强化学习基本原理及编程实现05：基于函数逼近的强化学习

郭宪

2019.10.27

人工智能学院
College of Artificial Intelligence

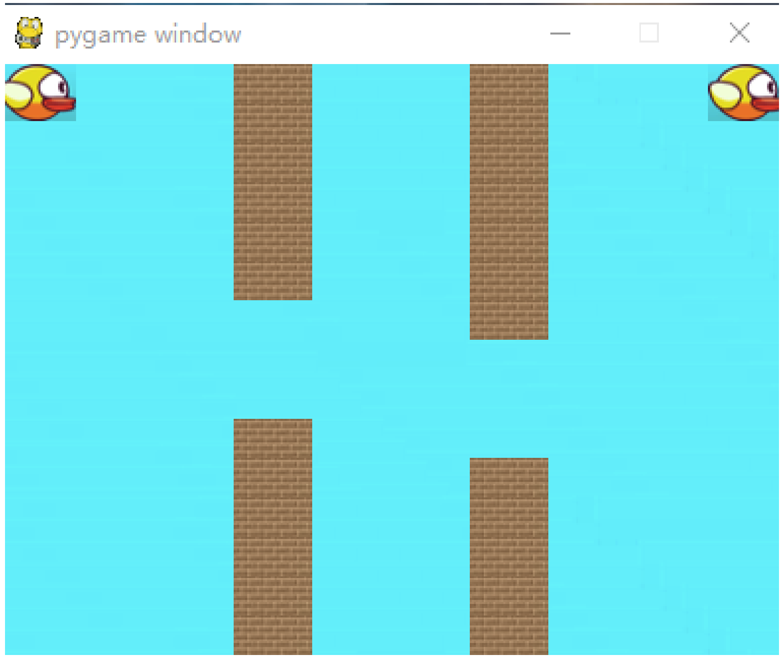
南開大學
Nankai University

第四次作业

1.阅读《Reinforcement Learning： An Introduction》第五、六章

2. 利用MC方法和TD方法实现右图游戏

3. 利用MC方法和TD方法实现你自己的小游戏



表格型强化学习算法

动态规划值函数迭代算法

输入：状态转移概率 $P_{ss'}^a$,回报函数R_s^a ，折扣因子γ

初始化值函数：$v(s) = 0$　初始化策略 π_0

Repeat l=0,1,…

for every s do

$$v_{l+1}(s) = \max_a R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_l(s')$$

Until $v_{l+1} = v_l$

输出：$\pi(s) = \underset{a}{\operatorname{argmax}} R_s^a + \gamma \sum_{s'\in S} P_{ss'}^a v_l(s')$

[1] 初始化所有：$s \in S, a \in A(s), Q(s,a) \leftarrow arbitrary$

$Returns(s,a) \leftarrow empty\ list$

$\pi(s) \leftarrow arbitrary\ \varepsilon\text{-}soft$策略,

Repeat:

[2] 从 S_0, A_0 开始以策略 π 生成一次实验(episode),

[3] 对每对在这个实验中出现的状态和动作，s, a:

$G \leftarrow s,a$第一次出现后的回报

将G附加于回报Returns(s, a)上

策略评估

$Q(s,a) \leftarrow average(Returns(s,a))$对回报取均值

[4] 对该实验中的每一个s:

策略改进

$$\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|A(s)|} & if\ a = \arg\max_a Q(s,a) \\ \dfrac{\varepsilon}{|A(s)|} & if\ a \neq \arg\max_a Q(s,a) \end{cases}$$

Sarsa：On-Policy TD

1. 初始化 $Q(s,a), \forall s \in S, a \in A(s)$，给定参数$\alpha$，$\gamma$

行动策略和评估策略都是 ε 贪婪策略

2. Repeat：

给定起始状态 s，并根据ε贪婪策略在状态 s 选择动作 a

Repeat（对于一幕的每一步）

（a）根据 ε 贪婪策略在状态 s 选择动作 a，得到回报 r 和下一个状态s’，在状态 s’ 根据 ε 贪婪策略得到动作a’

（b）$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$

（c）s=s’，a=a’

Until s 是终止状态

Until 所有的$Q(s,a)$收敛

3. 输出最终策略：$\pi(s) = \arg\max_a Q(s,a)$

Qlearning: Off-policy TD

1. 初始化 $Q(s,a), \forall s \in S, a \in A(s)$, 给定参数$\alpha$，$\gamma$

2. Repeat:

给定起始状态 s，并根据 ε贪婪策略在状态 s 选择动作 a

Repeat(对于一幕的每一步)

行动策略为 ε 贪婪策略

$\boxed{\text{(a) 根据 } \varepsilon \text{ 贪婪策略在状态 s\_t 选择动作a\_t，得到回报r\_t 和下一个状态 } s_{t+1}}$

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

目标策略为贪婪策略

(c) s=s’，a=a’

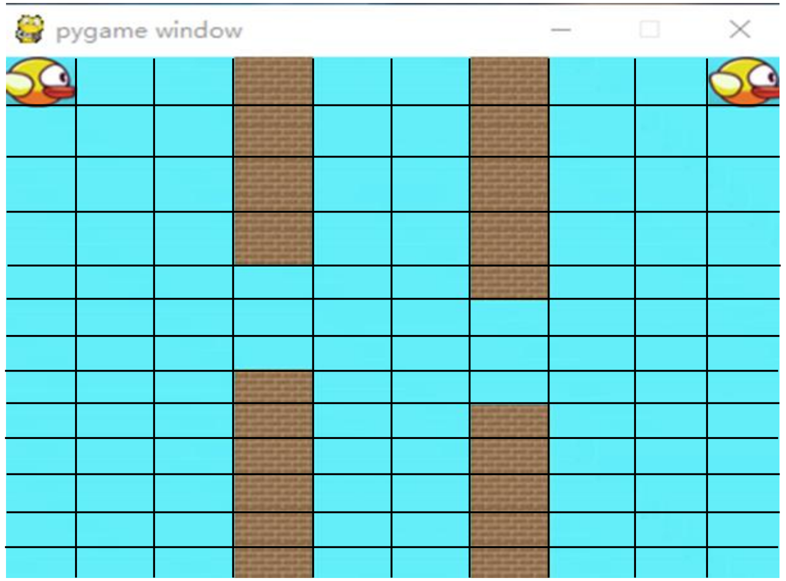
Until s 是终止状态

Until 所有的 $Q(s,a)$ 收敛

3. 输出最终策略：$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$

值函数的表示

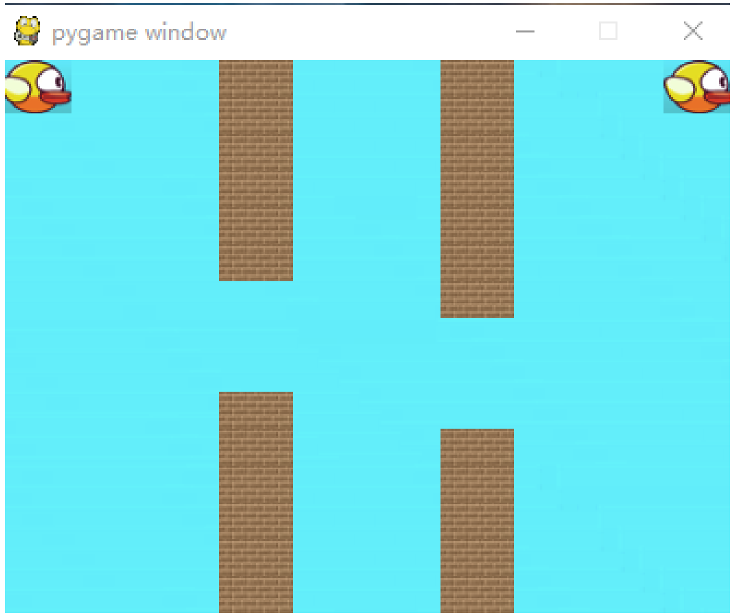
$q(s,a)$



行为值函数的数目： $|S|\cdot|A|$

大的MDP问题：状态空间连续或状态数无穷多

1. 存储空间无穷大
2. 学习速度很慢

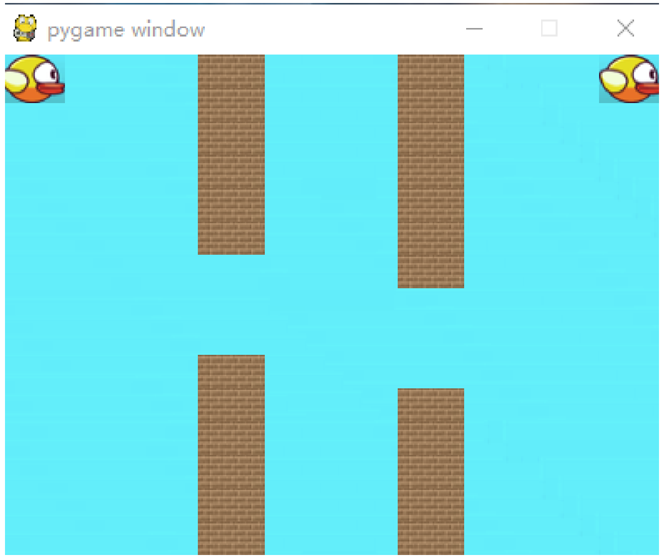


状态空间表示为：图像，如分辨率为（400，500）图像空间的大小为 256^{20000}

1.存储空间无穷大，计算时间无穷。

2. 几乎每次遇到的状态下次都不会再遇到（泛化能力），需要泛化能力强

值函数的表示

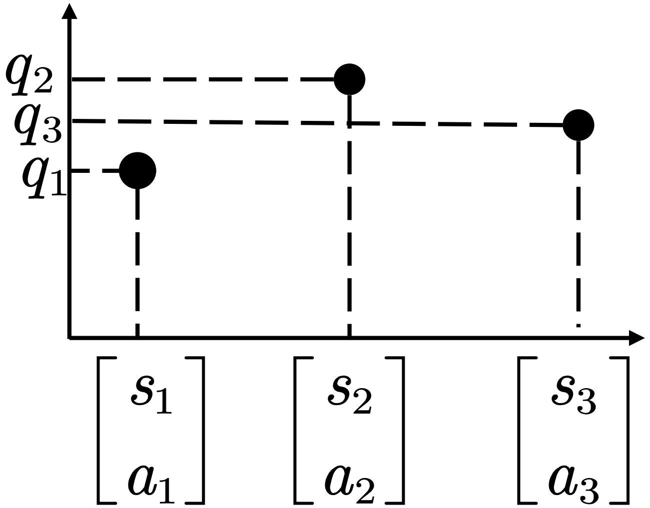


采集到的数据：

$$Q(s,a)=\{q_1,q_2,\cdots,q_n\}$$

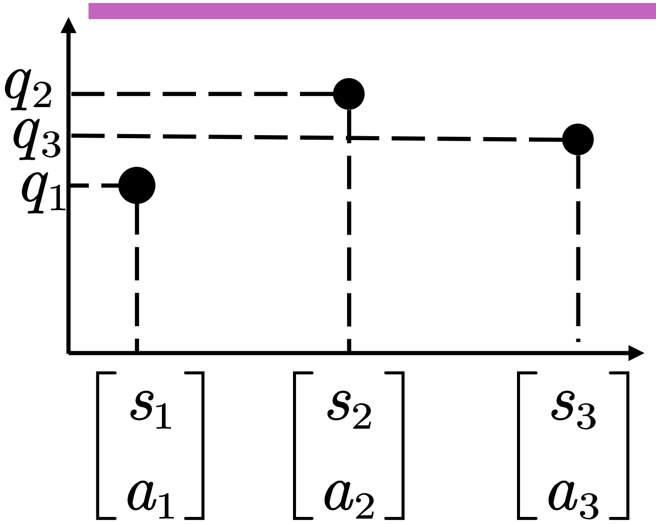
如何得到其他状态行为值函数?

$$Q(s',a')$$



利用泛化方法得到其他状态行为处的值函数

值函数的表示



利用泛化方法得到其他状态行为处的值函数

基于函数逼近的强化学习：

强化学习+泛化方法

泛化方法：函数逼近理论，机器学习！

Remark: 不同的地方，非静态性，自举和延迟目标

利用函数逼近方法估计值函数 $\hat{q}(s,a;\theta)$

1. 参数化逼近
2. 非参数化逼近，如基于核的方法

参数化方法：

线性参数化：各种人为基底

非线性参数化：人工神经网络，决策树，模糊网络

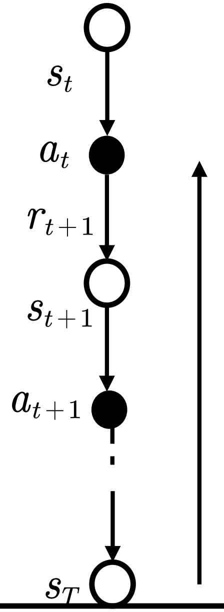
值函数估计过程

表格型值函数估计

Backup 值：

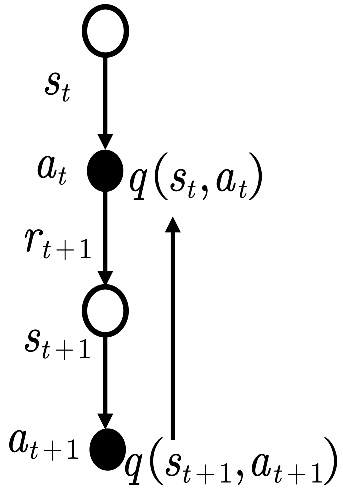
DP $Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s')\, Q_\pi(s',a')$

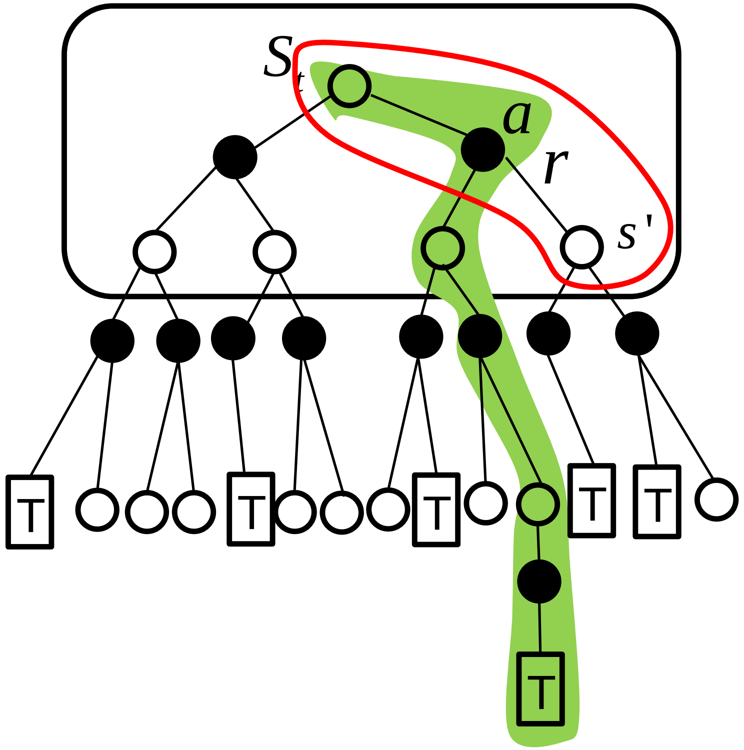
MC的回溯值为：$\sum_{k=1}^{N} \gamma^{k-1} r_{t+k}$



TD回溯值为：

$r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$



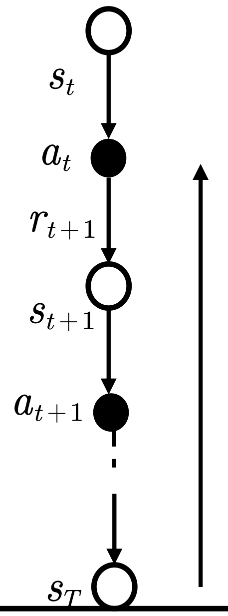


值函数估计过程

Backup 值：

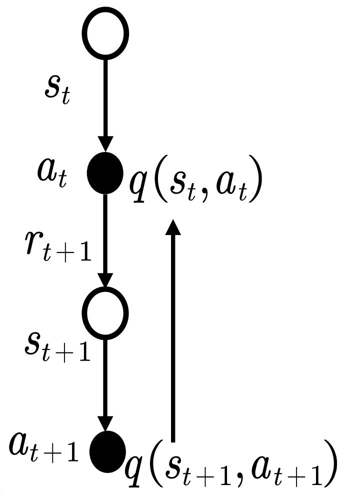
DP $Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s')\, Q_\pi(s',a')$

MC的回溯值为：$\sum_{k=1}^{N} \gamma^{k-1} r_{t+k}$

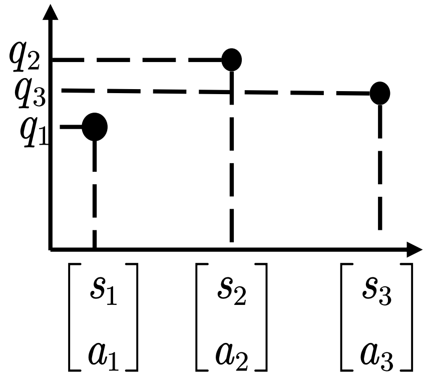


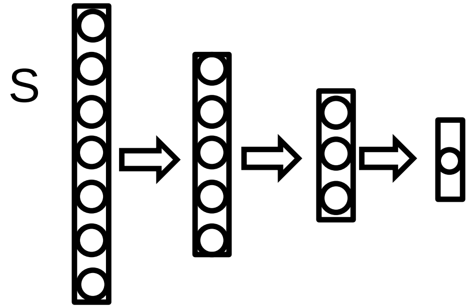
TD回溯值为：

$r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$



函数逼近：$\hat{q}(s,a,\theta)$





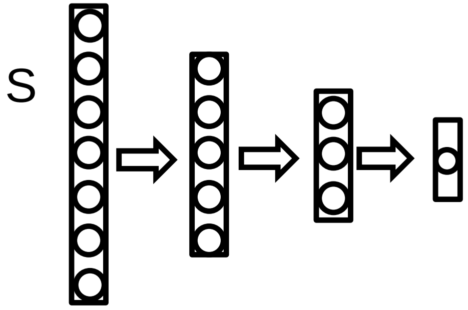
$\hat{q}(s,a,\theta)$

训练目标：$\arg\min_{\theta} \in \left(q(s,a) - \hat{q}(s,a,\theta)\right)^2$

强化学习：在线学习

值函数逼近的损失函数

函数逼近： $\hat{q}(s,a,\theta)$



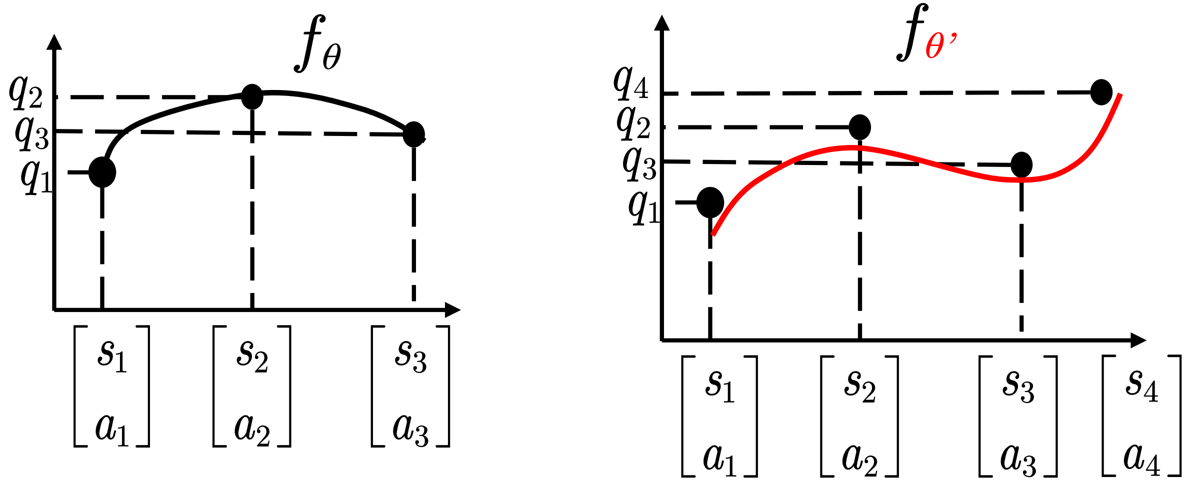
$\hat{q}(s,a,\theta)$

训练目标： $\arg\min_{\theta} \in \left(q(s,a) - \hat{q}(s,a,\theta)\right)^2$

强化学习：在线学习

参数改变后，其他状态值函数跟着也发生变化

非静态目标函数：增量式学习方法和批方法



目标函数的构建：

$$\overline{VE}(w) = \sum_{s \in \mathcal{S}} \mu(s) \left[q_\pi(s,a) - \hat{q}(s,w) \right]^2$$

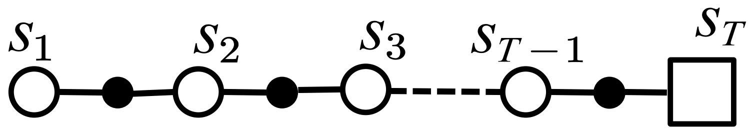
随机梯度下降

随机梯度下降法：

$$\theta_{t+1} = \theta_t + \alpha[U_t(s,a) - \hat{q}(s_t,a;\theta)]\nabla\hat{q}(s_t,a_t;\theta)$$

对于蒙特卡罗方法：$U_t(s) = G_t(s)$

给定策略 π 产生一次试验：



值函数估计过程：

监督学习，训练数据集为：$\langle s_1, G_1\rangle, \langle s_2, G_2\rangle, \cdots$

策略评估过程为：

$$\Delta\theta = \alpha[U_t(s,a) - \hat{q}(s_t,a;\theta)]\nabla\hat{q}(s_t,a_t;\theta)$$

α 比较小，用来平衡不同状态值函数的误差

基于梯度的蒙塔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 π 产生一幕数据 $S_0, A_0, R_1, S_1, A_1, \cdots, R_T, S_T$

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

$$\theta \leftarrow \theta + \alpha[G_t - \hat{q}(s_t,a;\theta)]\nabla\hat{q}(s_t,a;\theta)$$

半梯度下降

对于TD(0), DP等

Bootstrapping

$$U_t = R_{t+1} + \gamma \boxed{\hat{q}(S_{t+1}, a; \theta)}$$

半梯度方法:

$$\theta_{t+1} = \theta_t + \alpha[U_t(s,a) - \hat{q}(s_t, a; \theta)]\nabla\hat{q}(s_t, a_t; \theta)$$

$U_t(s)$ 依赖于当前的参数估计 θ

只考虑参数 θ 对估计值函数 $q(s_t, a_t; \theta_t)$ 的影响而忽略对目标函数 u_t 的 影响，称为半梯度法。

好处：学习速度快，可以应用到连续系统中

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 $\upsilon(S_t, \theta_t)$

采用动作 A 并观测回报 R, S'

$$\theta \leftarrow \theta + \alpha[G_t - \hat{q}(s_t, a; \theta)]\nabla\hat{q}(s_t, a; \theta)$$

$$S \leftarrow S'$$

直到 S’ 是终止状态

半梯度 Sarsa 算法

输入：一个要逼近的可微动作值函数：$\hat{q}:S \times A \times R^n \rightarrow R$ 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：

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 $\hat{q}(S',A',\theta)$

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

$S \leftarrow S'$

$A \leftarrow A'$

值函数的线性逼近

基于梯度或半梯度的的值函数逼近：

$$\theta_{t+1}=\theta_t+\alpha[U_t(s)-\hat{q}(s_t,a;\theta_t)]\nabla\hat{q}(s_t,a;\theta_t)$$

线性逼近：

$$\hat{q}(s,a;\theta)=\theta^T\phi(s,a)$$

线性逼近的好处：在线性情况，仅有一个最优值，

因此可收敛到全局最优。

$\phi(s)$ 称为状态 s 的特征函数。

常用的基函数类型：

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

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

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

蒙特卡罗方法值函数更新：

$$\Delta\theta=\alpha[U_t(s)-\hat{q}(s_t,a_t;\theta_t)]\nabla\hat{q}(s_t,a;\theta_t)$$

$$=\alpha[G_t-\theta^T\phi]\phi$$

TD(0)线性逼近值函数更新为：

$$\Delta\theta=\alpha[R+\gamma\theta^T\phi(s')-\theta^T\phi(s)]\phi(s)$$

$$=\alpha\delta\phi(s)$$

正向视角 $TD(\lambda)$

$$\Delta\theta=\alpha(G_t^\lambda-\theta^T\phi)\phi$$

反向视角 $TD(\lambda)$

$$\delta_t=R_{t+1}+\gamma\theta^T\phi(s')-\theta^T\phi(s)$$

$$E_t=\gamma\lambda E_{t-1}+\phi(s)$$

$$\Delta\theta=\alpha\delta_t E_t$$

批方法

增量式方法：计算简单，但往往存在样本效率不高的缺点。

批的方法寻找最好的拟合函数值 $\hat{q}(s,a;\theta)$ 。

给定经验数据集：

$$D=\{\langle s_1,v_1^{\pi}\rangle,\langle s_2,v_2^{\pi}\rangle,\cdots,\langle s_T,v_T^{\pi}\rangle\}$$

最小二乘方法：找到最优参数

$$LS(\theta)=\sum_{t=1}^{T}(q_t^{\pi}-\hat{q}(s_t,a;\theta))^2$$

$$=E_D[(q^{\pi}-\hat{q}(s,a;\theta))^2]$$

线性最小二乘逼近

$$\Delta\theta=\alpha\sum_{t=1}^{T}[q_t^{\pi}-\theta^T\phi(s_t,a)]\phi(s_t)=0$$

最小二乘蒙特卡罗方法：

LSMC:
$$\theta=\left(\sum_{t=1}^{T}\phi(s_t)\phi(s_t)^T\right)^{-1}\sum_{t=1}^{T}\phi(s_t)G_t$$

最小二乘时间差分方法：

LSTD:
$$\theta=\left(\sum_{t=1}^{T}\phi(s_t)(\phi(s_t)\text{-}\gamma\phi(\mathrm{s}_{t+1}))^T\right)^{-1}\sum_{t=1}^{T}\phi(s_t)R_{t+1}$$

最小二乘TD(λ)方法：

LSTD（λ）：
$$\theta=\left(\sum_{t=1}^{T}\mathrm{E}_t(\phi(s_t)\text{-}\gamma\phi(\mathrm{s}_{t+1}))^T\right)^{-1}\sum_{t=1}^{T}E_tR_{t+1}$$

表格型强化学习是一种特殊的函数逼近

行为值函数表示：

$$Q(s,a)=\phi(s,a)^T\Theta$$

表格型值函数可以看成是函数逼近方法的一种特殊形式，每个格点表示一个特征。我们以驾鸯系统为例，以表格的形式表示的行为-值函数共有 100×4 个元素，每个元素对应一个特征，则特征函数可以写为：

此时，参数 Θ 相应于原表格表示的行为值函数

固定稀疏表示

状态空间的维数为 n 维，即 $s=[s_1,\cdots,s_n]$，每个维度离散化为 d 个数，则状态空间第 i 维共有 d 个数，记为 υ_i^j，其中 $j=1,\cdots,d$。用固定稀疏来表示状态的特征为：

$$\phi(s)=[\phi_{11},\phi_{12},\cdots,\phi_{1d},\phi_{21},\cdots,\phi_{2d},\cdots,\phi_{n1},\cdots,\phi_{nd}]^T \qquad (6.3)$$

状态-行为值函数的特征可表示为：

$$\phi(s,a)=[\phi^1(s),\cdots,\phi^{|A|}(s)] \qquad (6.5)$$

固定稀疏表示的特征的个数和参数的个数为：$d\times n\times|A|$，该表示特征的个数随维数线性增长而非指数增长。

函数逼近方法

- 函数逼近方法
  - 基于非参数的函数逼近
    - 基于核的方法
    - 基于高斯回归的方法
  - 基于参数的函数逼近
    - 基于线性参数
    - 基于非线性参数
      - 神经网络

前向神经网络的正向传播

神经网络可看成是基函数参数化的一种方法

$z_i^{(l)}$ 表示第 l 层第 i 个单元输入加权和

$$z_i^{(2)}=\sum_{j=1}^{n_1} W_{ij}^{(1)} x_j + b_i^{(1)},\quad z_i^{(3)}=\sum_{j=1}^{n_2} W_{ij}^{(2)} x_j + b_i^{(2)}$$

$a_i^{(l)}$ 表示第 l 层的第 i 个单元激活，则 $a_i^{(l)}=f(z_i^{(l)})$。

神经网络的前向计算为：

$$z^{(2)}=W^{(1)}x+b^{(1)}$$

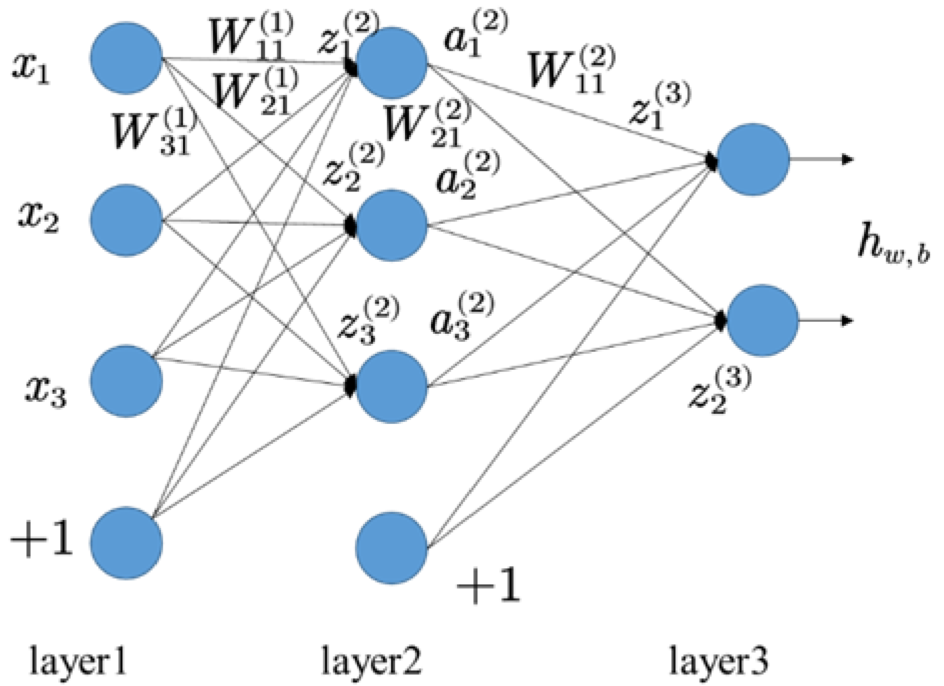
$$a^{(2)}=f(z^{(2)})$$

$$z^{(3)}=W^{(2)}a^{(2)}+b^{(2)}$$

$$h_{W,b}(x)=a^{(3)}=f(z^{(3)})$$

参数化的基函数

输出层： $y=z^{(n_l)}=W^{(n_l-1)}a^{(n_l-1)}+b^{(n_l-1)}$



前向神经网络的结构

前向神经网络的后向传播

已知训练样本：$T=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),\cdots,(x^{(N)},y^{(N)})\}$

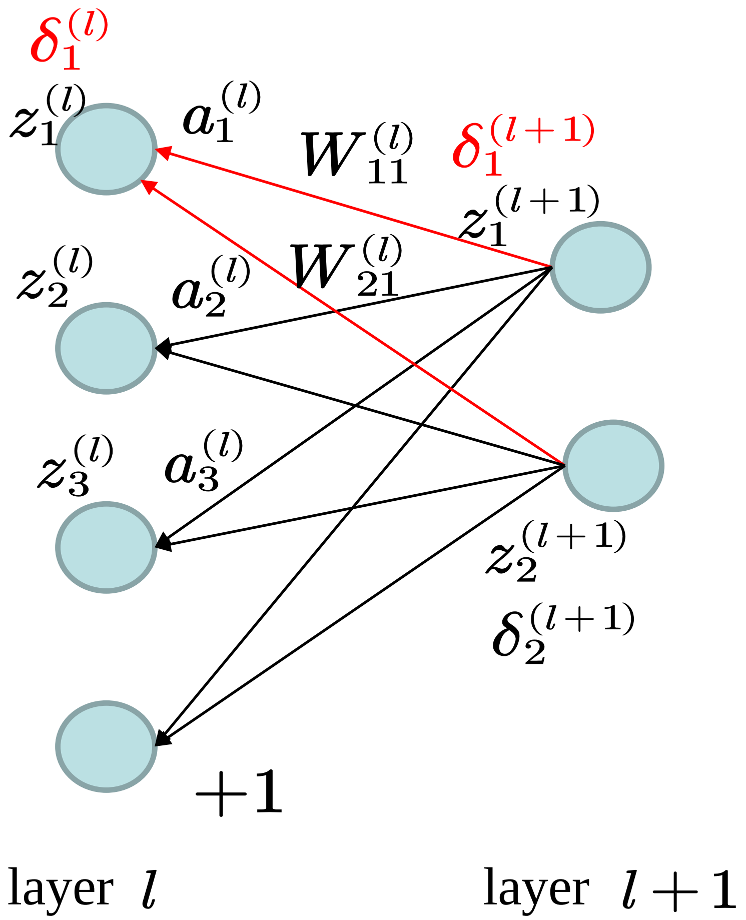
优化函数为：$J(W,b;x^{(i)},y^{(i)})=\frac{1}{2}\|h_{W,b}(x^{(i)})-y^{(i)}\|^2$

利用梯度下降法更新权值和偏置：

$$W_{ij}^{(l)}=W_{ij}^{(l)}-\alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)$$

$$b_i^{(l)}=b_i^{(l)}-\alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b)$$

关键是计算导数



前向神经网络的后向传播

Step1: 正向计算每层的输出

$$z^{(l+1)}=W^{(l)}a^{(l)}+b^{(l)}$$

$$a^{(l+1)}=f\left(z^{(l+1)}\right)$$

Step2: 计算最后一层的残差

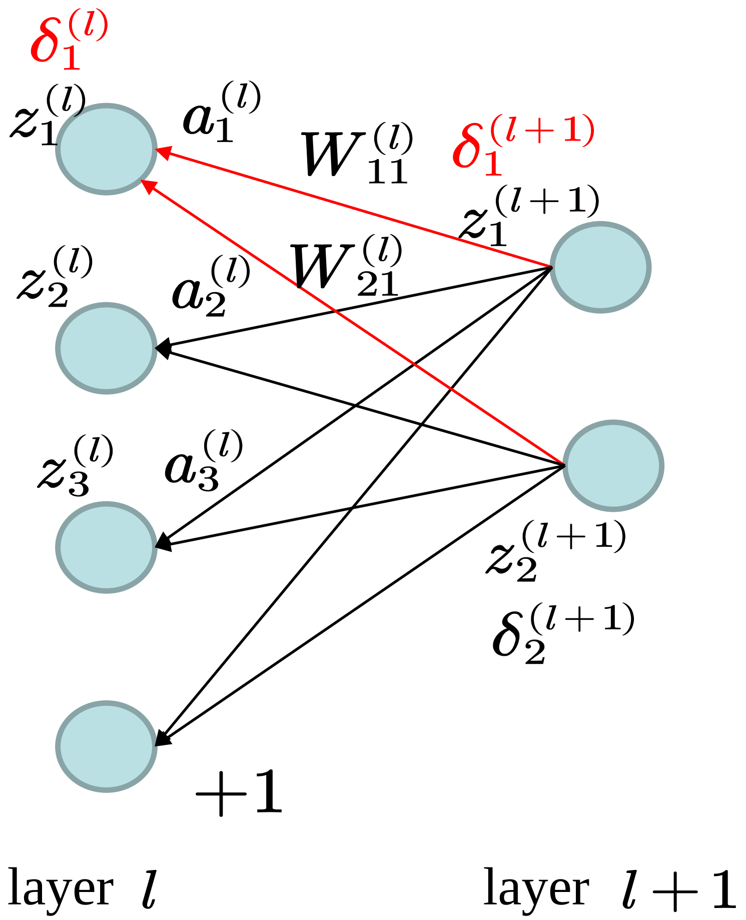
$$\delta_i^{(n_l)}=\frac{\partial}{\partial z_i^{n_l}}J(W,b;x,y)=\frac{\partial}{\partial z_i^{n_l}}\frac{1}{2}\|y-h_{W,b}(x)\|^2$$

$$=\frac{\partial}{\partial z_i^{n_l}}\frac{1}{2}\sum_{j=1}^{S_{n_l}}\left(y_j-f\left(z_j^{(n_l)}\right)\right)^2$$

$$=-\left(y_i-a_i^{(n_l)}\right)\cdot f'\left(z_i^{(n_l)}\right)$$

Step3: 残差从输出层往后逐渐传播

$$\delta_i^{(l)}=\left(\sum_{j=1}^{s_{l+1}}W_{ji}^{(l)}\delta_j^{(l+1)}\right)f'\left(z_i^{(l)}\right)$$



前向神经网络的后向传播

Step3: 残差从输出层往后逐渐传播

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

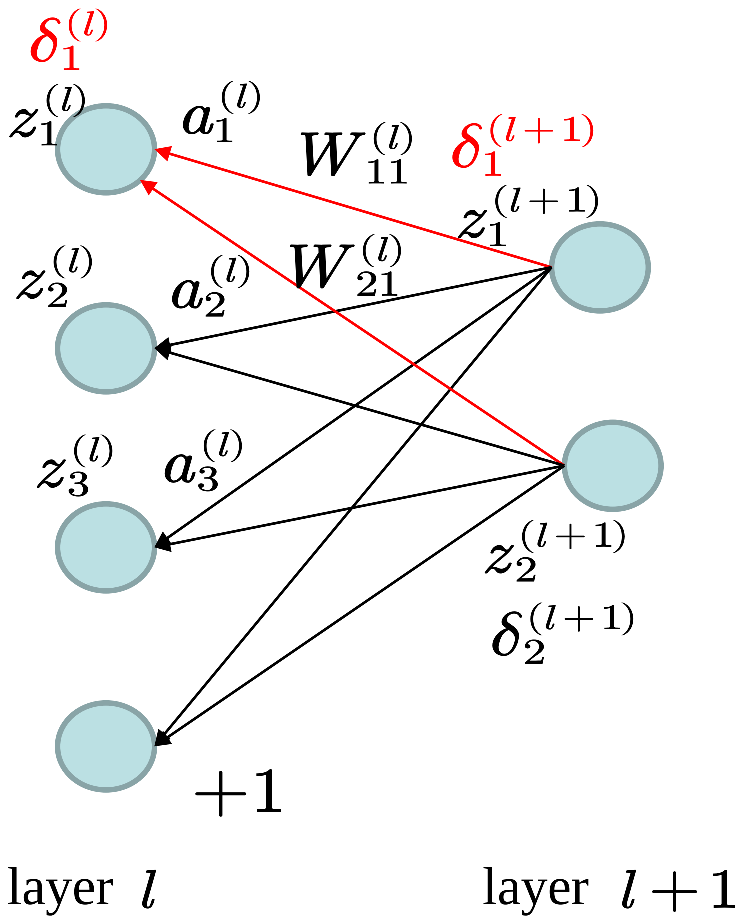
推导过程：

$$\delta_i^{(n_l-1)} = \frac{\partial}{\partial z_i^{n_l-1}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - a_j^{(n_l)})^2$$

$$= \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - f(z_j^{(n_l)}))^2$$

$$= \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot \frac{\partial}{\partial z_i^{(n_l-1)}} f(z_j^{(n_l)})$$

$$= \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot f'(z_j^{(n_l)}) \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{(n_l-1)}}$$



前向神经网络的后向传播

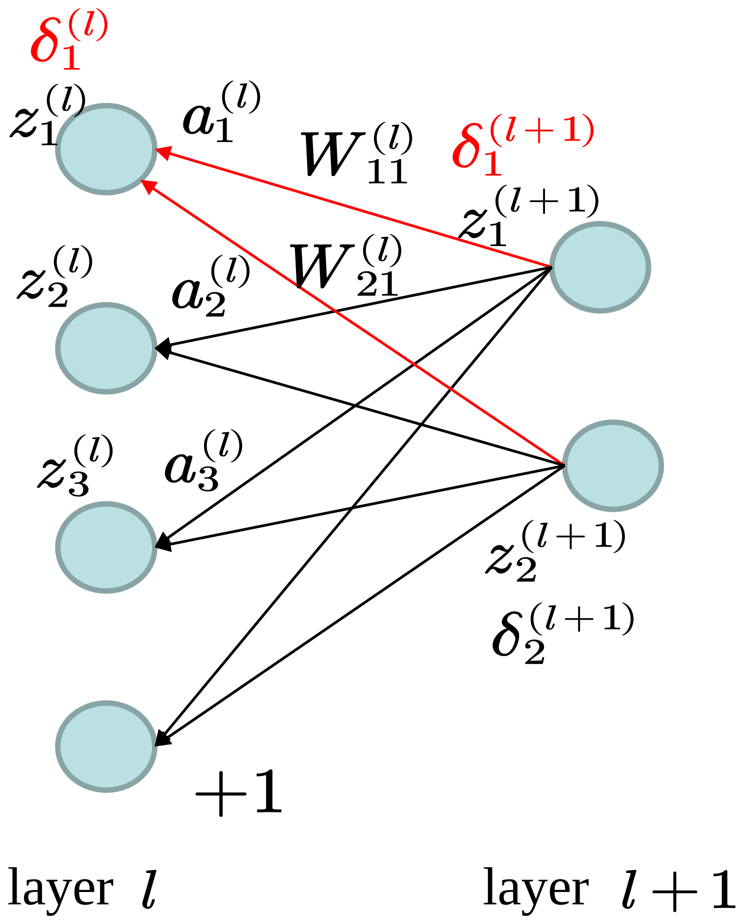
Step4：计算偏导数

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = \frac{\partial}{\partial z_j^{(l+1)}} J(W,b;x,y) \cdot \frac{\partial z_j^{(l+1)}}{\partial W_{ij}^{(l)}}$$

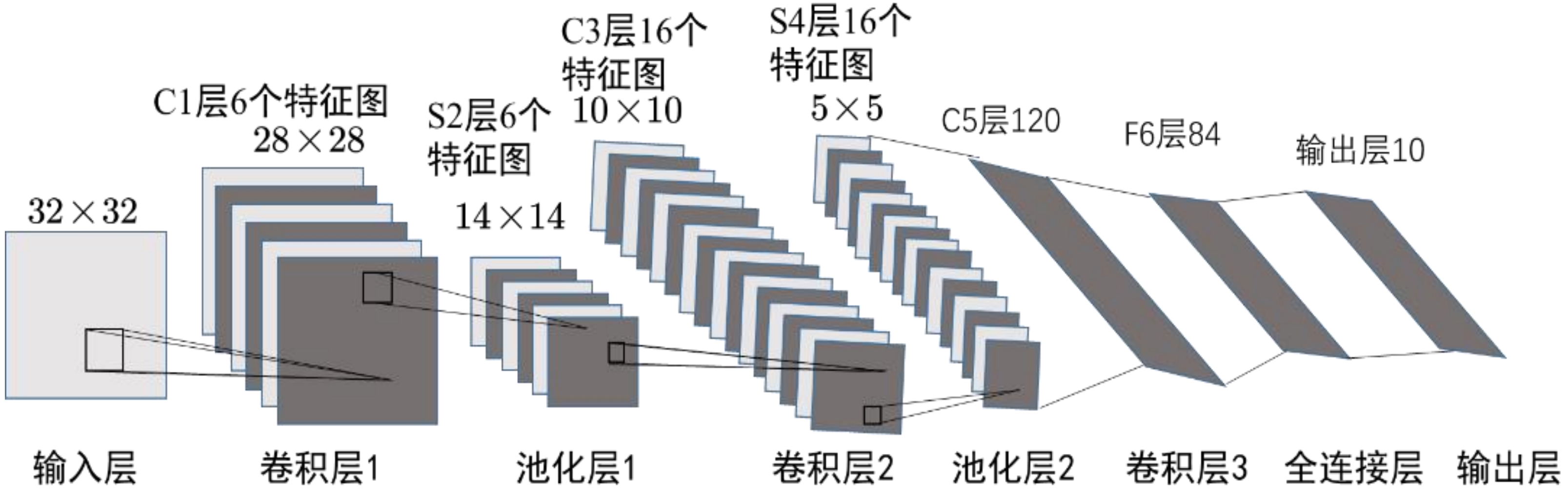
最后的计算公式为：

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_i^{(l)} \delta_j^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}$$



卷积神经网络

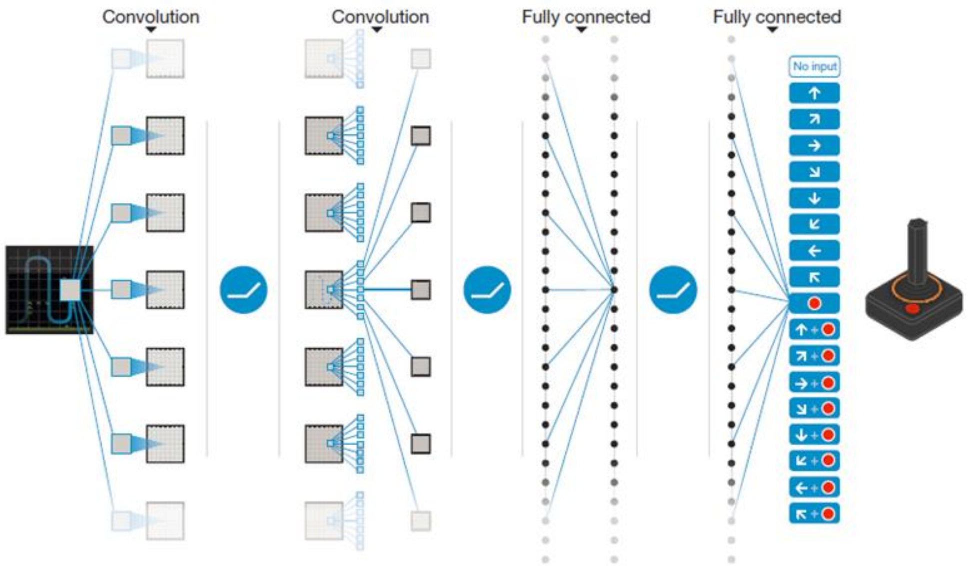


LeNet 网络

卷积：权值共享，稀疏连接

池化：抽象特征表示，进一步减少权值

DQN 介绍



DQN网络

1. 框架为qlearning
2. 值函数表示为卷积神经网络

DQN 介绍

Qlearning伪代码

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 ε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 ε 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：$\pi(s) = \underset{a}{\operatorname{argmax}} Q(s,a)$

行动策略为 s_t 贪婪策略

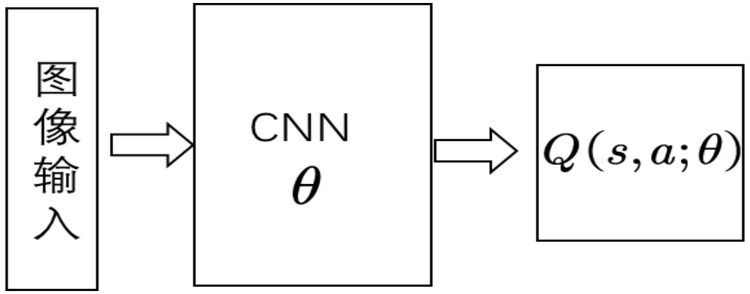
目标策略为贪婪策略

DQN伪代码

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function Q̂ with weights θ⁻ = θ
For episode = 1, M do
    Initialize sequence s_1 = {x_1} and preprocessed sequence φ_1 = φ(s_1)
    For t = 1,T do
        With probability ε select a random action a_t
        otherwise select a_t = argmax_a Q(φ(s_t),a;θ)
        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 φ_{t+1} = φ(s_{t+1})
        Store transition (φ_t,a_t,r_t,φ_{t+1}) in D
        Sample random minibatch of transitions (φ_j,a_j,r_j,φ_{j+1}) from D
        Set y_j = { r_j                                   if episode terminates at step j+1
                  { r_j + γ max_a' Q̂(φ_{j+1},a';θ⁻)      otherwise
        Perform a gradient descent step on (y_j − Q(φ_j,a_j;θ))² with respect to the
        network parameters θ
        Every C steps reset Q̂ = Q
    End For
End For
```

DQN 的技巧

（1）DQN利用卷积神经网络逼近行为值函数



（2）DQN利用经验回放对强化学习过程进行训练

$\langle s_1, a_1, r_2, s_2 \rangle$
$\langle s_2, a_2, r_3, s_3 \rangle$
$\langle s_3, a_3, r_4, s_4 \rangle$
$\langle s_4, a_4, r_5, s_5 \rangle$
$\langle s_5, a_5, r_6, s_6 \rangle$
$\vdots$

（3）DQN设置了目标网络来单独处理时间差分算法中的TD偏差。

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

Tensorflow 基础

最核心的：先定义后计算

Tensorflow 的使用包括两个阶段：

Step1: 构建阶段

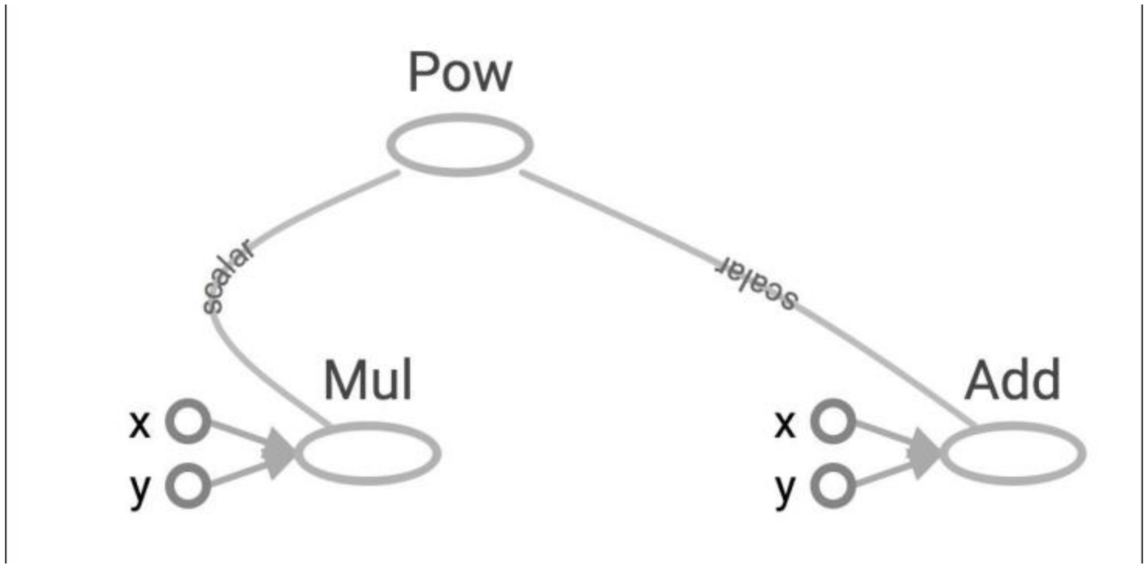
Step2: 执行阶段

构建阶段：构建图

```
x=2
y=3
op1 = tf.add(x,y)
op2 =tf.multiply(x,y)
op3 = tf.pow(op2,op1)
```

执行阶段：启动会话，计算值

```
with tf.Session() as sess:
    op3 = sess.run(op3)
```



节点：　常量，变量和操作

边：　张量

Tensorflow 基础

常量：x1=tf.constant([[1]])

变量， w1 = tf.Variable(0,name='w1')

变量定义后，一定要初始化！

tf.global_variables_initializer()

占位符：构建图

tf.placeholder(dtype, shape=None, name=None)

输入值先定义占位符，再用feed_dict输入

获得某个操作值时用：run

```
s1 = tf.placeholder(tf.float32)
s2 = tf.placeholder(tf.float32)
out = tf.multiply(s1,s2)
with tf.Session() as sess:
    output = sess.run(out, feed_dict={s1:[7.0], s2:[8.0]})
    print(output)
```

Tensorflow 基础

一个优化过程包括两大部分：

1. 组装模型

   (1) 定义占位符
   (2) 定义权重
   (3) 定义预测模型
   (4) 定义损失函数
   (5) 定义优化器

2 训练模型

   输入数据，启动会话，不断迭代

自动微分：

```
tf.gradients(y,[xs])
```

保存模型：

```
saver = tf.train.Saver()

saver.save(sess, seve_path, global_step=None,...)
```

恢复模型：

```
saver.restore(sess, 'checkpoints/name_of_the_checkpoint')
```

第五次作业

1.阅读《Reinforcement Learning： An Introduction》第9章

2.阅读github代码，FlappyBird，体会DQN算法

3.选一款雅达利游戏或自己选一款游戏，编写DQN算法进行训练。