import gym

import tensorflow as tf

import numpy as np

import random

from collections import deque

# Hyper Parameters for DQN

GAMMA = 0.9 # discount factor for target Q

INITIAL\_EPSILON = 0.5 # starting value of epsilon

FINAL\_EPSILON = 0.01 # final value of epsilon

REPLAY\_SIZE = 10000 # experience replay buffer size

BATCH\_SIZE = 32 # size of minibatch

class DQN():

# DQN Agent

def \_\_init\_\_(self, env):

# init experience replay

self.replay\_buffer = deque()

# init some parameters

self.time\_step = 0

self.epsilon = INITIAL\_EPSILON

self.state\_dim = env.observation\_space.shape[0]

self.action\_dim = env.action\_space.n

self.create\_Q\_network()

self.create\_training\_method()

# Init session

self.session = tf.InteractiveSession()

self.session.run(tf.initialize\_all\_variables())

def create\_Q\_network(self):

# network weights

W1 = self.weight\_variable([self.state\_dim,20])

b1 = self.bias\_variable([20])

W2 = self.weight\_variable([20,self.action\_dim])

b2 = self.bias\_variable([self.action\_dim])

# input layer

self.state\_input = tf.placeholder("float",[None,self.state\_dim])

# hidden layers

h\_layer = tf.nn.relu(tf.matmul(self.state\_input,W1) + b1)

# Q Value layer

self.Q\_value = tf.matmul(h\_layer,W2) + b2

def create\_training\_method(self):

self.action\_input = tf.placeholder("float",[None,self.action\_dim]) # one hot presentation

self.y\_input = tf.placeholder("float",[None])

Q\_action = tf.reduce\_sum(tf.mul(self.Q\_value,self.action\_input),reduction\_indices = 1)

self.cost = tf.reduce\_mean(tf.square(self.y\_input - Q\_action))

self.optimizer = tf.train.AdamOptimizer(0.0001).minimize(self.cost)

def perceive(self,state,action,reward,next\_state,done):

one\_hot\_action = np.zeros(self.action\_dim)

one\_hot\_action[action] = 1

self.replay\_buffer.append((state,one\_hot\_action,reward,next\_state,done))

if len(self.replay\_buffer) > REPLAY\_SIZE:

self.replay\_buffer.popleft()

if len(self.replay\_buffer) > BATCH\_SIZE:

self.train\_Q\_network()

def train\_Q\_network(self):

self.time\_step += 1

# Step 1: obtain random minibatch from replay memory

minibatch = random.sample(self.replay\_buffer,BATCH\_SIZE)

state\_batch = [data[0] for data in minibatch]

action\_batch = [data[1] for data in minibatch]

reward\_batch = [data[2] for data in minibatch]

next\_state\_batch = [data[3] for data in minibatch]

# Step 2: calculate y

y\_batch = []

Q\_value\_batch = self.Q\_value.eval(feed\_dict={self.state\_input:next\_state\_batch})

for i in range(0,BATCH\_SIZE):

done = minibatch[i][4]

if done:

y\_batch.append(reward\_batch[i])

else :

y\_batch.append(reward\_batch[i] + GAMMA \* np.max(Q\_value\_batch[i]))

self.optimizer.run(feed\_dict={

self.y\_input:y\_batch,

self.action\_input:action\_batch,

self.state\_input:state\_batch

})

def egreedy\_action(self,state):

Q\_value = self.Q\_value.eval(feed\_dict = {

self.state\_input:[state]

})[0]

if random.random() <= self.epsilon:

return random.randint(0,self.action\_dim - 1)

else:

return np.argmax(Q\_value)

self.epsilon -= (INITIAL\_EPSILON - FINAL\_EPSILON)/10000

def action(self,state):

return np.argmax(self.Q\_value.eval(feed\_dict = {

self.state\_input:[state]

})[0])

def weight\_variable(self,shape):

initial = tf.truncated\_normal(shape)

return tf.Variable(initial)

def bias\_variable(self,shape):

initial = tf.constant(0.01, shape = shape)

return tf.Variable(initial)

# ---------------------------------------------------------

# Hyper Parameters

ENV\_NAME = 'CartPole-v0'

EPISODE = 10000 # Episode limitation

STEP = 300 # Step limitation in an episode

TEST = 10 # The number of experiment test every 100 episode

def main():

# initialize OpenAI Gym env and dqn agent

env = gym.make(ENV\_NAME)

agent = DQN(env)

for episode in xrange(EPISODE):

# initialize task

state = env.reset()

# Train

for step in xrange(STEP):

action = agent.egreedy\_action(state) # e-greedy action for train

next\_state,reward,done,\_ = env.step(action)

# Define reward for agent

reward\_agent = -1 if done else 0.1

agent.perceive(state,action,reward,next\_state,done)

state = next\_state

if done:

break

# Test every 100 episodes

if episode % 100 == 0:

total\_reward = 0

for i in xrange(TEST):

state = env.reset()

for j in xrange(STEP):

env.render()

action = agent.action(state) # direct action for test

state,reward,done,\_ = env.step(action)

total\_reward += reward

if done:

break

ave\_reward = total\_reward/TEST

print 'episode: ',episode,'Evaluation Average Reward:',ave\_reward

if ave\_reward >= 200:

break

if \_\_name\_\_ == '\_\_main\_\_':

main()