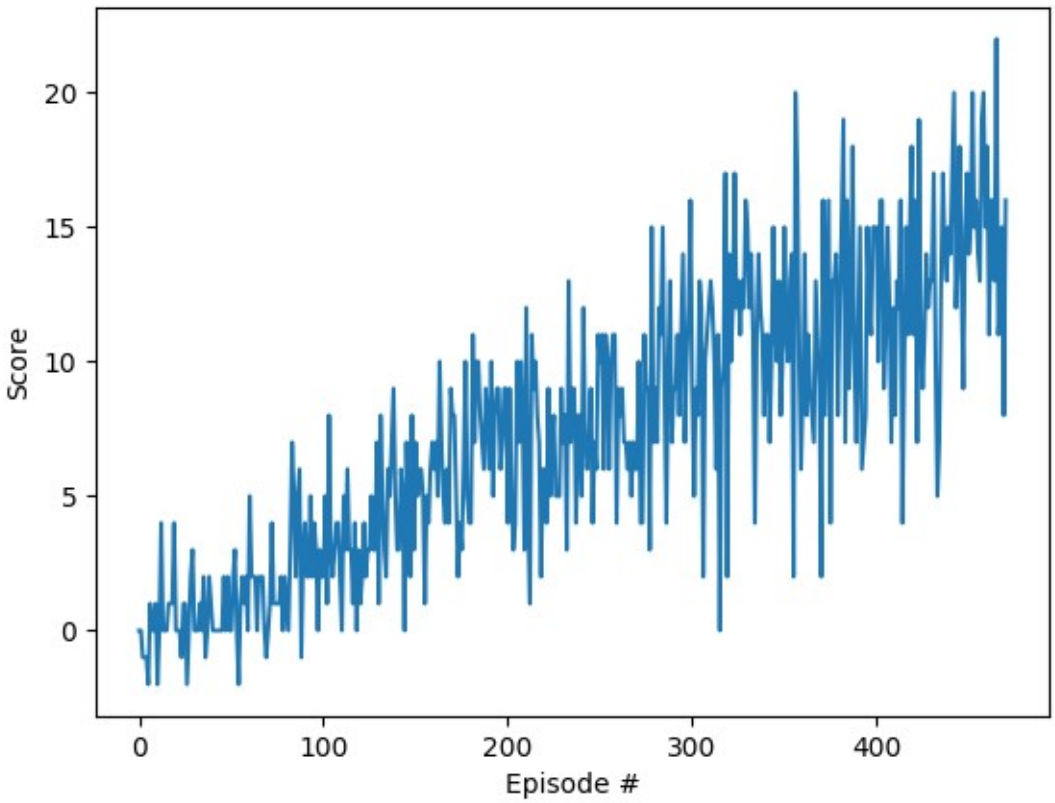
Udacity Deep Learning Project 1 – Navigation

I present the approach I used to solve the Banana Environment.

DQN Algorithm – The Deep Q Network algorithm is a model free RL algorithm for learning Q values of continuous state space Markov Decision Processes. The Q(s,a) function is approximated by a neural network with non linearities. The Q function is learned via Stochastic Gradient Descent over a moving target, the target given by an older copy of the Q network.

I started off with the vanilla DQN algorithm implemented by Udacity, which worked quite well out of the box.



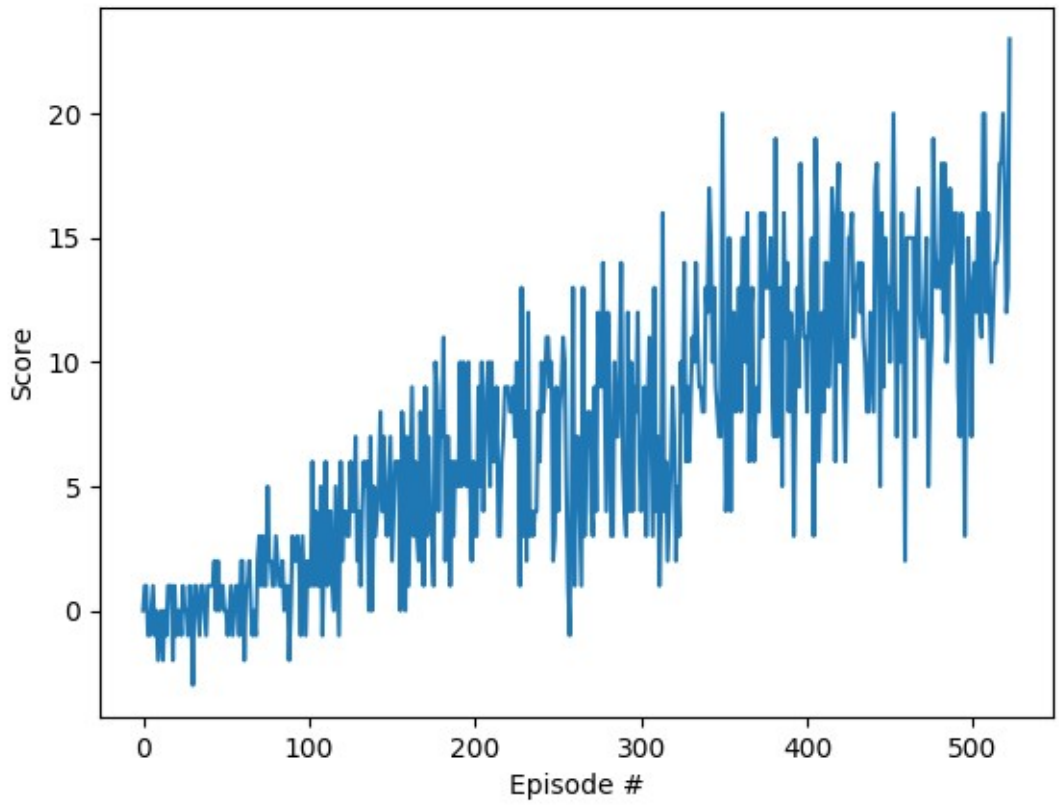
The figure shown above illustrates the performance of the vanilla DQN algorithm.

The Qnetwork has 3 Linear layers with ReLU non linearity
QNetwork(
  (fc1): Linear(in_features=37, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=4, bias=True)
)
Average Score after every 100 episodes
Episode 100 Average Score: 1.156
Episode 200 Average Score: 5.15
Episode 300 Average Score: 7.89
Episode 400 Average Score: 10.92
Episode 471 Average Score: 13.04
Hyperparameters
BUFFER_SIZE = int(1e5) # replay buffer size
BATCH_SIZE = 64 # minibatch size
GAMMA = 0.99 # discount factor
TAU = 1e-3 # for soft update of target parameters
LR = 5e-4 # learning rate
UPDATE_EVERY = 4 # how often to update the network

The checkpoint and resulting figure are saved in dqn64/

Experiments with different Architectures with same Hyperparameters

Increasing depth, reducing width, adding residual connections – Deeper networks with reduced width of hidden layers and residual connections are empirically known to have better convergence properties.



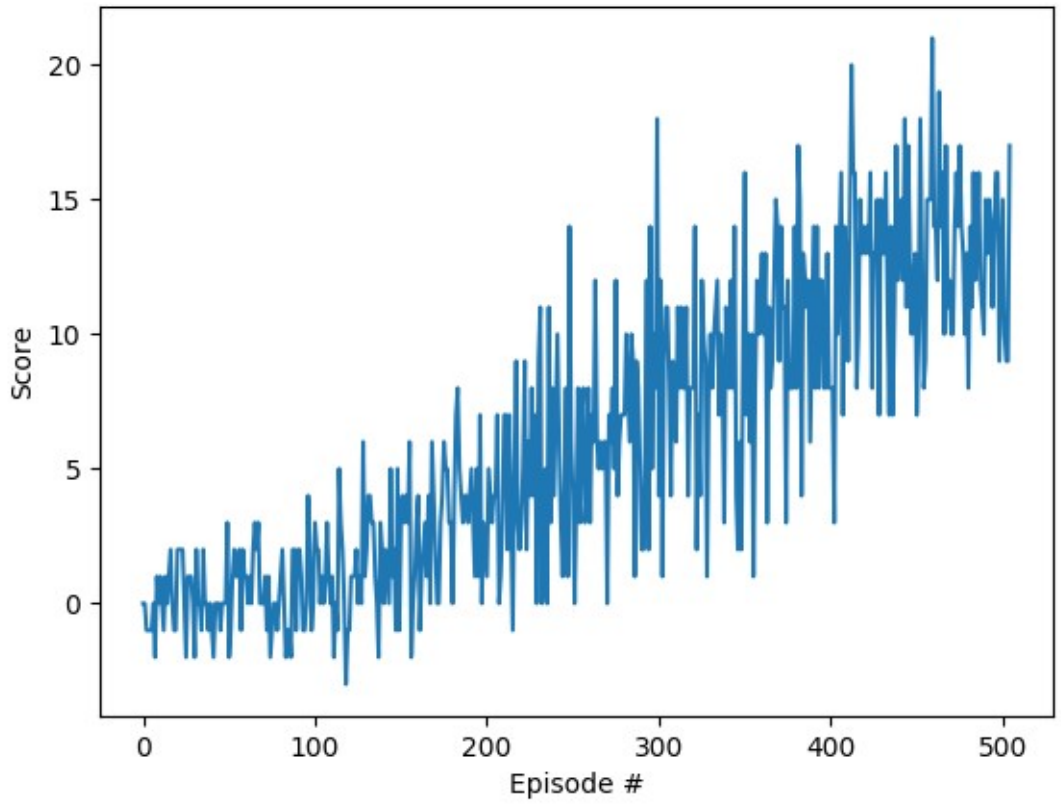
The figure shown above illustrates the performance of the vanilla DQN algorithm.

The Qnetwork has 4 Linear layers with ReLU non linearity
QNetworkResidual(
  (fc1): Linear(in_features=37, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=32, bias=True)
  (fc4): Linear(in_features=32, out_features=4, bias=True)
)

Average Score after every 100 episodes
Episode 100 Average Score: 0.555
Episode 200 Average Score: 4.42
Episode 300 Average Score: 7.01
Episode 400 Average Score: 9.64
Episode 500 Average Score: 12.30
Episode 523 Average Score: 13.00
Environment solved in 423 episodes! Average Score: 13.00

The checkpoint and resulting figure are saved in dqn32/

BatchNorm Layer – For supervised learning problems, Batchnorm helps immensely in faster convergence.



The figure shown above illustrates the performance of the vanilla DQN algorithm.

The Qnetwork has 3 Linear layers with ReLU non linearity before Batchnorm in fc1 and fc2.

```
QNetworkBN(
  (bn): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc1): Linear(in_features=37, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=4, bias=True)
)
```

Average Score after every 100 episodes
Episode 100 Average Score: 0.285
Episode 200 Average Score: 2.16
Episode 300 Average Score: 5.61
Episode 400 Average Score: 9.11
Episode 500 Average Score: 12.83
Episode 505 Average Score: 13.00

The checkpoint and resulting figure are saved in dqnbn/

Overall, the default neural network performs the best with the environment solved in 371 episodes, although the difference with other architectures isn't that great. It could be safely concluded that empirically, simpler neural networks tend to work well in RL problems.

Experiments TODO

1. Role of Priority Experience Replay Buffer
2. Role of Double DQN
3. Role of Dueling DQN
4. Combining multiple improvements mentioned above.
5. Role of Rainbow DQN
6. Role of different Optimizers - eg. RMSProp, SGD
7. Explore 2nd order optimizers - BFGS-L