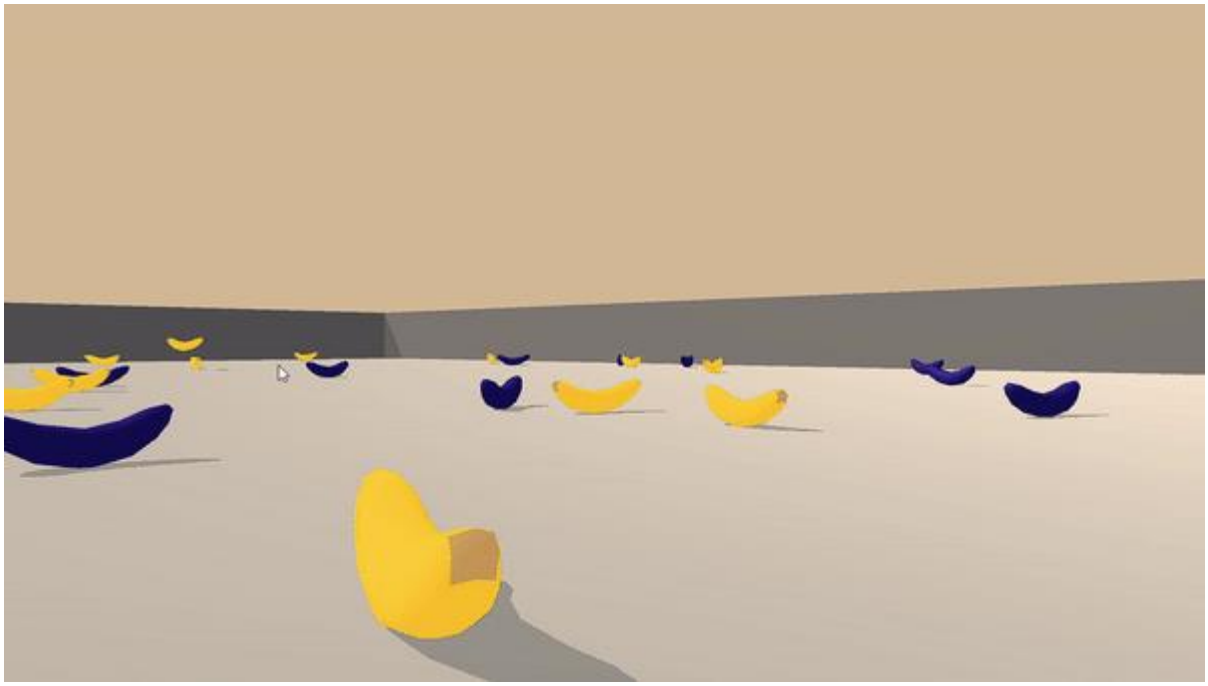
Navigation



Problem Statement

Using reinforcement learning techniques to train an agent to navigate (and collect bananas!) in a large, square world.

A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.

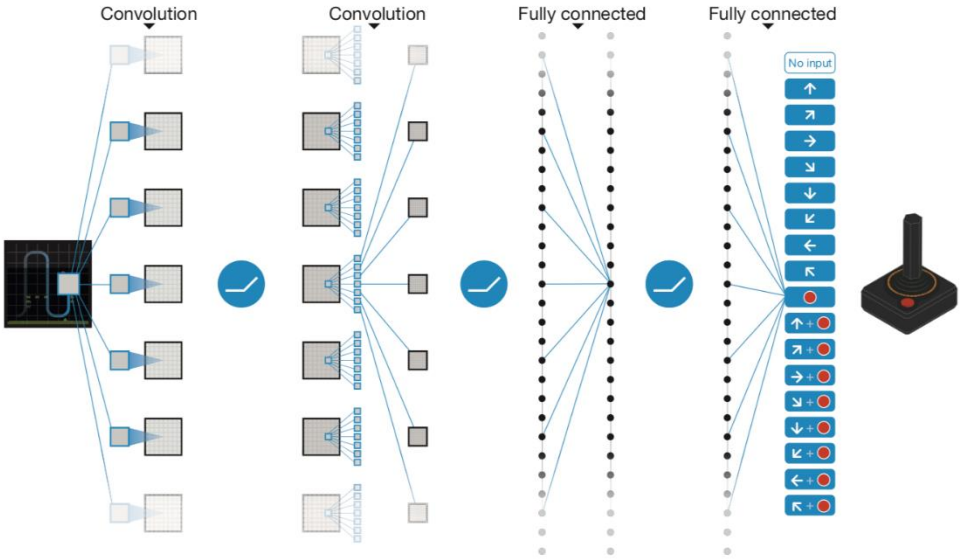
The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:

- ✓ 0 - move forward.
- ✓ 1 - move backward.
- ✓ 2 - turn left.
- ✓ 3 - turn right.

The task is episodic and ends when an average reward of a value greater the +13 is obtained.

Implementation

The learning algorithm used in this project is the Deep Q-Network Algorithm. In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output.



Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Figure 1:DQN Algorithm

Results

The agent was able to obtain an average reward of value greater than +13 within 529 episodes of learning.

```
end = time.time()
print("Total time taken :",end - start)
```

```
INFO:matplotlib.font_manager:generated new fontManager
```

```
Episode 100     Average Score: 1.87
Episode 200     Average Score: 6.33
Episode 300     Average Score: 9.11
Episode 400     Average Score: 10.96
Episode 500     Average Score: 11.84
Episode 529     Average Score: 13.02
Environment solved in 429 episodes!     Average Score: 13.02
```

Figure 2: Training using DQN

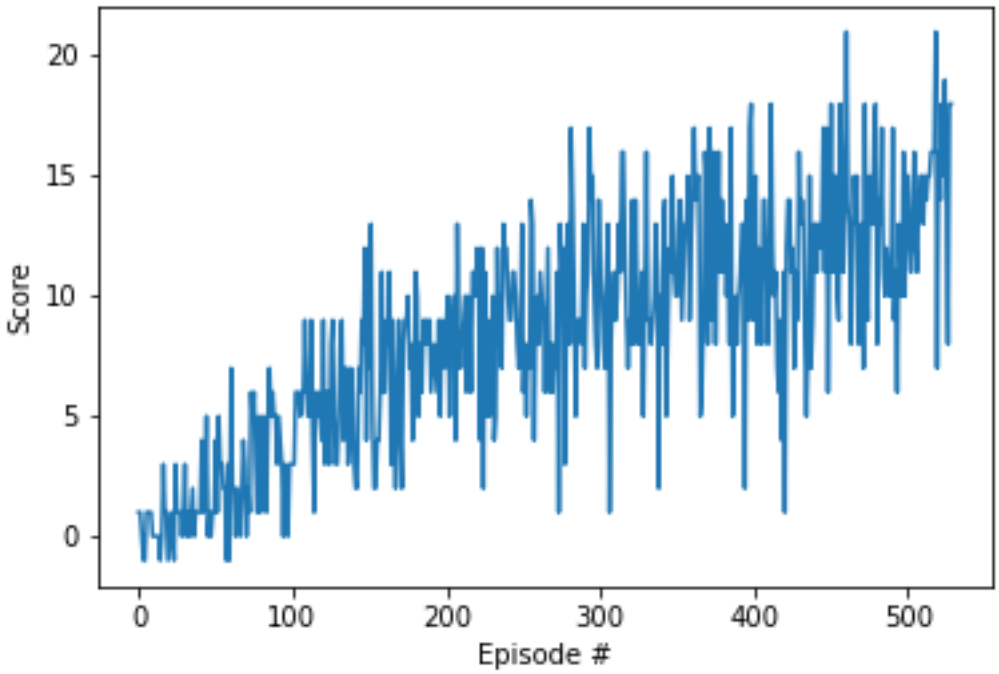


Figure 3: Episode # V/S Score

Conclusion

The agent is able to accumulate reward of +13.0 over 529 episodes of learning. Several other algorithms liken Double DQN, Duelling DQN etc. can be explored further to see how the agent performs. In this project agent learned from information such as its velocity, along with ray-based perception of objects around its forward direction, learning from pixel values which can be explored further. Prioritized replay could also be introduced which showed a huge improvement in learning of Atari Games.

References

- https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
- https://arxiv.org/abs/1511.05952