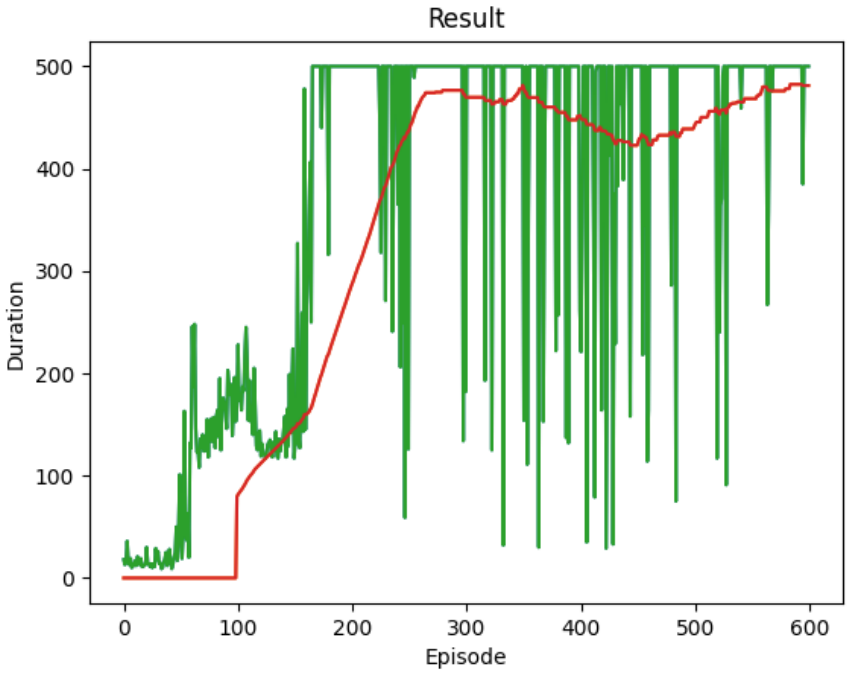
Billy Tobin

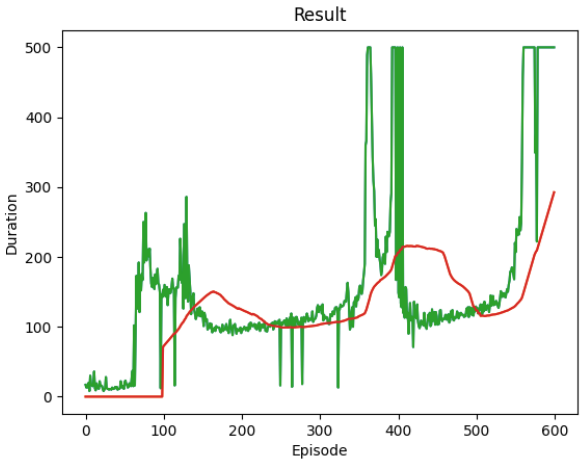
Assignment 6

DQN for CartPole

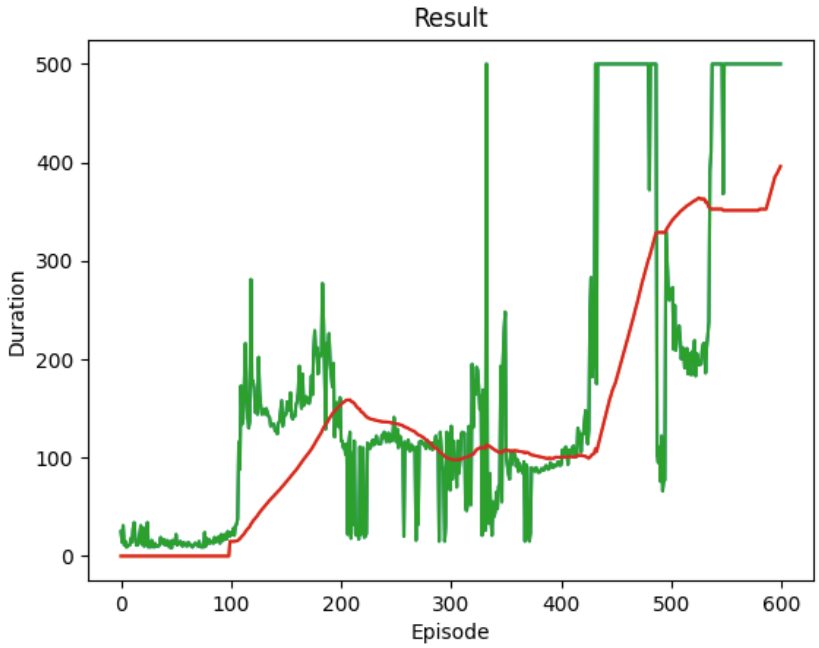
The initial NN architecture I tested had an input layer, two hidden layers, and an output layer. Here are the results from the first trial, with 256 nodes in each of the two hidden layers.



Next, I decided to decrease the number of nodes in each layer from 256 to 128. Here are the results from that trial.

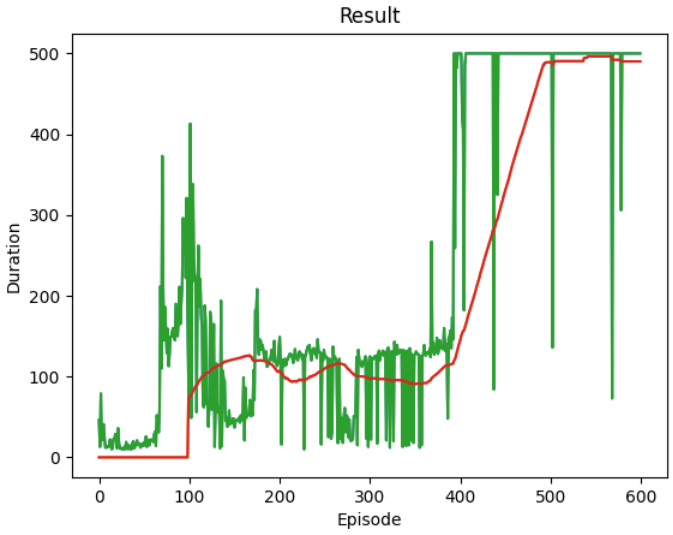


I then decreased the number of nodes in each layer from 128 to 64, resulting in this.

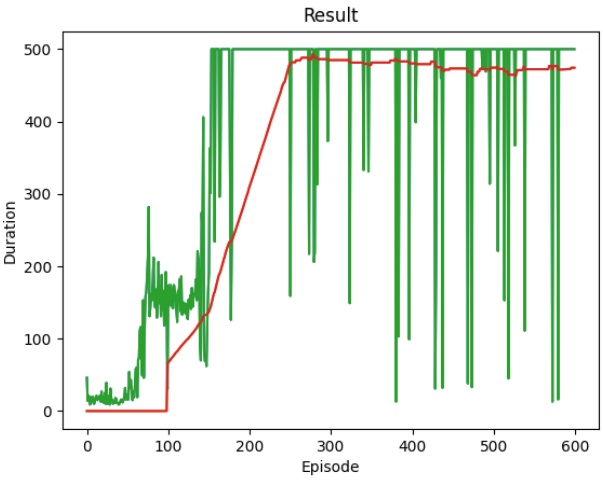


Next, I tested an architecture that had 64 nodes in the first hidden layer and 128 in the next layer. This tests one of the concepts we talked about in class, where you increase the number of nodes

as you go through the hidden layers. Here are the results from that trial.



Next, I decided to test the inverse, where you decrease the number of nodes in each hidden layer. The first hidden layer contains 128 nodes, and the second hidden has 64. That gave this result.



I decided to stick with the architecture of 128 nodes in the first hidden layer and 64 in the second. I did some testing with the hyperparameters and finalized on the following values. All architectures I tested used the following hyperparameter values and activation function:

```
BATCH_SIZE = 150
GAMMA = 0.99
EPS_START = 0.9
EPS_END = 0.05
EPS_DECAY = 1000
TAU = 0.005
LR = 0.0001
```

Activation function: ReLu

I decided on these hyperparameters, as they seemed to give the best results with the final architecture I chose. One challenge I faced was that the testing took a long time, as the model would usually take around 5 minutes to complete for each iteration. However, it's interesting to think about how theoretically, there is an optimal combination of hyperparameters. What I did with the hyperparameters seems similar to gradient descent. It would be cool to write a network that optimizes the hyperparameters given the architecture of a model, and just iteratively run it. I don't even want to begin to think about how computationally inefficient that is, but it seems like an interesting thought!

For the activation functions, I tested ReLu, sigmoid, and leaky ReLu. ReLu gave the best results, with it reliably reaching the max duration within the 600 episodes. Leaky ReLu and sigmoid both only reached a target network duration of approximately 150 seconds after the 600 episodes.

One insight I got from my experimentation with the architecture of the NN was that two hidden layers seemed to be the most optimal. In the future, when we run into more difficult problems and the inputs become more complicated, it will be interesting just how important the architecture of a DQN – basically a hyperparameter – will be. Given how computationally expensive training models can be, it seems that iteratively testing every single 'hyperparameter' will not be a viable option.