qlearning参数_深度强化学习之深度Q网络DQN详解

引言

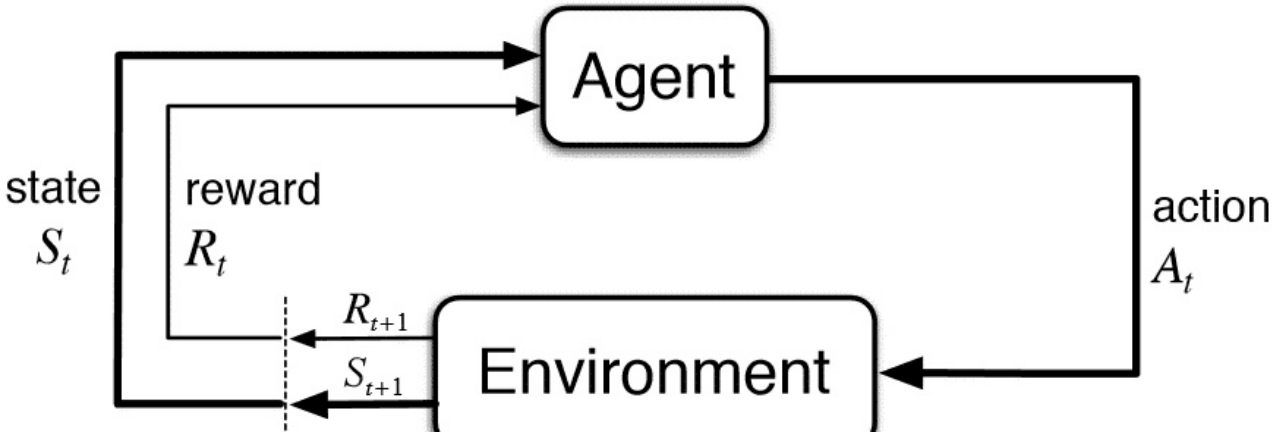
本文将对深度强化学习中经典算法DQN进行详细介绍，先分别介绍强化学习和Q-学习，然后再引入深度强化学习和DQN。本文所有参考资料及部分插图来源均列在文末，在文中不做额外说明。**强化学习**

讲强化学习先讲其适用的场景。强化学习多用在需要与环境交互的场景下，即给定一个环境的状态（State），程序根据某种策略（Policy）选出一个对应的行为（Action），而执行这个Action后环境又会发生改变，即状态会转换为新的状态S'，且每执行完一个Action后程序会得到一个激励值（Reward），而程序就依据得到的激励值的大小调整其策略，使得在所有步骤执行完后，即状态到达终止状态（Terminal）时，所获得的Reward之和最大。

上面说的可能比较抽象，举个例子，假如我们的程序是一只小狗，现在我们让它坐下（给它一个State），它如果听话（某种Policy）坐下（执行Action），那么我们就给它一个鸡腿（正激励），而如果它不听话（某种Policy）跑开了（执行另一种Action），我们就罚它一顿不许吃饭（负激励），而在它执行完这个行为后，我们可以再次对它提出要求，比如让它站起来（新的State），然后如此往复。小狗对我们给的每一个状态都要给出一个行为，而我们会在它每次给出行为后决定给它一个什么样的激励，且环境的状态在它执行完Action后可能会发生变化，然后它需要对新环境再继续根据某种策略选择执行新的动作，从而得到新的激励。而我们训练的目的，就是使得**总的激励值之和**最大。

总结一下，在强化学习中，我们关注的有如下几点：

- 环境观测值/状态 State
- 动作选择策略 Policy
- 执行的动作/行为 Action
- 得到的奖励 Reward
- 下一个状态 S’



- **状态State**：将每一帧作为一个状态，取小鸟离下一个地面上柱子在水平和竖直方向上的距离作为状态的观测值，即下图中的(△x, △y)；

- **行为Action**：对每一个状态（每一帧），只有两种选择：跳，不跳；
- **奖励Reward**：小鸟活着时给每帧奖励1，死亡时奖励-1000。

在该游戏中，程序是如何选择该跳还是不该跳呢？按照前面说的Q学习算法，那么它应该是需要有一个Q(S, A)函数的，可以知道在什么状态时采取什么样的行为能得到最大的Reward之和。在这个游戏中，很显然状态和动作的组合都是有限的，因此可以维护一个S-A表，其记录了在每个状态下，采用什么动作时能得到什么样的Q值。表格形式如下，只要程序在运行中不断更新这个表格，使其最终能收敛，难么程序就能拿得到的state通过查表的方式来判断它该选择什么样的行为，才能获得最大的Q值。

状态	飞	不飞
$(\Delta x_1, \Delta y_1)$	1	20
$(\Delta x_1, \Delta y_2)$	20	-100
…	…	…
$(\Delta x_m, \Delta y_{n-1})$	-100	2
$(\Delta x_m, \Delta y_n)$	50	-200

Q值更新方法

Q值大体上有两种更新方式，一种是类似上面例子中的情况，状态和行为的组合是可以穷尽的情况，这时候往往采用的是S-A表格的形式记录Q值，而如果状态和行为的组合不可穷尽，比如自动驾驶中输入的外界环境照片与车速之间的组合是有无穷种的，那么前一种方法显然就不适用了，这时候常用的方式为将深度学习与Q学习结合起来，也就是本文的重点，DQN，这个我们将在后面重点讲解。**S-A表格如何更新**

对于使用S-A表格的情况，需要如何更新其表格中的Q值，使得其在每一个状态下都能选择总体最优的策略呢？

这里首先引出Q值的更新方法：

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a))$$

解释一下，s为状态State，a为采取的行为Action，α参数用来表示新的值对更新后值所造成的影响大小，r为在状态s下采取动作a后获得的奖励Reward，γ也是一个discount值，即用来减小新值的影响的值。其中α和γ的范围都在0~1之间。

如果还不明白，那么可以将这个公式拆开理解。

先看左边一部分，如果不看(1 - α)，那么就是在状态s下采取动作a时的旧的Q值，乘以(1 - α)是因为要更新它，但是也不能把旧的Q值全盘否定了呀。

再看右边部分，同样先不看α，里面是激励值r加上一部分，而后面那部分先去掉γ也不看，就是在下一个状态下的最大Q值，想一想，状态s下采用动作a后得到的激励值r加上下一个状态下的最大Q值，不就是一个新的Q值么，它这是在不断使得激励值r收敛啊，而像α和γ只是用来控制新的Q值和旧的Q值各占多少权重罢了。**策略Policy选择注意事项**

前面讲了，在强化学习中最重要的部分就是策略的选择，S-A表格说白了也不过是给选择哪个策略提供了一个参考。而在实际实验中，如果对每个状态s值，都选择其能获得最大Q值的行为去执行，是有问题的！

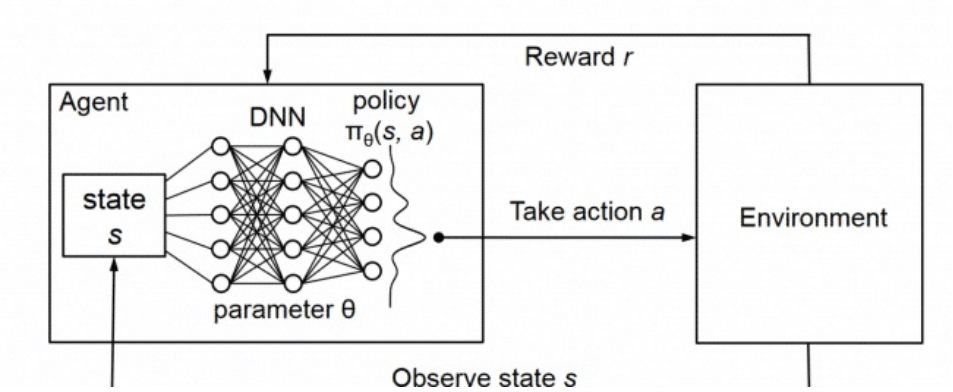
假使初始的S-A表格所有值全为0，那么在状态s采用随机一个行为（比如a1），并第一次获得reward后，如果reward值大于0，那么以后再遇见状态s时，程序都会直接采用行为a1，然而，还有很多种没有行为从来都没有常识过，说不定采取其他的行为会使得它能得到的Q值更大。

因此在强化学习中，往往需要设置一个阈值ε来保持一定的随机程度，即在每次做决定前，先生成一个随机数，如果这个随机数比ε小，那么就随机选取一个action，否则才选取当前已知条件下能使得Q值最大的action。这个阈值ε往往一开始被设置地很大，而其值也会随着程序不断地迭代而慢慢衰减，一般也需要给其设置一个最小值，即衰减到最小值后就停止衰减了。这样的好处是使得程序可以遍历所有的S-A对，以准确判断在给定状态下选择哪个行为最优，而这种做法被称为**exploration**，这种算法叫做**e-greddy**。**DQN**

终于讲到DQN了，DQN即Deep Q Network，深度Q网络。

DQN属于DRL（深度强化学习）的一种，它是深度学习与Q学习的结合体。前面讲了采用S-A表格的局限性，当状态和行为的组合不可穷尽时，就无法通过查表的方式选取最优的Action了。这时候就该想到深度学习了，想通过深度学习找到最优解在很多情况下确实不太靠谱，但是找到一个无限逼近最优解的次优解，倒是没有问题的。

因此DQN实际上，总体思路还是用的Q学习的思路，不过对于给定状态选取哪个动作所能得到的Q值，却是由一个深度神经网络来计算的了，其流程图如下：



DNN如何训练

现在我们的选择哪个动作，是由DNN来做决定的，因此我们需要训练DNN以使其能达到令人满意的表现。这显然是一个监督学习的问题，那么训练集是什么，标签是什么，损失函数又是什么？

首先，我们DNN的输出值，自然是在给定状态的情况下，执行各action后能得到的Q值。然而事实上我们在很多情况下并不知道最优的Q值是什么，比如自动驾驶、围棋等情况，所以似乎我们没法给出标签。但是什么是不变的呢？**Reward**！

对状态s，执行动作a，那么得到的reward是一定的，而且是不变的！因此需要考虑从reward下手，让预测Q值和真实Q值的比较问题转换成让模型实质上在拟合reward的问题。

如果不能很好的理解，请看下面公式，下面公式中target是什么我们会在后面说到，这里先忽略它，那个红色的θi-我们也暂且将它当成θi来看。这个公式描述的就是模型的损失函数，大括号外面就是求一个均方差，我们主要看括号里面。前面被target标出来的地方是这一步得到的reward+下一状态所能得到的最大Q值，它们减去这一步的Q值，那么实际上它们的差就是实际reward减去现有模型认为在s下采取a时能得到的reward值。

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left[r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right]^2$$

现在的问题就已经转换为需要一组训练集，它能够提供一批四元组（s, a, r, s’），其中s’为s执行a后的下一个状态。如果能有这样一个四元组，就能够用来训练DNN了，这就是我们要介绍的experience reply。Experience Reply

前面提到我们需要一批四元组（s, a, r, s’）来进行训练，因此我们需要缓存一批这样的四元组到经验池中以供训练之用。由于每次执行一个动作后都能转移到下一个状态，并获得一个reward，因此我们每执行一次动作后都可以获得一个这样的四元组，也可以将这个四元组直接放入经验池中。

我们知道这种四元组之间是存在关联性的，因为状态的转移是连续的，如果直接按顺序取一批四元组作为训练集，那么是容易过拟合的，因为训练样本间不是独立的！为解决这个问题，我们可以简单地从经验池中随机抽取少量四元组作为一个batch，这样既保证了训练样本是**独立同分布**的，也使得每个batch**样本量不大**，能加快训练速度。

训练的伪代码如下图（图源知乎）：

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t e-greedy策略
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$ 放到经验池
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$ 从经验池采样
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for

Target Network

上面的代码似乎已经能够正常运行了，为什么又冒出一个target network呢？回想下前面那个公式，这里重新搬到下面来，是不是之前说target和θi-都先忽略，现在就该解释一下了。

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2$$

这个公式里θi-和θi肯定是有区别的，不然也犯不着用两个符号了。事实上，我们需要设计两个DNN，它们结构完全一样，但是参数不一样，即神经网络中各层的权重、偏置等，一个的参数是θi-，而另一个是θi。我们每次迭代中，更新的是θi而不更新θi-，且规定每运行C步后让θi- = θi。而其θi-所在的网络就被称为target network。

为什么要弄这么奇怪的东西？

这也是为了防止过拟合。试想如果只有一个神经网络，那么它就在会不停地更新，那么它所追求的目标是在一直改变的，即在 θ 改变的时候，不止Q(s, a)变了，max $Q(s’, a’)$也变了。这样的好处是使得上面公式中target所标注的部分是暂时固定的，我们不断更新θ追逐的是一个固定的目标，而不是一直改变的目标。

如果做了这种考虑，那么伪代码就要做一点点修改了，修改后如下，大家可以对比着看：

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

DQN架构及实现方法已经介绍完毕了，希望能帮到大家理解！也感谢下面参考资料里提到的作者们！**参考资料**

- Ye, Hao, Geoffrey Ye Li, and Biing-Hwang Fred Juang. “Deep reinforcement learning based resource allocation for V2V communications.” *IEEE Transactions on Vehicular Technology* 68.4 (2019): 3163-3173.
- Mnih, Volodymyr, et al. “Human-level control through deep reinforcement learning.” *Nature* 518.7540 (2015): 529-533.
- Welcome to Deep Reinforcement Learning Part 1 : DQN
- @jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4">RL — DQN Deep Q-network
- 关于强化学习中经验回放（experience replay）的两个问题？ - 宝珠道人的回答 - 知乎
- memory replay 是不是就是在DQN中为训练提供训练样本的呢？ - 东林钟声的回答 - 知乎
- Using Deep Q-Learning in the Classification of an Imbalanced Dataset
- 强化学习的Q-learning可以从已经完成的episode学习吗？还是边学习边完成episode的？ - 郭祥昊的回答 - 知乎
- Reinforcement learning - Part 1: Introduction to Q-learning
- Reinforcement learning - Part 2: Getting started with Deep Q-Networks
- Deep Q Learning
- Deep Q-Learning
- 如何用简单例子讲解 Q - learning 的具体过程？ - 牛阿的回答 - 知乎