浅析强化学习及使用Policy Network实现自动化控制

浅析强化学习

强化学习（Reinforcement Learning）是机器学习的一个重要分支，主要用来解决连续决策的问题。强化学习可以在复杂、不确定的环境中学习如何实现我们设定的目标。强化学习的应用场景非常广，几乎包括了所有需要做一系列决策的问题，比如控制机器人的电机让它执行特定任务，给商品定价或者库存管理，玩视频或棋牌游戏等。强化学习也可以应用到有序列输出的问题中，因为它可以针对一系列变化的环境状态，输出一系列对应的行动。举个简单的例子，围棋（乃至全部棋牌类游戏）可以归结为一个强化学习问题，我们需要学习在各种局势下如何走出最好的招法。

一个强化学习问题包含三个主要概念，即环境状态（Environment State）、行动（Action）和奖励（Reward），而强化学习的目标是获得最多的累计奖励。在围棋中，环境状态就是已经下出来的某个局势，行动是在某个位置落子，奖励则是当前这步棋获得的目数（围棋中存在不确定性，在结束对弈后计算的目数是准确的，棋局中获得的目数是估计的），而最终目标就是在结束对弈时总目数超过对手，赢得胜利。我们要让强化学习模型根据环境状态、行动和奖励，学习
为目标，不能只看某个行动当下带来的利益（比如围棋中通过某一手棋获得的实地），还要看到
（比如围棋中外势可以带来的潜在价值）。我们回顾一下，AutoEncoder属于无监督学习，而MLP
习，但强化学习跟这两种都不同。它不像无监督学习那样完全没有学习目标，也不像监督学习
label），强化学习的目标一般是变化的、不明确的，甚至可能不存在绝对正确的标签。

强化学习已经有几十年的历史，但是直到最近几年深度学习技术的突破，强化学习才有了比较大的进展。Google DeepMind结合强化学习与深度学习，提出DQN（Deep Q-Network，深度Q网络），它可以自动玩Atari 2600系列的游戏，并取得了超过人类的水平。而DeepMind的AlphaGo结合了策略网络（Policy Network）、估值网络（Value Network，也即DQN）与蒙特卡洛搜索树（Monte Carlo Tree Search），实现了具有超高水平的围棋对战程序，并战胜了世界冠军李世石。

DeepMind使用的这些深度强化学习模型（Deep Reinforcement Learning，DRL）本质上也是神经网络，主要分为策略网络和估值网络两种。深度强化学习模型对环境没有特别强的限制，可以很好地推广到其他环境，因此对强化学习的研究和发展具有非常重大的意义。下面我们来看看深度强化学习的一些实际应用例子。

我们也可以使用深度强化学习自动玩游戏，如图1所示，用DQN可学习自动玩Flappy Bird。DQN前几层通常也是卷积层，因此具有了对游戏图像像素（raw pixels）直接进行学习的能力。前几层卷积可理解和识别游戏图像中的物体，后层的神经网络则对Action的期望价值进行学习，结合这两个部分，可以得到能根据游戏像素自动玩Flappy Bird的强化学习策略。而且，不仅是这类简单的游戏，连非常复杂的包含大量战术策略的《星际争霸2》也可以被深度强化学习模型掌握。目前，DeepMind就在探索如何通过深度强化学习训练一个可以战胜《星际争霸2》世界冠军的人工智能，这之后的进展让我们拭目以待。

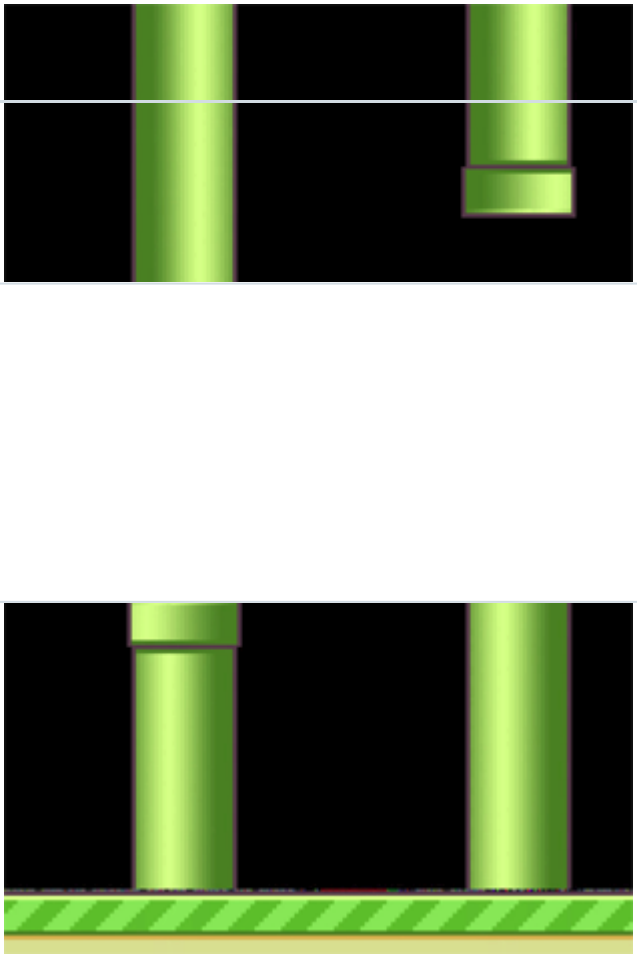

图1　使用深度强化学习自动玩Flappy Bird

深度强化学习最具有代表性的一个里程碑自然是AlphaGo。在2016年，Google DeepMind的Alph
的世界冠军李世石，如图2所示。围棋可以说是棋类游戏中最为复杂的，19×19的棋盘给它带来了3
违反游戏规则的状态，计算机也是无法通过像深蓝那样的暴力搜索来战胜人类的，要在围棋这个项
算机抽象思维的能力，而AlphaGo做到了这一点。

请输入标题
请输入链接地址

请输入推荐理由

请输入标签

在AlphaGo中使用了快速走子（Fast Rollout）、策略网络、估值网络和蒙特卡洛搜索树等技术。图3所示为AlphaGo的几种技术单独使用时的表现，横坐标为步数，纵坐标为预测的误差（可以理解为误差越低模型效果越好），其中简单的快速走子策略虽然效果比较一般，但是已经远胜随机策略。估值网络和策略网络的效果都非常好，相对来说，策略网络的性能更胜一筹。AlphaGo融合了所有这些策略，取得了比单一策略更好的性能，在实战中表现出了惊人的水平。

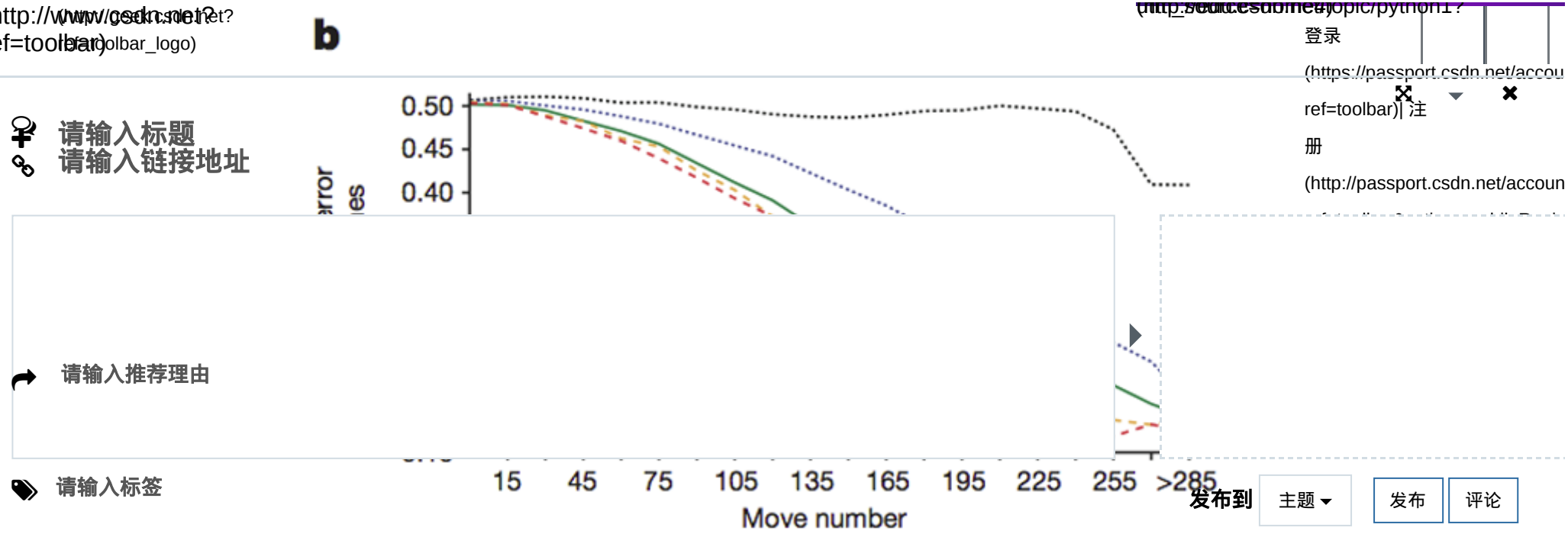

图3　AlphaGo中随机策略、快速走子、估值网络和策略网络（SL和RL两种）的性能表现

Policy-Based（或者Policy Gradients）和Value-Based（或者Q-Learning）是强化学习中最重要的两类方法，其主要区别在于Policy-Based的方法直接预测在某个环境状态下应该采取的Action，而Value Based的方法则预测某个环境状态下所有Action的期望价值（Q值），之后可以通过选择Q值最高的Action执行策略。这两种方法的出发点和训练方式都有不同，一般来说，Value Based方法适合仅有少量离散取值的Action的环境，而Policy-Based方法则更通用，适合Act值的Action的环境。而结合深度学习后，Policy-Based的方法就成了Policy Network，而Value Network。

图4所示为AlphaGo中的策略网络预测出的当前局势下应该采取的Action，图中标注的数值为策Action的概率，即我们应该在某个位置落子的概率。

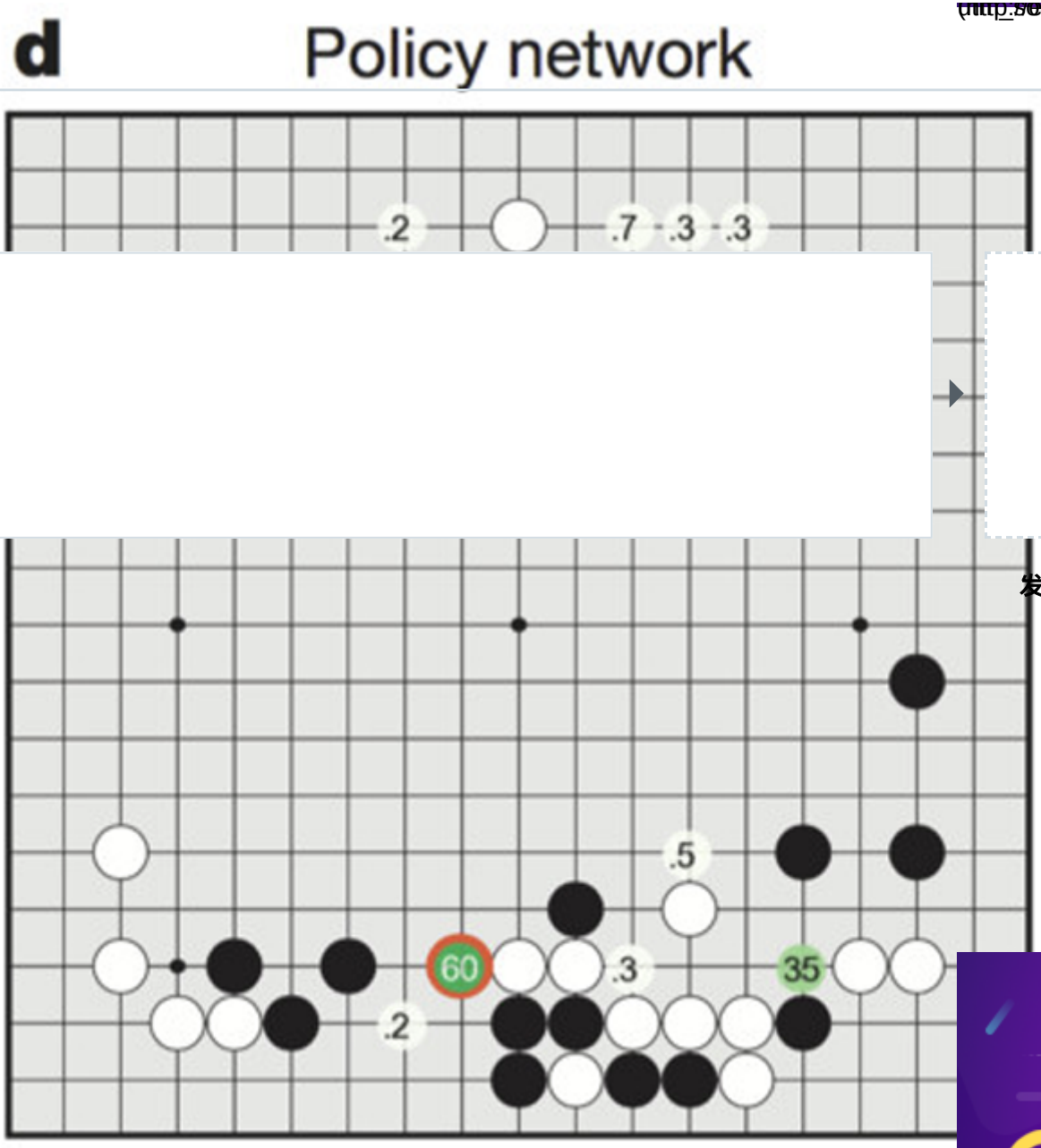

图4　AlphaGo中的策略网络，输出在某个位置落子的概率

图5所示为AlphaGo中估值网络预测出的当前局势下每个Action的期望价值。估值网络不直接输出策略，而是输出Action对应的价值，即在某个位置落子可以获得的期望价值。随后，我们可以直接选择期望价值最大的位置落子，或者选择其他位置进行探索。

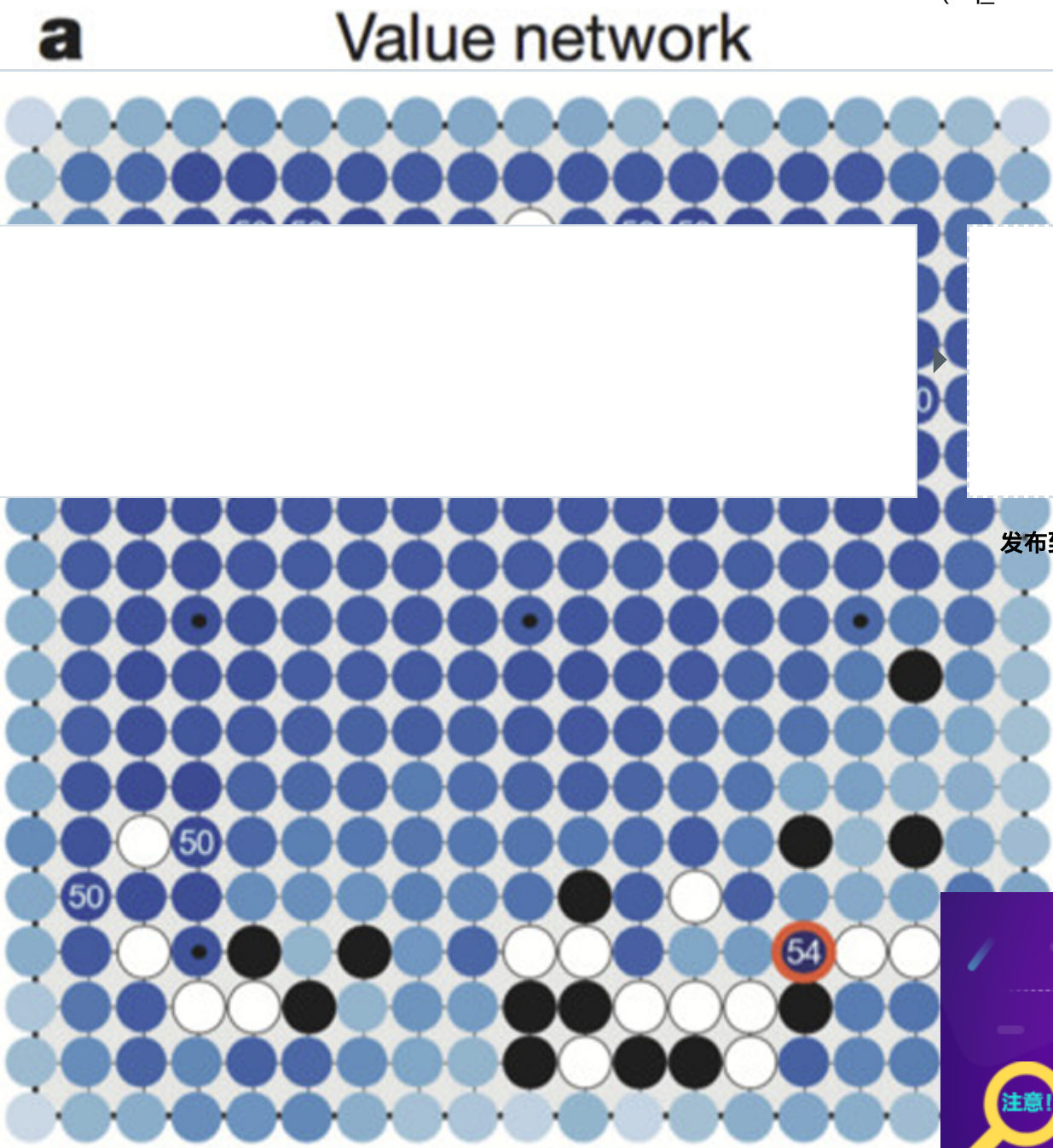

图5　AlphaGo中的估值网络，输出在某个位置落子的期望价值

在强化学习中，我们也可以建立额外的model对环境状态的变化进行预测。普通的强化学习直接根据环境状态预测出行动策略，或行动的期望价值。如果根据环境状态和采取的行动预测接下来的环境状态，并利用这个信息训练强化学习模型，那就是

Action的期望价值进行预测，因此计算效率非常高。当然，如果有一个很好的model可以高效、准确地对环境进行预测，会对训练RL带来益处；但是一个不那么精准的model反而会严重干扰RL的训练。因此，对大多数复杂环境，我们主要使用model-free RL，同时供给更多的样本给RL训练，用来弥补没有model预测环境状态的问题。

使用策略网络（Policy Network）实现自动化控制

前面提到了强化学习中非常重要的3个要素是Environment State、Action和Reward。在环境中，强化学习模型的载体是Agent，它负责执行模型给出的行动。环境是Agent无法控制的，但是可以进行观察；根据观察的结果，模型给出行动，交由Agent来执行；而Reward是在某个环境状态下执行了某个Action而获得的，是模型要争取的目标。在很多任务中，Reward是延迟获取的（Delayed），即某个Action除了可以即时获得Reward，也可能跟未来获得的Reward有很大关系。

所谓策略网络，即建立一个神经网络模型，它可以通过观察环境状态，直接预测出目前最应该执行的Action，使得按这个策略可以获得最大的期望收益（包括现在的和未来的Reward）。与普通的监督学习不同，在强化学习的学习目标，样本的feature不再和label一一对应。对某一个特定的环境状态，我们并不知道它对应的Action是什么，也不知道当前Action获得的Reward还有试验后获得的未来的Reward。我们需要让强化学习模型通过试验，学习在某个环境状态下比较好的Action，而不是告诉模型什么才是比较好的Action，因为我们也不知道正确的Action（没有现成的label，只有估算出的label）。我们的学习目标是期望价值，即当前获得的Reward，加上未来潜

更好地让策略网络理解未来的、潜在的Reward，策略网络不只是使用当前的Reward作为label，而是使用Discounted Future Reward，即把所有未来奖励依次乘以衰减系数γ。这里的衰减系数一般是一个略小于但接近1的数，防止没有损耗地积累导致Reward目标发散，同时也代表了对未来奖励的不确定性的估计。

$$r = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{n-1} r_n$$

Acti
望价
低选
对应
ion应

然），或者输出某个Action的具体数值（如果Action不是离散值，而是连续值）。因此策略网络是一种End-to-End（端对端）的方法，可以直接产生最终的策略。

Policy Based的方法相比于Value-Based，有更好的收敛性（通常可以保证收敛到局部最优，且不会发散），对高维或者连续值的Action非常高效（训练和输出结果都更高效），同时能学习出带有随机性的策略。例如，在石头剪刀布的游戏中，任何有规律的策略都会被别人学习到并且被针对，因此完全随机的策略反而可以立于不败之地（起码不会输给别的策略）。在这种情况下，可以利用策略网络学到随机出剪刀、石头、布的策略（三个Action的概率相等）。

我们需要使用Gym辅助我们进行策略网络的训练。Gym是OpenAI推出的开源的强化学习的环境生成工具。在Gym中，有两个核心的概念，一个是Environment，指我们的任务或者问题，另一个就是Agent，即我们编写的策略或算法。Agent会将执行的Action传给Environment，Environment接受某个Action后，再将结果Observation（即环境状态）和
中提供了完整的Environment的接口，而Agent则是完全由用户编写。

下面我们就以Gym中的CartPole环境作为具体例子。CartPole任务最早由论文《Neuronlike Ada
Solve Difficult Learning Control Problem》提出，是一个经典的可用强化学习来解决的控制问题
境中有一辆小车，在一个一维的无阻力轨道上行动，在车上绑着一个连接不太结实的杆，这个杆会

observation并不是图像像素，而只是一个有4个值的数组，包含了环境中的各种信息，比如小车位置、速度、杆的角度、速度等。我们并不需要知道每个数值对应的具体物理含义，因为我们不是要根据这些数值自己编写逻辑控制小车，而是设计一个策略网络让它自己从这些数值中学习到环境信息，并制定最佳策略。我们可以采取的Action非常简单，给小车施加一个正向的力或者负向的力。我们有一个Action Space的概念，即Action的离散数值空间，比如在CartPole里Action Space就是Discrete(2)，即只有0或1，其他复杂一点的游戏可能有更多可以选择的值。我们并不需要知道这里的数值会具体对应哪个

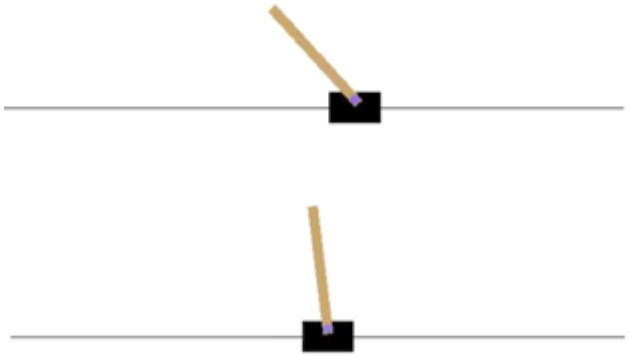

图6　CartPole环境中包含一个可以控制移动方向的小车和不稳的杆

当我们使用env.reset()方法后，就可以初始化环境，并获取到环境的第一个Observation。此后，根
取的Action，并使用env.step(action)在环境中执行Action，这时会返回Observation（在CartPole中
任务中可能是图像像素）、reward（当前这步Action获得的即时奖励）、done（任务是否结束的
或者小车偏离中心太远，其他游戏中可能是被敌人击中。如果为True，应该reset任务）和info（

游戏中一些随机事件的概率，但是不应该用来训练Agent）。这样我们就进入Action-Observation的循环，执行Action，获得Observation，再执行Action，如此往复直到任务结束，并期望在结束时获得尽可能高的奖励。我们可执行的Action在CartPole中是离散的数值空间，即有限的几种可能，在别的任务中可能是连续的数值，例如在赛车游戏任务中，我们执行的动作是朝某个方向移动，这样我们就有了0~360度的连续数值空间可以选择。

下面就使用TensorFlow创建一个基于策略网络的Agent来解决CartPole问题。我们先安装OpenAI Gym中获得本要实现

```
import numpy as np
import tensorflow as tf
import gym
env = gym.make('CartPole-v0')
```

先测试在CartPole环境中使用随机Action的表现，作为接下来对比的baseline。首先，我们使用env.reset()初始化环境，然后进行10次随机试验，这里调用env.render()将CartPole问题的图像渲染出来。使用np.random.randint(0,2)产生随机的Action，然后用env.step()执行随机的Action，并获取返回的observation、reward和done。如果done标记为True，则代表这次试验结束，即倾角超过15度或者偏离中心过远导致任务失败。在一次试验结束后，我们展示这次试验累计的奖励reward_sum并重启环境。

```
    env.reset()
random_episodes = 0
reward_sum = 0
while random_episodes < 10:
    env.render()
    observation, reward, done, _ = env.step(np.random.randint(0,2))
    reward_sum += reward
```

```
Reward for this episode was: 12.0
Reward for this episode was: 17.0
Reward for this episode was: 20.0
Reward for this episode was: 44.0
Reward for this episode was: 28.0
Reward for this episode was: 19.0
Reward for this episode was: 13.0
Reward for this episode was: 30.0
Reward for this episode was: 20.0
Reward for this episode was: 26.0
```

我们的策略网络使用简单的带有一个隐含层的MLP。先设置网络的各个超参数，这里隐含节点数
25，学习速率learning_rate为0.1，环境信息observation的维度D为4，gamma即Reward的disc
Action的期望价值（即估算样本的学习目标）时会考虑Delayed Reward，会将某个Action之后获得
累加起来，这样可以让模型学习到未来可能出现的潜在Reward。注意，一般discount比例要小于1
断累加导致发散，这样也可以区分当前Reward和未来Reward的价值（当前Action直接带来的Rew
的Reward因存在不确定性所以需要discount）。

```
H = 50
batch_size = 25
learning_rate = 1e-1
D = 4
gamma = 0.99
```

请输入标题
请输入链接地址

下面定义策略网络的具体结构。这个网络将接受observations作为输入信息，最后输出一个概率值用以选择Action（我们只有两

serva
D, H
们并
Sigm

请输入推荐理由

请输入标签

```
observations = tf.placeholder(tf.float32, [None,D] , name="input _x")
W1 = tf.get_variable("W1", shape=[D, H],
                     initializer=tf.contrib.layers.xavier_initializer())
layer1 = tf.nn.relu(tf.matmul(observations,W1))
W2 = tf.get_variable("W2", shape=[H, 1],
                     initializer=tf.contrib.layers.xavier_initializer())
score = tf.matmul(layer1,W2)
probability = tf.nn.sigmoid(score)
```


这里模型的优化器使用Adam算法。我们分别设置两层神经网络参数的梯度的placeholder——W1Grad和W2Grad，并使用adam.apply_gradients定义我们更新模型参数的操作updateGrads。之后计算参数的梯度，当积累到一定样本量的梯度，就传入W1Grad和W2Grad，并执行updateGrads更新模型参数。这里注意，深度强化学习的训练和其他神
training的方式。我们不逐个样本地更新参数，而是累计一个batch_size的样本的梯度再更新参数，
声对模型带来不良影响。

```
adam = tf.train.AdamOptimizer(learning_rate=learning_rate)
W1Grad = tf.placeholder(tf.float32,name="batch_grad1")
W2Grad = tf.placeholder(tf.float32,name="batch_grad2")
batchGrad = [W1Grad,W2Grad]
updateGrads = adam.apply_gradients(zip(batchGrad,tvars))
```

下面定义函数discount_rewards，用来估算每一个Action对应的潜在价值discount_r。因为CartPole问题中每次获得的Reward都

接获得的Reward外的潜在价值为running_add，running_add是从后向前累计的，并且需要经过discount衰减。而每一个Action的潜在价值，即为后一个Action的潜在价值乘以衰减系数gamma再加上它直接获得的reward，即running_add*gamma+r[t]。这样从最后一个Action开始不断向前累计计算，即可得到全部Action的潜在价值。这种对潜在价值的估算方法符合我们的期望，越靠前的Action潜在价值越大。

```
def discount_rewards(r):
    discounted_r = np.zeros_like(r)
    running_add = 0
    for t in reversed(range(r.size)):
        running_add = running_add * gamma + r[t]
        discounted_r[t] = running_add
    return discounted_r
```

我们定义人工设置的虚拟label（下文会讲解其生成原理，其取值为0或1）的placeholder——input_

值的placeholder——advangtages。这里loglik的定义略显复杂，我们来看一下loglik到底代表什

probability（即策略网络输出的概率），Action取值为0的概率为1-probability，label取值与Action

Action为1时，label为0，此时loglik=tf.log(probability)，Action取值为1的概率的对数；当Act

loglik = tf.log(1-probability)，即Action取值为0的概率的对数。所以，loglik其实就是当前Action对应的概率的对数，我们将loglik与潜在价值advantages相乘，并取负数作为损失，即优化目标。我们使用优化器优化时，会让能获得较多advantages的Action的概率变大，并让能获得较少advantages的Action的概率变小，这样能让损失变小。通过不断的训练，我们便能持续加大能获得较多advantages的Action的概率，即学习到一个能获得更多潜在价值的策略。最后，使用tf.trainable_variables()获取策略网络中全部可训练的参数tvars，并使用tf.gradients求解模型参数关于loss的梯度。

```
newGrads = tf.gradients(loss,tvars)
```

在正式进入训练过程前，我们先定义一些参数，xs为环境信息observation的列表，ys为我们定义的label的列表，drs为我们记录的每一个Action的Reward。我们定义累计的Reward为reward_sum，总试验次数total_episodes为10000，直到达到获取200的Reward才停止训练。

```
xs,ys,drs = [],[],[]
reward_sum = 0
episode_number = 1
total_episodes = 10000
```

我们创建默认的Session，初始化全部参数，并在一开始将render的标志关闭。因为render会带来比
太成熟的模型还没必要去观察。先初始化CartPole的环境并获得初始状态。然后使用sess.run执行
来创建储存参数梯度的缓冲器gradBuffer，并把gardBuffer全部初始化为零。接下来的每次试验中
到gradBuffer中，直到完成了一个batch_size的试验，再将汇总的梯度更新到模型参数。

```
        with tf.Session() as sess:
            rendering = False
            init = tf.global_variables_initializer()
            sess.run(init)
            observation = env.reset()

            gradBuffer = sess.run(tvars)
```

```
while episode_number <= total_episodes:

        if reward_sum/batch_size > 100 or rendering == True :
            env.render()
            rendering = True

        x = np.reshape(observation,[1,D])

        tfprob = sess.run(probability,feed_dict={observations: x})
        action = 1 if np.random.uniform() < tfprob else 0
```

然后将输入的环境信息observation添加到列表xs中。这里我们制造虚拟的label——y，它取值与Ac
将其添加到列表ys中。然后使用env.step执行一次Action，获取observation、reward、done
reward_sum，同时将reward添加到列表drs中。

```
        xs.append(x)
        y = 1 - action
        ys.append(y)

    observation, reward, done, info = env.step(action)
    reward_sum += reward
```



```
if done:
        episode_number += 1
        epx = np.vstack(xs)
        epy = np.vstack(ys)
        epr = np.vstack(drs)
        xs,ys,drs = [],[],[]

        discounted_epr = discount_rewards(epr)
        discounted_epr -= np.mean(discounted_epr)
        discounted_epr /= np.std(discounted_epr)
```

我们将epx、epy和discounted_epr输入神经网络，并使用操作newGrads求解梯度。再将获得的梯

```
tGrad = sess.run(newGrads,feed_dict={observations: epx,
                    input_y: epy, advantages: discounted_epr})
        for ix,grad in enumerate(tGrad):
            gradBuffer[ix] += grad
```

当进行试验的次数达到batch_size的整倍数时，gradBuffer中就累计了足够多的梯度，因此使用updateGrads操作将gradBuffer中的梯度更新到策略网络的模型参数中，并清空gradBuffer，为计算下一个batch的梯度做准备。这里注意，我们是使用一个batch的梯度更新参数，但是每一个梯度是使用一次试验中全部样本（一个Action对应一个样本）计算出来的，因此一个batch中的样本数实际上是25（batch_size）次试验的样本数之和。同时，我们展示当前的试验次数episode_number，和batch内每次试验平均获得的reward。当我们batch内每次试验的平均reward大于200时，我们的策略网络就成功完成了任务，并将终止循环试验

```
    for ix,grad in enumerate(gradBuffer):
        gradBuffer[ix] = grad * 0
    print('Average reward for episode %d : %f.' % \
     (episode_ number,reward_sum/batch_size))

if reward_sum/batch_size > 200:
    print("Task solved in",episode_number, 'episodes!')
    break

    reward_sum = 0

    observation = env.reset()
```

下面是模型的训练日志，可以看到策略网络在仅经历了200次试验，即8个batch的训练和参数更新，batch内平均230的reward，顺利完成预设的目标。有兴趣的读者可以尝试修改策略网络的结构、学习速率等参数来尝试优化策略网络的训练，加快其学习到好策略的速度。

```
Average reward for episode 25 : 19.200000.
Average reward for episode 50 : 30.680000.
Average reward for episode 75 : 41.360000.
Average reward for episode 100 : 52.160000.
Average reward for episode 125 : 70.680000.
Average reward for episode 150 : 84.520000.
Average reward for episode 175 : 153.320000.
```


SDCC 2017“人工智能技术实战线上峰会” (http://edu.csdn.net/huiyiCourse/series_detail/68)将在CSDN学院以直播互动的方式举行。

作为SDCC系列技术峰会的一部分，来自**阿里巴巴、微软、商汤科技、第四范式、微博、出门问问、菱歌科技的AI专家**，将针对机器学习平台、系统架构、对话机器人、芯片、推荐系统、Keras、分布式系统、NLP等热点话题进行分享。先行者们正在关注哪些关键技术？如何从理论跨越到企业创新实践？你将从本次峰会找到答案。**每个演讲时段均设有答疑交流环节，与会者和讲师可零距离互动。**

SDCC 2017“人工智能技术实战线上峰会”-日程

直播时间：10月28日（周六）

时间	议题/演讲嘉宾
00:00-00:55	**自然语言处理在“天猫精灵”的实践应用**
13:30-14:25	**深度学习在搜索的应用：学术前沿与工业方案解析** 张俊林　新浪微博AI lab资深算法专家
14:30-15:25	**深度学习部署系统构建** 刘文志　商汤科技高性能计算部门负责人
15:30-16:25	**多租户机器学习平台的权限模型与调度设计** 陈迪豪　第四范式先知平台架构师
16:30-17:25	**深度学习在推荐领域的应用和实践** 吴岸城　蘑歌科技首席算法科学家

(http://geek.csdn.net/user/publishlist/heyc861221)
何永灿CSDN (http://geek.csdn.net/user/publishlist/heyc861221)
发布于 人工智能 (http://geek.csdn.net/forum/43) 2017-10-01 21:05

AlphaGo的大脑value network和policy network

确实牛逼啊。

源码解析：http://www.iocoder.cn?csdn (http://www.iocoder.cn?csdn)

0 不进某宝不改名 (http://geek.csdn.net/user/publishlist/s13618912524) 2017-10-02 12:24

解析的不错，已经收藏啦~~