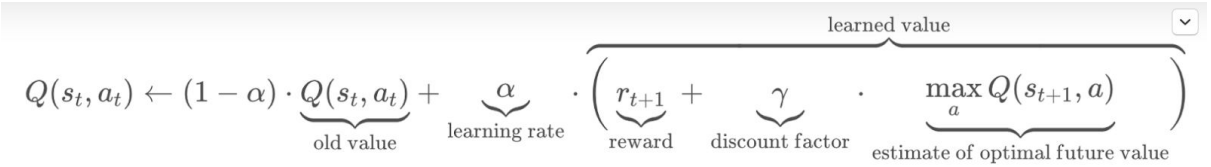
# Solving a rewarding problem using Deep Q-Network (DQN)

# Learning Algorithm

The problem we are going to solve is to collect as many yellow bananas as possible. It's a episodic problem about maximization of expected cumulative reward. It could be modelled as a Q-Learning algorithm:




Just like "Monte Carlo" process, the agent will learn through the Q-Learning algorithm by exploring or exploiting the environment many times.

Exploration is about learning new knowledge while Exploitation is exploiting existing limited knowledge to decide the best actio to take for the current environment.

To balance between Exploration and Exploitation, a Epsilon-Greedy Policy is introduced. It is a decaying probability percentage which the learning algorithm uses to choose between picking random action (Exploration) or greedy action based on existing knowledge (Exploitation). In the implementation, a decay percentage 0.995 is picked with start of 1.0 and end of 0.01.

The biggest difference between traditional Q-Learning and Deep Q-Network (DQN) is Q action-value function is replaced by the best "Function Approximator" which is "Deep Neural Network".

In this project, it is implemented in Python PyTorch with simple 2 layers network with 64 hidden units in each layer and input size of 37 dimensions and output actions size of 4. It uses the Adam optimizer as it is shown to perform well in the typical case. The agent can be trained in GPU or CPU. It indeeds can train fast enough using CPU in Macbook Pro to achieve good result. The agent will update the Neural Network in every 4 action steps during training.

While Neural Network is a good "Function Approximator", it also introduces instability to the reinforcement learning. To mitigate the issue, "Experienced Replay" is implemented. Intuitively, it can be thought of human dreaming, a process in the brain that can replay the recently experienced trajectories. Instead of following the playback sequence of actions which could be highly correlated, we stores the actions in a "Replay Buffer" of size 100,000 so that the agent can randomly pick the sequence of action values in the form of tuples

(State, Action, Reward, Next State). This sampling process is called “Experience Replay” which the agent will pick a batch size of 64 in the implementation. By doing this, the agent can learn in a better way by learning the same and rare experience multiple times.

# Training Result

The agent takes 595 episodes to achieves an average rewards of 13 scores in about 10 mins trained in CPU only machine. If GPU were used, it would have been quicker.

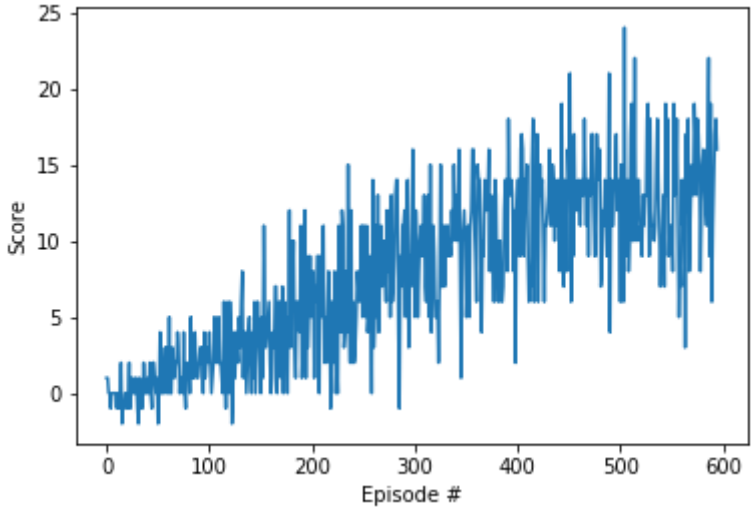



*Figure 1 - No. of episodes / Scores Plot*

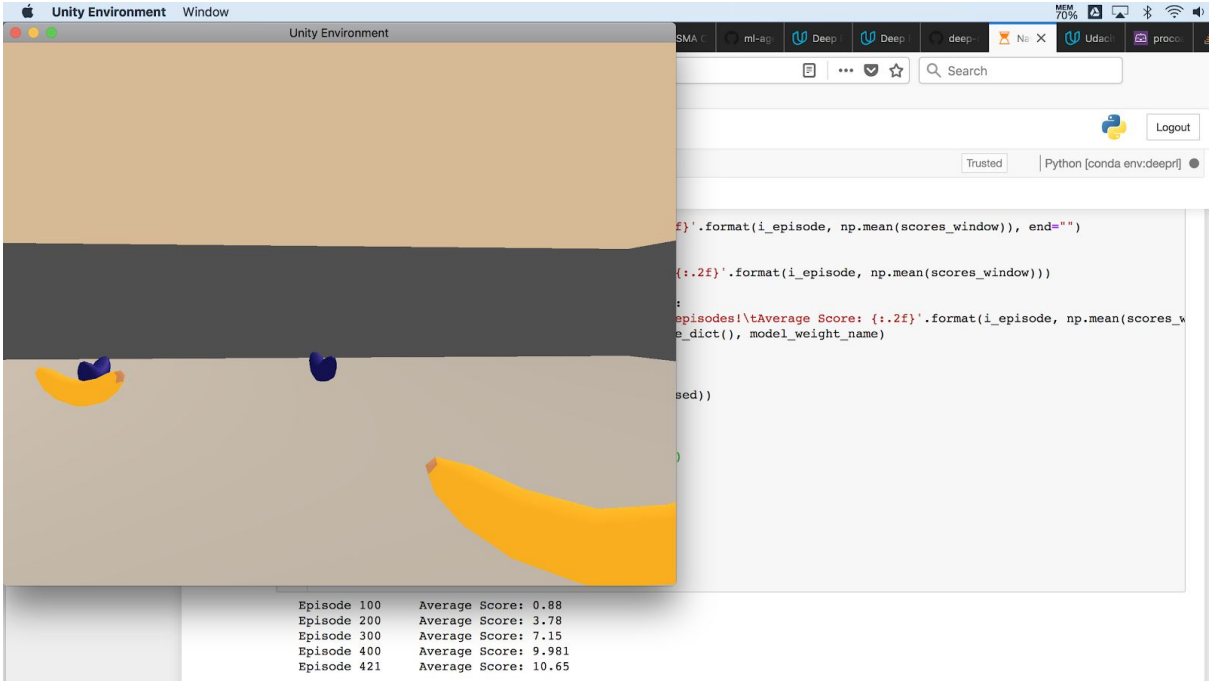



*Figure 2 - Screenshot showing Training In Progress*

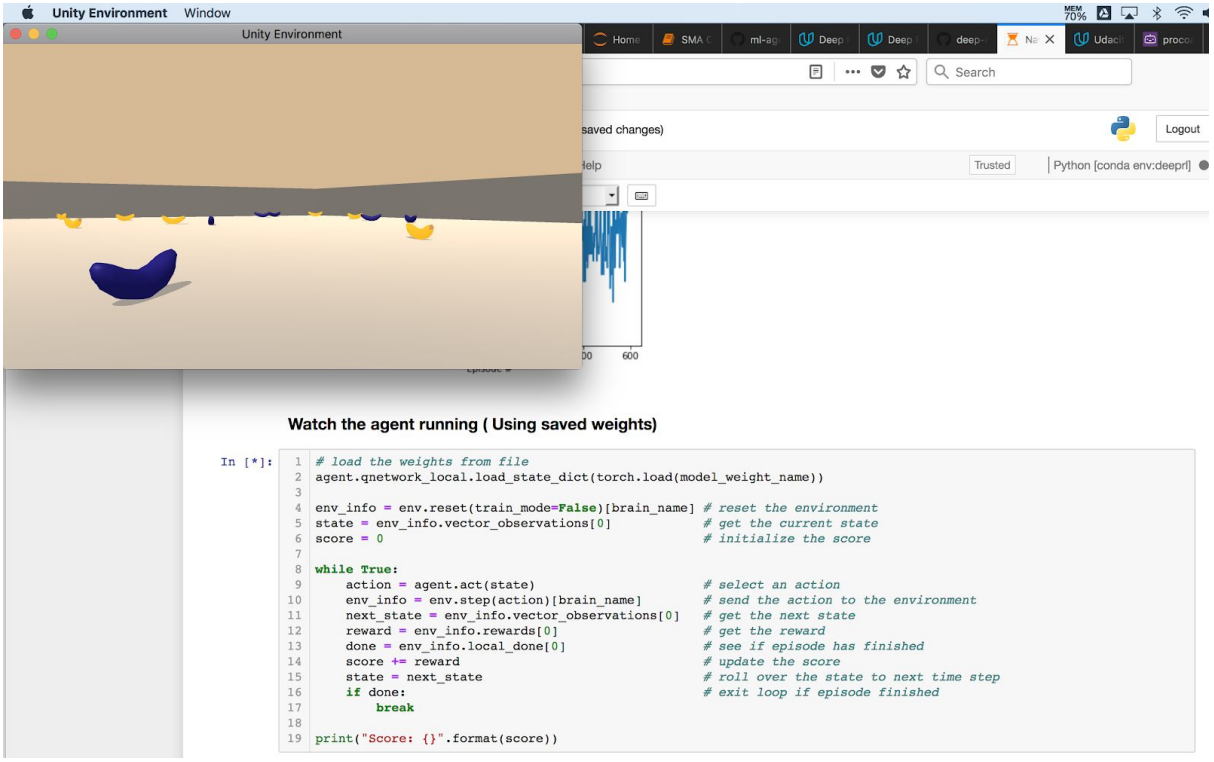



*Figure 3 - Running the agent with saved weights*

# Future Improvements

While the current agent can achieve pretty good result, there is number of areas we can improve. The Neural network we use has only 2 layers with 64 hidden units in each layer. We could try more sophisticated network model with more layers and hidden units. However, it would also mean it will need better machine and GPU instance. In terms of hyperparameters, the current batch size is 64. We could definitely try out a bigger batch size with more frequent update of the network.

Instead of tweaking hyperparameters, the other way is to explore improving DQN model. Standard DQN suffers from overestimation of Q-values in early stage while it's still evolving. **Double Q-Learning** mitigates the issue by having separate set of W" to evaluate the action, which makes it different from the set of original W used to select the best action. To improve it even further, we could adopt a completely separate Neural Network as proposed in **Dueling DQN** for assessing the value of the state without having to evaluate the value of each action.

Standard DQN uses Experienced Replay to help stablising the reinforcement learning. However, as the learning proceeds, the old experience would get lost. Just like human brain, some life experience is more important for learning and would better be replayed more

frequently. **Prioritized Experience Replay** achieves this by adding sampling probability to give priority to more important experience during replay.

While each individual model can improve the standard DQN model in certain way, it's interesting to see if combining them all would take it to a new level or worse or better be done individually. A research was done as mentioned in "*Rainbow: Combining Improvements in Deep Reinforcement Learning"* https://arxiv.org/abs/1710.02298 . It combines all major improvements to the DQN model.

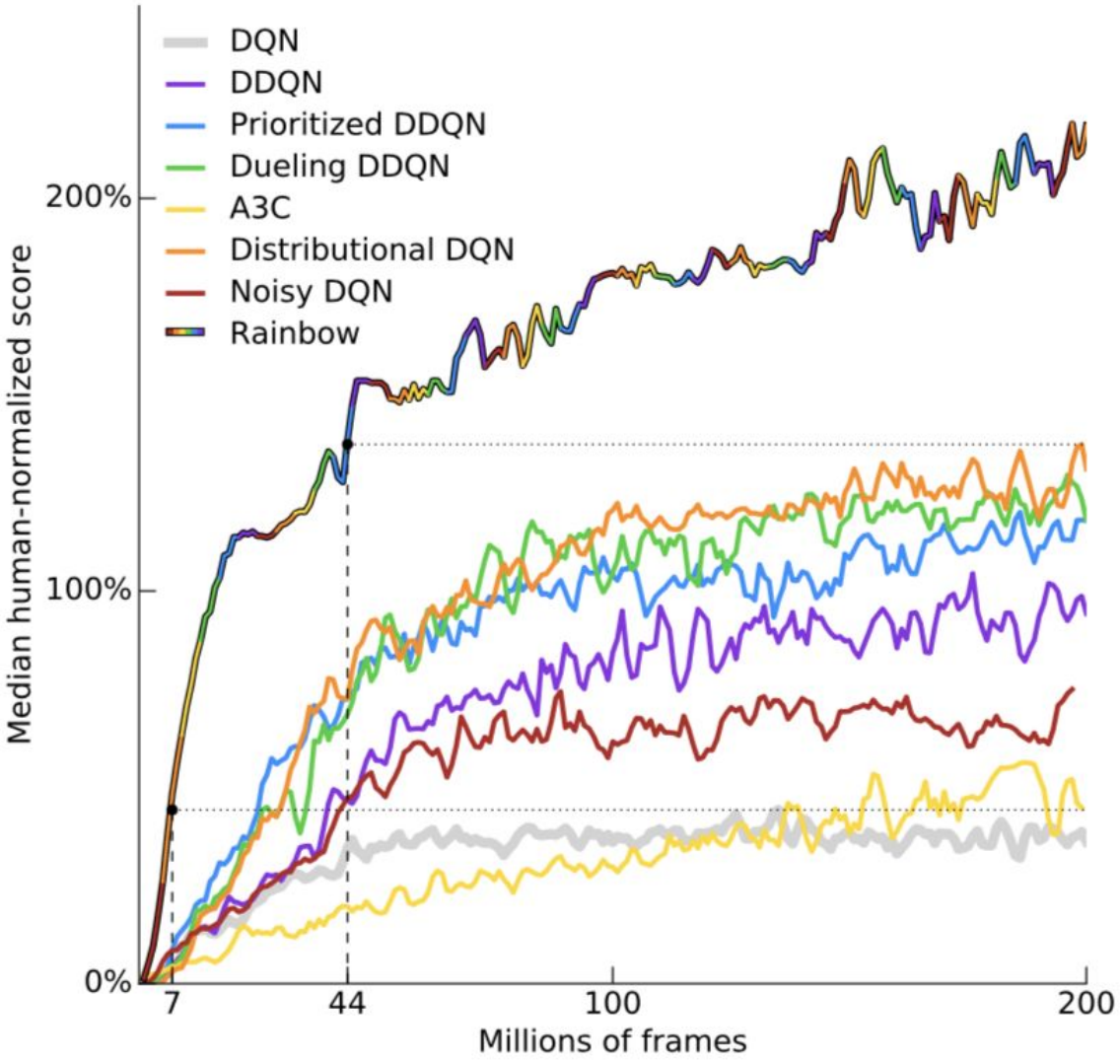



Figure 4 - Plot of scores from "*Rainbow: Combining All Improvements in Deep Reinforcement Learning"*

The result is pretty remarkable. "Rainbow" can achieve a good score much faster than the rest. If I were to pick a couple to implement, I would recommend **Dueling Double Q-Learning.**