TDL PROJECT REPORT

Flappy Bird using Deep Q learning

Shailesh Sridhar 01FB16ECS349

Srikumar Subramanian 01FB16ECS396

Taruni Sunder 01FB16ECS418

INTRODUCTION

The objective of this project was to train an agent to play the game of Flappy Bird. This is achieved by coupling a neural network with Q learning. In Q-learning we define a function *Q(s, a)* representing the maximum discounted future reward when we perform action a in state s, and continue optimally from that point on.

The way to think about *Q(s, a)* is that it is "the best possible score at the end of the game after performing action a in state s." It is called Q-function, because it represents the "quality" of a certain action in a given state.

Q learning works on the principle of exploration and exploitation. Initially, there is lots of exploration, (based on a parameter epsilon) and as training ensues exploitation is employed. Exploitation is a greedy approach wherein the action with the maximum rewards is chosen.

It follows the Bellman Equation which is as follows:

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

Where gamma is the discount factor, a way of giving higher rewards to states closer to the goal.

ABOUT THE GAME

Flappy bird is a game wherein a bird is to clear as many obstacles as it can. The obstacles manifest themselves in the form of pipes and the bird must adjust its movement so as to avoid crashing into them. Upon contact with the obstacles, the game terminates. The bird can move straight, or go up. In order to model the game to a reinforcement learning problem, the actions are defined as “do nothing”, or “flap”. The former action is indicative of remaining in the same state, that is, the bird will move downwards eventually, and the latter will cause the bird to jump.

At each timestep, the rewards for both actions are calculated. Initially, based on parameter epsilon, there is lots of exploration (random actions chosen) and slowly as epsilon is decreased, the greedy exploitation method is employed where actions with higher rewards are chosen.

ENVIRONMENT

The environment variables used are as follows. These variables are used to define the game at any given time and serve as the states of the games. Consequently, these states are what is fed to the network.

1. basex
2. playerx
3. playery,
4. pipeVelX,
5. playerAccY,
6. playerFlapped,

7. playerMaxVelY,
8. playerVelY,
9. lowerPipes[0]['x'],
10. lowerPipes[0]['y'],
11. lowerPipes[1]['x'],
12. lowerPipes[1]['y'],
13. upperPipes[0]['x'],
14. upperPipes[0]['y'],
15. upperPipes[1]['x'],
16. upperPipes[1]['y']

These parameters define the game in terms of geometry, in other words, it describes the physics of the game. I.e the coordinates of the player(the bird), its velocity, the position of the pipes, etc.

These 16 are passed as a vector to the neural network.

NETWORK ARCHITECTURE

The network architecture is as follows. There is an input layer which takes a vector of length 16 (corresponding to the game parameters defined in the previous section). This is connected to a hidden layer that has 512 neurons which in turn is connected to another similar hidden layer having another 512 neurons. The activation of both these layers is the ReLu function.

Finally, there is an output layer that has two output neurons that correspond to the rewards for the actions.

The output of the network are the rewards obtained for the two actions (Flap or do nothing) in the given state.

APPROACH

Two approaches have been adopted. One in which 16 states are fed to the network for each frame, and another in which states for four frames at a time serve as an input to the network.

The loss function is the RMS difference of the predicted rewards (by the network) and the actual reward calculated as per the Bellman equation. The minimization of this cost is the process of training.

TRAINING

The model was trained for 7 million timeframes. The model's weights were randomly initialized using a normal distribution with a standard deviation of 0.01, then the replay memory was set with a max size of 500,00 experiences.

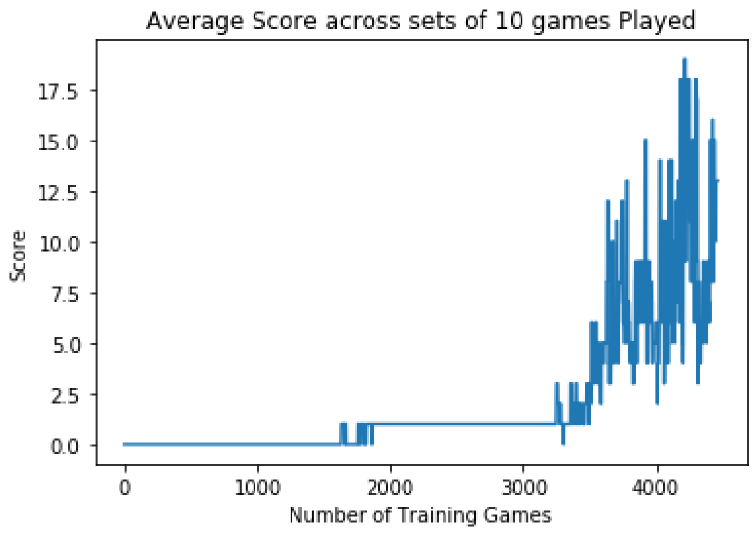
Actions are chosen uniformly at random for the first 10,000 time steps, without updating the network weights. This allows the system to populate the replay memory before training begins.

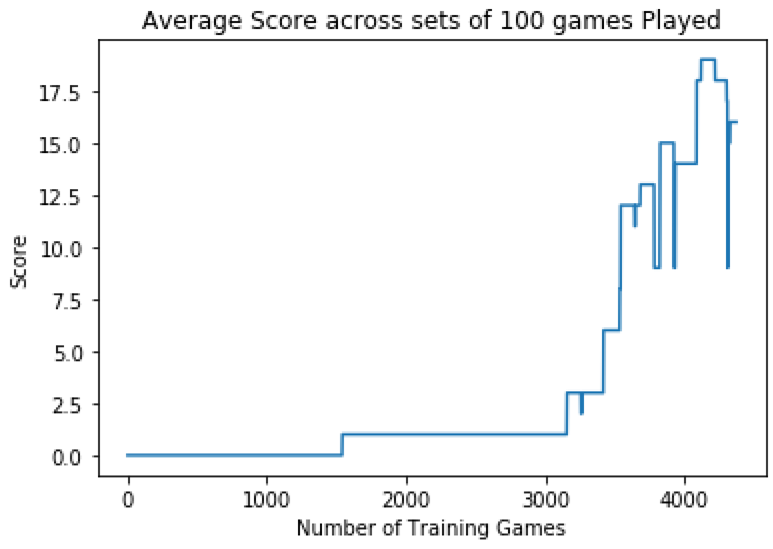
The value of epsilon is set to 0.1 initially and is made to linearly anneal to final epsilon of 0.0001.

At each timestep during training the network samples mini batches of size 32 from the replay memory to train on, and performs a gradient step on the loss function described above using the Adam optimization algorithm with a learning rate of 0.000001. After annealing finishes, the network continues to train indefinitely, with epsilon fixed at 0.001.

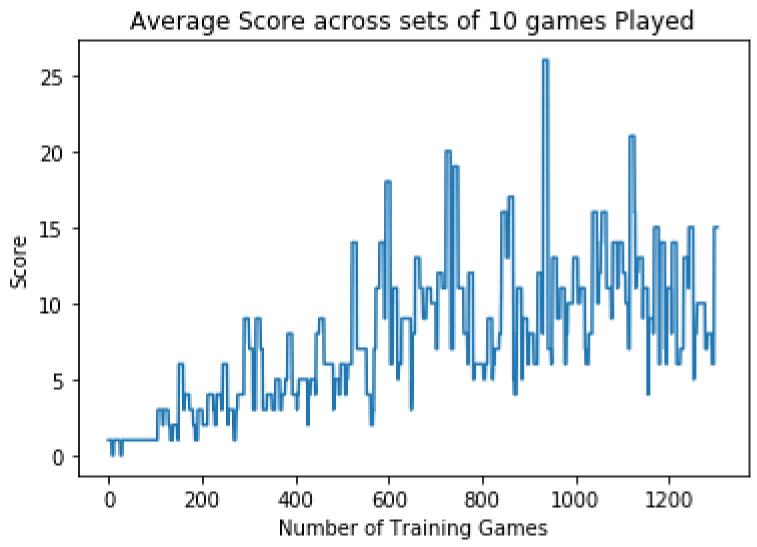
COMPARISON AND RESULTS

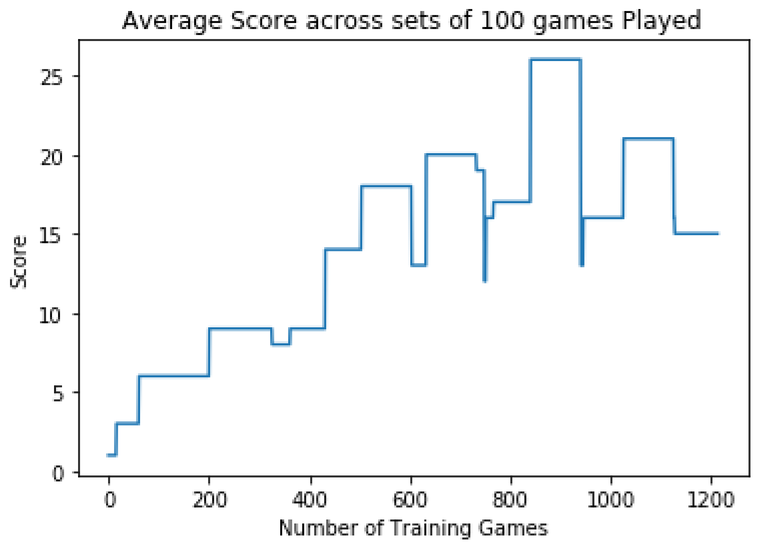
Performance when using a single frame





Performance when using multiple frames


Average Score across sets of 10 games Played
Score
Number of Training Games


Average Score across sets of 100 games Played
Score
Number of Training Games