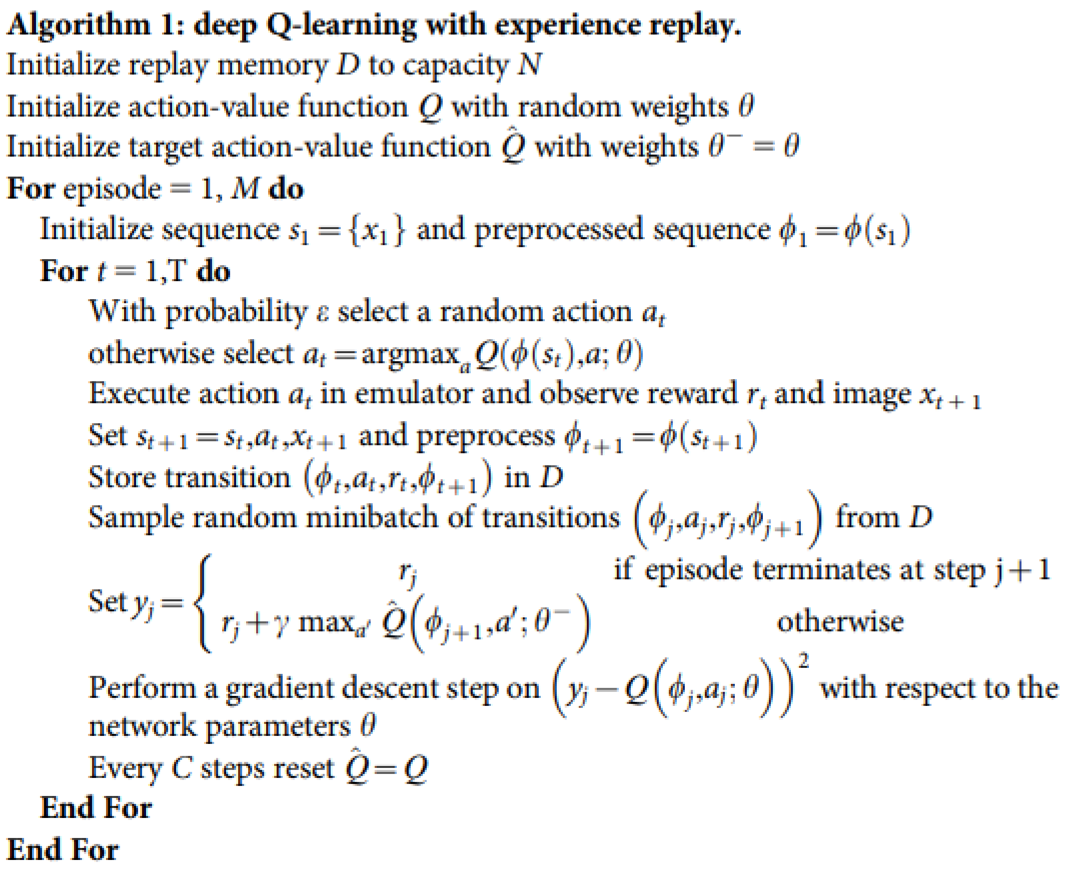
RL_exercise

Environment: CartPole-v0 from gym

Deep Q-Learning

Algorithm:



```
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ 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 $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```

Q-learning is a model-free algorithm that learns optimal Q(s,a) action value functions from the agent's history of interaction with the environment.

In Deep Q-learning, neural network is used to calculate action value function. To ensure stable performance, two tricks are used:

1. Experience Replay: At each time step the agent memorizes some experience (state, action, reward, next_action, done), and learns by replaying a batch of experience from its memory. This is done to reduce the effect of correlations between sequence of observations which would have made neural network behave poorly.

2. Delayed Update: A separate "target" network is used to generate target value during experience replay. Every C timesteps the target network is updated to the parameters of training network. Doing so can increase the stability of the model, because it reduces the issue that a very small update to training network may change the outcome of policy, causing possible divergence during training.

Double DQN

It was shown that Q-learning algorithm tend to overestimate action values, which results in worse policy under some conditions. DQN also suffers the same problems. Double DQN reduces this problem by generalizing the idea of Double Q-learning algorithm to DQN.

Double Q-learning algorithm uses two value functions during update, one to determine the greedy policy, the other to determine the action value from this policy. Doing so prevents the model using the same value for both selecting policy and evaluating action, which can cause overestimation. The two value function are updated separately during training, and experience is assigned randomly to update each value function.

Because there already exist two networks, the online training network and target network, in DQN. Double DQN can achieves similar result as Double Q-learning by using online network for policy selection, and target network for policy evaluation. Basically, the target changes from:

$$Y_t^{\mathrm{DQN}} = R_{t+1} + \gamma \mathrm{max}_a Q(S_{t+1}, a; \theta_t^-)$$

to:

$$Y_t^{\mathrm{DoubleDQN}} = R_{t+1} + \gamma Q(S_{t+1}, \mathrm{argmax}_a Q(S_{t+1}, a; \theta_t), \theta_t^-)$$

Current Results

Currently Deep Q-learning algorithm has been completely implemented including experience replay and delayed update. On top of that a stopping mechanism is implemented to stop learning when the agent can pass a consecutive number of episodes with maximum rewards. Now the agent can consistently pass the given task in around 400 episodes of training, usually getting maximum reward for all 100 test episodes.

Double DQN is also implemented. The small difference between DQN and Double DQN makes the implementation quite easy. For CartPole-v0, double DQN shows similar results with normal DQN. It seems that double DQN is less prone to divergence, but both DQN and double DQN can diverge from optimal policy if overtrained. Typically, both algorithms give good results after 400 episodes of training, but if training continues the both models are likely to diverge after 800 episodes.

For practice purposes, a basic Q-Learning agent is also implemented prior to all deep learning agents. The basic Q-learning agent does not perform very well and cannot solve the CartPole problem in 1000 episodes of training.

Reference

[Deep Q-Learning Nature]

Double Q-Learning

Double DQN

Mxnet Tutorial