Agent Implementation

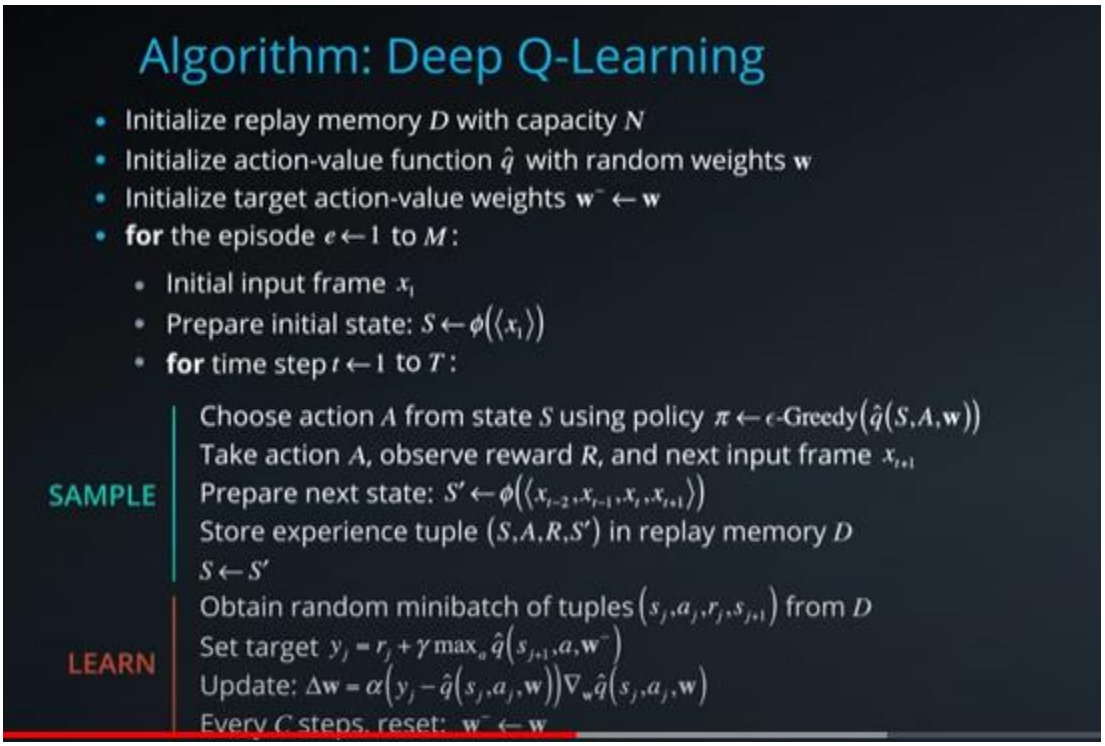
This project implements a Value Based method called Deep Q-Networks. DeepMind leveraged Deep Q-Network (DQN) to build Deep Q-Learning algorithm. This algorithm can learn to play Atari games.

Experience Replay is act of sampling batch of tuples from the replay buffer in order to learn more from individual tuples multiple times, recall rate occurrences, and in general make better use of our experiences. and Fixed Q-Targets.

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function[20]. This instability has several causes: the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$. We address these instabilities with a novel variant of Q-learning, which uses two key ideas. First, we used a biologically inspired mechanism termed experience replay[21–23] that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution (see below for details). Second, we used an iterative update that adjusts the action-values (Q) towards target values that are only periodically updated, thereby reducing correlations with the target.

The above article is taken from Human-level control through deep reinforcement learning.

Algorithm: Deep Q-learning



Code:

Model.py: QNetwork class is implemented. The network will predict the action to perform depending on the environment observed states. The QNetwork will be used by DQN agent and consists of two fully connected layers. The first layer's input size is provided through state_size and the output size is fc1_units. The second fully connect layter's input size is the output size of the first layer. The output size of the second layer is fc2_units. The default fc1_units and fc2_units are 64. The fc2_units is the input size of the last layer.

Below are the dimension of the QNetwork:

state_size (int): Dimension of each state

action_size (int): Dimension of each action

seed (int): Random seed

fc1_units (int): Number of nodes in first hidden layer

fc2_units (int): Number of nodes in second hidden layer

DQN_agent.py: Class Agent is defined. It has functions __init__, step, act, learn, and soft_update.

- The __init__ initializes state_size and action_size. It initializes the QNetwork and ReplayBuffer.
- The step function allows the memory of step taken to be saved for future learning and it allows to sample and learn from the tuples.
- The act function returns actions for given state as per current policy. It uses Epsilon Greedy action to balance between Exploitation and Exploration.
- The learn function updates the value parameters using given batch of experience tuples.
- The soft_update function is called by learn to update the target network. This is the Fixed Q-Targets.
- The ReplayBuffer is Fixed-size buffer to store experience tuples.

Navigation.ipynb: This notebook allows the training of Agent on the Banana environment.

- It installs the Banana's Brain environment.
- It exams the state and action spaces.
- It takes random actions in the environment.

The DQN agent uses the following parameters:

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

UPDATE_EVERY = 4 # how often to update the network

The input size is 37 and consists of two fully connected layers.

The Neural Networks use the Adam optimizer with a learning rate LR=5e-4 and are trained using a BATCH_SIZE=64

```
Episode 100     Average Score: 0.60
Episode 200     Average Score: 4.23
Episode 300     Average Score: 7.25
Episode 400     Average Score: 10.08
Episode 500     Average Score: 12.02
Episode 533     Average Score: 13.03
Environment solved in 433 episodes!     Average Score: 13.03
```