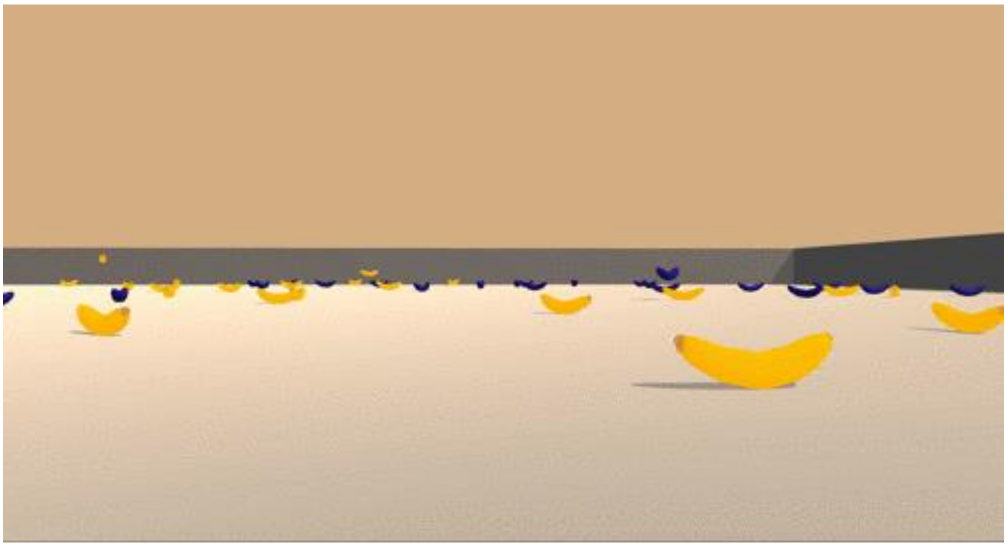
Deep Reinforcement Learning Nanodegree

Project 1: Navigation

Pradeep Dayanandam

Introduction:

For this project, we will train an agent to navigate (and collect bananas!) in a large, square world.



Overview of the environment

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 in order to solve the environment, your agent must get an average score of +13 over 100 consecutive episodes.

Algorithm Used:

The Algorithm used to solve this problem is Deep Q-Networks(DQN) with experience replay was proposed by Mnih et al. (2015). It takes agent's state as input and outputs Q action values. It uses experience replay and target network to stabilize the model training.

Pseudocode:

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 $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Taken from Human-level control through Deep Reinforcement Learning (Mnih et al. (2015))

Model Architecture:

The model is made of three fully connected layers. The number of neurons in first two layers is 64 and in the last layer it's equal to action size. Each layer's output except the last layer is transformed using the ReLU activation function.

Hyperparameters

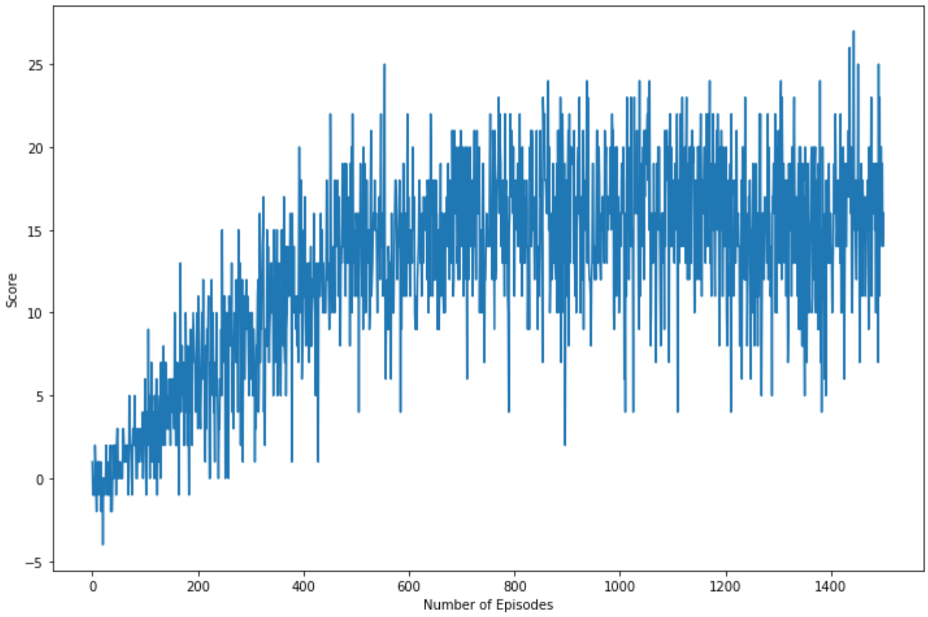
- BUFFER_SIZE = int(1e5) # replay buffer size
- BATCH_SIZE = 64 # minibatch size
- GAMMA = 0.99 # discount factor
- TAU = 1e-3 # for soft update of target parameters
- LR = 5e-4 # learning rate
- n_episodes = 1500 # maximum number of training episodes
- max_timesteps = 1000 # maximum number of time steps per episode
- epsilon_start = 1.0 # starting value of epsilon, for epsilon-greedy action selection
- epsilon_end = 0.01 # minimum value of epsilon
- epsilon_decay = 0.995 # decay factor (per episode) for decreasing epsilon

Results:

Average Score:

```
Episode 100    Average Score: 0.86
Episode 200    Average Score: 4.58
Episode 300    Average Score: 7.29
Episode 400    Average Score: 10.07
Episode 500    Average Score: 12.81
Episode 600    Average Score: 13.90
Episode 700    Average Score: 14.74
Episode 800    Average Score: 15.70
Episode 900    Average Score: 15.77
Episode 1000   Average Score: 16.06
Episode 1100   Average Score: 16.01
Episode 1200   Average Score: 16.90
Episode 1300   Average Score: 14.51
Episode 1400   Average Score: 15.02
Episode 1500   Average Score: 16.07
```

Plot of Rewards



Environment solved in 510 episodes!

Average score (when env was first solved): 13.03

Further the model was trained till 1500 episodes. Model.pt file contains the saved model weights of the successful agent.

The trained agent was able to perform well by scoring well in the three episodes tested.

- Episode 1 Score: 18.00
- Episode 2 Score: 17.00
- Episode 3 Score: 15.00

The trained agent was able to perform well by scoring well in the three episodes tested.

Future ideas to improve the agent's performance:

More experiments can be done to increase the performance of agent by applying different extensions of DQN:

- Double DQN (DDQN)
- Prioritized experience replay
- Dueling DQN
- A3C
- Distributional DQN
- Noisy DQN

We can also apply all the above extensions together. This was done by Deepmind's researchers and they have termed it Rainbow. This algorithm has outperformed each of the extension achieved SOTA results on Atari 2600.

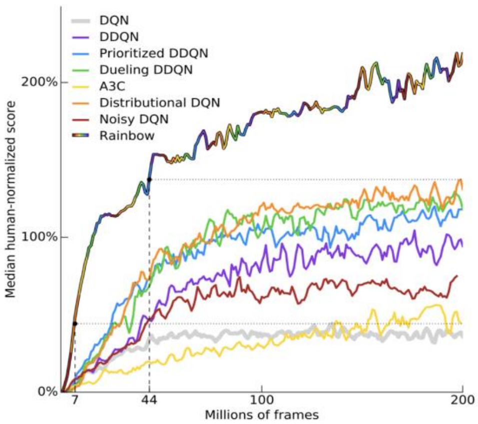


Figure 1: **Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final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

Taken from Rainbow: Combining Improvements in Deep Reinforcement Learning(Hessel et al. (2017))