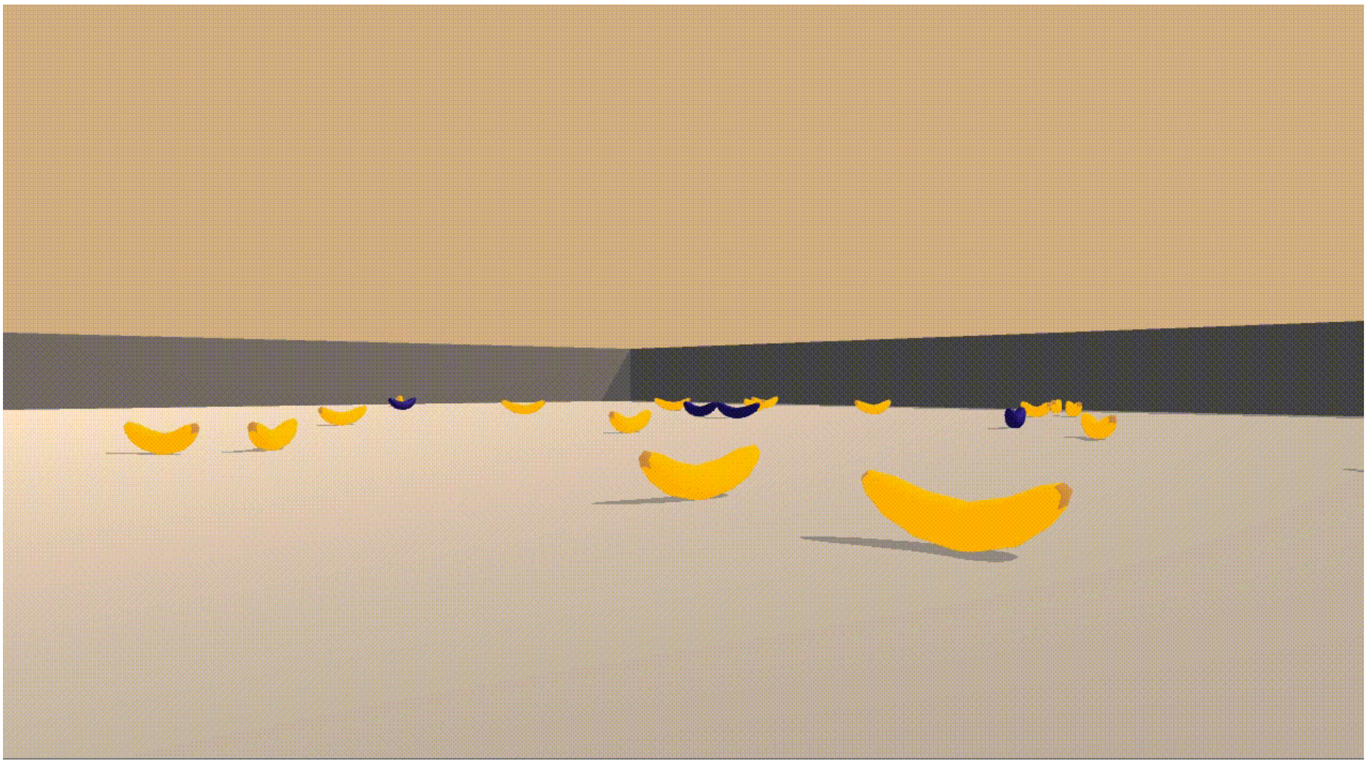
# Udacity Deep RL Project 1 Report: Navigation



GIF of the best performing agent

## Goal

The goal of this assignment was to train an agent in a simple environment using neural networks. The task is considered solved when a score of 13 or more is achieved over 100 consecutive episodes.

## Environment

The project environment is built in Unity and is a version of the Banana Collector, customised for the Udactity Deep Reinforcement Learning nanodegree. There were 37 different state-space dimensions with 4 different possible actions. Rewards of +1 were accumulated when a yellow banana is collected and -1 when a blue banana is collected. There are no rewards or penalties for moving. The task is considered solved when a score of 13 or more is achieved over 100 consecutive episodes.

## Approach

### Learning Algorithm

The simple Deep Q-Networks, Double Deep Q-Network and Dueling Deep Q-Networks were implemented, which all had a replay buffer.

Q-learning is a form of Temporal Difference (TD) learning, where the agent learns from each step instead of each episode. The state space is continuous and can be represented as a table. Thus a function approximator is used instead. This is where deep learning is used. The function approximator is represented by a neural network. Hence the name Deep Q-Networks (DQN). The Deep Q-Networks algorithm can be highly unstable and the most common methods used to stabilize them are Fixed Q-Targets and an

Experience Buffer. An experience buffer was added to our agent, the idea behind it is that we maintain a replay buffer of a certain size and after a few iterations we sample experiences from the buffer and use it to calculate the loss. this random sampling breaks the sequential nature of the experiences and helps to stabilize learning.

Deep Q-Networks are known to overestimate the value function. The idea behind Double Deep Q-Network (DDQN) is that one network is used to find the best action and the other network is used to evaluate the chosen action. Thus if one network chooses an action by mistake the other network would be able to determine it as a suboptimal action.

Duelling Deep Q-Networks (Duel DQN) use two output streams instead of one as is normal in DQN's. The one output stream is used to calculate the value function for the state while the other is used to calculate the advantage function for that action. The two streams are added together in an aggregated layer.

#### Model architecture

The architecture of the model was simple. It consisted of two hidden layers with the option to add a dropout layer. The input layer had the dimension 37x1 for the 37 different state spaces. Next, there were two fully connected layers with the number of nodes kept the same and adjustable. There was the option to add a dropout layer between the two fully connected layers to help prevent any overfitting that may occur, this had a dropout probability of 50%.

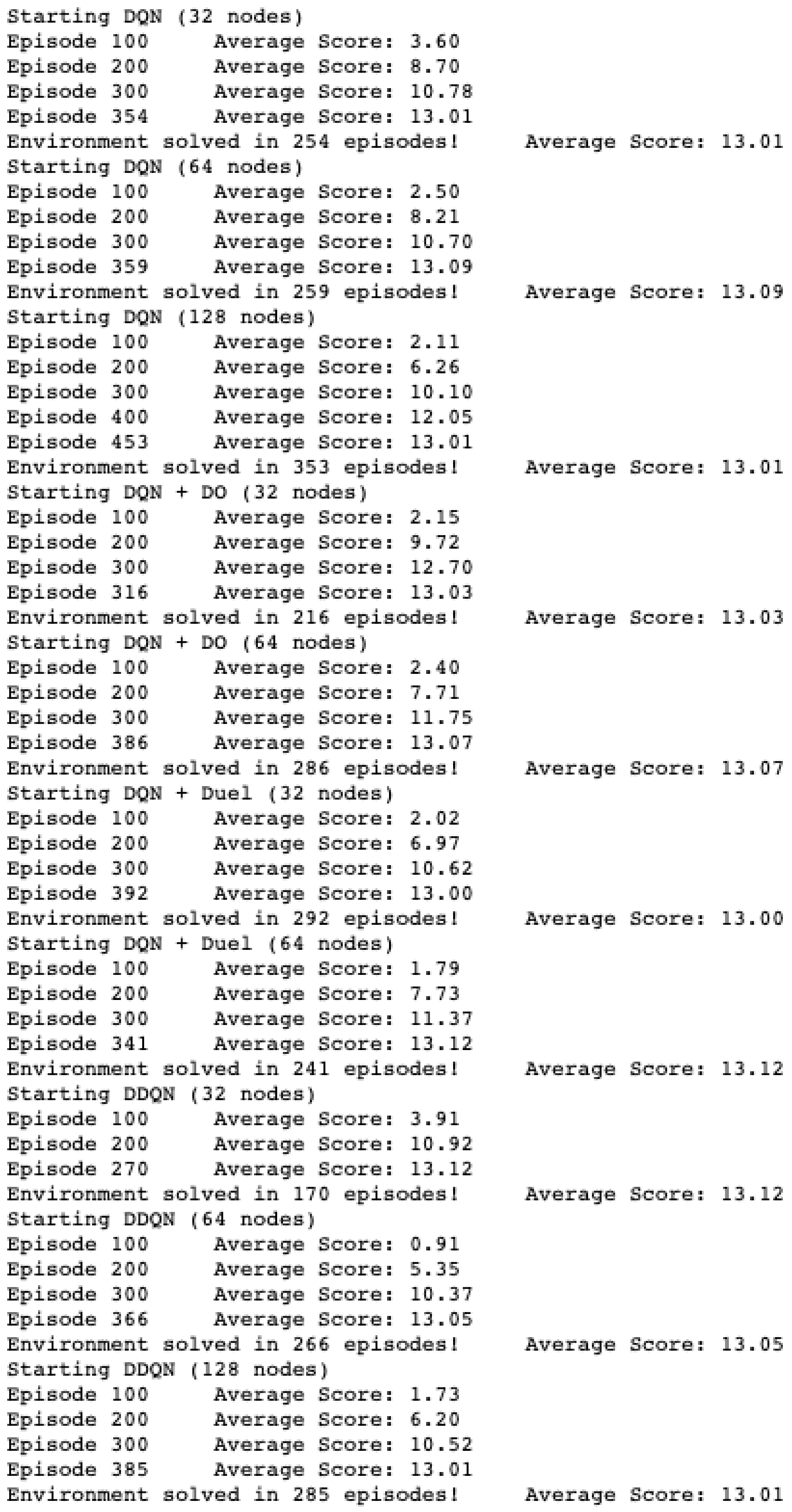
#### Hyperparameters

The following hyperparameters were used:

| Hyperparameter | Value |
|---|---|
| Epsilon start | 1.0 |
| Epsilon end | 0.1 |
| Epsilon decay | 0.995 |
| γ (Discount factor) | 0.99 |
| τ | 1e-3 |
| Learning rate | 5e-4 |
| Replay buffer size | 1e5 |
| Batch size | 64 |
| Update interval | 4 |
| Max number of episodes | 500 |

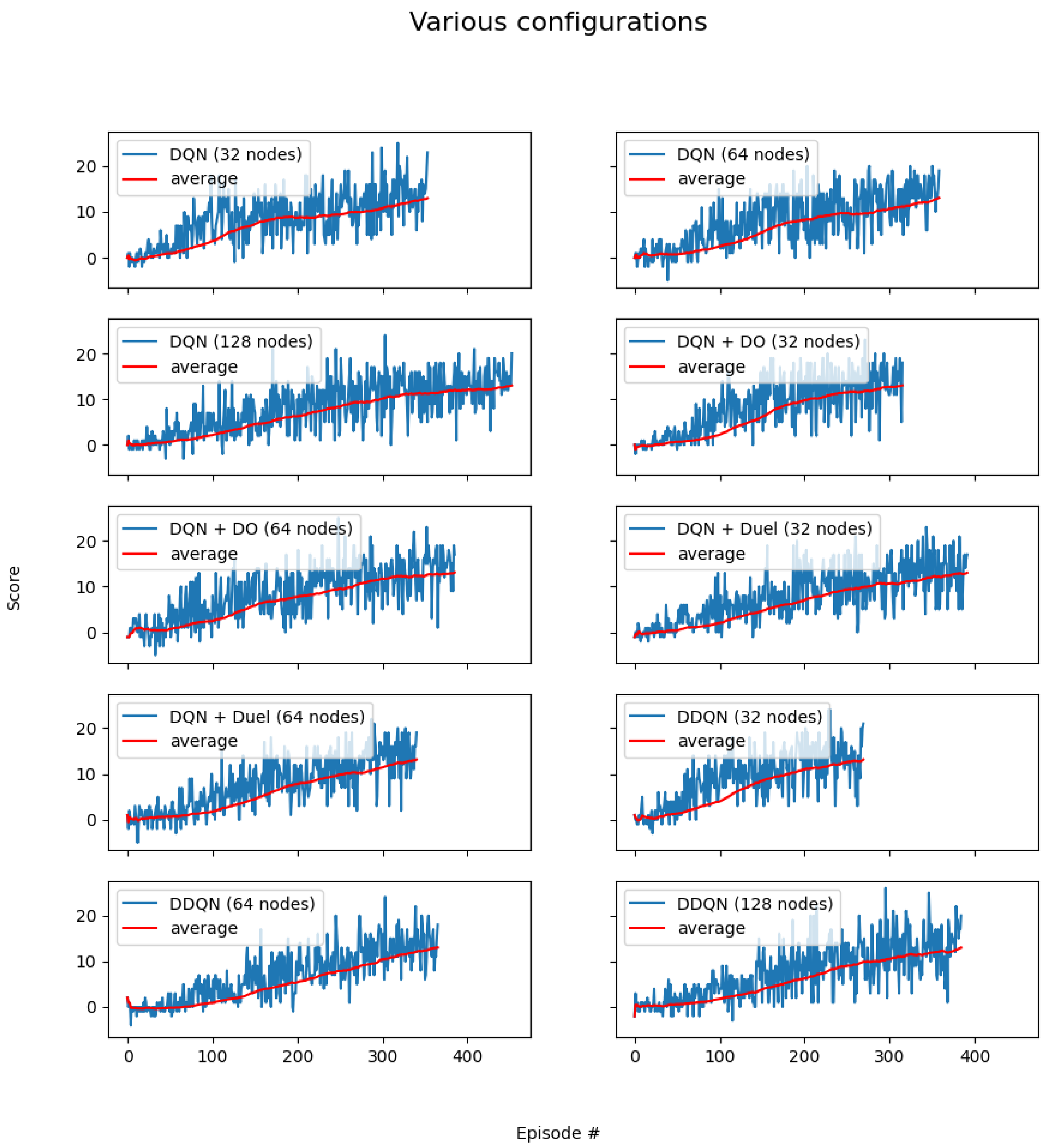
### Results

10 different configurations were run. The number of nodes, type of network and dropout layers was changed and added. The image below is the output from running all the configurations.



```
Starting DQN (32 nodes)
Episode 100     Average Score: 3.60
Episode 200     Average Score: 8.70
Episode 300     Average Score: 10.78
Episode 354     Average Score: 13.01
Environment solved in 254 episodes!     Average Score: 13.01
Starting DQN (64 nodes)
Episode 100     Average Score: 2.50
Episode 200     Average Score: 8.21
Episode 300     Average Score: 10.70
Episode 359     Average Score: 13.09
Environment solved in 259 episodes!     Average Score: 13.09
Starting DQN (128 nodes)
Episode 100     Average Score: 2.11
Episode 200     Average Score: 6.26
Episode 300     Average Score: 10.10
Episode 400     Average Score: 12.05
Episode 453     Average Score: 13.01
Environment solved in 353 episodes!     Average Score: 13.01
Starting DQN + DO (32 nodes)
Episode 100     Average Score: 2.15
Episode 200     Average Score: 9.72
Episode 300     Average Score: 12.70
Episode 316     Average Score: 13.03
Environment solved in 216 episodes!     Average Score: 13.03
Starting DQN + DO (64 nodes)
Episode 100     Average Score: 2.40
Episode 200     Average Score: 7.71
Episode 300     Average Score: 11.75
Episode 386     Average Score: 13.07
Environment solved in 286 episodes!     Average Score: 13.07
Starting DQN + Duel (32 nodes)
Episode 100     Average Score: 2.02
Episode 200     Average Score: 6.97
Episode 300     Average Score: 10.62
Episode 392     Average Score: 13.00
Environment solved in 292 episodes!     Average Score: 13.00
Starting DQN + Duel (64 nodes)
Episode 100     Average Score: 1.79
Episode 200     Average Score: 7.73
Episode 300     Average Score: 11.37
Episode 341     Average Score: 13.12
Environment solved in 241 episodes!     Average Score: 13.12
Starting DDQN (32 nodes)
Episode 100     Average Score: 3.91
Episode 200     Average Score: 10.92
Episode 270     Average Score: 13.12
Environment solved in 170 episodes!     Average Score: 13.12
Starting DDQN (64 nodes)
Episode 100     Average Score: 0.91
Episode 200     Average Score: 5.35
Episode 300     Average Score: 10.37
Episode 366     Average Score: 13.05
Environment solved in 266 episodes!     Average Score: 13.05
Starting DDQN (128 nodes)
Episode 100     Average Score: 1.73
Episode 200     Average Score: 6.20
Episode 300     Average Score: 10.52
Episode 385     Average Score: 13.01
Environment solved in 285 episodes!     Average Score: 13.01
```

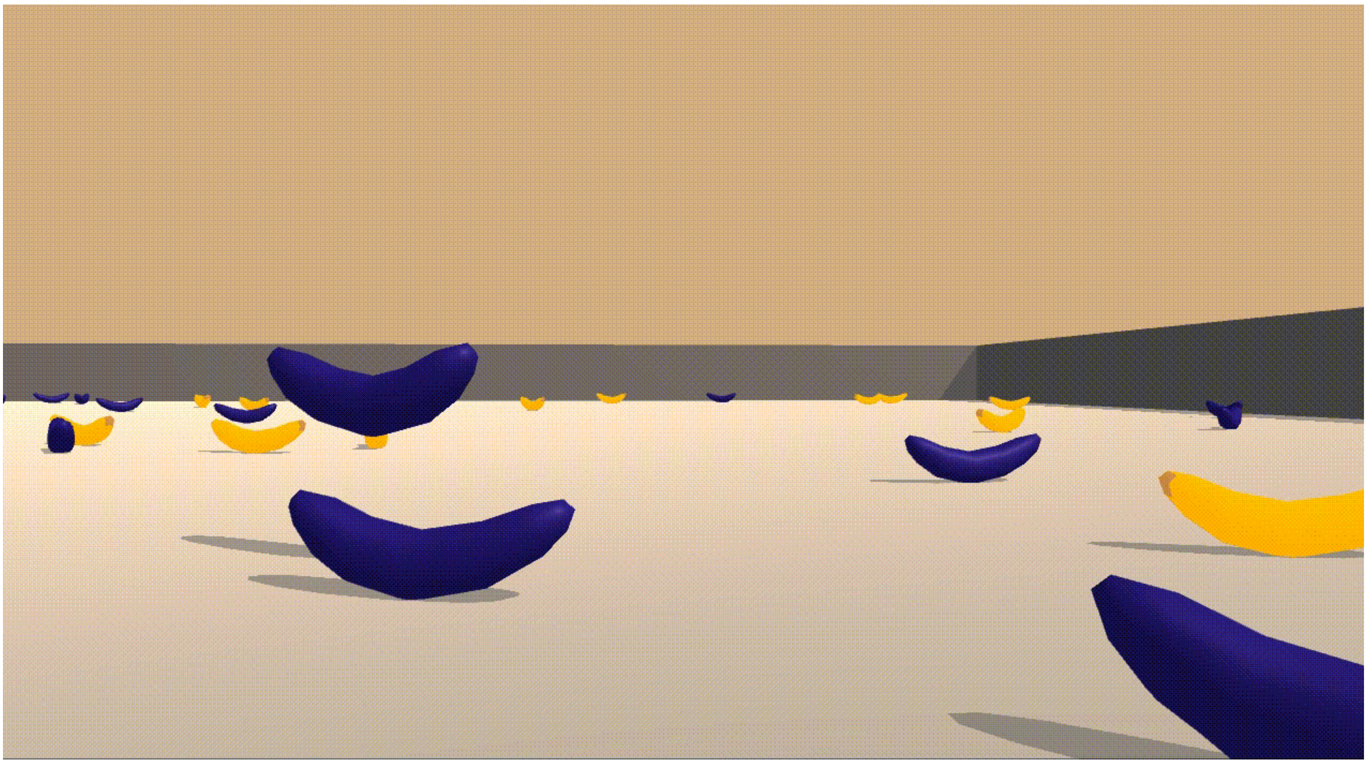
The next image is the plots for each of the configurations.



## Conclusion

Although the DDQN solved the environment the fastest it came with extra complexity. The simple DQN with dropout and 32 nodes also solved the environment quickly.

The agent didn't perform perfectly, when watching the chosen agent play the game it often got stuck in a loop of choosing between two actions as can be seen in the GIF below.



GIF of the best performing agent

## Future Improvements

- Find a solution for the agent getting stuck between two actions
- Test the effect of the replay buffer
- Using Prioritized Replay showed
- Hyper-parameter search would lead to better performances