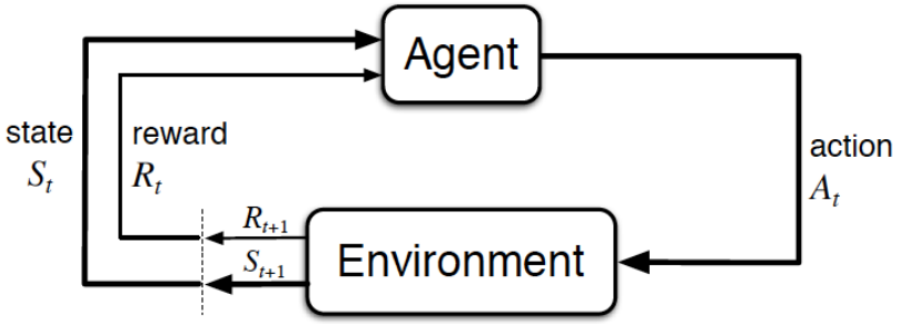
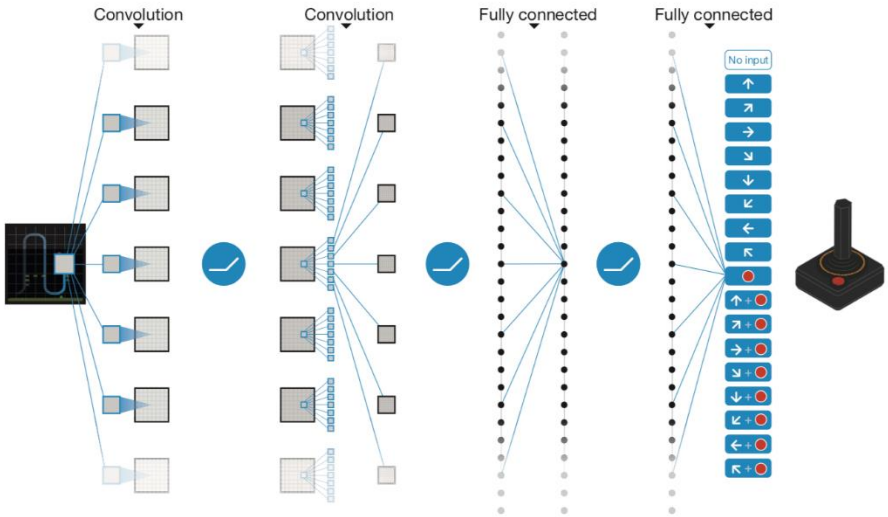
Learning Algorithm

The learning algorithm used a deep reinforcement learning neural network with ReLu activation convolution layers. Dqn_model.py contains the neural network for observing the state space. The state space of this project contains 37 states. Dqn_agent.py contains the hyperparameters and controls the agents actions. This project contains 4 agent actions. Below are the hyperparameters chosen to train the DQN agent.

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 256        # minibatch size
GAMMA = 0.999            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 16        # how often to update the network
```

Navigation.ipynb contains the function 'dqn', which codes the for-loop process shown below. The agent chooses an action, the environment is observed, a reward is given, and algorithm seeks to optimize its score. The environment state space parameters are feed into the dqn_model.py as depicted in the second picture below to get the action that would most likely result in a higher score.

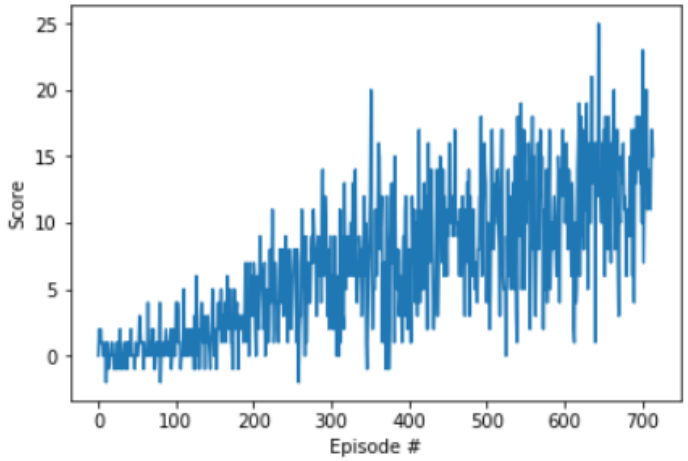




Plot of Rewards

The rewards are recorded every 100 episodes along with the average score. This training session used a DQN only approach to the task. Once the DQN agent achieves a score of 13, the program ends and plots the scores of each episode. The agent achieved a score of 13 within 715 episodes.

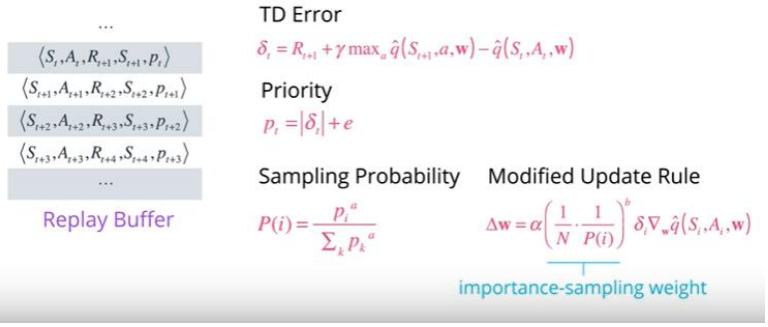
```
Episode 100     Average Score: 0.45
Episode 200     Average Score: 2.09
Episode 300     Average Score: 5.53
Episode 400     Average Score: 6.78
Episode 500     Average Score: 9.04
Episode 600     Average Score: 9.88
Episode 700     Average Score: 12.31
Episode 715     Average Score: 13.00
Environment solved in 715 episodes!     Average Score: 13.00
```



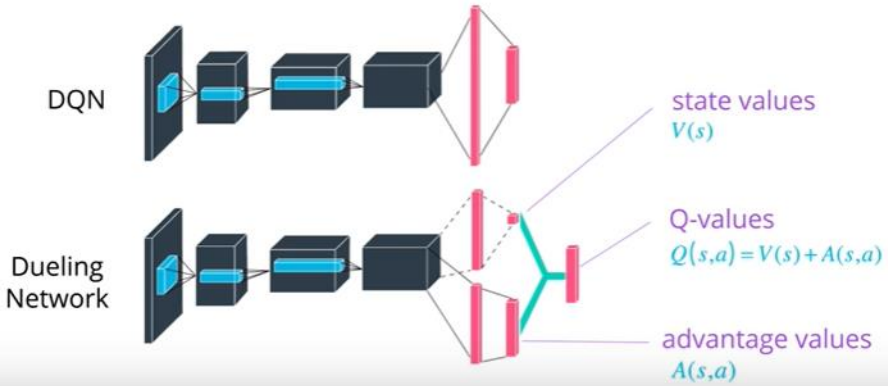
Ideas for Future Work

Although 'Prioritized Experience Replay' and 'Dueling DQN' networks were implemented in this project. The agent learned slower than a DQN only approach. This could be due to the chosen hyperparameters being different for different learning algorithm, because 'Prioritized Experience Replay' in combination with 'Dueling Networks' should have outperformed the DQN only algorithm as shown in the 'Rainbow' algorithm comparison chart below.

Prioritized Experience Replay



Dueling Networks



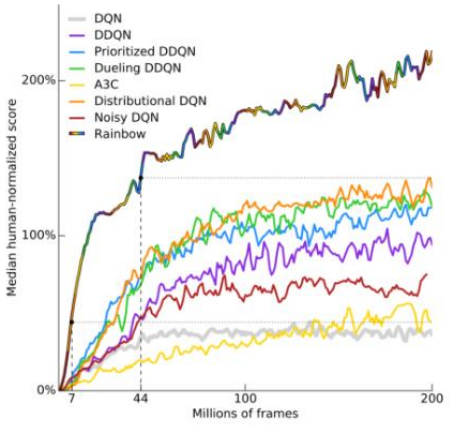


Figure 1: **Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final performance. Curves are smoothed with a moving average over 5 points.

Performance on Atari games: comparison of Rainbow to six baselines.