DQN for Banana collection game environment

Architecture

1. Introduction

Reinforcement learning is rooted in psychological and neurological perspective of animal behavior. In many cases the environment can be too complex that the number of states is not enumerable to a finite set. Animals and humans learn the complex environment around them through a harmonious combination of reinforcement learning and hierarchical sensory processing systems. DQN was one of the ground-breaking use of Deep neural networks in reinforcement learning to learn Atari games. The Deep Q-Network agent surpassed all previous algorithms and achieved levels comparable to professional human players.

This implementation to solve the Unit Environment Bananas, is based on the DQN algorithm used by Google DeepMind, to solve Atari games. The software agent (DQN) interacts with the environment thru a sequence of observations (37) actions (4) and rewards. The goal of the agent is to select an action that maximizes the cumulative future reward. This implementation uses a MLP (Multi-layer perceptron) network to approximate the optimal action-value function fig 1.

1.1 Q-Learning

The Bellman equation for Q-Learning is given below.

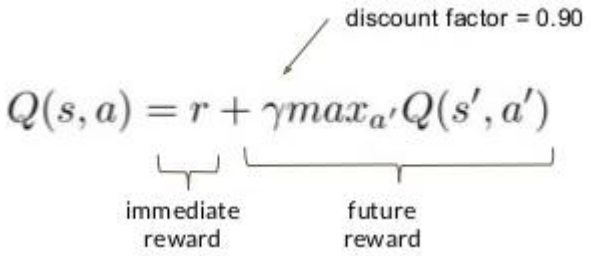

Fig. 1

As we take the samples from the environment, it is not a good idea to just assign new values on top of existing values. This can create lot of instability. What is usually done in practice is to update Q(s,a) with approximations using a blend of old and new values using a learning rate. The improved equation is shown in Fig 2.

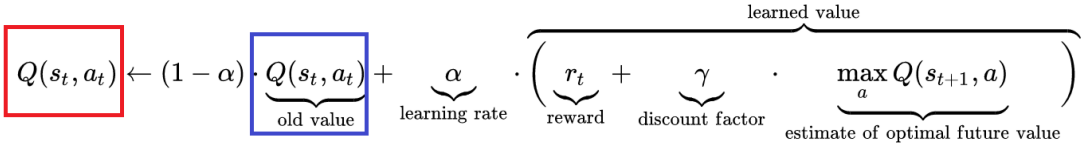

Fig. 2

1.2 Deep Q-Learning

Deep Q-learning network does not just use a network to get the next state values. Updating the same network with new values makes it unstable. DQN uses a replay buffer and a target network to make network updates stable.

$$\underbrace{Q(s,a)}_{\text{TD target}} = r(s,a) + \gamma Q(s', \underbrace{argmax_a Q(s',a)}_{\substack{\text{DQN Network choose} \\ \text{action for next state}}})$$

Target network calculates the Q value of taking that action at that state

2. Architecture.

The Architecture has five main components

1) Agent
2)The Network
3) Training
4) Saved Model

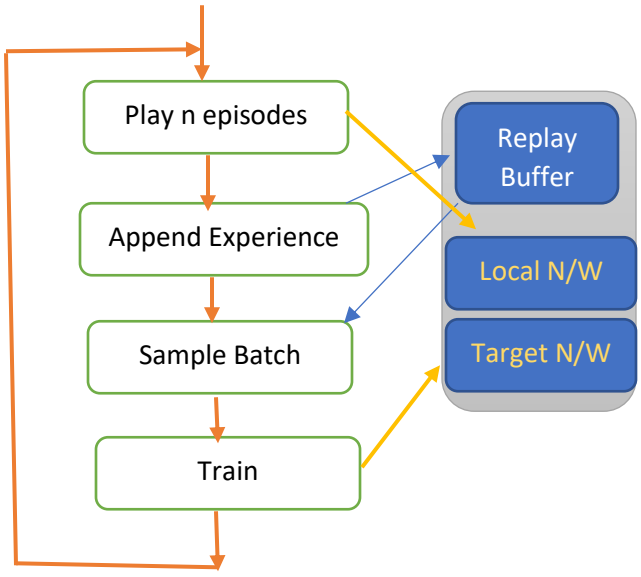

Fig. 3

2.1 Agent

The Agent has three components, two networks: local network to train and the other, target network to verify and a Replay buffer to save episodes of training. Data from the replay buffer is sampled to train the actor network while the target network is updated after specific time steps.

2.2 The Network

The Unity Banana environment has 37 observations for each state and 4 possible actions (see Environment below). The 37 observations form the input of a MLP network with 2 hidden layers and 4 output. The hidden layers have 128 neurons each. See Fig 4.

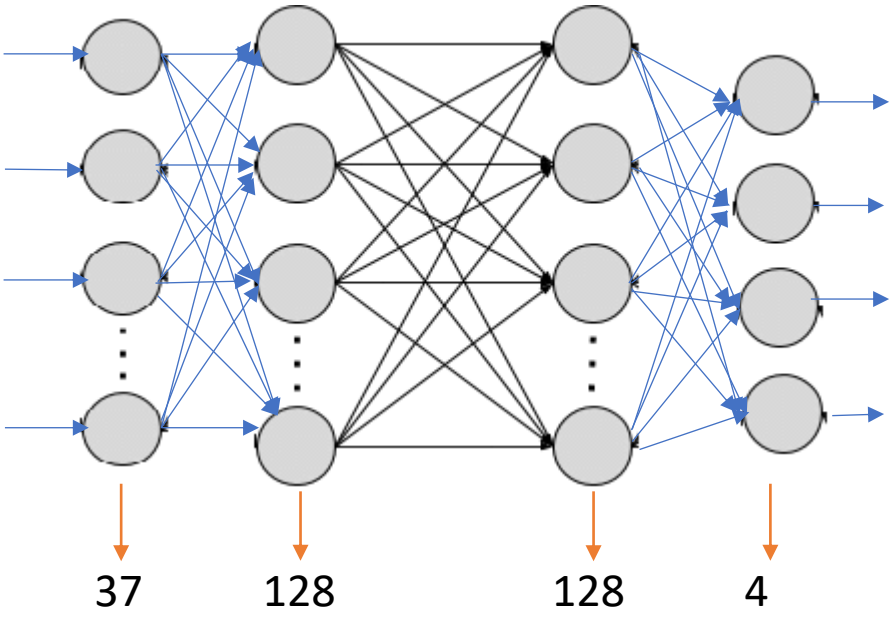

Fig 4

2.3 Training & Testing

The Agent is trained for 1800 episodes. With this network an average score of 13+ over 100 episodes was achieved within 600 episodes consistently. The training can be run by executing. The parameter used for the model are

Episode count :

Maximum : 1800

Epsilon :

```
starting value of epsilon, for epsilon-greedy action selection : 1.0
minimum value : 0.01
multiplicative factor (per episode) for decreasing epsilon
```

- Python Bananas.py

2.4 Saved Model execution.

The model is saved after 1800 episodes. This may not consistently perform and show a score of 13+ each time. But will gain a reasonable score in most runs and 13+ in few cases. To run saved model

- Python run_model.py

The code runs the game 3 times using saved model

3. Algorithm

3.1 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, the agent must get an average score of +13 over 100 consecutive episodes

3.2 Deep Q-learning Algorithm

The Deep Q-Learning algorithm represents the optimal action-value function. The algorithm is given below.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

4. Code

There are 4 files

1) Bananas.py
   This file has the code to train the networks. To run training execute this script.
2) dqn_agent.py
   The Agent and replay buffer is defined in this file. The Agent class instantiates two networks: local and target. The target network is the network against which the actions of local network is compared. Replay buffer is used to store episodes of game play. Data is randomly retrieved from the replay buffer to train the networks.
3) model.py
   Model.py has the MLP definition: both forward and backward pass.
4) run_model.py
   run_model.py if executed run the saved model. Make sure the Banana application path is valid before running this.

5. Improvements

There are several improvements to the original algorithm that is discussed in literature a few examples are

1. Double DQN.
   This improves estimation of action value.

2. Prioritized Experience Replay
   Prioritized experience replay is based on the idea that the agent can learn more effectively from some episodes of the game is of more value than other. So a priority is assigned to data in the replay buffer for retrieval and learning.

3. Dueling DQN
   Dueling DQN uses a dueling architecture to improve estimation of action values

6. References

1. Human-level control through deep reinforcement learning. The nature Volodymyr Mnih1 et. al

2. Reinforcement Learning An Introduction Richard S. Sutton & Andrew G. Barto

3. Deep reinforcement Learning hand-On. Maxim Lapan.