

基于tensorflow的最简单的强化学习入门-part0：Q-learning和神经网络



y_felix (/u/c706b1b8d11d) 作者 + 关注

2017.03.12 10:16* 字数 1762 阅读 612 评论 0 喜欢 4

(/u/c706b1b8d11d)

基于tensorflow的最简单的强化学习入门-part0：Q学习和神经网络

本文翻译自 Simple Reinforcement Learning with Tensorflow Part 0: Q-Learning with Tables and Neural Networks，作者是 Arthur Juliani，原文链接 (<https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0#.5m3361v1w>)。

在这个增强学习系列的教程中，我们打算探索一些列称为==Q-learning==的增强学习算法，它和之前教程介绍过的基于==策略梯度policy-base==的增强算法有所不同。

我们将从实现一个简单的查找表算法开始，然后展示如何使用tensorflow实现神经网络算法。考虑到上述安排，我们从基础开始，所以这篇教程作为整个系列的part-0。希望通过这个系列的教程，我们在理解Q-learning之后，能够结合policy gradient和Q-learning方法构建更好的增强学习网络。（如果你对策略网络更感兴趣或者你已经有一些Q-learning的经验，那么你可以从这里 (<http://note.youdao.com/>)开始阅读）



策略梯度算法(policy gradient)试着学习某个函数，该函数可以直接把==状态(state)映射为动作(action)的概率分布==。Q-learning和策略梯度算法不一样，它试着学习在每个状态下对应的值，并且依赖该状态执行某一个动作。虽然两种方法最终都允许我们给定情况下采取特定的行动，但是实现该方法的方法是不相同的。你也许已经听说深度Q-learning可以玩atari游戏，我们将要在这里讨论和实现这些更复杂和强大的Q-learning算法。

译者注：如果要深入了解基于策略梯度的增强学习算法，可以参考Andrej Karpathy的文章，Deep Reinforcement Learning: Pong from Pixels (<http://karpathy.github.io/2016/05/31/rl/>)

Tabular Approaches for Tabular Environment(表格算法)

| | |
|------|---|
| SFFF | (S: starting point, safe) |
| FHFH | (F: frozen surface, safe) |
| FFFH | (H: hole, fall to your doom) |
| FFFG | (G: goal, where the frisbee is located) |

FrozenLake环境的规则

在本篇教程中，我们将要试着去解决来自于OpenAI gym (<https://gym.openai.com/>)的FrozenLake (<https://gym.openai.com/envs/FrozenLake-v0>)问题。OpenAI gym提供了一种简单的环境，让研究者在一些简单的游戏中试验他们的方法。比如FrozenLake，该游戏包括一个4*4的网络格子，每个格子可以是==起始块，目标块、冻结块或者危险块==。我们的目标是让agent学习从开始块如何行动到目标块上，而不是移动到危险块上。agent可以选择向上、向下、向左或者向右移动，同时游戏中还有可能吹来一阵风，将agent吹到任意的方块上。在这种情况下，每个时刻都有完美的策略是不能的，但是如何避免危险洞并且到达目标洞肯定是可行的。



增强学习需要我们定义==奖励函数(reward function)==，那么定义每一步的奖励是0，如果进入目标快则奖励为1。因此我们需要一种算法能够学习到长期的期望奖励，而本教程要讲的Q-learning提供了这种机制。

Q - learning最简单的实现方式是一个基于所有可能的状态和执行动作的查找表。在表中的每个单元格中，我们学习到在给定状态下执行特定动作的是否有效的值。在FrozenLake游戏中，我们有16种可能的状态和4种可能的动作，给出了16*4的Q值表。我们一开始初始化表格中的所有值为0，然后根据我们观察到的各种动作获得的奖励，相应地更新表格。

我们使用称为贝尔曼方程 (https://en.wikipedia.org/wiki/Bellman_equation)(bellman equation)的更新来对Q表进行更新。==该方程表明，给定动作的长期预期奖励来自于当前动作的即时奖励，以及来自未来最佳动作的预期奖励==。公式如下：

$$\text{Eq 1. } Q(s, a) = r + \lambda \max_{a'} Q(s', a')$$

这个公式说明，给定动作a和状态s的q值，等于当前的奖励r加上未来预期的q值。 λ 可以看作未来期望q值和当前奖励相比的权重。以这种方式更新，q表就会慢慢的收敛到在给定状态下和动作下的期望收益。下面一段python代码就是基于FrozenLake游戏的Q - table的算法实现。



```
import gym
import numpy as np
#Load the environment
env = gym.make('FrozenLake-v0')
#Implement Q-Table learning algorithm
#Initialize table with all zeros
Q = np.zeros([env.observation_space.n,env.action_space.n])
# Set learning parameters
lr = .85
y = .99
num_episodes = 2000
#create lists to contain total rewards and steps per episode
#jList = []
rList = []
for i in range(num_episodes):
    #Reset environment and get first new observation
    s = env.reset()
    rAll = 0
    d = False
    j = 0
    #The Q-Table learning algorithm
    while j < 99:
        j+=1
        #Choose an action by greedily (with noise) picking from Q table
        a = np.argmax(Q[s,:] + np.random.randn(1,env.action_space.n)*(1./(i+1)))
        #Get new state and reward from environment
        s1,r,d,_ = env.step(a)
        #Update Q-Table with new knowledge
        Q[s,a] = Q[s,a] + lr*(r + y*np.max(Q[s1,:]) - Q[s,a])
        rAll += r
        s = s1
        if d == True:
            break
    #jList.append(j)
    rList.append(rAll)
print "Score over time: " + str(sum(rList)/num_episodes)
print "Final Q-Table Values"
print Q
```

神经网络和Q-learning

现在你可能会想，q-table方法效果不错，但是很难扩展。虽然很容易为一个简单的游戏建立16*4的表，但是在任何现代游戏或者现实世界环境中可能的状态数量几乎都是无限大的。对于大多数有趣的问题，q-table方法太简单了。所以我们需要另一种方法能够描



述状态，并且生成Q值。这就是为什么我们需要神经网络，我们可以将任意数量的可能状态表示为向量，并学习将它们映射为Q值。

在FrozenLake例子中，我们将采用单层网络，该网络会将状态编码为独热码(one-hot vector)，并且生成一个四维的Q值向量。我们可以使用Tensorflow来选择网络的层数，激活函数和输入类型，这都上文中Q表格方法办不到的。使用神经网络时，更新的方法和Q表是不同的，我们将使用反向传播算法更新损失函数。

$$\text{Eq 2. Loss} = \sum (Q_{\text{target}} - Q)^2$$

如果读者不熟悉深度学习/神经网络，可以从这里 (<http://deeplearning.net/>)开始学习。

这里给出基于tensorflow实现的简单Q网络：



```

import gym
import numpy as np
import random
import tensorflow as tf
import matplotlib.pyplot as plt
#Load the environment
env = gym.make('FrozenLake-v0')
#The Q-Network Approach
#Implementing the network itself

tf.reset_default_graph()
These lines establish the feed-forward part of the network used to choose actions
inputs1 = tf.placeholder(shape=[1,16],dtype=tf.float32)
W = tf.Variable(tf.random_uniform([16,4],0,0.01))
Qout = tf.matmul(inputs1,W)
predict = tf.argmax(Qout,1)

#Below we obtain the loss by taking the sum of squares difference between the target
nextQ = tf.placeholder(shape=[1,4],dtype=tf.float32)
loss = tf.reduce_sum(tf.square(nextQ - Qout))
trainer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
updateModel = trainer.minimize(loss)

#Training the network
init = tf.initialize_all_variables()

# Set learning parameters
y = .99
e = 0.1
num_episodes = 2000
#create lists to contain total rewards and steps per episode
jList = []
rList = []
with tf.Session() as sess:
    sess.run(init)
    for i in range(num_episodes):
        #Reset environment and get first new observation
        s = env.reset()
        rAll = 0
        d = False
        j = 0
        #The Q-Network
        while j < 99:
            j+=1
            #Choose an action by greedily (with e chance of random action) from the
            a,allQ = sess.run([predict,Qout],feed_dict={inputs1:np.identity(16)[s:s+1]})
            if np.random.rand(1) < e:
                a[0] = env.action_space.sample()
            #Get new state and reward from environment

```



```


s1,r,d,_ = env.step(a[0])
#Obtain the Q' values by feeding the new state through our network
Q1 = sess.run(Qout, feed_dict={inputs1:np.identity(16)[s1:s1+1]})
#Obtain maxQ' and set our target value for chosen action.
maxQ1 = np.max(Q1)
targetQ = allQ
targetQ[0,a[0]] = r + y*maxQ1
#Train our network using target and predicted Q values
_,W1 = sess.run([updateModel,W], feed_dict={inputs1:np.identity(16)[s:s+1]})
rAll += r
s = s1
if d == True:
    #Reduce chance of random action as we train the model.
    e = 1./((i/50) + 10)
    break
jList.append(j)
rList.append(rAll)
print "Percent of succesful episodes: " + str(sum(rList)/num_episodes) + "%"

```

当神经网络学习解决FrozenLake问题时，结果证明它并不像Q-table方法一样有效。==虽然神经网络允许更大的灵活性，但是它们以Q-learning的稳定性为代价==。我们简单的Q网络还可以有很多的期缴来达到更好的效果，==一个称为经验回放(Experience Replay)另一个称为目标网络冻结(Freezing Target Networks)==。这些改进方法和其他的调整方法是深度增强学习能够起作用的关键，我们回来后续的章节中更详细的探讨有关理论。

如果你觉得这篇文章对你有帮助，可以关注原作者，或者打赏作者。

如果你想要继续看到我翻译的文章，也可以专注专栏。第一次翻译，希望能和大家一起交流。

 machine learning (/nb/147541)

[举报文章](#) © 著作权归作者所有



y_felix (/u/c706b1b8d11d)

写了 9709 字，被 20 人关注，获得了 8 个喜欢
(/u/c706b1b8d11d)

+ 关注



♡ 喜欢 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-like-button) | 4



更多分享

(http://cwb.assets.jianshu.io/notes/images/10060553/weibo/image_

