CartPole

January 17, 2020

```
In [ ]: import gym
        import os
        import time
        import random
        import numpy as np
        from collections import deque
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.optimizers import Adam
        import matplotlib.pyplot as plt
        env = gym.make('CartPole-v0')
        obs = env.reset()
        state_size = env.observation_space.shape[0]
        action_size = env.action_space.n
        batch_size = 32
        n_episodes = 1001
        output_dir = 'model_output/cartpole'
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)


        # Define agent

        class DQNAgent:
            def __init__(self, state_size, action_size):
                self.state_size = state_size
                self.action_size = action_size
                self.memory = deque(maxlen = 2000)
                self.gamma = 0.95
                self.epsilon = 1.0
                self.epsilon_decay = 0.995
                self.epsilon_min = 0.01
                self.learning_rate = 0.1

                self.model = self._build_model()

            def _build_model(self):
                model = Sequential()
        model.add(Dense(32,input_dim = self.state_size, activation = 'relu'))
        model.add(Dense(32,activation = 'relu'))
        model.add(Dense(self.action_size, activation = 'linear'))
        model.compile(loss = 'mse', optimizer = Adam(lr = self.learning_rate))
        return model

    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))

    def act(self, state):
        if np.random.rand() <= self.epsilon:
            return random.randrange(self.action_size)
        act_values = self.model.predict(state)
        return np.argmax(act_values[0])

    def replay(self, batch_size):
        minibatch = random.sample(self.memory, batch_size)

        for state, action, reward, next_state, done in minibatch:
            target = reward
            if not done:
                target = reward + self.gamma * np.max(self.model.predict(next_state)[0]

            target_f = self.model.predict(state)
            target_f[0][action] = target

            self.model.fit(state, target_f, epochs = 1, verbose = 0)

            if self.epsilon > self.epsilon_min:
                self.epsilon *= self.epsilon_decay

    def load(self, name):
        self.model.load_weights(name)
    def save(self, name):
        self.model.save_weights(name)

agent = DQNAgent(state_size, action_size)

# interact with the environment

done = False

for e in range(n_episodes):

    state = env.reset()
    state = np.reshape(state, [1, state_size])
```

```
env.render()

action = agent.act(state)
next_state, reward, done, _ = env.step(action)
reward = reward if not done else -100
next_state = np.reshape(next_state, [1, state_size])
agent.remember(state, action, reward, next_state, done)
state = next_state

if done:
    print('episode: {}/{}, score:{}, e:{:.2}'.format(e, n_episodes,time,agent.epsil
    break
    env.close()

if len(agent.memory) > batch_size:
    agent.replay(batch_size)
if e % 50 == 0:
    agent.save(output_dir + 'weights_' + '{:4d}'.format(e) + '.hdf5')
```