Lab 6-2: Q Network for Cart Pole

Reinforcement Learning with TensorFlow & OpenAI Gym
Sung Kim <hunkim+ml@gmail.com>
Cart Pole

```python
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample())  # take a random action
```

It should look something like this:

https://gym.openai.com/docs
import gym

env = gym.make('CartPole-v0')
env.reset()
random_episodes = 0
reward_sum = 0

while random_episodes < 10:
    env.render()
    action = env.action_space.sample()
    observation, reward, done, _ = env.step(action)
    print(observation, reward, done)
    reward_sum += reward
    if done:
        random_episodes += 1
        print("Reward for this episode was:", reward_sum)
        reward_sum = 0
        env.reset()
Rewards

<table>
<thead>
<tr>
<th>Values</th>
<th>Action</th>
<th>Reward</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-0.1760681, -0.21040623, 0.0050548, 0.33178224]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.02181493, -0.40559977, 0.01169044, 0.6260549]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.02992693, -0.60088294, 0.02421154, 0.92239656]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.04194459, -0.79632351, 0.04265947, 1.22258898]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.05787106, -0.60177631, 0.06711125, 0.94357177]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.06990658, -0.79773512, 0.08598269, 1.25656419]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.08586128, -0.60381257, 0.11111397, 0.99200273]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.09793754, -0.41033868, 0.13095402, 0.7361819]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.10614431, -0.60700289, 0.14567766, 1.06704293]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.11828437, -0.80372008, 0.16701852, 1.40167111]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.13435877, -0.61102015, 0.19505194, 1.16551887]</td>
<td>1.0 False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[-0.14657917, -0.41889634, 0.21836232, 0.93978019]</td>
<td>1.0 True</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reward for this episode was: 12.0

# Get new state and reward from environment
s1, reward, done, _ = env.step(a)

if done:
    Qs[0, a] = -100

else:
    x1 = np.reshape(s1, [1, input_size])
    # Obtain the Q' values by feeding the new state through our network
    Qs1 = sess.run(Qpred, feed_dict={X: x1})
    Qs[0, a] = reward + dis * np.max(Qs1)
Cart Pole Q-network

\( (1) s \rightarrow W_s (2) \)
Q-Network training (Network construction)

\[(1)s \rightarrow \text{input layer} \rightarrow \text{hidden layer 1} \rightarrow \text{hidden layer 2} \rightarrow (2)Ws \rightarrow \text{output layer}\]

```
# First layer of weights
W1 = tf.get_variable("W1", shape=[input_size, output_size],
                   initializer=tf.contrib.layers.xavier_initializer())
Qpred = tf.matmul(X, W1)
```
Q-Network training (linear regression)

\[
cost(W) = (Ws - y)^2
\]
import numpy as np
import tensorflow as tf

import gym
env = gym.make('CartPole-v0')

# Constants defining our neural network
learning_rate = 1e-1
input_size = env.observation_space.shape[0]
output_size = env.action_space.n

X = tf.placeholder(tf.float32, [None, input_size], name="input_x")

# First layer of weights
W1 = tf.get_variable("W1", shape=[input_size, output_size],
                        initializer=tf.contrib.layers.xavier_initializer())
Qpred = tf.matmul(X, W1)

# We need to define the parts of the network needed for learning a policy
Y = tf.placeholder([None, output_size], dtype=tf.float32)

# Loss function
loss = tf.reduce_sum(tf.square(Y - Qpred))

# Learning
train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

# Values for q learning
num_episodes = 2000
dis = 0.9
rList = []
for i in range(num_episodes):
    e = 1. / ((i / 10) + 1)
rAll = 0
step_count = 0
s = env.reset()
done = False

# The Q-Network training
while not done:
    step_count += 1
    x = np.reshape(s, [1, input_size])
    # Choose an action by greedily (with e chance of random action) from the Q-network
    Qs = sess.run(Qpred, feed_dict={X: x})
    if np.random.rand(1) < e:
        a = env.action_space.sample()
    else:
        a = np.argmax(Qs)

    # Get new state and reward from environment
    s1, reward, done, _ = env.step(a)
    if done:
        Qs[0, a] = -100
    else:
        x1 = np.reshape(s1, [1, input_size])
        # Obtain the Q' values by feeding the new state through our network
        Qs1 = sess.run(Qpred, feed_dict={X: x1})
        Qs[0, a] = reward + dis * np.max(Qs1)

    # Train our network using target and predicted Q values on each episode
    sess.run(train, feed_dict={X: x, Y: Qs})

rList.append(step_count)
print("Episode: {} steps: {}\n".format(i, step_count))
# If last 10's avg steps are 500, it's good enough
if len(rList) > 10 and np.mean(rList[-10:]) > 500:
    break

preprocess $\phi_{t+1} = \phi(s_{t+1})$

Set $y_j = \begin{cases} r_j \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) \end{cases}$ for terminal $\phi_{j+1}$ for non-terminal $\phi_{j+1}$
Code: apply

```python
rList.append(step_count)
print("Episode: {} steps: {}".format(i, step_count))
# If last 10's avg steps are 500, it's good enough
if len(rList) > 10 and np.mean(rList[-10:]) > 500:
    break

# See our trained network in action
observation = env.reset()
reward_sum = 0
while True:
    env.render()
    x = np.reshape(observation, [1, input_size])
    Qs = sess.run(Qpred, feed_dict={X: x})
    a = np.argmax(Qs)

    observation, reward, done, _ = env.step(a)
    reward_sum += reward
    if done:
        print("Total score: {}".format(reward_sum))
        break
```
Results: really poor!

Episode: 1988  steps: 14
Episode: 1989  steps: 25
Episode: 1990  steps: 15
Episode: 1991  steps: 23
Episode: 1992  steps: 19
Episode: 1993  steps: 17
Episode: 1994  steps: 46
Episode: 1995  steps: 20
Episode: 1996  steps: 17
Episode: 1997  steps: 15
Episode: 1998  steps: 33
Episode: 1999  steps: 22

2017-02-08 16:59:31.216 Python[7525:2769691]
Total score: 15.0
Why does not work? Too shallow?

```
X = tf.placeholder(tf.float32, [None, input_size], name="input_x")

# First layer of weights
W1 = tf.get_variable("W1", shape=[input_size, output_size],
                      initializer=tf.contrib.layers.xavier_initializer())
Qpred = tf.matmul(X, W1)
```

- But **diverges** using neural networks due to:
  - Correlations between **samples**
  - Non-stationary targets
Excise

• Why does not work?
• Hint: DQN