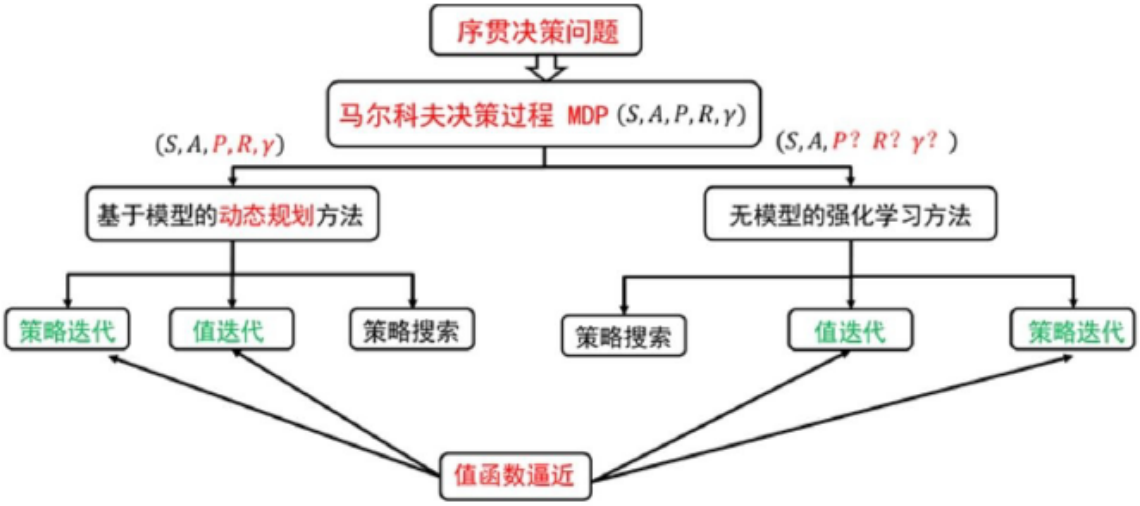
5 Value Function Approximation-Based Reinforcement Learning Method

5.1 基于值函数逼近的理论分析

5.1.1 值函数逼近理论

对于表格型值函数，可以采用前述的基于动态规划的方法，基于蒙特卡洛的方法和基于时间差分的方法。这些方法有一个基本的前提条件：状态空间和动作空间是离散的，而且状态空间和动作空间不能太大。对于状态值函数，其索引是状态；对于行为值函数，其索引是状态-行为对。值函数的迭代更新实际上就是这张表的迭代更新。若状态空间的维数很大，或者状态空间为连续空间，此时值函数无法用一张表格来表示。这时，我们需要利用函数逼近的方法表示值函数，如下图所示。当值函数利用函数逼近的方法表示后，可以利用策略迭代和值迭代方法构建强化学习算法。



在表格型强化学习中，值函数对应着一张表。在值函数逼近方法中，值函数对应着一个逼近函数 $\hat{v}(s)$ 。从数学角度看，函数逼近方法可以分为参数逼近和非参数逼近。其中参数逼近又分为线性化参数逼近和非线性化参数逼近。

对于参数化逼近，是指值函数可以由一组参数 θ 来近似。我们将逼近的值函数写为 $\hat{v}(s,\theta)$ 。

当逼近的值函数结构确定时，值函数的逼近就等价于参数的逼近，值函数的更新也就等价与参数的更新。也就是说，我们需要利用试验数据来更新参数值。

对于表格型值函数的更新过程，都是朝着一个目标值更新的，这个目标值在蒙特卡洛方法中是 G_t ，在时间差分方法中是 $r+\gamma Q(s',a')$ ，在 $TD(\lambda)$ 中是 G_t^{λ} 。

将表格型强化学习值函数的更新过程推广到值函数逼近过程，有如下形式。

函数逼近 $\hat{v}(s,\theta)$ 的过程是一个监督学习的过程，其数据和标签对为 (S_t,U_t) ，其中 U_t 等价于表格型值函数的目标值。

训练的目标函数为

$$\arg\min_{\theta}\left(q(s,a)-\hat{q}(s,a,\theta)\right)^2 \tag{1}$$

下面我们比较总结一下表格型强化学习和函数逼近方法的强化学习值函数更新时的异同点。

（1）表格型强化学习在更新值函数时，只有当前状态 S_t 处的值函数改变，其他地方的值函数不改变。

（2）值函数逼近方法更新值函数时，更新的是参数 θ，而估计的值函数为 $\hat{v}(s,\theta)$，所以当参数 θ 发生改变，任意状态处的值函数都会发生改变。

5.1.2 值函数更新方法

值函数更新可以分为增量式学习方法和批学习方法。

对于增量式学习方法，随机梯度下降法是最常用的增量式学习方法。

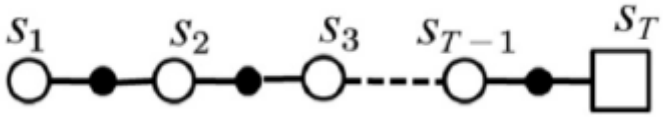
增量式学习方法：随机梯度下降法

由（1）式我们可以得到参数的随机梯度更新为：

$$\theta_{t+1}=\theta_t+\alpha[U_t-\hat{v}(S_t,\theta_t)]\nabla_\theta\hat{v}(S_t,\theta) \tag{2}$$

基于蒙特卡洛方法的函数逼近：

给定要评估的策略 π，产生一次试验：



值函数的更新过程实际是一个监督学习的过程，其中监督数据集从蒙特卡洛的试验中得到，数据集为 $<s_1,G_1>$，$<s_2,G_2>$，$\cdots$，$<s_T,G_T>$。

值函数的更新如下。

$$\Delta\theta=\alpha(G_t-\hat{v}(S_t,\theta))\nabla_\theta\hat{v}(S_t,\theta) \tag{3}$$

下图为基于梯度的蒙特卡洛值函数逼近更新过程。蒙特卡洛方法的目标值函数使用一次试验的整个回报返回值。

基于梯度的蒙特卡罗值函数评估算法

输入：要评估的策略 π，一个可微逼近函数 $\hat{v}:S\times R^n\to R$

恰当地初始化的值函数权重 θ （例如 $\theta=0$ ）

Repeat:

利用策略 产生一幕数据

For $t=0,1,\cdots,T-1$

$\theta\leftarrow\theta+\alpha[G_t-\hat{v}(S_t,\theta)]\nabla\hat{v}(S_t,\theta)$

基于时间差分方法的函数逼近

$TD(0)$ 方法中目标值函数为 $U_t=R_{t+1}+\gamma\hat{v}(S_{t+1},\theta)$，即目标值函数用到了 bootstrapping 的方法。

此时要更新的参数 θ 不仅出现在要估计的值函数 $\hat{v}(S_t,\theta)$ 中，还出现在目标值函数 U_t 中。若只考虑参数 θ 对估计值函数 $\hat{v}(S_t,\theta)$ 的影响而忽略对目标值函数的影响，这种方法就不是完全的梯度法，因此也称为基于半梯度的 $TD(0)$ 值函数评估算法，如下图所示。

$$\theta_{t+1}=\theta_t+\alpha[R+\gamma\hat{v}(S',\theta)-\hat{S_t,\theta_t}]\nabla\hat{v}(S_t,\theta_t) \tag{4}$$

基于半梯度的TD(0)值函数评估算法

输入：要评估的策略 π，一个可微逼近函数 $\hat{v}:S\times R^n \to R$

恰当地初始化的值函数权重 θ （例如 $\theta=0$ ）

Repeat：

初始化状态S，

Repeat（对于一幕中的每一步）

选择动作 $A\sim\pi(\cdot|S)$

采用动作 A 并观测回报 R,S'

$\theta_{t+1}=\theta_t+\alpha[R+\gamma\hat{v}(S',\theta)-\hat{v}(S_t,\theta_t)]\nabla\hat{v}(S_t,\theta_t)$

$S\leftarrow S'$

直到 S' 是终止状态

下图为基于半梯度的 Sarsa 算法。与表格型强化学习相比，值函数逼近方法中把对值函数的更新换成了对参数的更新，参数的学习过程为监督学习。

输入：一个要逼近的可微动作值函数：$\hat{q}:S\times A\times R^n \to R$ 任意地初始化的值函数权重 θ（例如$\theta=0$）

Repeat（for each episode）：

初始化状态行为对 S，A

Repeat（对于每一幕数据中的每一步）：

采用动作 A，得到回报 R，和下一个状态 S'

如果 S' 是终止状态：$\theta\leftarrow\theta+\alpha[R-\hat{q}(S,A,\theta)]\nabla\hat{q}(S,A,\theta)$

进入下一幕

利用软策略选择一个动作 A'，以便估计动作值函数 $\hat{q}(S',A',\theta)$

$\theta\leftarrow\theta+\alpha[R+\gamma\hat{q}(S',A',\theta)-\hat{q}(S,A,\theta)]\nabla\hat{q}(S,A,\theta)$

$S\leftarrow S'$

$A\leftarrow A'$

值函数形式

值函数可以采用线性逼近和非线性逼近，非线性逼近常用的是神经网络。

下面仅讨论线性逼近：$\hat{v}(s,\theta)=\theta^T\phi(s)$。

相比于非线性逼近，线性逼近的好处是只有一个最优值，因此可以收敛到全局最优。其中，$\phi(s)$ 为状态 s 处的特征函数，或者称为基函数。

常用的基函数的类型如下。

多项式基函数，如 $(1,s_1,s_2,s_1s_2,s_1^2,s_2^2,\ \cdots)$。

傅里叶基函数：$\phi_i(s)=cos(i\pi s),\ s\in[0,1]$。

径向基函数：$\phi_i(s)=\exp\left(-\frac{||s-c_i||^2}{2\sigma_i^2}\right)$。

将线性逼近值函数带入随机梯度下降法和半梯度下降法中，可以得到参数的更新公式，不同强化学习方法更新公式如下。

蒙特卡洛方法值函数更新公式：

$$\begin{aligned}\Delta\theta &= \alpha[U_t(s) - \hat{v}(S_t,\theta_t)]\nabla\hat{v}(S_t,\theta_t)\\ &= \alpha[G_t - \theta^T\phi]\phi\end{aligned} \tag{5}$$

$TD(0)$ 线性逼近值函数更新公式：

$$\begin{aligned}\Delta\theta &= \alpha[R + \gamma\theta^T\phi(s') - \theta^T\phi(s)]\phi(s)\\ &= \alpha\delta\phi(s)\end{aligned} \tag{6}$$

正向视角的 $TD(\lambda)$ 更新公式：

$$\Delta\theta = \alpha(G_t^\lambda - \theta^T\phi)\phi \tag{7}$$

后向视角的 $TD(\lambda)$ 更新公式：

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma\theta^T\phi(s') - \theta^T\phi(s)\\ E_t &= \gamma\lambda E_{t-1} + \phi(s)\\ \Delta\theta &= \alpha\delta_t E_t\end{aligned} \tag{8}$$

批学习方法

增量式方法参数更新过程随机性比较大，尽管计算简单，但样本数据的利用效率并不高。

批学习方法虽然计算复杂，但计算效率高。

所谓批的方法是指给定经验数据集 $D = <s_1, v_1^\pi>, <s_2, v_2^\pi>, \cdots, <s_t, v_T^\pi>$，找到最好的拟合函数 $\hat{v}(s,\theta)$，使得 $LS(\theta) = \sum_{t=1}^T(v_t^\pi - \hat{v}_t^\pi(s_t,\theta))^2$ 最小。

可利用线性最小二乘逼近：

$$\Delta\theta = \alpha\sum_{t=1}^T[v_t^\pi - \theta^T\phi(s_t)]\phi(s_t) = 0 \tag{9}$$

最小二乘蒙特卡洛方法参数为

$$\theta = (\sum_{t=1}^T\phi(s_t)\phi(s_t)^T)^{-1}\sum_{t=1}^T\phi(s_t)G_t \tag{10}$$

最小二乘差分方法为

$$\theta = (\sum_{t=1}^T\phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^T)^{-1}\sum_{t=1}^T\phi(s_t)R_{t+1} \tag{11}$$

最小二乘 $TD(\lambda)$ 方法为

$$\theta = (\sum_{t=1}^T E_t(\phi(s_t) - \gamma\phi(s_{t+1}))^T)^{-1}\sum_{t=1}^T E_t R_{t+1} \tag{12}$$

5.2 DQN及其变种

5.2.1 DQN方法

DQN 方法是 DeepMind 发表在 Nature 上的第一篇论文，名字是 Human-level Control through Deep Reinforcement Learning 。这篇论文有两个创新点，即经验回放和设立单独的目标网络。算法的大体框架是传统强化学习中的 Q-learning 。

Q-learning 方法是异策略时间差分方法。其伪代码如下图所示。

1. 初始化 $Q(s,a),\forall s \in S, a \in A(s)$,给定参数$\alpha$，$\gamma$
2. Repeat:
3. 给定起始状态 s，并根据 $\mathcal{E}$贪婪策略在状态 s 选择动作 a
4. Repeat(对于一幕的每一步)
5. (a) 根据 $\mathcal{E}$ 贪婪策略在状态 S_t 选择动作 a_t ，得到回报r_t和下一个状态S_{t+1}
6. (b) $Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha\left[r_t+\gamma \max_a Q(s_{t+1},a)-Q(s_t,a_t)\right]$
7. (c) s=s' ， a=a'
8. Until s 是终止状态
9. Until 所有的 $Q(s,a)$ 收敛
10. 输出最终策略：$\pi(s)=\operatorname{argmax}_a Q(s,a)$

行动策略为 s，贪婪策略

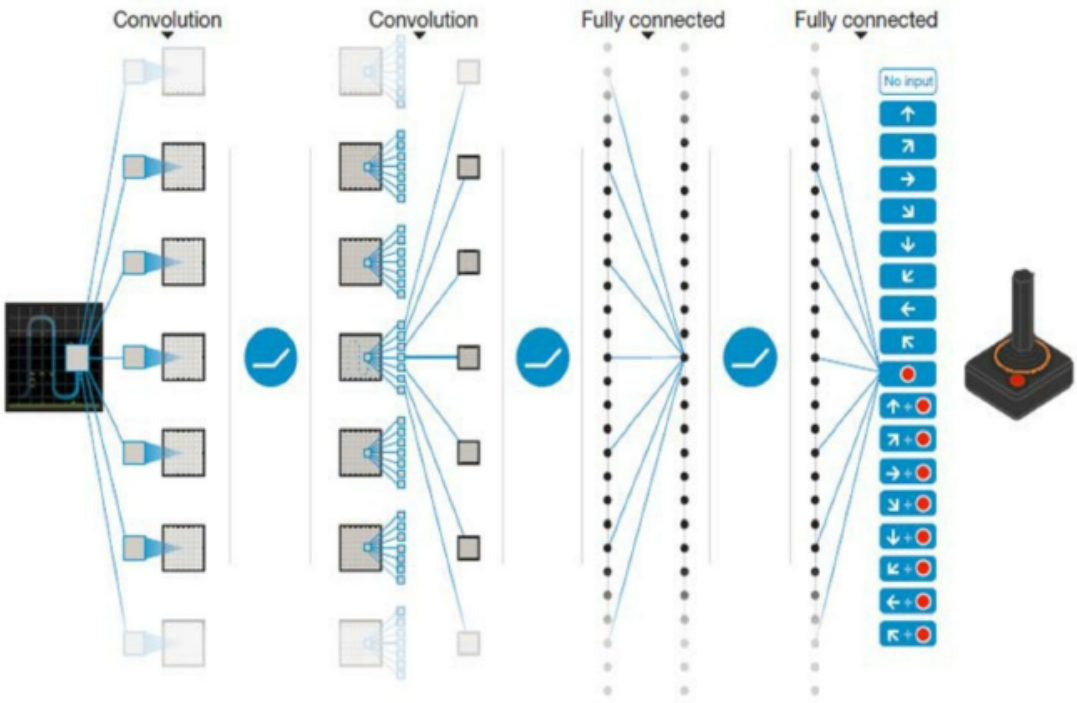
目标策略为贪婪策略

异策略是指行动策略（产生数据的策略）和要评估的策略不是一个策略。在上图中，行动策略是 $\epsilon-greedy$ 策略，而要评估和改进的策略是贪婪策略。时间差分方法是指利用时间差分目标来更新当前行为值函数，即 $r_t+\gamma \max_a Q(s_{t+1},a)$ 。

DQN 对 Q-learning 的修改主要体现在以下三个方面。

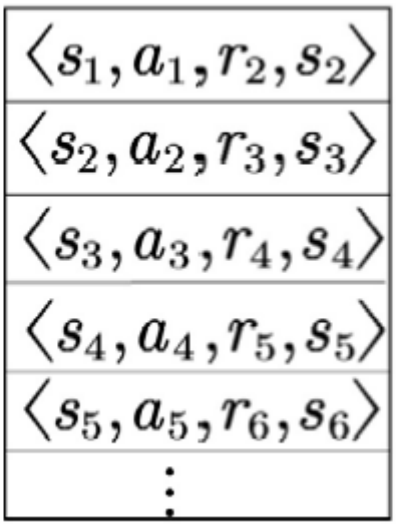
（1）DQN 使用深度卷积神经网络逼近值函数。

下图为 DQN 行为值函数逼近网络。此处的值函数对应着一组参数，在神经网络里参数是每层网络的权重，我们用 θ 表示。用公式可以表示值函数为 $Q(s,a;\theta)$ 。此时更新值函数即为更新参数 θ ，当网络结构确定时， θ 就代表值函数。 DQN 所用的网络结构是三个卷积层加两个全连接层，整体框架如下图所示。



（2）DQN 利用了经验回放训练强化学习的学习过程。

通过经验回放可以令神经网络的训练收敛且稳定，因为训练神经网络时，存在的假设是训练数据是独立同分布的，但是通过强化学习采集的数据之间存在着关联性，利用这些数据进行顺序训练，神经网络当然不稳定。经验回放可以打破数据间的关联。如下图所示。



在强化学习过程中，智能体将数据存储到一个数据库中，再利用均匀随机采样的方法从数据库中抽取数据，然后利用抽取的数据训练神经网络。

（3）DQN 独立设置了目标网络来单独处理时间差分算法中的 TD 偏差。

DQN 中的参数更新方法为如下公式。

$$\theta_{t+1} = \theta_t + \alpha[r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)] \nabla Q(s, a; \theta) \tag{13}$$

其中， $r + \gamma \max_{a'} Q(s', a'; \theta)$ 为 TD 目标，在计算 $\max_{a'} Q(s', a'; \theta)$ 值时用到的网络参数为 θ 。

我们称计算 TD 目标时所用的网络为 TD网络。在 DQN 算法出现之前，利用神经网络逼近值函数时，计算 TD　目标的动作值函数所用的网络参数，与梯度计算中要逼近的值函数所用的网络参数相同，这样就容易导致数据间存在关联性，从而使训练不稳定。为了解决此问题， DeepMind 提出计算 TD 目标的网络表示为 θ^- ；计算值函数逼近的网络表示为 θ ；用于动作值函数逼近的网络每一步都更新，而用于计算 TD 目标的网络则是每个固定的步数更新一次。

因此，值函数的更新变为

$$\theta_{t+1} = \theta_t + \alpha[r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)] \nabla Q(s, a; \theta) \tag{14}$$

最后，我们给出 DQN 的伪代码，如下图所示。

[1] Initialize replay memory D to capacity N
[2] Initialize action-value function Q with random weights θ
[3] Initialize target action-value function $\hat{Q}$ with weights $\theta^{-}=\theta$
[4] **For** episode = 1, M **do**
[5] Initialize sequence $s_1=\{x_1\}$ and preprocessed sequence $\phi_1=\phi(s_1)$
[6] **For** t = 1,T **do**
[7] With probability ε select a random action a_t
[8] otherwise select $a_t=\text{argmax}_a Q(\phi(s_t),a;\theta)$
[9] Execute action a_t in emulator and observe reward r_t and image x_{t+1}
[10] Set $s_{t+1}=s_t,a_t,x_{t+1}$ and preprocess $\phi_{t+1}=\phi(s_{t+1})$
[11] Store transition $(\phi_t,a_t,r_t,\phi_{t+1})$ in D
[12] Sample random minibatch of transitions $(\phi_j,a_j,r_j,\phi_{j+1})$ from D
[13] Set $y_j=\begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j+\gamma \max_{a'} \hat{Q}(\phi_{j+1},a';\theta^{-}) & \text{otherwise} \end{cases}$
[14] Perform a gradient descent step on $\left(y_j-Q(\phi_j,a_j;\theta)\right)^2$ with respect to the
[15] network parameters θ
[16] Every C steps reset $\hat{Q}=Q$
[17] **End For**
[18] **End For**

5.2.2 Double DQN

DQN 无法克服 Q-learning 本身所固有的缺点——过估计。过估计是指估计的值函数比真实的值函数要大。一般来说，Q-learning 之所以存在过估计的问题，根源在于 Q-learning 中的最大化操作。

Q-learning 评估值函数的数学公式有如下两类。

对于表格型，值函数评估的更新公式为

$$Q(s_t,a_t)\leftarrow Q(s_t,a_t)+\alpha[r_t+\gamma\max_a Q(s_{t+1},a)-Q(s_t,a_t)] \tag{15}$$

对于基于值函数逼近的方法，值函数的更新公式为

$$\theta_{t+1}=\theta_t+\alpha[R_{t+1}+\gamma\max_a Q(S_{t+1},a;\theta_t)-Q(S_t,A_t;\theta_t)]\nabla_{\theta_t}Q(S_t,A_t;\theta_t) \tag{16}$$

max 操作使得估计的值函数比值函数的真实值大。如果值函数每一点的值都被过估计了相同的幅度，即过估计是均匀的，那么由于最优策略是贪婪策略，即找到最大的值函数所对应的动作，这时候最优策略是保持不变的。也就是说，在这种情况下，即使值函数被过估计了，也不影响最优的策略。然而，在实际情况中，过估计量并非是均匀的，因此值函数的过估计会影响最终的策略决策，从而导致最终的策略并非最优，而只是次优。

为了解决值函数过估计的问题， Hasselt 提出了 Double Q-learning 方法。所谓 Double Q-learning 是将动作的选择和动作的评估分别用不同的值函数来实现。

（1）动作选择。

在 Q-learning 的值函数更新中， TD 目标为

$$Y_t^Q=R_{t+1}+\gamma\max_a Q(S_{t+1},a;\theta_t) \tag{17}$$

在求 TD 目标 Y_t^Q 的时候，我们首先需要选择一个动作即 a^* ，该动作 a^* 应该满足在状态 S_{t+1} 处 $Q(S_{t+1},a)$ 最大，这就是动作选择。

（2）动作评估。

动作评估是指选出 a^* 后，利用 a^* 处的动作值函数构造 TD 目标。

一般 Q-learning 利用同一个参数 θ_t 来选择和评估动作。 Double Q-learning 分别用不同的行为值函数选择和评估动作。 Double Q-learning 的 TD 目标公式为

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t') \tag{18}$$

从该公式我们看到，动作的选择所用的动作值函数为

$$\arg\max_a Q(S_{t+1}, a; \theta_t) \tag{19}$$

这时动作值函数网络的参数为 θ_t 。当选出最大的动作 a^* 后，动作评估的公式为

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, a^*; \theta_t') \tag{20}$$

动作评估所用的动作值函数网络参数为 θ_t' 。

将 Double Q-learning 的思想应用到 DQN 中，则得到 Double DQN 即 DDQN ，其 TD 目标为

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t^-) \tag{21}$$

5.2.3 优先回放（Prioritized Replay）

经验回放时利用均匀分布采样并不是高效利用数据的方法。因为，智能体的经验即经历过的数据，对于智能体的学习并非具有同等重要的意义。智能体在某些状态的学习效率比其他状态的学习效率高。优先回放的基本思想就是打破均匀采样，赋予学习效率高的状态以更大的采样权重。

符合上述思想的一个选择是 TD 偏差 δ 。 TD 偏差越大，说明该状态处的值函数与 TD 目标的差距越大，智能体的更新量越大，因此该处的学习效率越高。

我们设样本 i 处的 TD 偏差为 δ_i ，则该样本处的采样概率为

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{22}$$

其中 p_i^α 由 TD 偏差 δ_i 决定。一般有两种方法，第一种方法是 $p_i = |\delta_i| + \epsilon$ ；第二种方法是 $p_i = \frac{1}{rank(i)}$ ，其中 $rank(i)$ 根据 $|\delta_i|$ 的排序得到。

当我们采用优先回放的概率分布采样时，动作值函数的估计值是一个有偏估计。因为采样分布与动作值函数的分布是两个完全不同的分布。为了矫正这个偏差，我们需要乘以一个重要性采样系数 $w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$ 。

带有优先回放的 Double DQN 的伪代码如下图所示。

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

5.2.4 Dueling DQN

Dueling DQN 从网络结构上改进了 DQN 。动作值函数可以分解为状态值函数和优势函数，即

$$Q^\pi(s,a) = V^\pi(s) + A^\pi(s,a) \tag{23}$$

前面的各类 DQN 方法直接利用神经网络逼近 $Q^\pi(s,a)$， Dueling DQN 则对 $V^\pi(s)$ 和 $A^\pi(s)$ 分别利用神经网络逼近，其网络结构如下图所示。