Navigation based on DQN

Project Objectives:

The project demonstrates how value-based methods can be used to learn the optimal policy in a model-free Reinforcement Learning setting using a Unity environment, in which I will train an agent to navigate (and collect bananas!) in a large, square world.

Reinforcement learning-related projects need to first understand the three elements of environmental state, action, and reward.

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, agent must get an average score of 13 over 100 consecutive episodes.

Project Flow:

This project will use the DQN algorithm to solve environmental problems. This algorithm is an improved algorithm based on Q-learning, so its update method is similar to Q-learning. The essence is to obtain labeled sample data through Q-learning, and then train the data through a neural network. The update of the parameter θ in the neural network is first performed by the agent that selects the initial value $Q(s_1, a_1)$ or the current value of $Q(s_i, a_i)$ predicted by the neural network, and then gets the reward in the environment r. At the same time, another Q target of the neural network in the algorithm also has its calculated $Q(s_i, a_i)$ estimated. By multiplying the neural network parameters of the current Q value and the learning rate α by the current Q and Q target values The gap can be obtained from the neural network parameters of the new Q current value. After the above process is performed a certain number of times, the neural network parameters of the Q target are updated.

The neural networks in the DQN algorithm all use convolutional neural networks as the approximation value function. The structure is usually three convolutional layers, two fully connected layers, and Relu is used as the activation function. By inputting the value calculated by the approximation function and action A to the neural network, output $Q(s_i, a_i, \theta)$, or you can input S to output $Q(s'_i, a'_i, \theta')$, Its error function formula can be expressed as:

$$\mathrm{L} = \frac{1}{2}\sum_a \sum_s (Q(s, a, \theta) - Q(s', a', \theta'))^2$$

Where $Q(s, a, \theta)$ is the cumulative return value of the real model, and $Q(s', a'; \theta')$ is the cumulative return value predicted by the value function model. In order to minimize the difference between them, gradient descent Method, its update formula is:

$$\theta_{t+1} = \theta_t + \alpha\big(r + \gamma max Q(s', a', \theta_t) - Q(s, a, \theta_t)\big)$$

The DQN algorithm is shown in the flowchart, and its detailed steps are as follows:

```
Initialize the neural network parameters $\theta^Q$ of the empirical playback pool, current value and target value;
Set the number of samples of the stored data to N, and randomly assign a neural network weight parameter $\theta$;
For each input data M:
    Initialize network input, set data dimensions, and calculate network output;
        For each time step t:
        ① When the time step is 1, a random starting action is selected according to the probability $\epsilon$. When the time step is not 1, the current state is input into the current value network, and the action with the largest value is selected according to its calculation;
        ② After the agent executes action $a_t$, it gets reward $r_t$ and the next state;
        ③ Four parameters (s, a, r, s') are stored in the experience playback pool as the state at time t;
        ④ When data is stored to N, minibatch states are randomly taken out;
        ⑤ Calculate the data action value, and update the Q value by the reward value obtained after the action is performed;
        ⑥ Update weights via gradient descent;
        ⑦And update the parameters of the current value network to the parameters of the target value network after every X iterations;
End the loop.
```

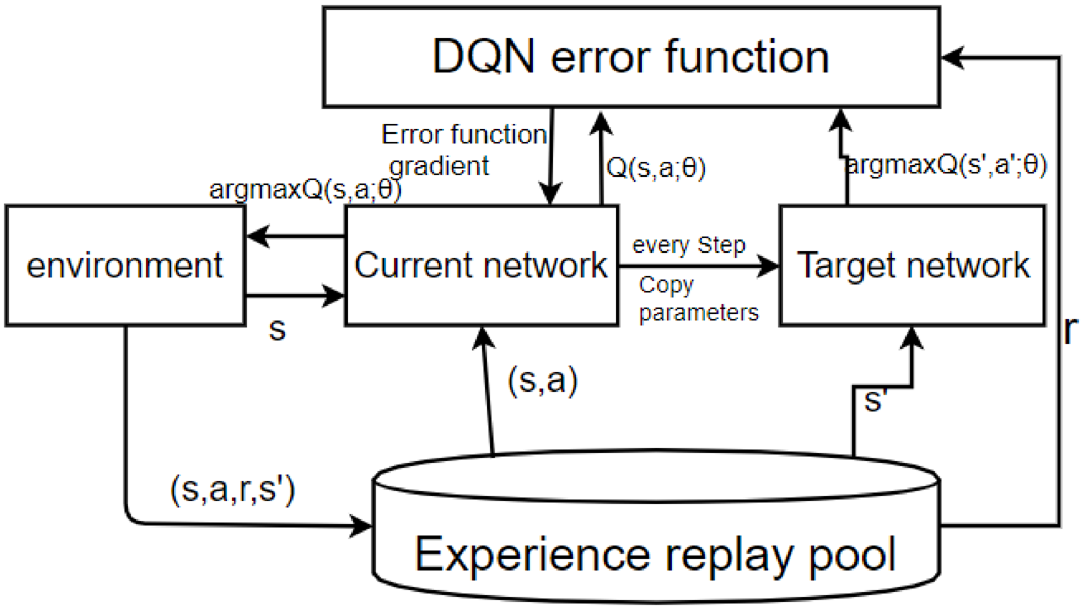


Fig. 1 DQN algorithm framework

Code composition:

In the code part, I mainly refer to the completed multi-agent report to write, because the part of the DDPG algorithm is DQN.

My code is mainly composed of four parts, namely neural network, hyperparameters, agent and training. In the agent part, it is mainly divided into three parts: behavioral function, learning function and experience playback. The main action is to update the weights by evaluating the target Q network after the agent takes action through the Q network.

In the agent-defined code, there is an important hyperparameter epsilon that controls the exploration of the agent. As the value approaches 1, the future total reward becomes more important. When the value is equal to 0, the agent only pays attention to the return of the current state, and has nothing to do with the future. In order to explore the action space more, the EPS_START parameter is set to 1.0, and then multiplied by EPS_DECAY after each training to achieve the effect of gradual convergence. This approach reduces the likelihood of exploration over time.

This project uses the SGD optimizer to update the learning rate by multiplying the decreasing factor in each cycle. Here, use a method similar to Epsilon Greedy to achieve it. Finally, the experience playback function was used. In order to solve the gradient explosion problem, the model uses the utils.clip_grad_norm function to implement gradient clipping, and sets the gradient range to 1. Prevent the gradient from growing exponentially by setting an upper limit on the size of the parameter update. This can make the agent learn more stably and quickly.

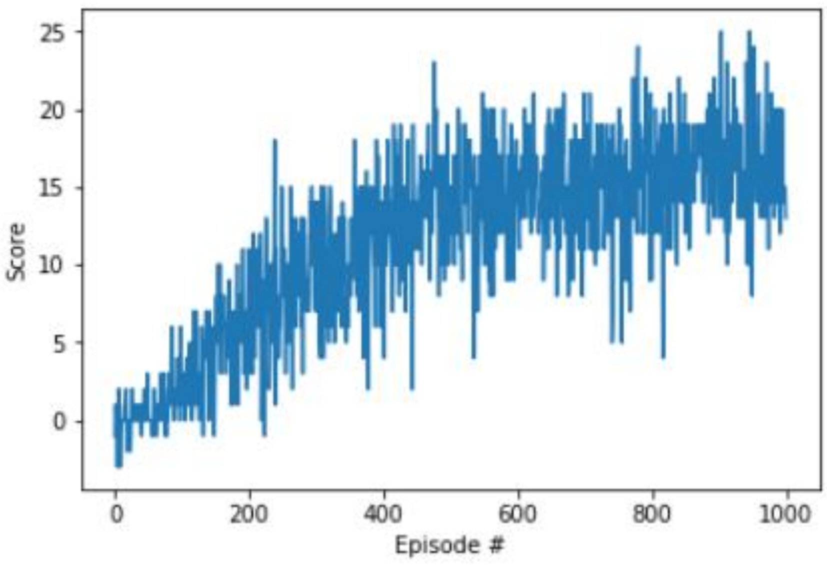
After the experience pool is full, by setting, each step can be learned multiple times in order to converge faster and get higher scores. However, setting too much will slow down the training speed, which is a problem worth balancing. Finally, I set the learning count to 10 so that the agent can learn better.

About hyperparameters:

```
EXPERIENCE_POOL_SIZE = 100000    # replay buffer size
BATCH_SIZE = 64                  # minibatch size
GAMMA = 0.99                     # discount factor
TAU = 0.001                      # for soft update of target parameters
LR = 0.0005                      # learning rate
LEARN_EVERY = 4                  # how often to update the network
WINDOW_LENGTH = 100              # limited number of actions
PRINT_EVERY = 30                 # print the score every tenth time
EPS_START = 1.0                  # Initial value of ε during noise attenuation
EPS_DECAY = 0.995                # to decay exploration as it learns
EPS_FINAL = 0.01                 # final value of ε after attenuation
```

Operation result:

The following figure shows the final training results.



After 379 episodes of training, the agent can solve the problem of the project environment. The highest score is 24.22 and the highest average is 17.00

Future Development:

Algorithmically:

DDPG achieves DQN and decreases convergence speed in tasks in continuous action space, but it cannot use random environment problems. Then, I think the most important improvement of reinforcement learning is the use of transfer learning. If the agent can continuously improve through memory, then general artificial intelligence is just around the corner. Finally, the priority experience playback algorithm is not used in this model training. The priority experience reply does not learn the data stored in the experience pool through random selection, but chooses the experience based on the priority value related to the error size.

At Work:

I will consider using MADDPG for medical diagnosis. By treating the patient's physical state as a dynamic environment and each external drug dose as an action, I will explore the application of artificial intelligence in medicine in this way.