基于tensorflow的最简单的强化学习入门-part1.5: 基于上下文老虎机问题(Contextual Bandits)

> 本文翻译自 Simple Reinforcement Learning with Tensorflow Part 1.5: Contextual Bandits， 作者是 Arthur Juliani， 原文链接。

介绍

在这个系列的前一部分文章中，我们介绍了增强学习的一些概念，并且演示了如何通过建立一个agent来解决多臂老虎机问题(Multi-arm bandits)。多臂老虎机可以当作一种特殊的增强学习问题，没有状态(state)，只需要采取行动(action)并获取最大的奖励(reward)即可。由于没有给定的状态，那么任意时刻的最佳动作始终都是最佳的动作。而在第二部分的文章展示了完整的强化学习问题，其中包括**环境状态**和**延迟奖励**。

事实上，在无状态问题和完整的强化学习问题上还存在着一些不同，我想提供一个这样的例子来展示如何解决它。我希望对强化学习不太了解的朋友们可以通过在逐步的学习中有所收获。这这篇文章中，作为第一篇文章和第二篇文章的过渡，我将展示如何解决有状态的问题，但是我们不会考虑延迟奖励，所有这些都将出现在第二部分的文章中。这种简化的强化学习问题称为上下文老虎机问题。

> 多臂老虎机问题(只有行动和回报)，上下文老虎机问题(有状态，行动和回报)：完全RL问题(奖励有可能在时间上延迟

上下文老虎机问题

在第一部分讨论多臂老虎机问题中，我们可以认为只有一个老虎机。agent可能的动作就是拉

状态，环境状态不会影响我们采取的动作和回报，所以对于所有的动作来说只有一种给定的**状态**。

上下文老虎机问题中带来了**状态**的概念。状态包含agent能够利用的一系列环境的描述和信息。在这个例子中，有多个老虎机而不是一个老虎机，状态可以看做我们正在操作哪个老虎机。我们的目标不仅仅是学习单一老虎机的操作方法，而是很多老虎机。在每一个老虎机中，转动每一个机臂带来的回报都会不一样，我们的agent需要学习到在不同状态下(老虎机)执行动作所带来的回报。为了实现这个功能，我们会基于tensorflow构造一个简单的神经网络，输入状态并且得到动作的权重。通过策略梯度更新方法，我们的agent就可以学习到不同状态下如何获得最大的回报。下面是实现上述过程的python的代码：

定义上下文老虎机

这里我们定义上下文老虎机，在这个例子中，我们使用三个多臂老虎机，不同的老虎机有不同的概率分布，因此需要执行不同的动作获取最佳结果。getbandit函数随机生成一个数字，数字越低就越可能产生正的回报。我们希望agent可以一直选择能够产生最大收益的老虎机臂

```
import tensorflow as tf
import tensorflow.contrib.slim as slim
import numpy as np


class contextual_bandit():
    def __init__(self):
        self.state = 0
        #List out our bandits. Currently arms 4, 2, and 1 (respectively) are
```

```
        self.num_actions = self.bandits.shape[1]

    def getBandit(self):
        self.state = np.random.randint(0,len(self.bandits)) #Returns a random
        return self.state

    def pullArm(self,action):
        #Get a random number.
        bandit = self.bandits[self.state,action]
        result = np.random.randn(1)
        if result > bandit:
            #return a positive reward.
            return 1
        else:
            #return a negative reward.
            return -1
```

策略梯度的agent

这段代码建立了一个简单的基于神经网络的agent，其中输入为当前的状态，输出为执行的动作。这使得agent可以根据当前的状态执行不同的动作。agent使用一组权重，每一个作为在给定状态下执行特定动作的回报的估计。

```
class agent():
    def __init__(self, lr, s_size,a_size):
        #These lines established the feed-forward part of the network. The age
        self.state_in= tf.placeholder(shape=[1],dtype=tf.int32)
```

```
            biases_initializer=None,activation_fn=tf.nn.sigmoid,weights_initia
        self.output = tf.reshape(output,[-1])
        self.chosen_action = tf.argmax(self.output,0)

        #The next six lines establish the training proceedure. We feed the rew
        #to compute the loss, and use it to update the network.
        self.reward_holder = tf.placeholder(shape=[1],dtype=tf.float32)
        self.action_holder = tf.placeholder(shape=[1],dtype=tf.int32)
        self.responsible_weight = tf.slice(self.output,self.action_holder,[1])
        self.loss = -(tf.log(self.responsible_weight)*self.reward_holder)
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=lr)
        self.update = optimizer.minimize(self.loss)

tf.reset_default_graph() #Clear the Tensorflow graph.
```

训练

```
cBandit = contextual_bandit() #Load the bandits.
myAgent = agent(lr=0.001,s_size=cBandit.num_bandits,a_size=cBandit.num_actions
weights = tf.trainable_variables()[0] #The weights we will evaluate to look in

total_episodes = 10000 #Set total number of episodes to train agent on.
total_reward = np.zeros([cBandit.num_bandits,cBandit.num_actions]) #Set scorel
e = 0.1 #Set the chance of taking a random action.

init = tf.initialize_all_variables()
```

```
    sess.run(init)
    i = 0
    while i < total_episodes:
        s = cBandit.getBandit() #Get a state from the environment.

        #Choose either a random action or one from our network.
        if np.random.rand(1) < e:
            action = np.random.randint(cBandit.num_actions)
        else:
            action = sess.run(myAgent.chosen_action,feed_dict={myAgent.state_

        reward = cBandit.pullArm(action) #Get our reward for taking an action

        #Update the network.
        feed_dict={myAgent.reward_holder:[reward],myAgent.action_holder:[actio
        _,ww = sess.run([myAgent.update,weights], feed_dict=feed_dict)

        #Update our running tally of scores.
        total_reward[s,action] += reward
        if i % 500 == 0:
            print "Mean reward for each of the " + str(cBandit.num_bandits) +
        i+=1
for a in range(cBandit.num_bandits):
    print "The agent thinks action " + str(np.argmax(ww[a])+1) + " for bandit
    if np.argmax(ww[a]) == np.argmin(cBandit.bandits[a]):
        print "...and it was right!"
    else:
        print "...and it was wrong!"
```

希望本教程能够有助于你直观的理解强化学习如何解决不同的问题。如果你已经掌握了这个方法，并且已经准备好探索完整的深度强化问题，你可以直接看第二部分或者以后的文章。

如果你觉得这篇文章对你有帮助，可以关注原作者。

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

深度学习（Deep Learning） 人工智能

强化学习 (Reinforcement Learning)

写下你的评论...

推荐阅读

基于tensorflow的最简单的强化学习入门-part2: Policy-based Agents

本文翻译自 imple Reinforcement Learning with Tensorflow: Part 2 - Policy-based Agents… 查看全文 >

felix · 6 个月前 · 发表于 有意思的数据挖掘

基于tensorflow的最简单的强化学习入门-part1：多臂老虎机问题

本文翻译自 Simple Reinforcement Learning in Tensorflow: Part 1 - Two-armed Bandit， 作… 查看全文 >

felix · 7 个月前 · 发表于 有意思的数据挖掘

除了杀死丈夫，广东女性在遭遇家庭暴力时还有这2种

广东一母子长期遭受家庭暴力，在一次父亲对母亲家暴时，儿子爆发并在母亲的帮助下将父亲杀害... 查看全文 >

吴杰臻律师 · 3 天前 · 编辑精选 · 发表于 离婚大师说

在免签天堂摩洛哥，我差点成了羊羔的爹

有些城市，光听名字就让人觉得很美好；比如维也纳，比如赫尔辛基，比如卡萨布兰卡。在西班牙... 查看全文 >

陈不为 · 5 个月前 · 编辑精选