基于tensorflow的最简单的强化学习入门-part1：多臂老虎机问题

y_felix (/u/c706b1b8d11d) 作者 + 关注
2017.03.21 11:01* 字数 1895 阅读 610 评论 0 喜欢 0
(/u/c706b1b8d11d)

题图

> 本文翻译自 Simple Reinforcement Learning in Tensorflow: Part 1 - Two-armed Bandit， 作者是 Arthur Juliani， 原文链接 (https://medium.com/@awjuliani/super-simple-reinforcement-learning-tutorial-part-1-fd544fab149#.2mruvzm39)。

介绍

强化学习不仅提供了指导人工智能agent如何行动的能力，还允许它通过和环境的相互作用自主学习。同时结合神经网络强大的表达能力和目标驱动学习方式，深度强化学习成为了强大的人工智能基本方法。深度强化学习已经完成了一些惊人的壮举，例如在Atari游戏中战胜了人类 (https://deepmind.com/dqn)，在在围棋项目上击败了世界冠军 (https://deepmind.com/alpha-go)，最近又在德州扑克 (http://sports.sina.com.cn/go/2017-02-01/doc-ifxzyxmt1717342.shtml)中大放异彩。

构建这些人工智能程序和构建可监督学习程序有所不同。可监督学习只是简单的学习模型对某个输入的反馈，而强化学习算法使agent能够通过**观察(observation)、奖励(reward)和动作(action)**来学习对于输入的**正确**的反馈。在特定情况下，强化学习和有监督学习是不同的，agent并不知道何谓**正确**的行动，这使事情变得很棘手。在这篇文章和后续的文章中，我将介绍如何构造和训练强化学习agent。为了使大家概念清晰，处理的任务和agent的设计都会从简单到复杂。

双臂老虎机问题(Two-Armed bandit)

最简单的强化学习问题就是多臂老虎机问题了。多臂老虎机问题本质上可以看做一个拥有n个槽的老虎机，转动每个槽都有固定回报概率。我们的目标就是找到回报概率最高的的槽并且不断的选择它来获取最高的回报。为了简化这个问题，假设这个机器只有两个槽，我们要做的就是从这两个槽中找到回报更高的那一个。事实上，这个问题非常简单，但是可以看作真正RL问题的一个原型。一般的RL问题需要符合如下条件

- 不同的动作导致不同的回报。举个例子，在迷宫中寻找宝藏，如果往左就能获得宝藏，往右就什么都得不到。
- 回报在时间上有延迟。沿用上述的例子，在迷宫中往左时，我们并不是立即知道我们走的就是正确的方向。
- 某个动作下的回报跟当时的环境有关。继续刚才的例子，往左边是当前情况下的最佳选择，在其他情况下就不一定了。

多臂老虎机是学习强化学习良好的开端，我们不需要去担心#2和#3的问题。我们只需要关注哪个动作可以带来怎样的回报，并且确保我们能够选择理想的动作。用RL的术语来说，这就叫做**Policy**。我们将要用一种叫做**策略梯度(policy gradients)**的方法，该方法中我们的简单的神经网络通过和环境的的不断交互同时结合**BP算法**就可以学习到如何执行该动作的策略(policy)。在强化学习中，还有另一种方法叫做价值函数(value functions)，在这个方法中，agent并不是学习某种状态下特定的动作，而是学习如何预测当前的状态和动作的好坏(价值)。两种方法都可以使得agent可以学习良好的策略，不过策略梯度方法更直接一些。

策略梯度算法

简单来说，策略梯度网络可以直接产生输出。在我们这个简单的例子中，我们不需要根据当前的状态来调节输出。因此，我们的网络只包括一组的权重，每个对应着老虎拉动机臂可能的动作，输出代表对应的动作的好坏。*如果我们将这些权重初始化为1，那么agent可能会对每个分支的潜在回报过于乐观。*

为了更新网络参数，我们将采用一种称为e-greedy的策略（后续章节会详细介绍这个方法）。应用这个策略意味着在大多数情况下，我们的agent会选择带来最大预期好处的动作，但是偶尔的以e的概率，它将随机选择。这样agent就可以尝试每一个可能的状态，一旦我们的agent采取行动，它就会收到1或者-1的奖励。有了这个奖励，我们可以使用损失函数更新我们的网络参数：

> 译者注：在Andrej Karpathy文中也有对这个方法的介绍。

```
Loss = log(\Pi) * A
```

- A称为优势(advantage)， 是所有强化学习方法中一个重要的概念。直观的看，它对应于某个动作跟某些baseline相比的好坏。在未来的算法中，我们会开发更复杂的baseline和我们的回报对比，当在当前的问题中，我们假设baseline为0，而A可以简单的认为是每个行动的回报。

- `\pi` 称为策略，在这个例子中它代表选择这个动作的权重。

显而易见，这样的损失函数允许我们增加产生积极奖励动作时的权重，并且减少产生负奖励时的权重。通过这样方式，agent或多或少的能够在未来学习如何采取动作，获得奖励，并且更新我们的网络。我们会很快收敛到(学习到)一个agent，该agent可以解决我们的多臂老虎机问题。如果不相信我说的，那你可以自己试试看。

基于tensorflow强化学习代码:

Bandits

在这个例子中我们采用一个四个臂的老虎机。pullBandit函数随机从正态分布函数采样一个值，如果该值越小，那么就会更有可能产生一个正的回报。我想要我们的agent能够学习到产生回报最高的那个老虎机臂。bandits是一个数组，bandit 4(index#3)被设置为产生最高回报。

```
import tensorflow as tf
import numpy as np

bandits = [0.2,0,-0.2,-5]
num_bandits = len(bandits)
def pullBandit(bandit):
    #Get a random number.
    result = np.random.randn(1)
    if result > bandit:
        #return a positive reward.
        return 1
    else:
        #return a negative reward.
        return -1
```

Agent

下述代码实现了一个简单的基于神经网络的agent。该agent主要包括每一个bandit对应的权重，每一个权重就是选择该bandit预期的回报。通过不断的选择老虎臂并且获得回报，我们将采用策略梯度方法来更新权重，

```
tf.reset_default_graph()

#These two lines established the feed-forward part of the network. This does the act
weights = tf.Variable(tf.ones([num_bandits]))
chosen_action = tf.argmax(weights,0)

#The next six lines establish the training proceedure. We feed the reward and chosen
#to compute the loss, and use it to update the network.
reward_holder = tf.placeholder(shape=[1],dtype=tf.float32)
action_holder = tf.placeholder(shape=[1],dtype=tf.int32)
responsible_weight = tf.slice(weights,action_holder,[1])
loss = -(tf.log(responsible_weight)*reward_holder)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
update = optimizer.minimize(loss)
```

Training the Agent

通过不断的选择动作并且获得回报，利用reward和action，我们能够合适的更新神经网络中的权重。最终该网络会倾向于选择带来回报更多的bandit。

```
total_episodes = 1000 #Set total number of episodes to train agent on.
total_reward = np.zeros(num_bandits) #Set scoreboard for bandits to 0.

e = 0.1 #Set the chance of taking a random action.

init = tf.initialize_all_variables()

# Launch the tensorflow graph
with tf.Session() as sess:
    sess.run(init)
    i = 0
    while i < total_episodes:

        #Choose either a random action or one from our network.
        if np.random.rand(1) < e:
            action = np.random.randint(num_bandits)
        else:
            action = sess.run(chosen_action)

        reward = pullBandit(bandits[action]) #Get our reward from picking one of the

        #Update the network.
        _,resp,ww = sess.run([update,responsible_weight,weights], feed_dict={reward_

        #Update our running tally of scores.
        total_reward[action] += reward
        if i % 50 == 0:
            print "Running reward for the " + str(num_bandits) + " bandits: " + str(
        i+=1
print "The agent thinks bandit " + str(np.argmax(ww)+1) + " is the most promising...
if np.argmax(ww) == np.argmax(-np.array(bandits)):
    print "...and it was right!"
else:
    print "...and it was wrong!"
```

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。

machine learning (/nb/147541)

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

如果觉得我的文章对您有用，请随意赞赏。您的支持将鼓励我继续创作！

赞赏支持

♡ 喜欢 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-like-button) 0

更多分享

(http://cwb.assets.jianshu.io/notes/images/10419522/weibo/image_

登录 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-comment-form)后发表评论

评论

智慧如你，不想发表一点想法 (/sign_in?utm_source=desktop&utm_medium=not-signed-in-nocomments-text)咩~