强化学习基础篇（二十八）值函数近似法（Value Function Approximation）

在大规模的强化学习任务求解中，精确获得状态值或动作值Q较为困难。而值函数近似法通过寻找状态值或动作值Q的近似替代函数$\hat{v}(s,w)$或$\hat{q}(s,a,w)$的方式来求解大规模强化学习任务，既避免了表格求解法所需大规模存储空间的问题，又提升了求解效率，是实际求解任务中被泛采纳的一种算法。

1、大规模强化学习

强化学习是可以去解决很多非常大型的问题的，比如像Backgammon游戏中有着10^{20}个状态，在围棋游戏中有着10^{170}个状态，此外一些连续状态空间的环境也有着无数的状态。

那么如何把model-free方法中的预测与控制的算法应用在这些大规模强化学习任务之中呢？

这就需要使用值函数近似法，在之前我们所介绍的算法都是基于表格查找，也就是每一个状态s在$V(s)$的表格中都有相应的条目索引，或者说每个state-action对(s,a)在$Q(s,a)$的表格中都有相应的条目索引。

但是表格型的表述方法在大型的MDP问题中有着致命性的问题：

- 一是，存储这些大量的状态或者动作信息将会耗费海量的内存
- 二是，去单独学习每个状态的价值非常得慢。

所以，对于大型的MDP问题，我们只能使用函数去近似价值函数：

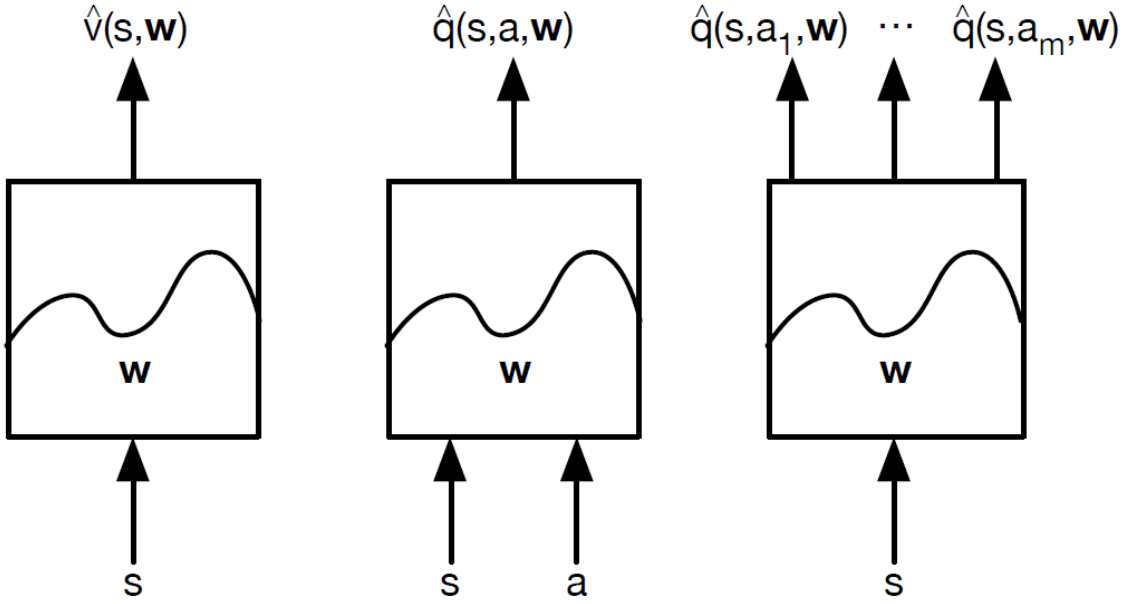
$$\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$$
$$\text{or } \hat{q}(s,a,\mathbf{w}) \approx q_\pi(s,a)$$

使用这种近似价值方法一方面可以将价值函数的近似从可见状态泛化到不可见状态，另一方面我们可以使用MC或TD学习的方法更新参数w。

2、函数近似的模式

价值函数的近似可以有三种主要的方式：

- 输入状态s，输出价值函数$\hat{v}(s,w)$
- 输入状态s与动作a，输出Q函数$\hat{q}(s,a,w)$.
- 输入状态s，输出所有可能动作的Q函数$\hat{q}(s,a_1,w)$，$\hat{q}(s,a_2,w)\dots\hat{q}(s,a_m,w)$



近似函数可以有很多的选择，比如线性函数，神经网络，决策树，最近邻算法，傅里叶基，小波变化等等。我们主要会集中于可微分的近似函数，比如线性函数和神经网络。

强化学习应用的场景其数据通常是非静态（non-stationary）、非独立同分布（non-iid）的，因为下一个状态通常与前一个状态高度相关。因此，函数近似器也需要适用于非静态、非独立同分布的数据

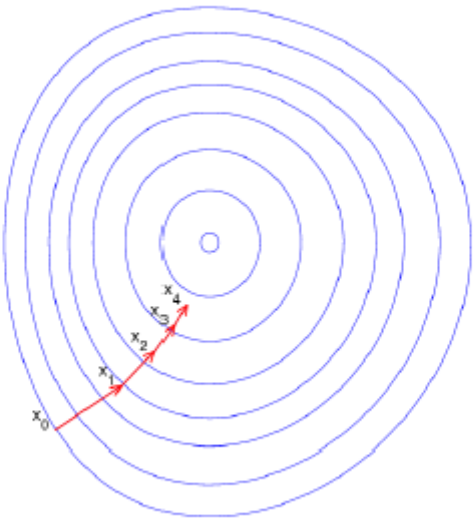
3、梯度下降（Gradient Descent）

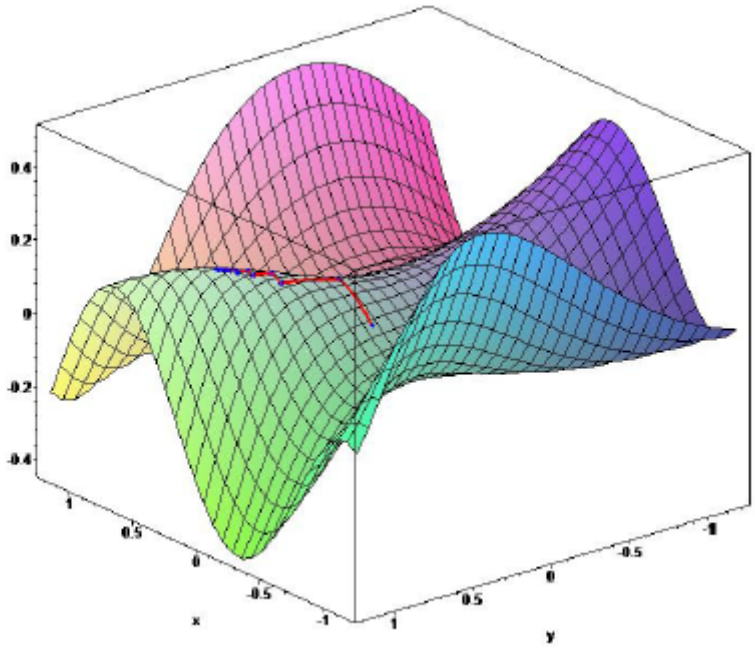
如果目标函数$J(w)$是一个关于参数w的可微分函数，那么目标函数$J(w)$的梯度就是：

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix}$$

为了去找到目标函数$J(w)$的最小值，我们只需要将参数w想着梯度的反方向进行调整。
$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w})$，这里的超参数$\alpha$是学习率。

梯度下降的可视化就可以看成沿着图形最陡峭的方向进行参数调整：





4、随机梯度下降（Stochastic Gradient Descent)

我如果已经知道策略π的真实的价值函数$v_\pi(s)$，那么我们可以将目标函数定义近似函数$\hat{v}(s,w)$和真实的价值函数$v_\pi(s)$之间的均方误差（Mean-squared error，MSE），我们需要找到为参数为w去最小化MSE的方法。

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[\left(v_\pi(S) - \hat{v}(S,\mathbf{w}) \right)^2 \right]$$

其关于参数w梯度为：

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[\left(v_\pi(S) - \hat{v}(S,\mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{V}(S,\mathbf{w}) \right] \end{aligned}$$

上面计算期望在大型环境中是很困难的，他需要求出策略π的概率分布，然后再加权求和。这样操作过于复杂，实际操作的时候会采用蒙特卡洛的形式，只用一个样本或者一批样本进行更新，此时会产生一定的样本误差。当样本足够多的时候，所有更新方向加权后就能逼近真实的梯度方向。）
随机梯度下降算法会对梯度进行采样：

$$\Delta \mathbf{w} = \alpha \left(v_\pi(S) - \hat{v}(S,\mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{V}(S,\mathbf{w})$$

5、线性函数近似

特征向量

状态我们是可以拆解为有一堆特征组成，所有的特征可以组成特征向量：

$$\mathbf{x}(S)=\begin{pmatrix}\mathbf{x}_1(S)\\ \vdots\\ \mathbf{x}_n(S)\end{pmatrix}$$

线性值函数近似

如果将值函数由线性方程来表示，可以是：

$$\hat{v}(S,\mathbf{w})=\mathbf{x}(S)^\top\mathbf{w}=\sum_{j=1}^{n}\mathbf{x}_j(S)\mathbf{w}_j$$

其MSE的目标函数可以表示为：

$$J(\mathbf{w})=\mathbb{E}_\pi\left[\left(v_\pi(S)-\mathbf{x}(S)^\top\mathbf{w}\right)^2\right]$$

对于线性值函数近似，我们使用SGD方法是可以收敛到全局最优解的，其梯度更新的过程也非常得简洁：

$$\begin{aligned}\nabla_{\mathbf{w}}\hat{V}(S,\mathbf{w})&=\mathbf{x}(S)\\ \Delta\mathbf{w}&=\alpha\left(v_\pi(S)-\hat{v}(S,\mathbf{w})\right)\mathbf{x}(S)\end{aligned}$$

这个梯度的更新过程，直观的看是：

$$Update=step\ size\times prediction\ error\times feature\ value$$

表格检索特征

表格检索是线性值函数近似的一种特殊形式，他吧每一个状态看成一个特征，个体具体处在某一个状态时，该状态特征取1，其余取0。参数的数目就是状态数，也就是每一个状态特征有一个参数。特征的形式是：

$$\mathbf{x}^{\text{table}}(S)=\begin{pmatrix}\mathbf{1}\left(S=s_1\right)\\ \vdots\\ \mathbf{1}\left(S=s_n\right)\end{pmatrix}$$

参数向量w本质上相当于给了每个状态对应的值函数:

$$\hat{v}(S,\mathbf{w})=\begin{pmatrix}\mathbf{1}\left(S=s_1\right)\\ \vdots\\ \mathbf{1}\left(S=s_n\right)\end{pmatrix}\cdot\begin{pmatrix}\mathbf{w}_1\\ \vdots\\ \mathbf{w}_n\end{pmatrix}$$

6、增量预测算法

之前是假设了给定了真实的值函数$v_\pi(s)$，但是在RL环境中，并不知道真实的值函数，只有奖励值，所以在实际中，我们会用目标（target）去替换真实的值函数$v_\pi(s)$，目标可以使用MC,$TD(0)$或$TD(\lambda)$的方法计算：

- 如果使用MC方法，目标是回报值G_t:

  $\Delta\mathbf{w}=\alpha\left(G_t-\hat{v}\left(S_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{v}\left(S_t,\mathbf{w}\right)$

- 如果使用$TD(0)$方法，则用TD目标$R_{t+1}+\gamma\hat{v}\left(S_{t+1},\mathbf{w}\right)$。

$$\Delta\mathbf{w}=\alpha\left(R_{t+1}+\gamma\hat{v}\left(S_{t+1},\mathbf{w}\right)-\hat{v}\left(S_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{v}\left(S_t,\mathbf{w}\right)$$

- 如果使用$TD(\lambda)$方法，则用$\lambda-return\ G_t^\lambda$

$$\Delta\mathbf{w}=\alpha\left(G_t^\lambda-\hat{v}\left(S_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{v}\left(S_t,\mathbf{w}\right)$$

使用MC方法的价值函数近似

如果使用MC方法，目标是回报值G_t，G_t是一个对真实价值函数$v_\pi(S_t)$的无偏差，但噪声很大的采样。我们在收集到很多训练数据后应用监督学习的方法进行训练。训练数据就是没个状态得到的回报对：

$$\left\langle S_1,G_1\right\rangle,\left\langle S_2,G_2\right\rangle,\ldots,\left\langle S_T,G_T\right\rangle$$

如果使用线性函数作为价值函数的近似，那么其MC方法下的梯度为：

$$\begin{aligned}\Delta\mathbf{w}&=\alpha\left(G_t-\hat{v}\left(S_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{v}\left(S_t,\mathbf{w}\right)\\&=\alpha\left(G_t-\hat{v}\left(S_t,\mathbf{w}\right)\right)\mathbf{x}\left(S_t\right)\end{aligned}$$

MC评估方法在应用中就算是使用非线性价值函数近似，也可以收敛到局部最优解，

使用TD方法的价值函数近似

如果使用TD方法，则用TD目标$R_{t+1}+\gamma\hat{v}\left(S_{t+1},\mathbf{w}\right)$，这是对真实价值函数$v_\pi(S_t)$的有偏差采样，其收集到的训练数据是这样的：

$$\left\langle S_1,R_2+\gamma\hat{v}\left(S_2,\mathbf{w}\right)\right\rangle,\left\langle S_2,R_3+\gamma\hat{v}\left(S_3,\mathbf{w}\right)\right\rangle,\ldots,\left\langle S_{T-1},R_T\right\rangle$$

如果使用线性函数作为价值函数的近似，那么其$TD(0)$方法下的梯度为：

$$\begin{aligned}\Delta\mathbf{w}&=\alpha\left(R+\gamma\hat{v}\left(S',\mathbf{w}\right)-\hat{v}(S,\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{V}(S,\mathbf{w})\\&=\alpha\delta\mathbf{x}(S)\end{aligned}$$

在收敛性上$TD(0)$几乎可以收敛到全局最优解。

使用$TD(\lambda)$方法的价值函数近似

如果使用$TD(\lambda)$方法，则用$\lambda-return\ G_t^\lambda$，其收集到的训练数据是这样的：

$$\left\langle S_1,G_1^\lambda\right\rangle,\left\langle S_2,G_2^\lambda\right\rangle,\ldots,\left\langle S_{T-1},G_{T-1}^\lambda\right\rangle$$

如果使用线性函数作为价值函数的近似，那么前向$TD(\lambda)$方法下的梯度为：

$$\begin{aligned}\Delta\mathbf{w}&=\alpha\left(G_t^\lambda-\hat{v}\left(S_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{V}\left(S_t,\mathbf{w}\right)\\&=\alpha\left(G_t^\lambda-\hat{v}\left(S_t,\mathbf{w}\right)\right)\mathbf{x}\left(S_t\right)\end{aligned}$$

后向$TD(\lambda)$方法下的梯度为：

$$\begin{aligned}\delta_t&=R_{t+1}+\gamma\hat{v}\left(S_{t+1},\mathbf{w}\right)-\hat{V}\left(S_t,\mathbf{w}\right)\\E_t&=\gamma\lambda E_{t-1}+\mathbf{x}\left(S_t\right)\\\Delta\mathbf{w}&=\alpha\delta_t E_t\end{aligned}$$

7、增量控制算法

控制的算法是指使用函数近似Q函数，$\hat{q}(S,A,\mathbf{w})\approx q_\pi(S,A)$。

在知道真实的action-value的时候，我们可以还是可以使用MSE作为目标函数：

$$J(\mathbf{w})=\mathbb{E}_\pi\left[\left(q_\pi(S,A)-\hat{q}(S,A,\mathbf{w})\right)^2\right]$$

然后用随机梯度下降（SGD）找到局部最优解：

$$
\begin{aligned}
-\frac{1}{2}\nabla_{\mathbf{w}}J(\mathbf{w}) &= \left(q_\pi(S,A)-\hat{q}(S,A,\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{q}(S,A,\mathbf{w}) \\
\Delta\mathbf{w} &= \alpha\left(q_\pi(S,A)-\hat{q}(S,A,\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{q}(S,A,\mathbf{w})
\end{aligned}
$$

使用线性函数近似Q函数

如果使用线性函数近似Q函数，我们将state-action对通过特征向量进行描述：

$$
\mathbf{x}(S,A)=\begin{pmatrix}\mathbf{x}_1(S,A)\\ \vdots \\ \mathbf{x}_n(S,A)\end{pmatrix}
$$

然后Q函数就可以表示为：

$$
\hat{q}(S,A,\mathbf{w})=\mathbf{x}(S,A)^\top\mathbf{w}=\sum_{j=1}^{n}\mathbf{x}_j(S,A)\mathbf{w}_j
$$

其SGD的更新方式为：

$$
\begin{aligned}
\nabla_{\mathbf{w}}\hat{q}(S,A,\mathbf{w}) &= \mathbf{x}(S,A) \\
\Delta\mathbf{w} &= \alpha\left(q_\pi(S,A)-\hat{q}(S,A,\mathbf{w})\right)\mathbf{x}(S,A)
\end{aligned}
$$

使用MC方法的Q函数近似

使用MC方法的Q函数近似，是将回报值G_t代替真实的Q函数值$q_\pi(S,A)$，其更新方式为：

$$
\Delta\mathbf{w}=\alpha\left(G_t-\hat{q}\left(S_t,A_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{q}\left(S_t,A_t,\mathbf{w}\right)
$$

使用$TD(0)$方法的Q函数近似

使用$TD(0)$方法的Q函数近似，是将$R_{t+1}+\gamma Q\left(S_{t+1},A_{t+1}\right)$代替真实的Q函数值$q_\pi(S,A)$，其更新方式为：

$$
\Delta\mathbf{w}=\alpha\left(R_{t+1}+\gamma\hat{q}\left(S_{t+1},A_{t+1},\mathbf{w}\right)-\hat{q}\left(S_t,A_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{q}\left(S_t,A_t,\mathbf{w}\right)
$$

使用$TD(\lambda)$方法的Q函数近似

使用前向$TD(\lambda)$方法的Q函数近似，是将λ回报代替真实的Q函数值$q_\pi(S,A)$，其更新方式为：

$$
\Delta\mathbf{w}=\alpha\left(q_t^\lambda-\hat{q}\left(S_t,A_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{q}\left(S_t,A_t,\mathbf{w}\right)
$$

其等价的使用后向观点的方法中，使用资格迹进行如下更新：

$$
\begin{aligned}
\delta_t &= R_{t+1}+\gamma\hat{q}\left(S_{t+1},A_{t+1},\mathbf{w}\right)-\hat{q}\left(S_t,A_t,\mathbf{w}\right) \\
E_t &= \gamma\lambda E_{t-1}+\nabla_{\mathbf{w}}\hat{q}\left(S_t,A_t,\mathbf{w}\right) \\
\Delta\mathbf{w} &= \alpha\delta_t E_t
\end{aligned}
$$

8、收敛性

几种算法之间的收敛性比较在下面几张图。

首先是预测算法的收敛性，在On-policy与Off-policy下都是MC最优。

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

从预测算法的收敛性上可以看到TD算法的Off-policy上的收敛性较差，但是如果使用Gradient TD会好很多。

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD	✓	✓	✗
	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
	TD	✓	✗	✗
	Gradient TD	✓	✓	✓

在控制算法的收敛性上的比较，其中 (✓) 表示接近最优值函数。

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

9、批量方法（Batch Method）

前面讲的部分都是使用梯度下降进行增量更新，以使得目标函数接近最优解。但是其采样效率很低，批量方法进行批量得对训练数据进行学习，具有更高的效率。批方法要在最大化利用数据的同时去找到对于值函数最好的拟合。

MSE

假设给定近似函数$\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$，并且给定包含大量state-value对的经验池D，

$$\mathcal{D} = \left\{\left\langle s_1, v_1^\pi \right\rangle, \left\langle s_2, v_2^\pi \right\rangle, \ldots, \left\langle s_T, v_T^\pi \right\rangle\right\}$$

我们需要找到一个最佳的参数w能够使得函数$\hat{v}(s,\mathbf{w})$能够拟合经验池D中的所有数据，MSE作为目标函数就可以找到最优解：

$$
\begin{aligned}
LS(\mathbf{w}) &= \sum_{t=1}^{T} \left(v_t^{\pi} - \hat{v}\left(s_t, \mathbf{w}\right)\right)^2 \\
&= \mathbb{E}_{\mathcal{D}}\left[\left(v^{\pi} - \hat{v}(s, \mathbf{w})\right)^2\right]
\end{aligned}
$$

使用经验回放的SGD

在给定经验池时，我们可以重复下面两个步骤应用SGD：

- 从经验池D中进行（state，value）对的采样：$\langle s, v^{\pi} \rangle \sim \mathcal{D}$
- 应用SGD进行参数更新：$\Delta \mathbf{w} = \alpha \left(v^{\pi} - \hat{v}(s, \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$

这样的结果可以收敛到最小二乘解：$\mathbf{w}^{\pi} = \underset{\mathbf{w}}{\operatorname{argmin}}\, LS(\mathbf{w})$

DQN算法

DQN算法的成功就是主要使用了经验池与固定的目标Q网络两种技术，其关键在于以下几点：

- 1. 使用$\epsilon - greedy$策略选取动作a_t
  2. 将产生的转移数据$(s_t, a_t, r_{t+1}, s_{t+1})$存储到经验池$D$中
  3. 从经验池D中随机采样一个小批量的经验数据$(s_t, a_t, r_{t+1}, s_{t+1})$
  4. 根据旧的，并且固定的参数w^{-}计算Q函数的目标值
  5. 优化Q-network和Q函数目标值之间的目标函数MSE：

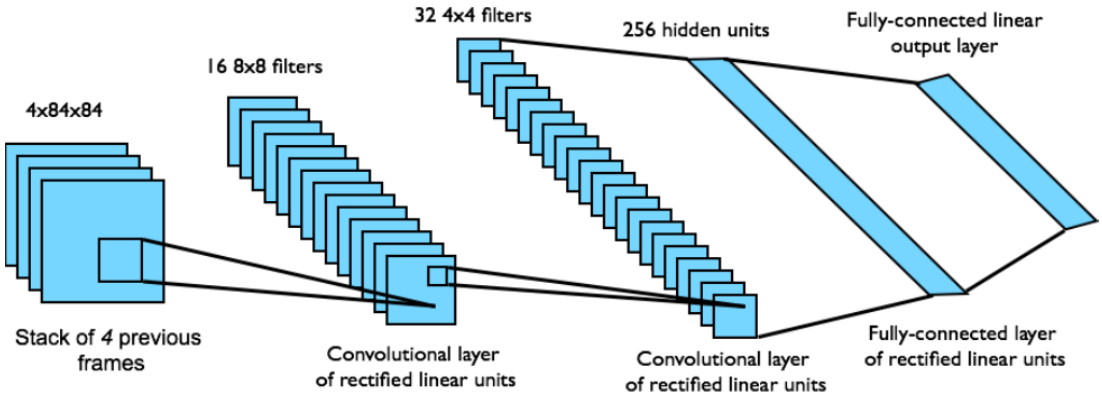
$$
\mathcal{L}_i\left(w_i\right) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; w_i^{-}\right) - Q\left(s, a; w_i\right)\right)^2\right]
$$

  6. 使用SGD进行参数更新

DQN算法在Atari游戏中的应用

DQN算法在Atari游戏训练过程中，端到端得从游戏像素s中学习Q函数$Q(s, a)$，其状态s是根据游戏最后4帧画面叠加而成，然后输出游戏手柄中18个动作按钮的Q值，其奖励就是在执行动作后游戏中的分数。

网络拓扑图如下所示：



线性最小二乘预测

在使用经验池的时候我们可以找到最小二乘解，但是知道最优解还是会花费非常多的迭代。如果使用线性近似函数$\hat{v}(s,\mathbf{w})=\mathbf{x}(s)^{\top}\mathbf{w}$，我们就可以直接找到最小二乘解。

在最小化$LS(w)$的过程中，我们$LS(w)$的期望为0进行推导：

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[\Delta\mathbf{w}]&=0\\ \alpha\sum_{t=1}^{T}\mathbf{x}\left(s_t\right)\left(v_t^{\pi}-\mathbf{x}(s_t)^{\top}\mathbf{w}\right)&=0\\ \sum_{t=1}^{T}\mathbf{x}\left(s_t\right)v_t^{\pi}&=\sum_{t=1}^{T}\mathbf{x}\left(s_t\right)\mathbf{x}(s_t)^{\top}\mathbf{w}\\ \mathbf{w}&=\left(\sum_{t=1}^{T}\mathbf{x}\left(s_t\right)\mathbf{x}(s_t)^{\top}\right)^{-1}\sum_{t=1}^{T}\mathbf{x}\left(s_t\right)v_t^{\pi}\end{aligned}$$

如果有N个特征，这种直接求解的复杂度是$O(N^3)$，如果使用Shermann-Morrison方法进行迭代更新可以将复杂度降低到$O(N^2)$

在不知道真实价值v_t^{π}的时候，在实际中就需要使用到MC,TD的方法进行采样代替v_t^{π}。

- 如果使用MC方法，就有LSMC（Least Squares Monte-Carlo）算法，使用回报代替真值：$v_t^{\pi}\approx G_t$
- 如果使用TD方法，就有LSTD（Least Squares Temporal-Di□erence）算法，$v_t^{\pi}\approx R_{t+1}+\gamma\hat{v}\left(S_{t+1},\mathbf{w}\right)$
- 如果使用$TD(\lambda)$方法，则使用λ回报代替真值:$v_t^{\pi}\approx G_t^{\lambda}$

在这三种方法的时候都可以直接进行求解最小二乘解：

对于LSMC:

$$\begin{aligned}0&=\sum_{t=1}^{T}\alpha\left(G_t-\hat{v}\left(S_t,\mathbf{w}\right)\right)\mathbf{x}\left(S_t\right)\\ \mathbf{w}&=\left(\sum_{t=1}^{T}\mathbf{x}\left(S_t\right)\mathbf{x}(S_t)^{\top}\right)^{-1}\sum_{t=1}^{T}\mathbf{x}\left(S_t\right)G_t\end{aligned}$$

对于LSTD：

$$0=\sum_{t=1}^{T}\alpha\left(R_{t+1}+\gamma\hat{v}\left(S_{t+1},\mathbf{w}\right)-\hat{v}\left(S_t,\mathbf{w}\right)\right)\mathbf{x}\left(S_t\right)$$

$$\mathbf{w}=\left(\sum_{t=1}^{T}\mathbf{x}\left(S_t\right)\left(\mathbf{x}\left(S_t\right)-\gamma\mathbf{x}\left(S_{t+1}\right)\right)^{\top}\right)^{-1}\sum_{t=1}^{T}\mathbf{x}\left(S_t\right)R_{t+1}$$

对于LSTD(λ):

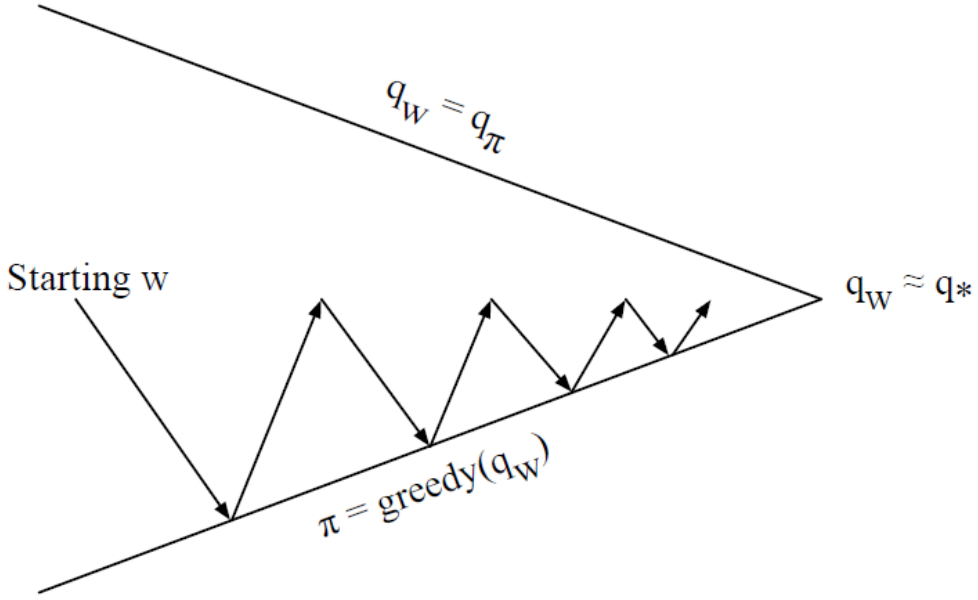
$$\begin{aligned}0&=\sum_{t=1}^{T}\alpha\delta_t E_t\\ \mathbf{w}&=\left(\sum_{t=1}^{T}E_t(\mathbf{x}\left(S_t\right)-\gamma\mathbf{x}\left(S_{t+1}\right))^{\top}\right)^{-1}\sum_{t=1}^{T}E_t R_{t+1}\end{aligned}$$

这三种预测算法的收敛性如下：

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✗	✗
	LSTD	✓	✓	-

最小二乘的策略迭代

使用最小二乘方法进行策略迭代的过程，只是在策略评估过程中使用least square Q-learning。



线性最小二乘控制

如果线性函数对q函数$q_\pi(s,a)$进行近似，

$$\hat{q}(s,a,\mathbf{w}) = \mathbf{x}(s,a)^\top \mathbf{w} \approx q_\pi(s,a)$$

然后最小化真实$q_\pi(s,a)$与预测的$\hat{q}(s,a,\mathbf{w})$之间的MSE，其中$\hat{q}(s,a,\mathbf{w})$来自于经验池的⟨(state,action), value⟩对：

$$\mathcal{D} = \left\{ \left\langle (s_1, a_1), v_1^\pi \right\rangle, \left\langle (s_2, a_2), v_2^\pi \right\rangle, \ldots, \left\langle (s_T, a_T), v_T^\pi \right\rangle \right\}$$

在策略评估中，我们是要去高效得利用已有的经验，但是在控制中，我们是要去提升策略。经验池中的经验是从许多的策略中采样产生的，所以对$q_\pi(s,a)$进行评估必须进行off-policy学习，需要使用和Q-learning一样的想法：

- 使用旧的策略产生经验：$S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{\text{old}}$
- 根据后续状态产生动作：$A' = \pi_{\text{new}}(S_{t+1})$
- 更新q函数$\hat{q}(S_t, A_t, \mathbf{w})$：$R_{t+1} + \gamma \hat{q}(S_{t+1}, A', \mathbf{w}))$

然后我们继续考虑线性函数的增量更新方式为：

$$\begin{aligned}\delta &= R_{t+1} + \gamma \hat{q}\left(S_{t+1}, \pi\left(S_{t+1}\right), \mathbf{w}\right) - \hat{q}\left(S_t, A_t, \mathbf{w}\right) \\ \Delta \mathbf{w} &= \alpha \delta \mathbf{x}\left(S_t, A_t\right)\end{aligned}$$

如果使用最小二乘的方法直接求解：

$$0 = \sum_{t=1}^{T} \alpha\left(R_{t+1} + \gamma \hat{q}\left(S_{t+1}, \pi\left(S_{t+1}\right), \mathbf{w}\right) - \hat{q}\left(S_t, A_t, \mathbf{w}\right)\right) \mathbf{x}\left(S_t, A_t\right)$$

$$\mathbf{w} = \left(\sum_{t=1}^{T} \mathbf{x}\left(S_t, A_t\right)\left(\mathbf{x}\left(S_t, A_t\right) - \gamma \mathbf{x}\left(S_{t+1}, \pi\left(S_{t+1}\right)\right)\right)^{\top}\right)^{-1} \sum_{t=1}^{T} \mathbf{x}\left(S_t, A_t\right) R_{t+1}$$

我们利用这里的LSTDQ对值函数进行估计，得到下面的LSPI算法：

function LSPI-TD$(\mathcal{D}, \pi_0)$
  $\pi' \leftarrow \pi_0$
  repeat
    $\pi \leftarrow \pi'$
    $Q \leftarrow$ **LSTDQ**$(\pi, \mathcal{D})$
    for all $s \in \mathcal{S}$ **do**
      $\pi'(s) \leftarrow \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s, a)$
    end for
  until $(\pi \approx \pi')$
  return π
end function

控制算法的收敛性：

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
LSPI	✓	(✓)	-