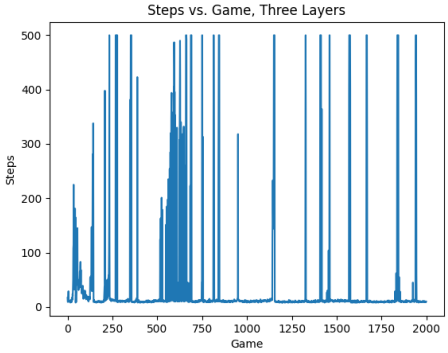
(a) One layer (64)



(b) One layer (32)



(c) Two layers (32, 32)



(d) Three layers (64, 32, 16)

Figure 2: Different hidden layer architectures

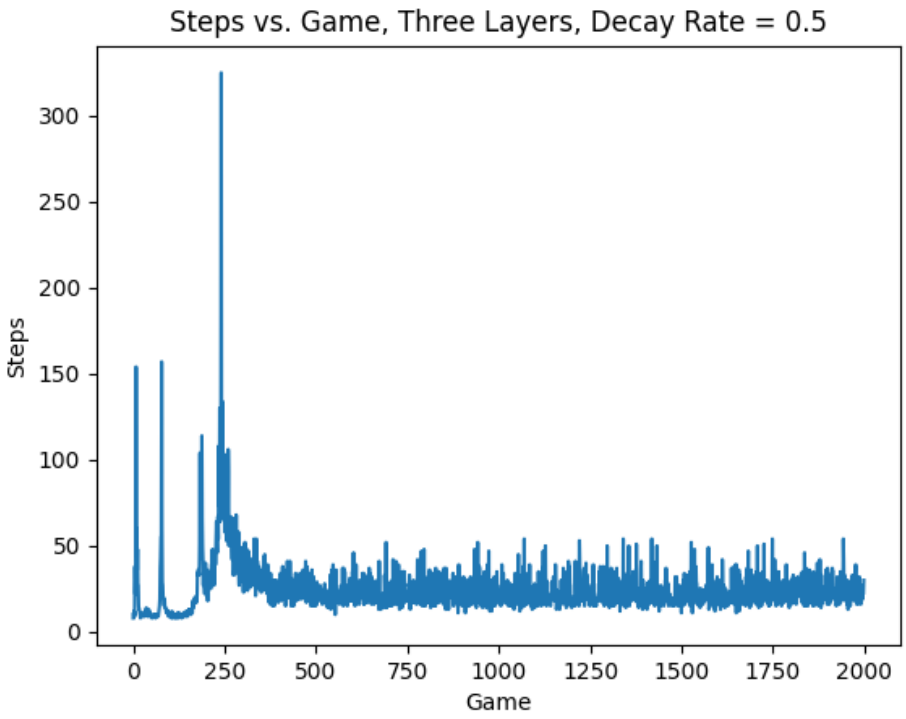
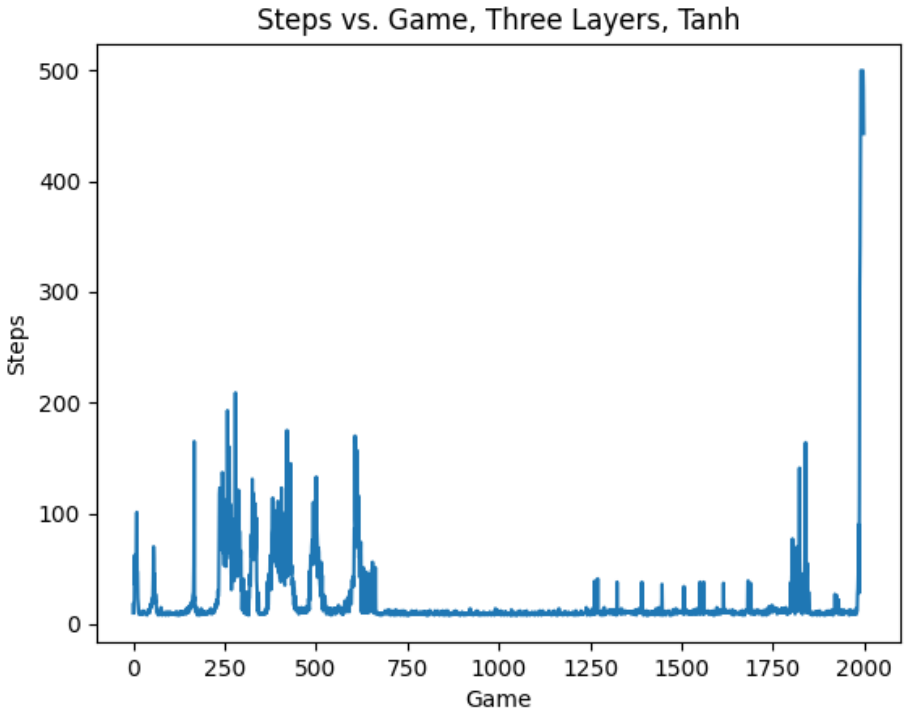
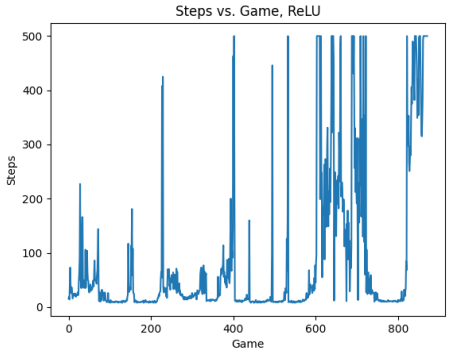


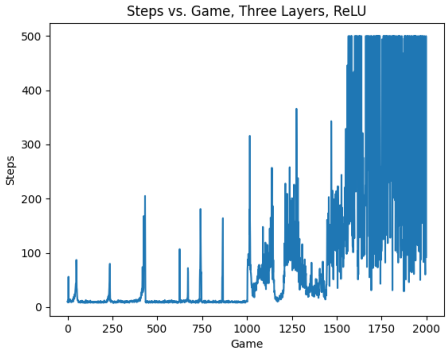
Figure 3: Three layers (64, 32, 16), decay rate = 0.5



(a) Three layers (64, 32, 16), tanh



(b) Two layers (64, 64), ReLU



(c) Three layers (64, 32, 16), ReLU

Figure 4: Different activation functions

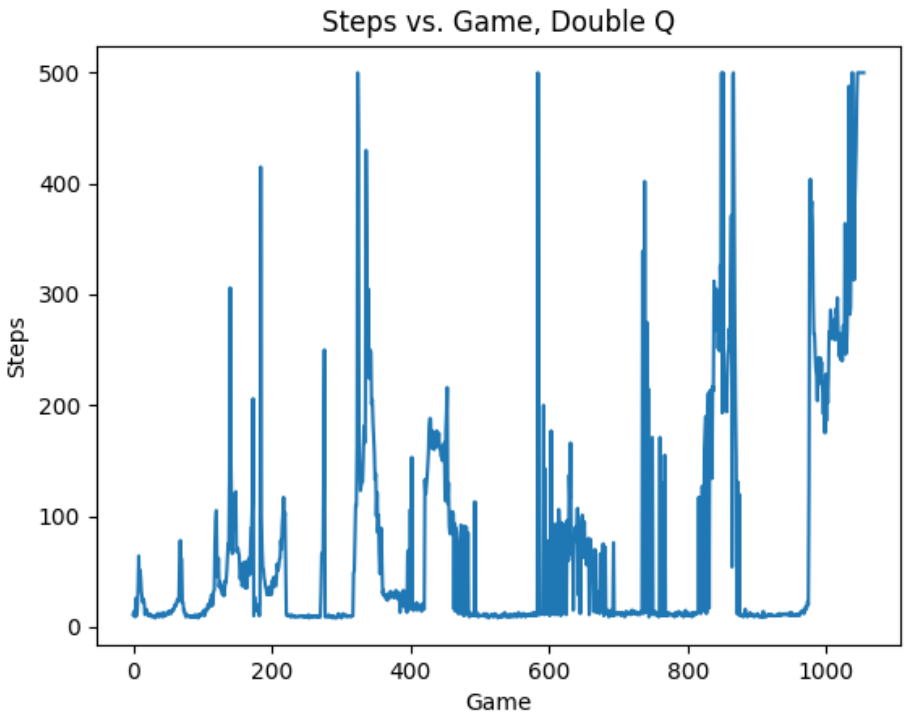


Figure 5: Two layers (64, 64), ReLU, double Q-learning

5 Discussion

For traditional Q-learning, the best networks by performance were essentially those with the most parameters, which were the network with two layers of 64 neurons and the three layer setup. According to Figures 1 and 2d, they were able to reach the maximum number of steps multiple times, and the former averaged out to a respectable 200 steps per game. Both, however, were awfully inconsistent at times, but decreasing the learning rate's decay did not improve upon the model, according to Figure 3. What did help, however, was a different choice of activation function. While tanh did not perform well until the very end, according to Figure 4a, ReLU showed promising results. Both networks with the ReLU activation function completed all 10 test games with the maximum number of steps, something no other network successfully did. In fact, the two layer version was unique in that it stopped early, and the three layer network was quite close to doing so itself.

Using the parameters which seemed to provide the optimal performance, the double Q-learning algorithm performed quite similarly, according to 5. This is surprising, as I would have expected it to outperform its traditional counterpart greatly, due to its ability to avoid overestimation. Perhaps my implementation of the algorithm is missing a nuance of the algorithm which would reveal such a difference, or maybe the difference is quite small to begin with. Still, it came to a solution which stopped early and lasted the maximum number of steps for all the test games, so no harm was done by its utilization.

6 Conclusion

In the end, it appears that the cart-pole problem can be effectively solved using RL techniques. When using the Q-learning algorithm with a neural network, a small number of layers with about 3,000 to 4,500 parameters seems to do the trick. Double Q-learning often helps aid convergence by preventing overestimation, though its benefits were not easily discernible within this context.

References

[1] OpenAI. Cartpole-v1, 2016.