DeepQ Banana Environment

Assumptions

This assumes the reader has basic understanding of deep networks and reinforcement learning

Learner

Based on Udacity Deep Q course, Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236, and Van Hasselt, Hado & Guez, Arthur & Silver, David. (2015). Deep Reinforcement Learning with Double Q-learning. Implemented using:

- Experience Replay
- Fixed Q-Targets
- Double Deep Q-Learning

As Deep-Q networks tend to overestimate the action values we are using Double Deep Q-Learning where the target network is fixed while the local network trains. Then the target network is reset with a weighted factor (Tau) using local networks values at a set update rate, in this case every 4 episodes. The new target weights are a factor of the old target and the new local weights such that Q_t = Tau*Q_{Lt} + (1-Tau) * Q_{Tt}. Double DQN does not fully decouple the local and the target networks but provides good reduction in the over overestimate of reward by splitting the selection and action evaluation steps. The update and selection functions are similar to DQN TD target is using the Q values from the frozen network instead of the updating one:

$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q(S_{t+1}, \text{argmaxa } Q(S_{t+1}, a; \theta_t), \theta_t^-)$

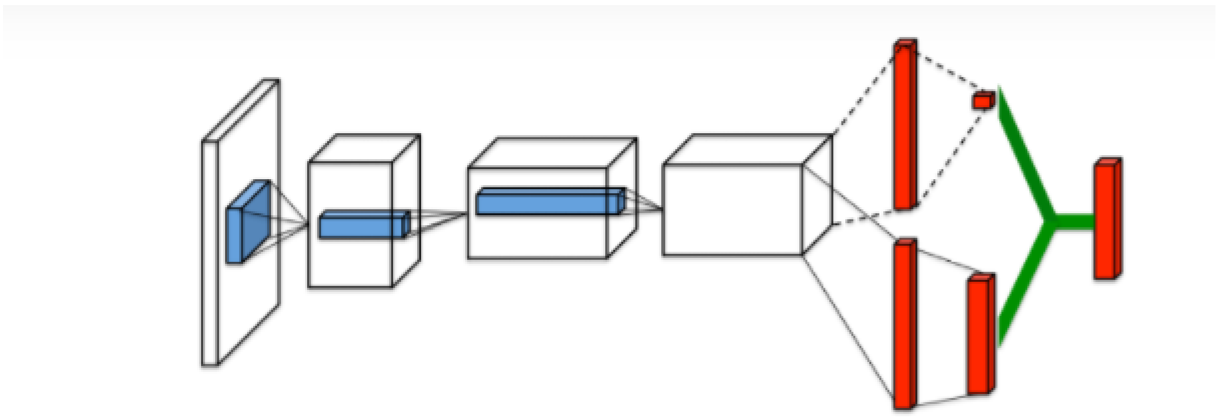


Illustration from Udacity's Deep Reinforcement Learning Nanodegree

The neural network being used is a linear feed forward network with 3 hidden layers consisting of 30 x 15 x 10 nodes. This model was chosen to reduce the number of nodes in the network for faster training and easier iterations. It trains and preforms better than the example network of 64 x 64 nodes.

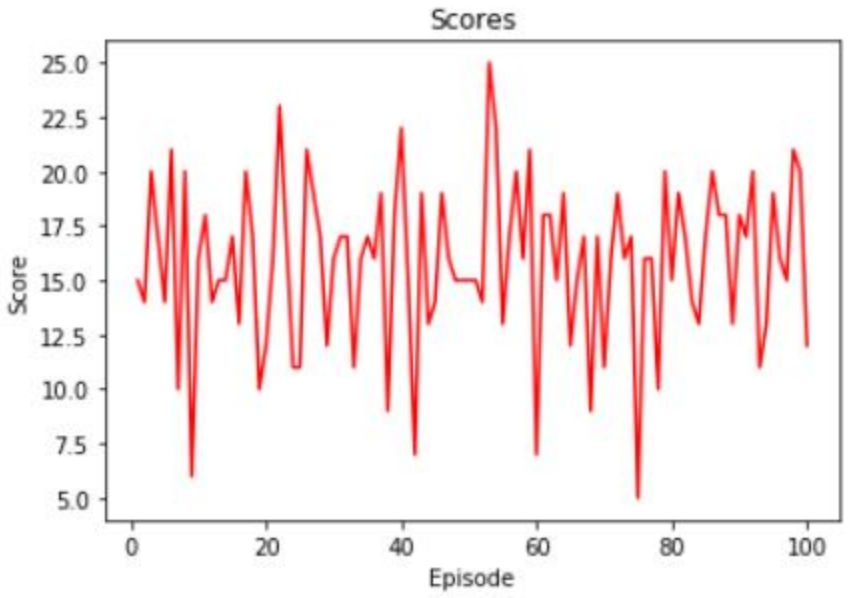
Hyperparameters

Discount Factor	.99
Buffer Size	10,000
Sample Size	64
Learning Rate	.00065
Tau	.0035
Update target rate	4
Epsilon start	1
Epsilon end	.01
Epsilon decay	.9948
Activation Function	ReLU

These hyperparameters were obtained by trial and error adjusting one value at a time, trying to minimize training time to an average of 13.5 score. Which is above the solve point by .5 points. The neural network was also selected this way and the reduction of nodes has a significant effect on training times even with similar number of episodes.

Results

On average these parameters take 530ish episodes to achieve 13-point average over episodes and 600 episodes to achieve 13.5 average. Running the fully trained model over 100 episodes gives a typical score of 15.8.



```
Average score: 15.79
```

Future work

- Implement dueling DQN which should reduce the overestimate even further but will slow down training. One idea is to have both train simultaneously using a cached copy of the other network, where A is training form an old copy of B and vice versa. Each only updating from their own datasets.
- Implement Prioritized Experience Replay instead of random.

- Optimize for GPU learning, currently all calculations must be sent back to the CPU for processing. Moving the networks and all calculations to the GPU would eliminate the transfer time and allow for faster training.