# Manual of DQN

Xinneng XU

July 2, 2019

## 1 Preparation

**Python Environment.** Before execute the programme, you should install the environment of python3, il doesn't work with python2.

**Modules.** In this programme, it is necessary to install numpy, tensorflow, pandas, matplotlib. You can use *pip* to install the modules.

## 2 Execution

**Files.** In the dossier *Deep Q-Learning/src*, there are the files:

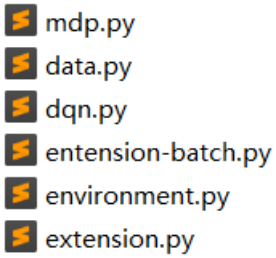


Figure 1: files

- *data.py*: This file defines the data that will be used to make simulations.
- *environment.py*: This file defines all variables global and functions necessary for assigning the variables. Before execution, you can modify the values of the variables for achieving your requirements, such as the variable $T$ is the number of iteration.
- *dqn.py*: In this file there is a class *DQN* which is in charge of the Q-network, it has several functions which can build the network, train the network etc.

  Functions used outside class *DQN*:

  - *get_Q_for_all_actions(self, state)* : For a state, calculate the Q for each arm in class *MDP* by the Q-network, then return the result.

- *train(self,state,arm,target)*: For a state and an arm, and target, train the network.
- *get_target(self, r, next_state)*: calculate the target by current network.

- *mdp.py*: In this file there is a class *DMP* which is in charge of managing the *state*, *action*, *transition* and several functions which can calculating new state, get optimal action etc.

  Functions used outside class *MDP*:

  - *get_next_state(self, a, y_a)* : For current state, with the input an arm and feedback of arm(1 or 0), calculate the next state $ns$ and the probability to transfer to $ns$, and return them.
  - *transition(self, next_state)*: Transfer to *next_state*
  - *get_action(self, dqn)*: Base on current state, network and $\epsilon$, get action by $\epsilon - greddy$.
  - *get_optimal_interval(self, dqn)*: Base on current network and arms selected to test, get the interval in which the optimal arm will be.
  - *get_estimated_theta(self, state, a)*: Calculate the estimated theta of arm a $\theta_a$ by the state input.
  - *get_estimated_expected_cost(self, state, a)*: Calculate the estimated cost of arm a by the state input.

- *extension.py*: This file defines the whole algorithm without batch and the entrance of programme executed without batch.
- *extension-batch.py*: This file defines the whole algorithm with batch and the entrance of programme executed with batch.

**For execution.**

1. Open terminal or other IDE, then go to the dossier *Deep Q-Learning/src*. Here there are 5 python files Figure 1.
2. To execute programme using algorithm normal(without batch), you can use the command *python extension.py*.
3. To execute programme using algorithm with batch, you can use the command *python extension-batch.py*.
4. After the programme finishes, the results necessary will be printed in terminal and they will be also saved in the variables in *environment.py*:

   For details, see *environment.py*.

# 3 Modify the code

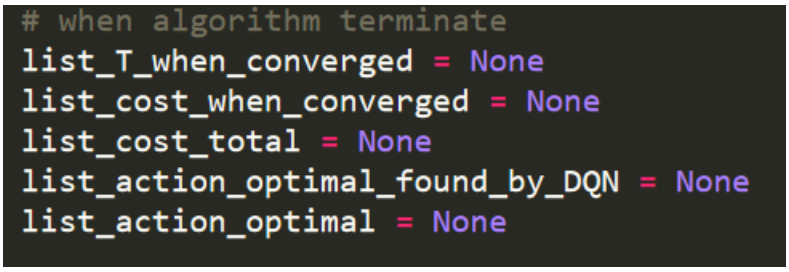


Figure 2: variables in *environment.py*

**Pour executer mon code** il faut —> installer jupyter notebook –> installer les bibliothèques tenserflow, numpy

**Voici la procédure pour exécuter le code**

1. Il faut tout d'abord être dans le répertoire DQN.
2. iI faut lancer la commande `jupyter notebook` (dans le répertoire DQN).

   Ici, il y a une ouverture de fenêtre. Il y a tous les fichiers de codes.

   Mettre une capture d'écran

   Ici donner la liste des fichiers

   (a) fichier "DQN1.0.ipynb : " source de l' algorithme DQN en considérant que l'apprentissage se fait "sans batch".
3. iI faut ouvrir le fichier : il faut cliquer sur le nom du fichier.
4. Pour executer le code, il faut aller dans le menu Kernel $--> $ Rstart & Run all. il faut cliquer pour lancer les code

   - Remarque : à chaque fin d'instructions ou partie de code (*In [xx]*), il y a un résultat intermédiaire (*Out [xx]*)