深度学习基础、DQN算法、DQN算法进阶

深度学习基础

略（比较熟悉，就不写了）

DQN算法

深度网络

类似于 Q 表，神经网络也可以用来近似动作价值函数$Q(\boldsymbol{s}, \boldsymbol{a})$，即将状态向量 s 作为输入，并输出所有动作$\boldsymbol{a}=(a_1, a_2, \ldots, a_n)$$对应的价值，

$$\boldsymbol{y}=Q_\theta(\boldsymbol{s}, \boldsymbol{a})$$

对比：

1. Q表的局限性会更大一些，它只能处理离散的状态和动作空间，而神经网络则可以处理连续的状态和动作空间。
2. 神经网络只用了两个维度的输入就表示了原来 Q 表中无穷多个状态，这就是神经网络的优势所在。因此，在 Q 表中我们**描述状态空间的时候一般用的是状态个数，而在神经网络中我们用的是状态维度**。

共同点：

输出的都是每个动作对应的 Q 值，即预测，而不是直接输出动作。要想输出动作，就需要额外做一些处理，例如结合贪心算法选择 Q 值最大对应的动作等，这就是控制过程。

神经网络的参数也使用梯度下降的方法来求解。回顾Q-learning算法的更新公式：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)+\alpha[r_t+\gamma \max_a Q'(s_{t+1}, a)-Q(s_t, a_t)]$$

期望的Q值：$y_t=r_t+\gamma \max_a Q'(s_{t+1}, a)$
实际的Q值：$Q(s_t, a_t)$

在 Q-learning 算法中，由于没有额外的参数，因此只需要直接一步步迭代更新 Q 值即可。而在 DQN 中，我们用神经网络来近似 Q 函数，引入了额外的网络参数 θ

$$Q\left(s_i, a_i ; \theta\right) \leftarrow Q\left(s_i, a_i ; \theta\right)+\alpha[y_i-Q\left(s_i, a_i ; \theta\right)]$$

$r_t+\gamma \max_a Q'(s_{t+1}, a)$和$Q(s_t, a_t)$之间的绝对差值也就是 TD 误差，也可以写成损失函数的形式并用梯度下降的方式来求解参数：

$$\begin{aligned} L(\theta) &= \left(y_i-Q\left(s_i, a_i ; \theta\right)\right)^2 \\ \theta_i &\leftarrow \theta_i-\alpha \nabla_{\theta_i} L_i\left(\theta_i\right) \end{aligned}$$

同样也有终止状态判断：

$$y_i = \begin{cases} r_i & \text{对于终止状态} s_i \\ r_i + \gamma \max_{a'} Q\left(s_{i+1}, a'; \theta\right) & \text{对于非终止状态} s_i \end{cases}$$

经验回放

——想解决单个样本更新参数导致的不稳定性（不满足独立同分布假设）。

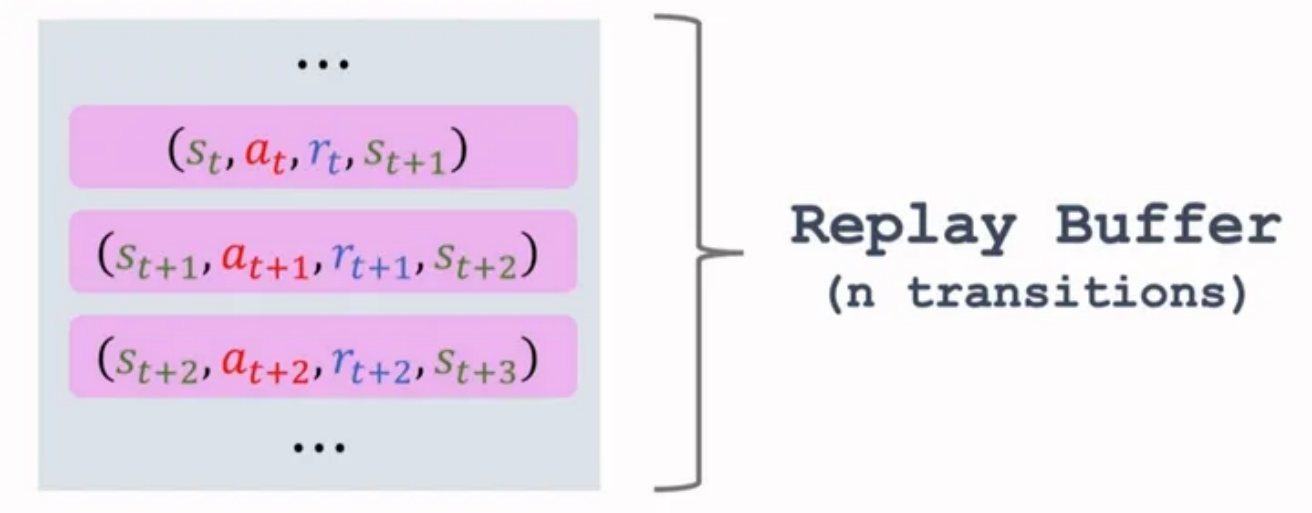
不使用经验回放的缺点：

1. 缺点一：浪费经验（Shortcoming 1：Watse of Experience）。传统的TD算法每次仅使用一个transition，使用完后就丢掉，不再使用，这会造成浪费。
2. 缺点二：关联性更新（Shortcoming 2：Correlated Updates）。传统TD算法会按照顺序使用transition(st,at,rt,st+1)，前后两条transition之间有很强的关联性，即st和st+1非常接近，实验表明这并不利于把DQN训练的很好。

而在 DQN 中，我们会把每次与环境交互得到的样本都存储在一个经验回放中，然后每次从经验池中随机抽取一批样本来训练网络。

好处：

1. 每次迭代的样本都是从经验池中随机抽取的，因此每次迭代的样本都是独立同分布的，这样就满足了梯度下降法的假设。
2. 经验池中的样本是从环境中实时交互得到的，因此每次迭代的样本都是相互关联的，这样的方式相当于是把每次迭代的样本都进行了一个打乱的操作，这样也能够有效地避免训练的不稳定性。



目标网络

目标网络和当前网络结构都是相同的，都用于近似 Q 值，在实践中每隔若干步才把每步更新的当前网络参数复制给目标网络，这样做的好处是保证训练的稳定，避免 Q值的估计发散。在计算损失函数的时候，使用的是目标网络来计算 Q 的期望值。

DQN算法进阶

Double DQN算法

在DQN算法中，我们直接拿目标网络中各个动作对应最大的Q值来当作估计值，这样会存在过估计问题。

Double DQN：现在当前网络中找出最大 Q 值对应的动作，然后再将这个动作代入到目标网络中去计算 Q 值

$$a_\theta^{max}(s_{t+1}) = \arg\max_a Q_\theta(s_{t+1}, a)$$

然后将这个找出来的动作代入到目标网络里面去计算目标的 Q 值，进而计算估计值

$$y_t = r_t + \gamma \max_a Q_{\hat{\theta}}(s_{t+1}, a_\theta^{max}(s_{t+1}))$$

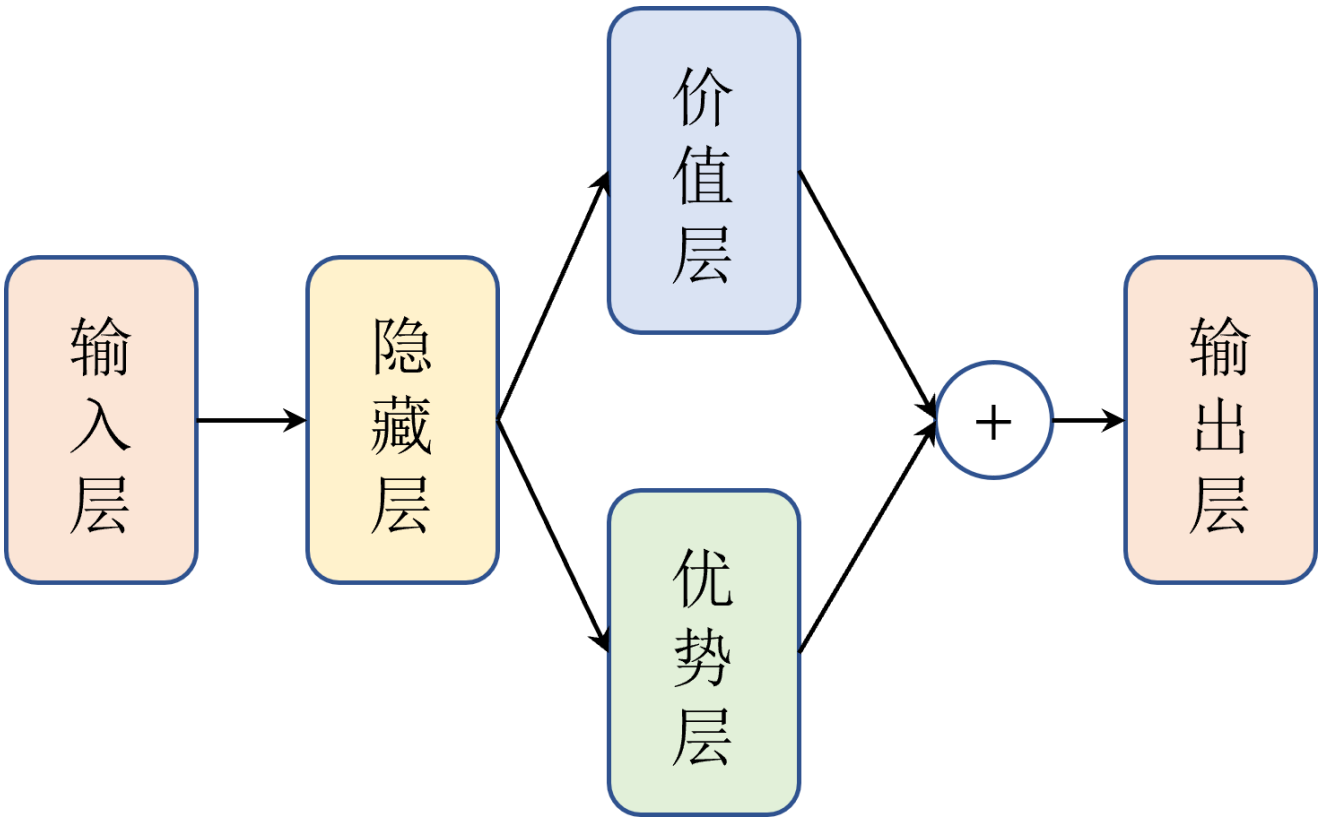
Double DQN 并不是每隔 C 步复制参数到目标网络，而是每次随机选择其中一个网络选择动作进行更新。假设两个网络分别为Q^A, Q^B， 那么在更新Q^A 的时候就用把 Q^B 当作目标网络估计动作值，同时 Q^B 也是用来选择动作的，反之亦然。

$$a^* = \arg\max_a Q^A(s_t, a)$$
$$Q^A(s_t, a) \leftarrow Q^A(s_t, a) + \alpha[r_t + \gamma Q^B(s_{t+1}, a^*) - Q^A(s_t, a)]$$

Dueling DQN算法

在 Dueling DQN 算法中我们通过优化神经网络的结构来优化算法。

Dueling DQN 算法在输出层之前分流（ dueling ）出了两个层，一个是优势层（advantage layer），用于估计每个动作带来的优势，输出维度为动作数一个是价值层（value layer），用于估计每个状态的价值，输出维度为 1 。



网络结构可表示为式：

$$Q_{\theta,\alpha,\beta}(\boldsymbol{s},\boldsymbol{a}) = A_{\theta,\alpha}(\boldsymbol{s},\boldsymbol{a})(\text{优势层，}demeaned) + V_{\theta,\beta}(\boldsymbol{s})(\text{价值层})$$

这一算法的好处：分开评估每个状态的价值以及某个状态下采取某个动作的 Q 值。当某个状态下采取一些动作对最终的回报都没有多大影响时，使得目标值更容易计算，因为通过使用两个单独的网络，我们可以隔离每个网络输出上的影响，并且只更新适当的子网络。

Noisy DQN算法

相对于Dueling DQN，Noisy DQN是为了增强网络的探索能力。

Noisy DQN 算法在 DQN 算法基础上在神经网络中引入了噪声层来提高网络性能的，即将随机性应用到神经网络中的参数或者说权重，增加了Q 网络对于状态和动作空间的探索能力，从而提高收敛速度和稳定性。

方法：通过添加随机性参数到神经网络的线性层，对应的 Q 值则可以表示为$Q_{\theta+\epsilon}$，这里 ϵ 是由高斯分布生成的总体分类噪声参数。

PER DQN算法

PER DQN 算法进一步优化了经验回放的设计从而提高模型的收敛能力和鲁棒性。PER 可以翻译为优先经验回放（prioritized experience replay），跟数据结构中优先队列与普通队列一样，会在采样过程中赋予经验回放中样本的优先级。

通常用 SumTree 这样的二叉树结构来实现在采样的时候取出优先级较大的样本。

缺陷：

1. 考虑到算法效率问题，我们在每次更新时不会把经验回放中的所有样本都计算 TD 误差并更新对应的优先级，而是只更新当前取到的一定批量的样本。这样一来，每次计算的 TD 误差是对应之前的网络，而不是当前待更新的网络。因此单纯根据 TD 误差进行优先采样有可能会错过对当前网络“信息量”更大的样本。
2. 被选中样本的 TD 误差会在当前更新后下降，然后优先级会排到后面去，下次这些样本就不会被选中，导致过拟合。

解决方法：**随机优先级采样**（ stochastic prioritization ），即在每次更新时，我们不再是直接采样 TD 误差最大的样本，而是定义一个采样概率

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

可以通过该式调节不同优先级样本被采样的概率。

重要性采样（importance sampling ）是一种用于估计某一分布性质的方法，它的基本思想是，我们可以通过与待估计分布不同的另一个分布中采样，然后通过采样样本的权重来估计待估计分布的性质。

$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int f(x)p(x)dx \\
&= \int f(x)\frac{p(x)}{q(x)}q(x)dx \\
&= \int f(x)\frac{p(x)}{q(x)}\frac{q(x)}{p(x)}p(x)dx \\
&= \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]
\end{aligned}
$$

我们已经从之前的网络中采样了一批样本，也就是 $q(x)$已知，然后只要找到之前网络分布与当前网络分布之前的权重 $\frac{p(x)}{q(x)}$，就可以利用重要性采样来估计出当前网络的性质。我们可以定义权重为

$$
w_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^{\beta}
$$

其中，N 是经验回放中的样本数量，P(i)是样本 i 的采样概率。**可以对权重进行归一化处理。**

超参数 β用于**调节重要性采样的程度**。在实践中，我们希望 β从 0 开始，随着训练步数的增加而逐渐增加，以便更好地利用重要性采样，这就是**热偏置（annealing the bias）**的思想。

$$
\beta = \min(1, \beta + \beta_{\text{step}})
$$

C51算法

也叫做Distributed DQN， Categorical DQN。该算法的核心思想是将传统 DQN 算法中的值函数 $Q(s,a)$换成了值分布$Z(x,a)$)，即将值函数的输出从一个数值变成了一个分布，这样就能更好地处理值函数估计不准确以及离散动作空间的问题。

我们可以将值函数 Q 看成是一个随机变量，它的期望值就是 Q 函数，而它的方差就是 Q 函数的不确定性，

$$
Q^{\pi}(x,a) := \mathbb{E}Z^{\pi}(x,a) = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R\left(x_t, a_t\right)\right]
$$

其中状态分布$x_t \sim P\left(\cdot \mid x_{t-1}, a_{t-1}\right), a_t \sim \pi\left(\cdot \mid x_t\right), x_0 = x, a_0 = a.$