- Graph Convolution Networks (GCN)
- Generative Adversarial Nets (GAN)
- Reinforcement Learning (RL)
- Analysis of IRGAN

Graph Convolution Networks (GCN)

预备知识

动机与原理

一代GCN

二代GCN

Thomas Kpif GCN

Q&A

Form: *David I Shuman*, **The Emerging Field of Signal Processing on Graphs**[Paper], *IEEE Signal Processing Magazine,2013*

Form: *Michaël Defferrard*, **Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering**[Paper], *NIPS,2016*

Form: *Thomas N. Kipf*, **Semi-supervised Classification With Graph Convolutional Networks**[Paper], *ICLR,2017*

Graph Convolution Networks (GCN)

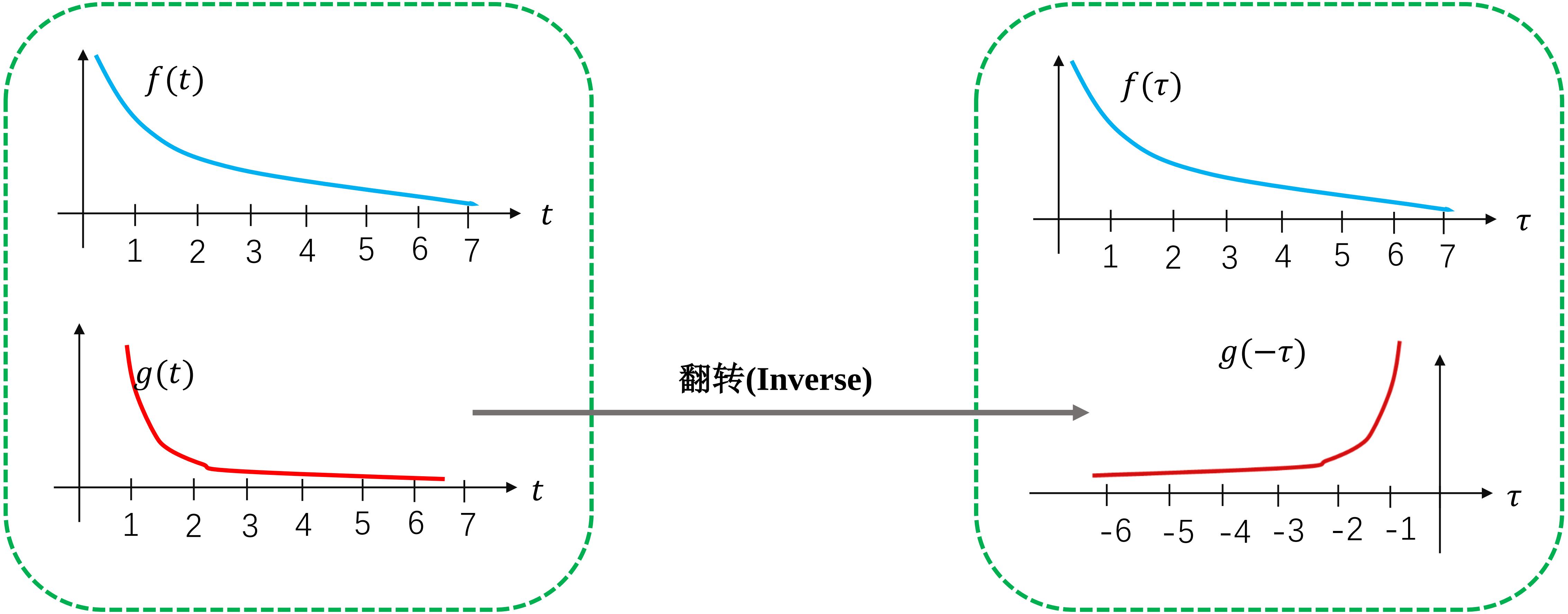
- *预备知识–什么是卷积*

设$f(t)$和 $g(t)$是$\mathbb{R}^n$ 上的两个可积函数， $f(t)$和 $g(t)$的卷积(Convolution)可记为 $Y = f(t) * g(t)$，它是其中一个函数翻转并做平移运动后与另一个函数乘积的积分。

$$\boxed{Y(t) = f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau}$$

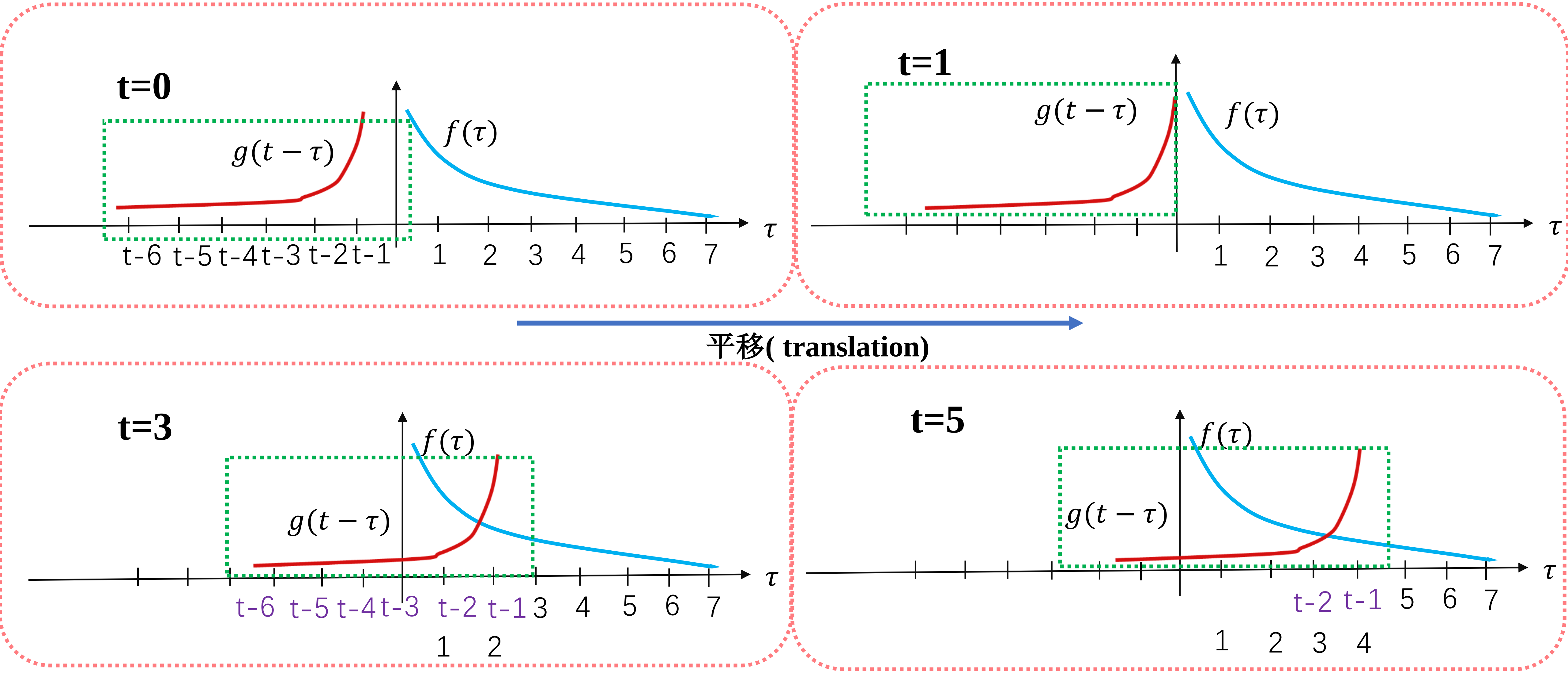
Graph Convolution Networks (GCN)

$$Y(t) = f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau$$



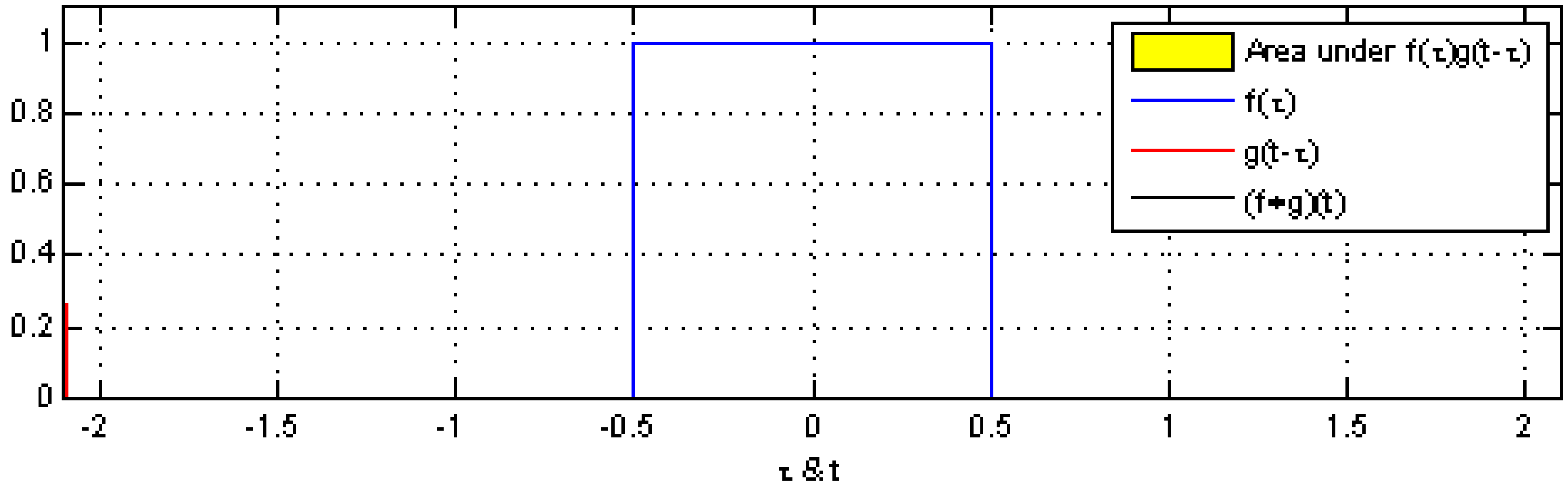
Graph Convolution Networks (GCN)

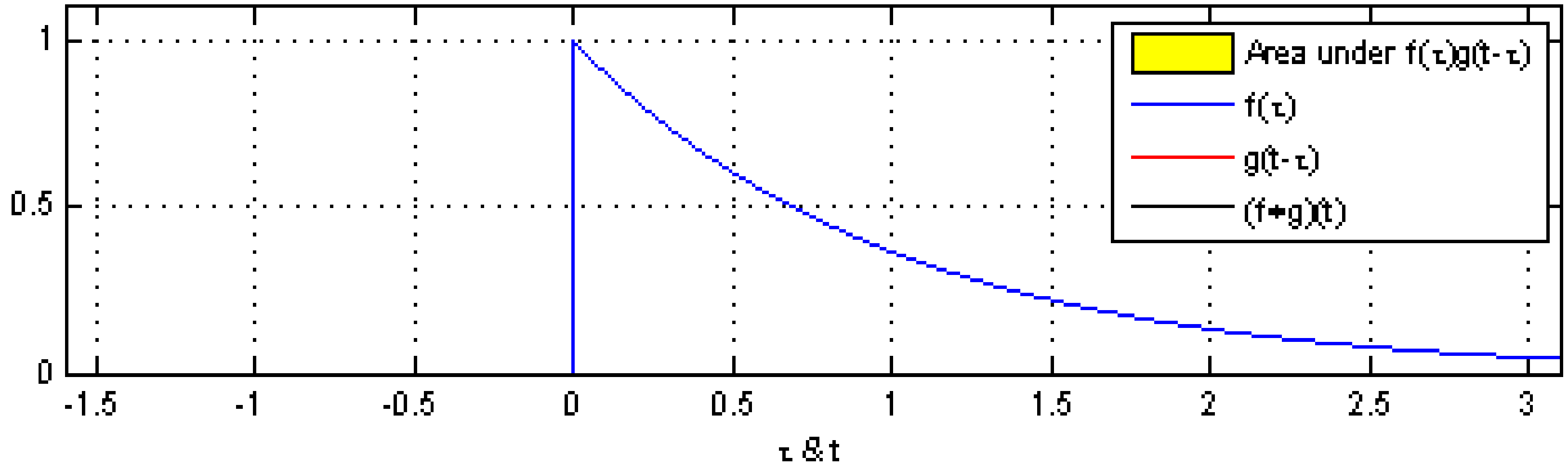
$$Y(t) = f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau) g(t-\tau) d\tau$$



Graph Convolution Networks (GCN)

$$Y(t) = f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau$$





Graph Convolution Networks (GCN)

离散卷积过程

$$Y(n) = f(n) * g(n) = \sum_{m=-\infty}^{+\infty} f(m)g(n-m)$$

F(n)	G(n)
0 0 0 2 0 0 0 0	0 1 2 3 4 5 6

F(m)	G(-m)
0 0 0 2 0 0 0 0	6 5 4 3 2 1 0

一句话总结：滑动加权求和

卷积定理：函数卷积的傅里叶变换是函数傅立叶变换的乘积

N=0

0 1 0 2 0 0 0 0
6 5 4 3 2 1 0

0

N=1

0 1 0 2 0 0 0 0
6 5 4 3 2 1 0

0 0

N=2

0 1 0 2 0 0 0 0
6 5 4 3 2 1 0

0 0 1

N=3

0 1 0 2 0 0 0 0
6 5 4 3 2 1 0

0 0 1 2

N=4

0 1 0 2 0 0 0 0
6 5 4 3 2 1 0

0 0 1 2 5

N=5

0 1 0 2 0 0 0 0
6 5 4 3 2 1 0

0 0 1 2 5 8

N=6

0 1 0 2 0 0 0 0
6 5 4 3 2 1 0

0 0 1 2 5 8 11

Graph Convolution Networks (GCN)

- *预备知识（ 拉普拉斯算子与拉普拉斯矩阵）*

拉普拉斯算子：定义为梯度的散度

(1) $\Delta f = \nabla \cdot \nabla f$

(2) $\Delta f = \sum_i \frac{\partial^2 f}{\partial x_i^2}$

Tips

a. 散度:某点在单位体积内通量和
b. 单变量的实值函数的梯度 仅是其导数

一维空间拉普拉斯算子

$$\Delta f = \frac{\partial^2 f}{\partial x^2} = \left(\frac{f(x+\delta x)-f(x)}{\delta x}\right)' = \frac{f(x+\delta x-\delta x)-f(x-\delta x)}{\delta x} - \frac{f(x+\delta x)-f(x)}{\delta x} / -\delta x$$

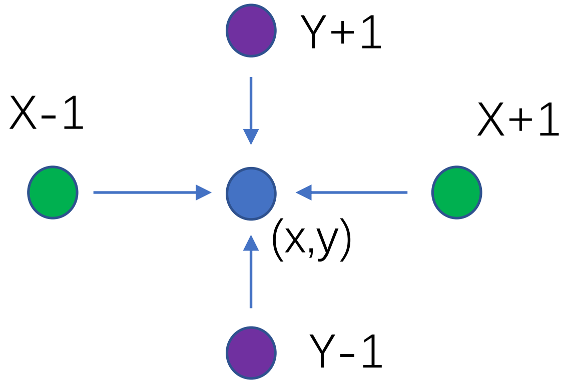
$$= \frac{f(x+\delta x)+f(x-\delta x)-2f(x)}{\delta x^2}$$

二阶导的二阶差分近似

二维空间拉普拉斯算子

$$\frac{\partial^2 f}{\partial x^2} \approx \mathrm{f(x+1,y)+f(x-1,y)-2f(x,y)}$$

$$\frac{\partial^2 f}{\partial y^2} \approx \mathrm{f(x,y+1)+f(x,y-1)-2f(x,y)}$$

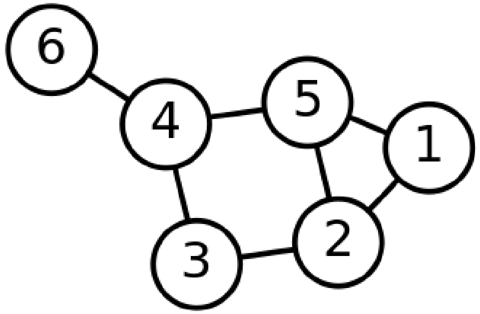


$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \mathrm{f(x+1,y)+f(x-1,y)+f(x,y+1)+f(x,y-1)-4f(x,y)}$$

Graph Convolution Networks (GCN)

拉普拉斯矩阵

$$L = D - A$$



$$D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad L = \begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

$$L = I_N - D^{-1/2} A \, D^{-1/2}$$

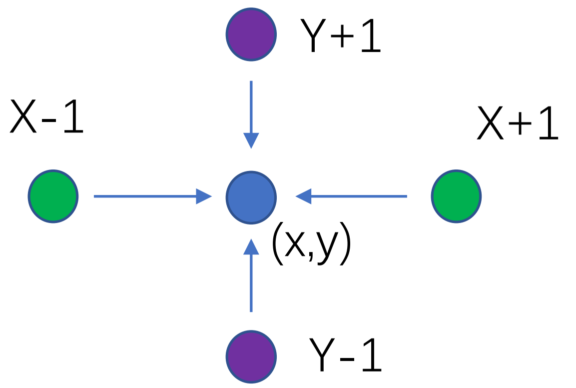
对称归一化拉普拉斯矩阵

半正定对称矩阵	顺序主子式都大于等于0	有N个特征值且都非负
有N个线性无关的特征向量	特征向量相互正交	转置等于其逆

Graph Convolution Networks (GCN)

拉普拉斯算子与拉普拉斯矩阵关系

1. **信息增益解释**



对于图像处理的拉普拉斯算子来说，某点的拉普拉斯算子值为，
在该点四个自由度方向扰动(位移）并对增益求和（二阶差分和）

对于一个图G来说由N个节点构成，我们可以把图G看作一个函数F
$F=(f_1,\cdots,f_i\cdots,f_N)$ 在这里f_i表示一个结点的值。类比f(x,y)为f在(x,y)处的值
对一个节点i, 其自由度方向由和它相连接的结点j 构成 , $j\in \mathrm{N}_i$ ，
N_i表示结点i的一阶邻域，结点i的自由度数为$O(\mathrm{N}_i)$

结点i的增益和为： $(\Delta F)_i=\frac{\partial^2 f}{\partial i^2}\approx\sum_{j\in N_i}(f_j-f_i)$

2. **散度解释**

散度可以简单解释为局部区域的通量和，对于一个图G来说，
G在某点的散度为 该点与其相连接的点一阶导的梯度之和，
这里用二阶差分近似。

$(\Delta F)_i=\frac{\partial^2 f}{\partial i^2}\approx\sum_{j\in N_i}(f_j-f_i)$

Graph Convolution Networks (GCN)

$(\Delta F)_i = \frac{\partial^2 f}{\partial i^2} \approx \sum_{j \in N_i}(f_j - f_i) = \sum_{j \in N_i} W_{i,j}(f_j - f_i)$, $W_{i,j}$对于i和j相邻 取值为-1，不相邻取值为0

$$\sum_{j \in N_i} W_{i,j}(f_j - f_i) = \sum_j W_{i,j} f_i - \sum_j W_{i,j} f_j$$

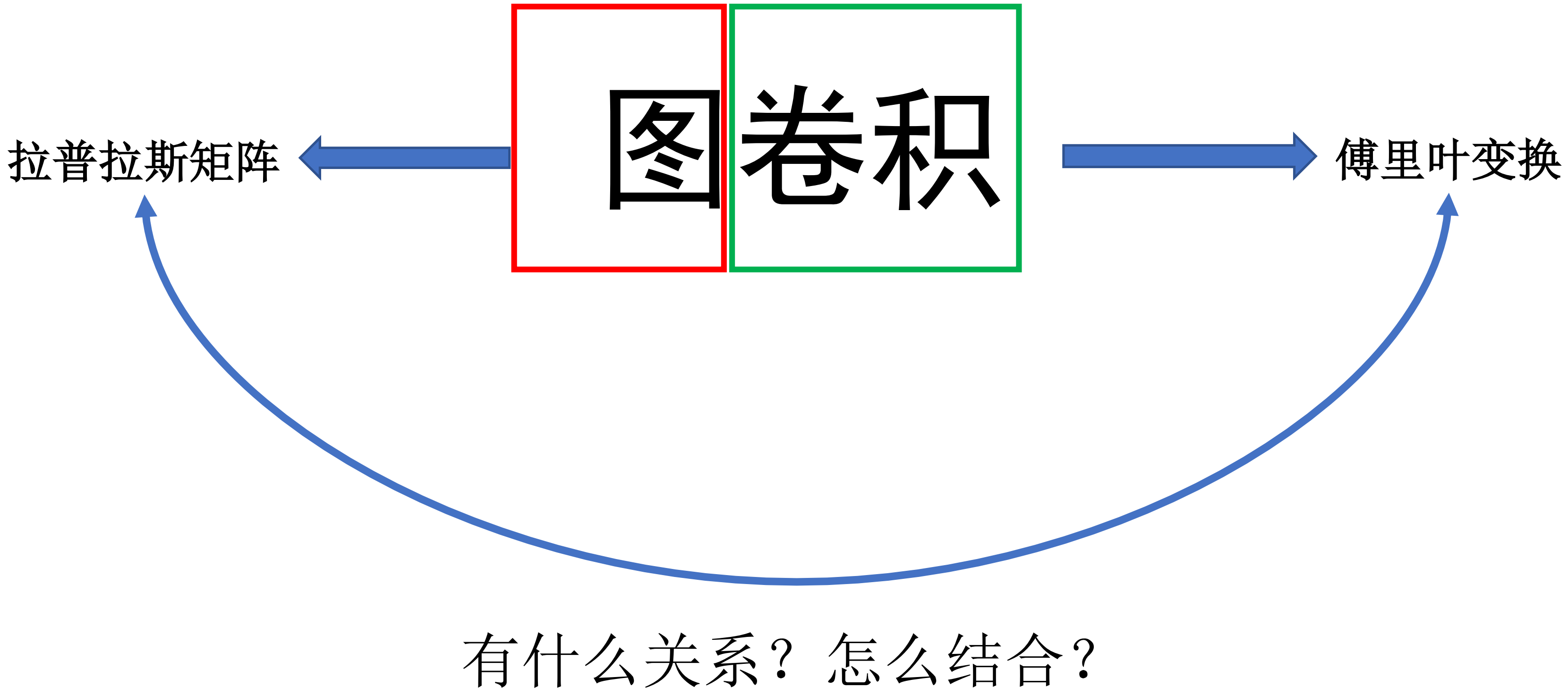
$$= D(i) f_i - \sum_j W_{i,j} f_j = D(i) \sum_j \delta_{i,j} f_j - \sum_j W_{i,j} f_j = \sum_j D_{i,j} f_j - \sum_j W_{i,j} f_j = \sum_j L_{i,j} f_j$$

$$(\Delta F)_i = \sum_j L_{i,j} f_j$$

$$(\Delta F) = \sum_i \sum_j L_{i,j} f_j = LF$$

Graph Convolution Networks (GCN)

- ***动机与原理***



Graph Convolution Networks (GCN)

Tips
[齐次]调和方程(拉普拉斯方程） $\Delta u = 0$
[非齐次] 泊松方程 $\Delta u = \mathrm{F}$

传统傅里叶变换: $F(\omega) = F[f(t)] = \int f(t)e^{-i\omega t}dt$

$f(t)$ 为时域函数， $e^{-i\omega t}$为变换基函数。 $e^{-i\omega t}$的拉普拉斯算子为$\Delta e^{-i\omega t}=\frac{\partial^2}{\partial t^2}e^{-i\omega t} = -\omega^2 e^{-i\omega t}$

广义特征方程定义为 $Av = \lambda v$. A是一种变换，v<u>是特征向量或者特征函数</u>，λ是特征值。
$\Delta e^{-i\omega t}=-\omega^2 e^{-i\omega t}$ 根据上述定义可得Δ作为一种变换， $e^{-i\omega t}$是特征函数，ω与特征值有关。

Δ 我们类比为L (拉普拉斯矩阵) , $e^{-i\omega t}$类比为v (拉普拉斯矩阵特征向量) 故有$Lv = \lambda v$

$$F(\omega) = F[f(t)] = \int f(t)e^{-i\omega t}dt \xrightarrow{\text{类比}} F(\lambda_l) = \hat{f}(\lambda_l) = \sum_{i=1}^{N} f(i)u_l^*(i)$$

f是图上的N维向量， $f(\mathrm{i})$与图上的顶点(i)一一对应， $u_l^*(i)$表示第l个特征向量的第i个分量，λ_l与u_l^*对应， u_l^*是u_l 共轭向量

Graph Convolution Networks (GCN)

$$\begin{pmatrix} \hat{f}(\lambda_1) \\ \hat{f}(\lambda_2) \\ \vdots \\ \hat{f}(\lambda_N) \end{pmatrix} = \begin{pmatrix} u_1(1) & u_1(2) & \cdots & u_1(N) \\ u_2(1) & u_2(2) & \cdots & u_2(N) \\ \vdots & \vdots & \ddots & \vdots \\ u_N(1) & u_N(2) & \cdots & u_N(N) \end{pmatrix} \begin{pmatrix} f(1) \\ f(2) \\ \vdots \\ f(N) \end{pmatrix} \Longleftrightarrow \hat{f} = U^T f$$

$$\mathcal{F}^{-1}[F(\omega)] = \frac{1}{2\pi}\int F(\omega)e^{i\omega t}d\omega$$

类比

$$f(i) = \sum_{l=1}^{N} \hat{f}(\lambda_l)u_l(i)$$

$$\begin{pmatrix} f(1) \\ f(2) \\ \vdots \\ f(N) \end{pmatrix} = \begin{pmatrix} u_1(1) & u_2(1) & \cdots & u_N(1) \\ u_1(2) & u_2(2) & \cdots & u_N(2) \\ \vdots & \vdots & \ddots & \vdots \\ u_1(N) & u_2(N) & \cdots & u_N(N) \end{pmatrix} \begin{pmatrix} \hat{f}(\lambda_1) \\ \hat{f}(\lambda_2) \\ \vdots \\ \hat{f}(\lambda_N) \end{pmatrix} \Longleftrightarrow$$

$$U^{T^{-1}}\hat{f} = U^{T^{-1}}U^T f$$

$$U^{-1^T}\hat{f} = f$$

$$U^{T^T}\hat{f} = f$$

$$U\hat{f} = f$$

Graph Convolution Networks (GCN)

卷积定理：函数卷积是函数傅立叶变换的乘积的逆傅里叶变换 $f * h = \mathcal{F}^{-1}\left[\hat{f}(\omega)\hat{h}(\omega)\right] = \frac{1}{2\pi}\int \hat{f}(\omega)\hat{h}(\omega)e^{i\omega t}d\omega$

f在图上的傅里叶变换为 $\hat{f}(\lambda_l) = \sum_{l=1}^{N} f(i)u_l(i)$ 其矩阵形式为 $\hat{f} = U^T f$

h在图上的傅里叶变换为 $\hat{h}(\lambda_l) = \sum_{l=1}^{N} h(i)u_l(i)$ 其矩阵形式为 $\hat{h} = U^T h$

为了方便乘积运算我们将$\hat{h}$写成对角矩阵形式 $\begin{bmatrix} \hat{h}(\lambda_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \hat{h}(\lambda_N) \end{bmatrix}$

两个向量f，h在图上的卷积可表示为： $(f * h)_G = U \begin{bmatrix} \hat{h}(\lambda_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \hat{h}(\lambda_N) \end{bmatrix} U^T f = U((U^T h)\odot(U^T f))$

Graph Convolution Networks (GCN)

- **一代GCN**

$$out = \sigma\left(U\begin{bmatrix}\theta_1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \theta_N\end{bmatrix}U^T x\right)$$

我们将$diag\left(\hat{h}(\lambda_l)\right)$变为$diag(\theta_l)$[卷积核]，$\theta$为可学习的参数，$\sigma$为激活函数

每一次前向传播，都要计算 $U, diag(\theta_l)$及 U^T三者的乘积，计算复杂度高。

卷积核参数不能共享

卷积核需要 n 个参数

Graph Convolution Networks (GCN)

- ***二代GCN***

$$\hat{h}(\lambda_l) = \sum_{l=1}^{N} h(i) u_l(i) \Rightarrow \sum_{j=0}^{K} \alpha_j \lambda_l^j$$

$$out = \sigma\left(U \begin{bmatrix} \sum_{j=0}^{K} \alpha_j \lambda_1^j & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sum_{j=0}^{K} \alpha_j \lambda_N^j \end{bmatrix} U^T x \right)$$

$$\begin{bmatrix} \sum_{j=0}^{K} \alpha_j \lambda_1^j & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sum_{j=0}^{K} \alpha_j \lambda_N^j \end{bmatrix} = \sum_{j=0}^{K} \alpha_j \Lambda^j$$

$$U \left(\sum_{j=0}^{K} \alpha_j \Lambda^j \right) U^T = \sum_{j=0}^{K} \alpha_j U \Lambda^j \ U^T = \sum_{j=0}^{K} \alpha_j L^j$$

$$out = \sigma\left(\sum_{j=0}^{K} \alpha_j L^j x \right)$$

卷积核只有K个参数, K远小于n

不需要做特征分解，直接用拉普拉斯矩阵 L进行变换.

K的大小为感受野

Graph Convolution Networks (GCN)

- ***Thomas Kpif GCN***

$$\hat{h}(\lambda_l) = \sum_{l=1}^{N} h(i)u_l(i) \Rightarrow \sum_{j=0}^{K-1} \theta_j T_j(\tilde{\Lambda}) \quad \tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I_N$$

$$U \sum_{j=0}^{K-1} \theta_j T_j(\tilde{\Lambda}) U^T = \sum_{j=0}^{K-1} \theta_j U T_j(\tilde{\Lambda}) U^T = \sum_{j=0}^{K-1} \theta_j T_j(\tilde{L})$$

$$out = \sum_{j=0}^{K-1} \theta_j T_j(\tilde{L}) x \qquad \tilde{L} = \frac{2L}{\lambda_{max}} - I_N$$

$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x),\ T_0(x) = 1, T_1(x) = x,$ ⟵ 切比雪夫多项式

$$out = (\theta_0 T_0(\tilde{L}) + \theta_1 T_1(\tilde{L}))\, x = \theta_0 x + \theta_1 \tilde{L} x$$

$out = \theta_0 x + \theta_1 (L - I_N) x$ ⟵ 让λ_{max}=2

$out = \theta\ (L - I_N) x$ ⟵ 为了防止过拟合 减少拟合参数

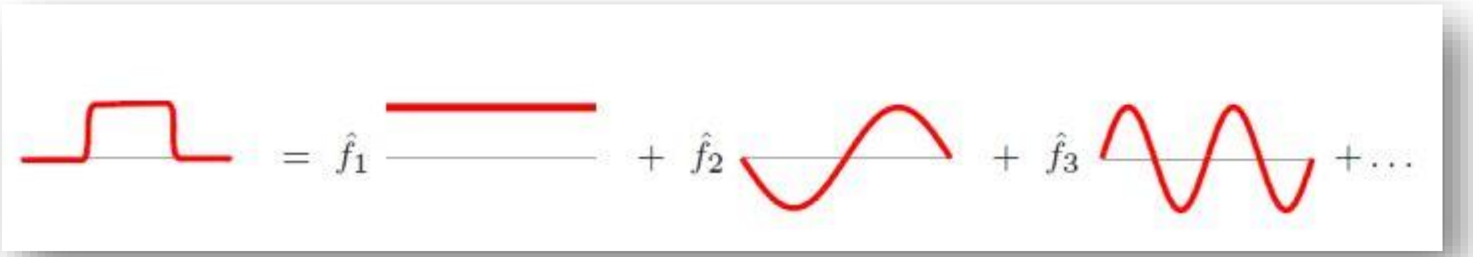
$\tilde{A} = A + I_N \qquad \tilde{D}_i = \sum_j \tilde{A}_{i,j}$

$\boxed{out = \theta\ D^{-1/2} A\, D^{-1/2} x}$ $\xrightarrow{\text{简化扩展}}$ $\boxed{out = \tilde{D}^{-1/2} \tilde{A}\, \tilde{D}^{-1/2} X\, \Theta}$ $\quad \Theta \in \mathcal{R}^{CxF} \quad X \in \mathcal{R}^{NxC}$

Graph Convolution Networks (GCN)

- ***Q&A***

为什么拉普拉斯矩阵的特征向量能够作为傅里叶变换的基？



傅里叶变换一个本质理解就是：**把任意一个函数表示成了若干个正交函数（由sin,cos 构成）的线性组合。**

图上的傅里叶变换对图上的任意向量f 可表示为拉普拉斯矩阵特征向量的线性组合：

$$f = \hat{f}(\lambda_1)u_1 + \hat{f}(\lambda_2)u_2 + \cdots + \hat{f}(\lambda_N)u_N$$

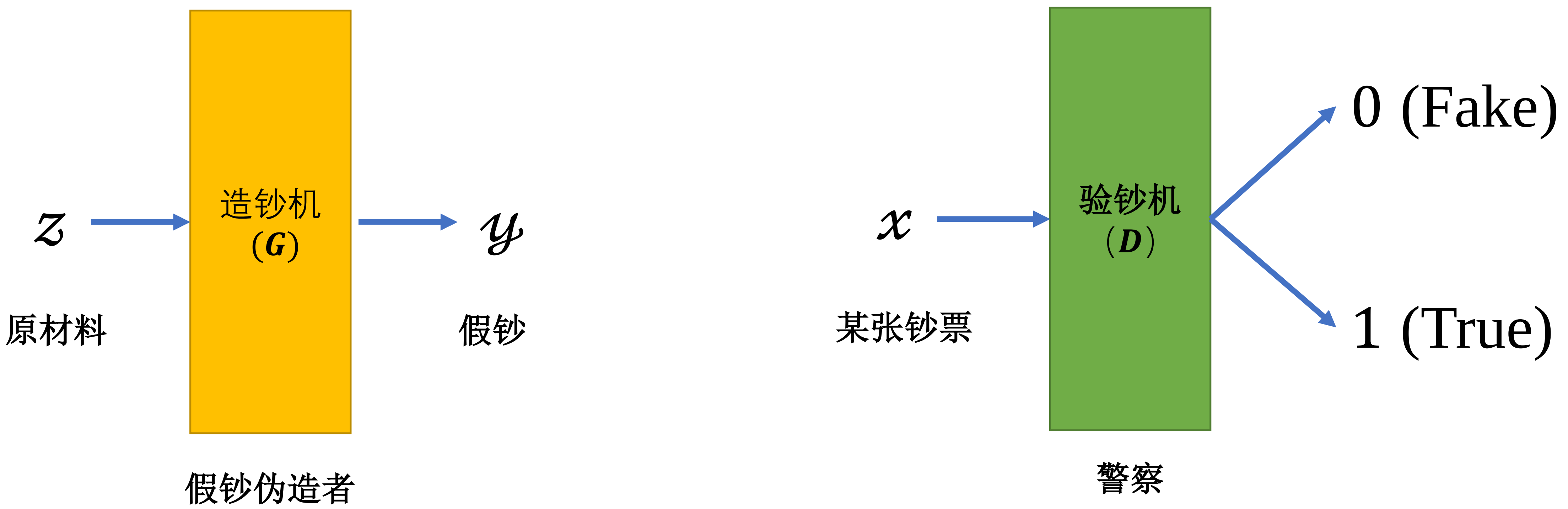
为什么任意向量f 可表示为拉普拉斯矩阵特征向量的线性组合？

拉普拉斯矩阵由于是半正定对称阵，则有N个线性无关的特征向量，且相互正交。
N维空间的n个线性无关的向量可以构成空间的一组基，于是可以这些特征向量可以组成一组基，且是正交基。

Generative Adversarial Nets (GAN)

Form: *Ian J. Goodfellow* ,**Generative Adversarial Nets** [Paper], *NIPS,2014*

感性认识



我们的目标:做最牛的造钞机和做最牛的验钞机

Generative Adversarial Nets (GAN)

理性认识

- ***建模判别器(验钞机)***

定义: 假设存在一个可微函数$\mathcal{D}(x)$, 其值域为[0,1], 即$0 \le \mathcal{D}(x) \le 1$ 。$\mathcal{D}(x)$输出值表现为输入x 的置信度，对于某一输入 x， 其$\mathcal{D}(x)$ 的值越小，则x越<u>不可信</u>(Fake),其$\mathcal{D}(x)$ 的值越大，则x越<u>可信</u>(True)。 我们也称$\mathcal{D}(x)$为判别器。

我们定义存在某样本集$\mathrm{X} = \{x_1, x_2, x_3, \cdots, x_m\}$, 样本容量为$m$ 。根据上述定义,$\mathcal{D}(x)$可以看作是服从某分布的条件概率函数$\mathcal{D}(\mathrm{Y} = \mathrm{C}|X)$ 。$\mathcal{D}(\mathrm{Y} = 1|x_1)$ 表示在x_1条件下$\mathrm{Y} = 1$的概率，换句话说，输入x_1被判别为**<u>真</u>**(True) 的概率 (x_1 的置信度值)。

存在一组真实样本集$\mathcal{X} = \{x_1, x_2, \cdots, x_m\}$, 样本容量为$m$ 。$\forall x \in \mathcal{X}$, $x \sim P_{data}(x)$ 表示对任意$x \in \mathcal{X}$， x服从P_{data}分布。<u>*对于真实样本集，什么样*的$\mathcal{D}(x)$ *是最好*</u>? So easy， 对于 $\forall x_i \in \mathcal{X}$,$\mathcal{D}(\mathrm{Y} = 1|x_i)$ 最大，由于独立同分布 对于整个样本集，我们可以表示为 $\prod_{i=1}^{m} D(Y = 1|x_i)$ 的最大化。$\mathcal{D}(x)$ 服从什么分布我们不知道，且我们希望把$\prod_{i=1}^{m} D(Y = 1|x_i)$ 转化为可优化的问题，于是我们可以构造最大似然函数 $J(\theta) = \prod_{i=1}^{m} D(Y = 1|x_i; \theta)$。问题可以转化为$\max_\theta J(\theta) = \max_\theta \prod_{i=1}^{m} D(Y = 1|x_i; \theta)$， 找到一个合适的$\theta$使得 样本集上的联合概率最大化。

Generative Adversarial Nets (GAN)

$$
\begin{aligned}
argmax_{\theta} J(\theta) &= argmax_{\theta} \prod_{i=1}^{m} D(Y=1|x_i;\theta) \\
argmax_{\theta} \log(J(\theta)) &= argmax_{\theta} \textstyle\sum_{i=1}^{m} \log(D(Y=1|x_i;\theta))
\end{aligned}
$$

$$
\begin{aligned}
\hat{\theta} &= argmax_{\theta} \sum_{i=1}^{m} \log(D(Y=1|x_i;\theta)) \\
&\approx argmax_{\theta} \textstyle\sum_{i=1}^{m} P_{data}(x_i)\log(D(Y=1|x_i;\theta)) \ \forall x \in \mathcal{X}\ ,\ x\sim P_{data}(x) \\
&= argmax_{\theta} \mathbb{E}_{x\sim P_{data}(x)}[\log(D(Y=1|x;\theta)]
\end{aligned}
$$

对于非真实样本，什么样的$\mathcal{D}(x)$ 是最好？这里我们需要引入生成器的定义。
生成器定义：存在一个可微函数$\mathcal{G}(x)$, 随机输入一组噪声$\mathcal{Z} = \{z_1, z_2, \cdots, z_m\}$ 对$\forall z \in \mathcal{Z}, z\sim P_{\mathcal{Z}}(z)$
$x = \mathcal{G}(z)$, x满足 $x\sim P_G(x)$, 由于$\mathcal{G}(x)$需要被优化我们对其参数化$\mathcal{G}(x;\theta_G)$

依据上述原理我们可以得出对于非真实样本, $\mathcal{D}(x)$ 的优化目标：

$$
\hat{\theta} = argmax_{\theta} \sum_{i=1}^{m} P_G(x_i)\log(D(Y=0|x_i;\theta))
$$

$$
\begin{aligned}
\hat{\theta} &= argmax_{\theta} \sum_{i=1}^{m} P_G(x_i)\log(1-D(Y=1|x_i;\theta)) \\
&= argmax_{\theta} \mathbb{E}_{x\sim P_G(x)}[\log(1-D(Y=1|x;\theta)]
\end{aligned}
$$

Generative Adversarial Nets (GAN)

$$\max_D V(D) = argmax_{\theta_D} \mathbb{E}_{x\sim P_{data}(x)}[\log(D(Y=1|x;\theta_D)] + \mathbb{E}_{x\sim P_G(x)}[\log(1-D(Y=1|x;\theta_D)]$$
$$= argmax_{\theta_D} \mathbb{E}_{x\sim P_{data}(x)}[\log(D(Y=1|x;\theta_D)] + \mathbb{E}_{z\sim P_Z(z)}[\log(1-D(Y=1|\mathcal{G}(z),;\theta_D)]$$

- ***建模生成器(造钞机)***

生成器定义：存在一个可微函数$\mathcal{G}(x)$, 随机输入一组噪声$\mathcal{Z} = \{z_1, z_2, \cdots, z_m\}$ 对$\forall z \in \mathcal{Z}$, $z\sim P_Z(z)$
$x = \mathcal{G}(z)$, x满足 $x\sim P_G(x)$, 由于$\mathcal{G}(x)$需要被优化我们对其参数化$\mathcal{G}(x;\theta_G)$

什么样的$\mathcal{G}(x)$ 是最好的？根据上述原理我们类比可以得到：

$$\hat{\theta} = argmax_{\theta_G} \sum_{i=1}^{m} P_G(x_i)\log(D(Y=1|x_i;\boxed{\theta_D}))$$

θ_D被固定住，判别器被固定住

$$\hat{\theta} = argmin_{\theta_G} \sum_{i=1}^{m} P_G(x_i)\log(1-D(Y=1|x_i;\theta_D))$$
$$\hat{\theta} = argmin_{\theta_G} \mathbb{E}_{x\sim P_G(x)}[\log(1-D(Y=1|x;\theta_D)]$$
$$\hat{\theta} = argmin_{\theta_G} \mathbb{E}_{z\sim P_Z(z)}[\log(1-D(Y=1|\mathcal{G}(z;\theta_G);\theta_D)]$$

Generative Adversarial Nets (GAN)

$$min_G max_D V(G,D) = argmin_{\theta_G} argmax_{\theta_D} \mathbb{E}_{x \sim P_{data}(x)}[\log(D(Y=1|x;\theta_D)] + \mathbb{E}_{z \sim P_Z(z)}[\log(1 - D(Y=1|\mathcal{G}(z;\theta_G);\theta_D)]$$

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Generative Adversarial Nets (GAN)

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

- ***存在最优判别器证明***

Proof: 对于任意给定的G, 最大化V ,即 $\max_{\mathrm{D}} V(D,G)$

$\max_{\mathrm{D}} V(D,G) = \mathbb{E}_{x\sim P_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim P_Z(z)}[\log(1-D(G(z))]$

$= \mathbb{E}_{x\sim P_{data}(x)}[logD(x)] + \mathbb{E}_{x\sim P_g(x)}[\log(1-D(x))]$ 对$\forall z \in \mathcal{Z}$, $z\sim P_Z(z)$ $G(z)=x$, x满足 $x\sim P_g(x)$

$=\int_x P_{data}(x)logD(x)\,dx + \int_x P_g(x)\log(1-D(x))\,dx$ 将满足$P_g(x)$ 和$P_{data}(x)$ 的样本合并

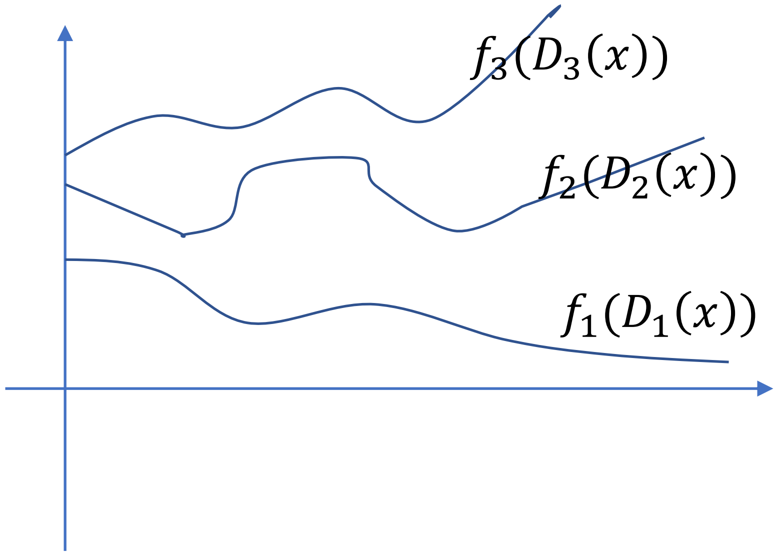
$= \int_x P_{data}(x)logD(x) + P_g(x)\log(1-D(x))\,dx$ $x\sim P_g(x) \cup P_{data}(x)$

$= \int_x f(D(x))\,dx$

$f(D(x)) = P_{data}(x)logD(x) + P_g(x)\log(1-D(x))$

$$\frac{\partial f(D(x))}{\partial (D(x))} = 0$$

$$\frac{P_{data}(x)}{D(x)} + \frac{P_g(x)}{1-D(x)} = 0 \rightarrow D(x) = \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}$$



Generative Adversarial Nets (GAN)

- *存在最优生成器证明*

Proof: 在最优判别器的条件下优化生成器，最小化V ,即 $\min_G V(D,G)$。若V 存在最小值则存在最优生成器。
令$C(G) = \min_G V(D,G)$, 已知最优判别器$D_G^*(x) = \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}$, 故 $C(G) = \min_G V(D_G^*, G)$

$$C(G) = \int_x P_{data}(x) log D_G^*(x)\,dx + \int_x P_g(x)\log(1 - D_G^*(x))\,dx$$

$$= \int_x P_{data}(x) log \frac{P_{data}(x)/2}{(P_{data}(x)+P_g(x))/2} dx + \int_x P_g(x)\log(1 - \frac{P_{data}(x)/2}{(P_{data}(x)+P_g(x))/2})\,dx$$

$$= \int_x P_{data}(x) log \frac{P_{data}(x)/2}{(P_{data}(x)+P_g(x))/2} dx + \int_x P_g(x)\log(\frac{P_g(x)/2}{(P_{data}(x)+P_g(x))/2})\,dx$$

$$= \int_x P_{data}(x)(log \frac{P_{data}(x)}{(P_{data}(x)+P_g(x))/2} - \log(2))\,dx + \int_x P_g(x)(\log\left(\frac{P_g(x)}{(P_{data}(x)+P_g(x))/2}\right) - \log(2))\,dx$$

$$= \int_x P_{data}(x)(\log(\frac{P_{data}(x)}{(P_{data}(x)+P_g(x))/2}))\,dx + \int_x P_g(x)(\log\left(\frac{P_g(x)}{(P_{data}(x)+P_g(x))/2}\right))\,dx - \log(4)$$

$$= KL(P_{data}(x) \parallel (P_{data}(x) + P_g(x))/2) + KL(P_g(x) \parallel (P_{data}(x) + P_g(x))/2) - \log(4)$$

KL散度大于等于0,故$C(G)$存在最小值$-\log(4)$，当且仅当$P_{data} = P_g$。所以存在最优判别器，当且仅当$P_{data} = P_g$。
当$P_{data} = P_g$时， $D(x)$恒为1/2

Generative Adversarial Nets (GAN)

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

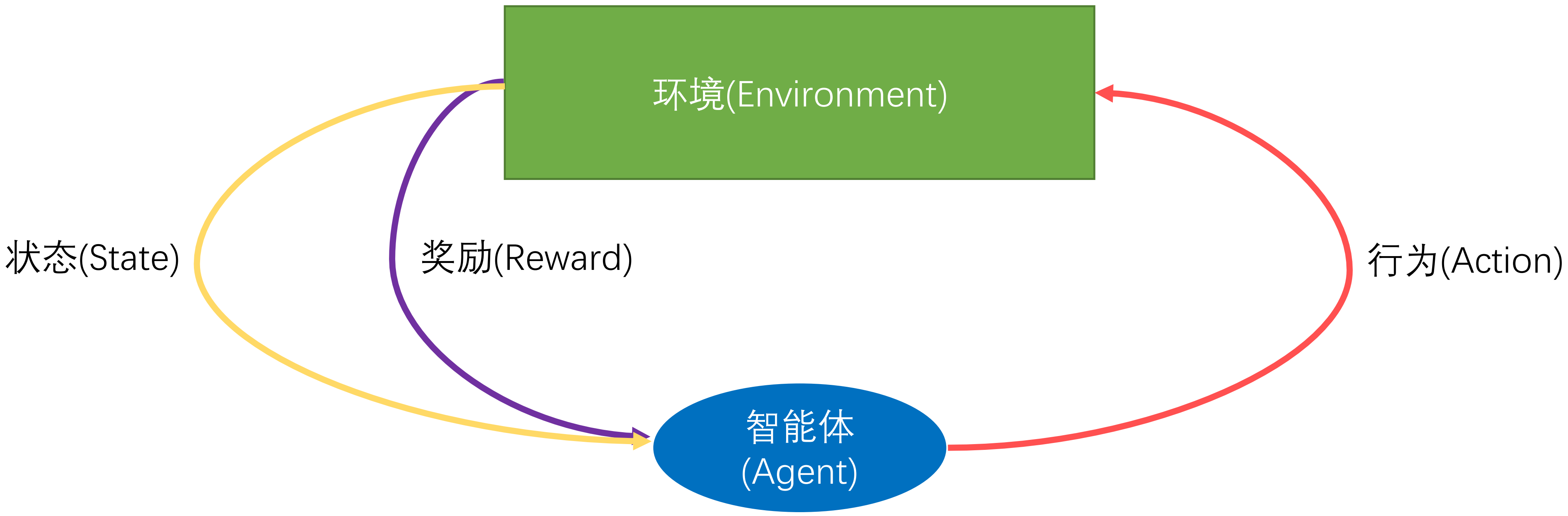
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Reinforcement Learning (RL)

Form: *Richard S. Sutton and Andrew G. Barto*,
Reinforcement Learning:An Introduction[Book], *2016*

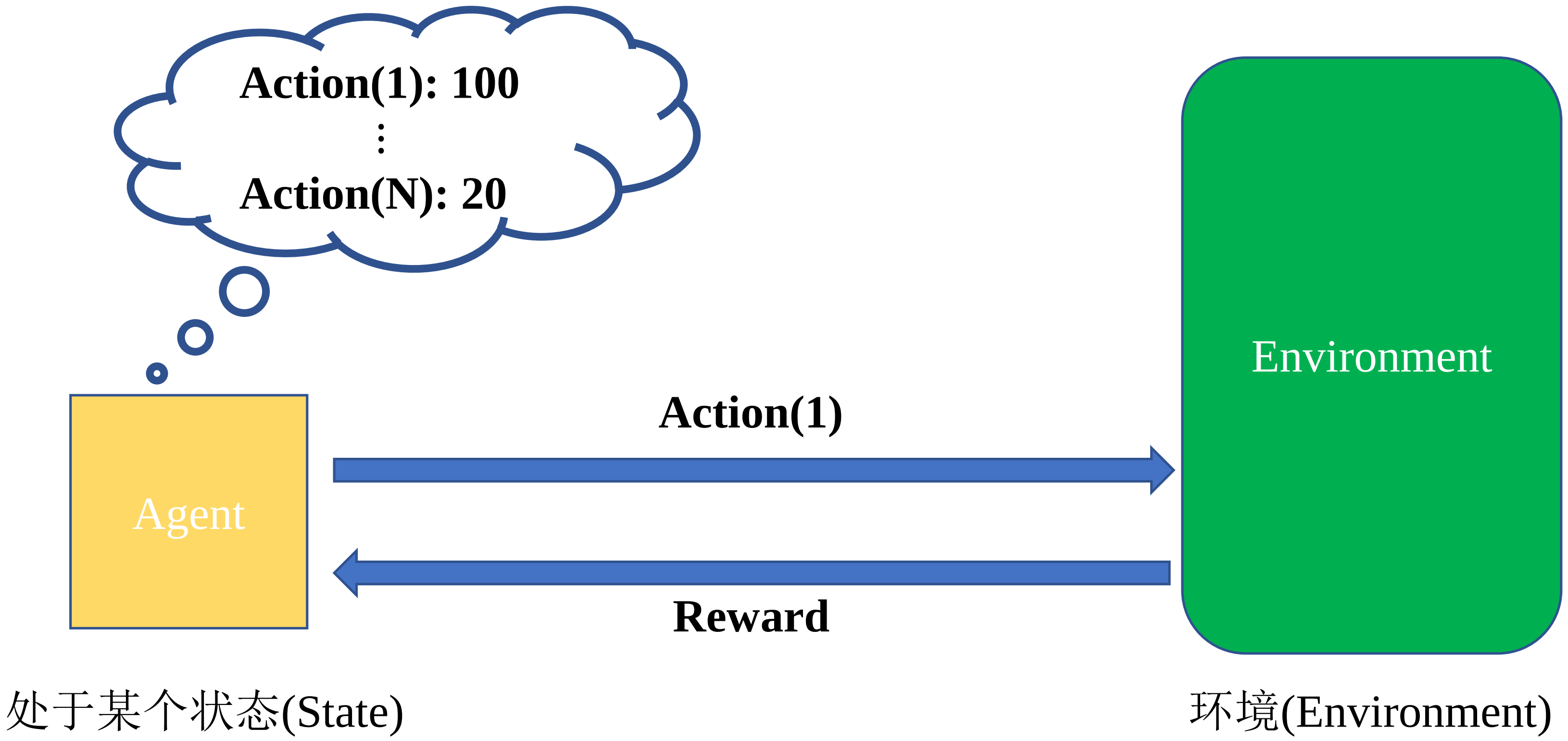
感性认识



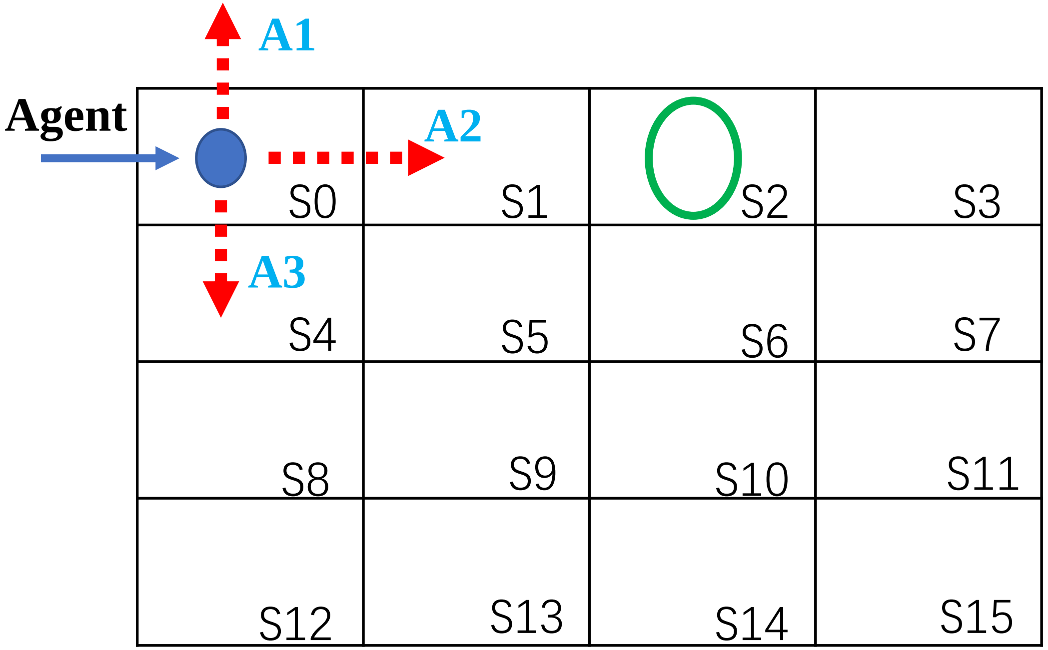
Reinforcement Learning (RL)

- ***(基于价值的强化学习) Value-based RL***

建模Q Learning



Reinforcement Learning (RL)



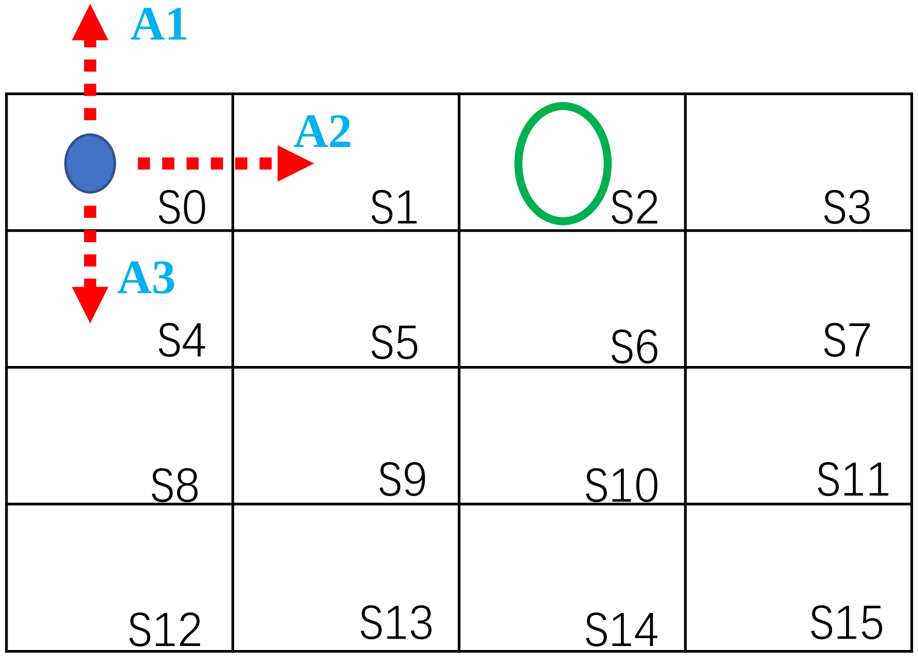
Q Table

	A1	**A2**	**A3**
S0	0	2	1

(S0 状态下A1的估计收益)
Agent 在S0 状态下执行A1可能带来的潜在收益。

Reinforcement Learning (RL)

第一轮



Q Table

	A1	A2	A3
S0	0	2	1

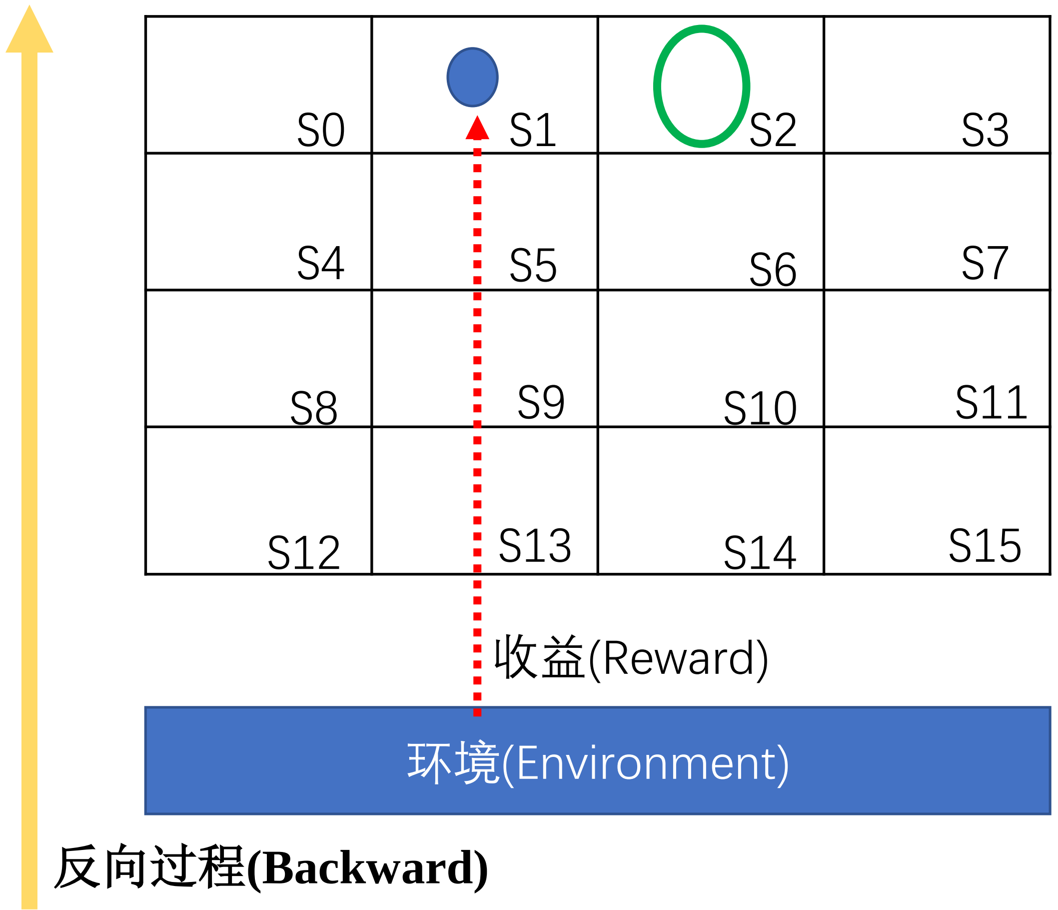
	A1	A2	A3
S1	0	5	4

⋮

前向过程(Forward)

Reinforcement Learning (RL)

第一轮



Q Table

	A1	**A2**	**A3**
S0	0	2	1

Update

	A1	**A2**	**A3**
S0	0	0	1

$Reward = 0$ ⟵ 真实收益

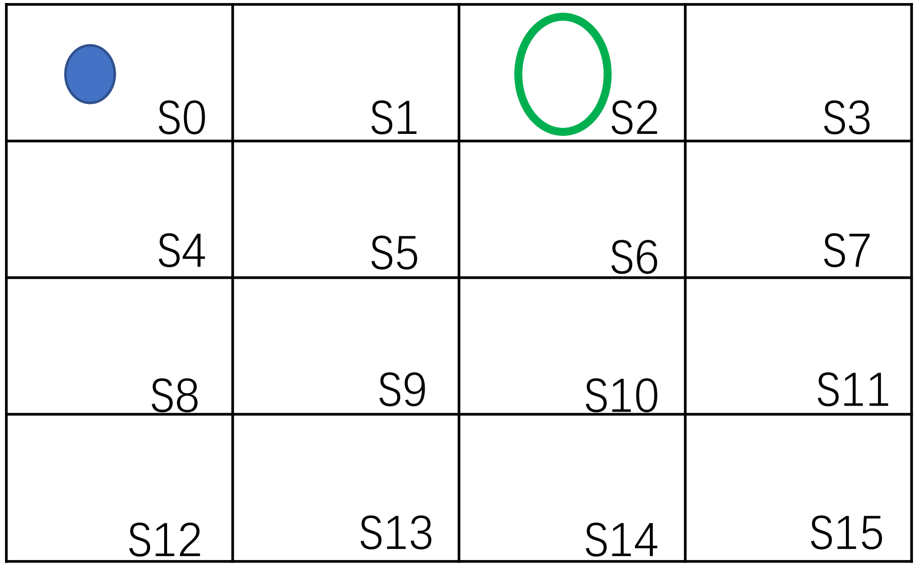
$Q(S_0, A_2) = 2$ ⟵ 估计收益

Update

$$Q(S_0, A_2) = Q(S_0, A_2) + (\text{Reward} - Q(S_0, A_2))$$

Reinforcement Learning (RL)

第二轮

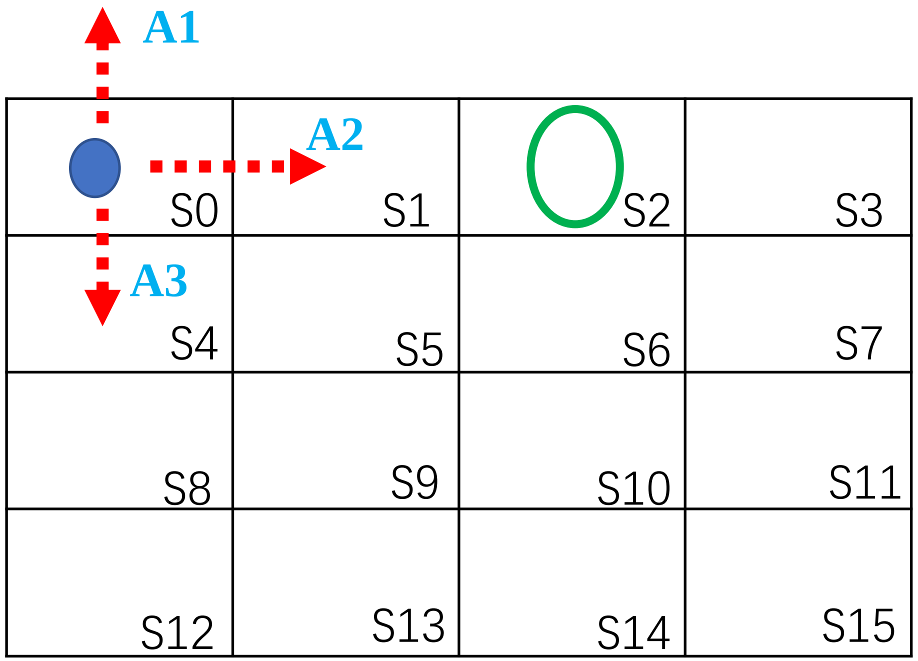


Q Table

	A1	A2	A3
S0	0	0	1

Reinforcement Learning (RL)

第一轮



Q Table

	A1	A2	A3
S0	0	2	1

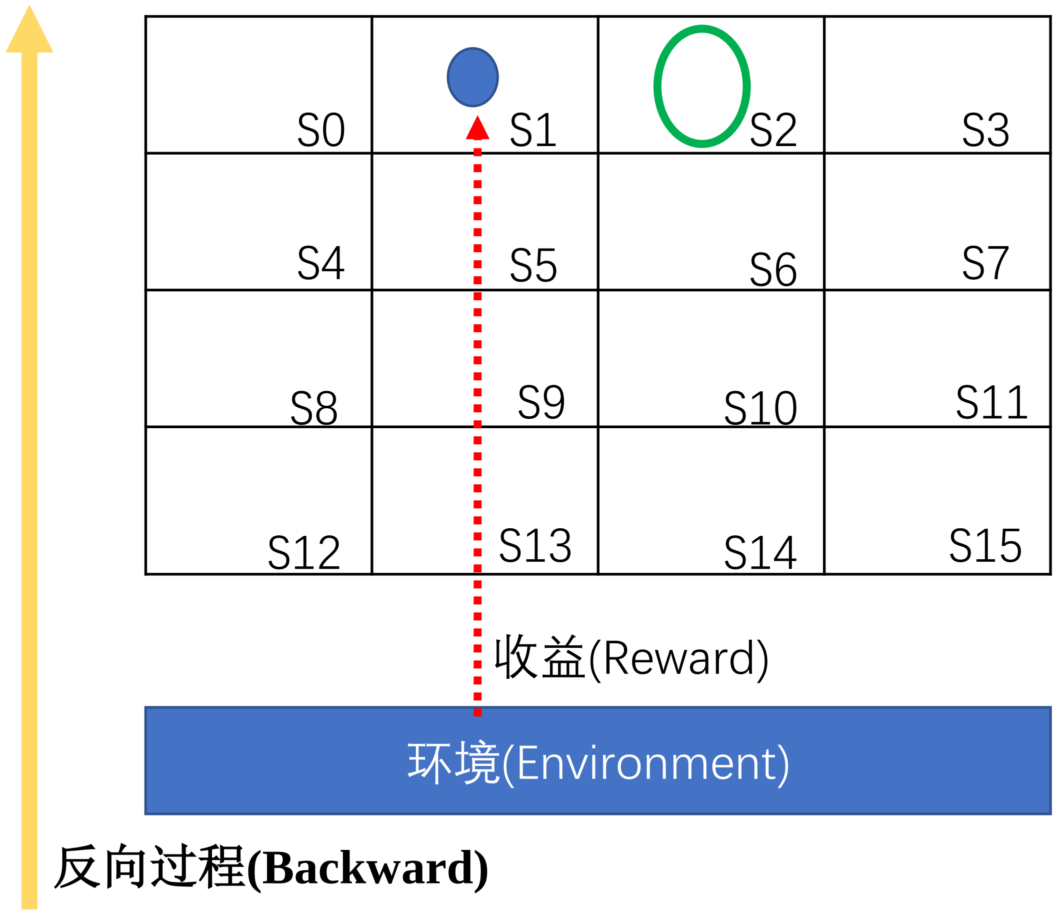
	A1	A2	A3
S1	0	5	4

	A1	A2	A3
S2			

前向过程(Forward)

Reinforcement Learning (RL)

<u>*第一轮*</u>



Q Table

	A1	A2	A3
S0	0	2	1

	A1	A2	A3
S1	0	5	4

Update

	A1	A2	A3
S0	0	5	1

$Reward = 0$ ⟵ 真实收益+$\max Q(S_1, A) = 5$未来收益

$Q(S_0, A_2) = 2$ ⟵ 估计收益

Update

$$Q(S_0, A_2) = Q(S_0, A_2) + (\text{Reward} + \max Q(S_1, A) - Q(S_0, A_2))$$

Reinforcement Learning (RL)

第二轮

S0	S1	S2	S3
S4	S5	S6	S7
S8	S9	S10	S11
S12	S13	S14	S15

Q Table

	A1	A2	A3
S0	0	5	1

	A1	A2	A3
S1	0	5	4

Reinforcement Learning (RL)

旧的估计收益 ***实际收益=当前真实收益+未来估计收益*** ***旧的估计收益***

$$Q(S,A) \leftarrow \underline{Q(S,A)} + \boxed{\alpha}\left(\underline{Reward + \boxed{\gamma}\,\underline{maxQ(S',A')}} - \underline{Q(S,A)}\right)$$

学习率

衰减因子:表示远见程度

下一个状态的最大收益估计值
(在下一个状态执行某个动作
可以带来的最大收益的估计值)

Reinforcement Learning (RL)

Q-learning: An off-policy TD control algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Reinforcement Learning (RL)

- ***(基于策略的强化学习) Policy-based RL***

建模Policy Gradient

S

A1	A2	A3	…	AN
P_{A1}	P_{A2}	P_{A3}	…	$\boxed{P_{AN}}$

对应行为执行概率

环境(Environment)

A1	A2	A3	…	AN
$Score_{A1}$	$Score_{A2}$	$Score_{A3}$	…	$\boxed{Score_{AN}}$

对应行为执行后获得收益

Reinforcement Learning (RL)

S

A1	A2	A3	…	AN
P_{A1}	P_{A2}	P_{A3}	…	P_{AN}

环境(Environment)

A1	A2	A3	…	AN
$Score_{A1}$	$Score_{A2}$	$Score_{A3}$	…	$Score_{AN}$

建模出发点：
我们希望得分高的行为在下一轮遇到S状态时，执行该行为的概率高

Reinforcement Learning (RL)

建模出发点:
我们希望<u>得分高的行为</u>在<u>下一轮</u>遇到S状态时，<u>执行该行为的概率高 （得分高的行为被执行的概率高）</u>

在S状态下，最大化行为得分期望

A1	A2	A3	E
0.5	0.3	0.2	3.8
5	3	2	

A1	A2	A3	E
0.8	0.1	0.1	4.5
5	3	2	

Reinforcement Learning (RL)

参数化的条件概率
(在s状态下行为a执行的概率)

在s状态下行为a所获得的收益值

$P_\theta(a|s,\theta)$

$R_{s,a}$

$$J(\theta) = max_\theta \sum_{a \in A} P_\theta(a|s,\theta) R_{s,a}$$

梯度下降优化

Trick：最大似然比

$$\nabla_\theta log(P_\theta(a|s,\theta)) = \frac{1}{P_\theta(a|s,\theta)} \nabla_\theta P_\theta(a|s,\theta)$$

$$P_\theta(a|s,\theta) \nabla_\theta log(P_\theta(a|s,\theta)) = \nabla_\theta P_\theta(a|s,\theta)$$

$$\nabla_\theta J(\theta) = \nabla_\theta \textstyle\sum_{a \in A} P_\theta(a|s,\theta) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{a \in A} \nabla_\theta P_\theta(a|s,\theta) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{a \in A} P_\theta(a|s,\theta) \nabla_\theta log(P_\theta(a|s,\theta)) R_{s,a}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{P_\theta}[\nabla_\theta log(P_\theta(a|s,\theta))\ R_{s,a}]$$

Reinforcement Learning (RL)

$$\nabla_\theta J(\theta) = \sum_{s \in S} d(s) \sum_{a \in A} P_\theta(a|s,\theta) \nabla_\theta log(P_\theta(a|s,\theta)) R_{s,a}$$

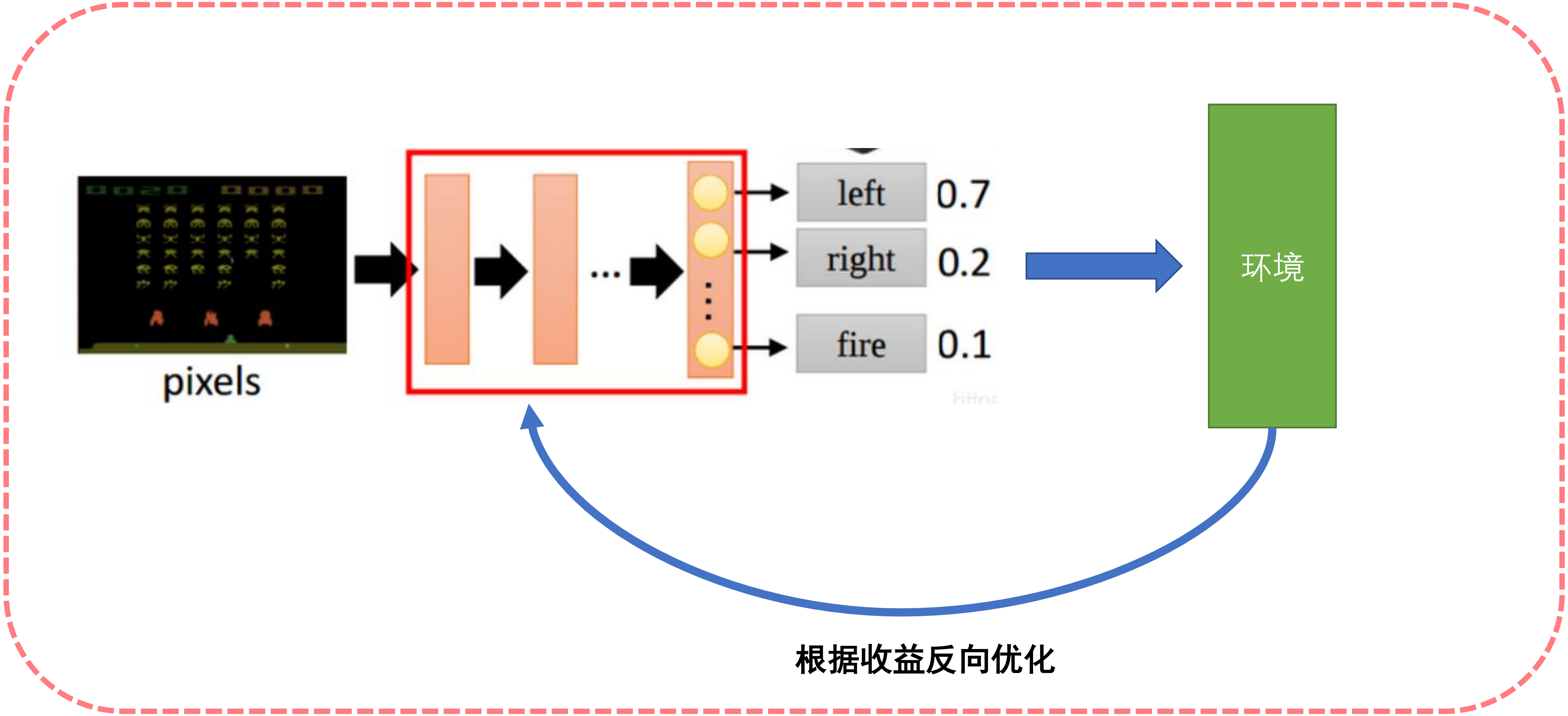
$$\nabla_\theta J(\theta) = \sum_{a \in A} P_\theta(a|s,\theta) \nabla_\theta log(P_\theta(a|s,\theta)) Q(S,a)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{P_\theta}[\nabla_\theta log(P_\theta(a|s,\theta)) Q(S,a)]$$

Monte-Carlo Policy Gradient

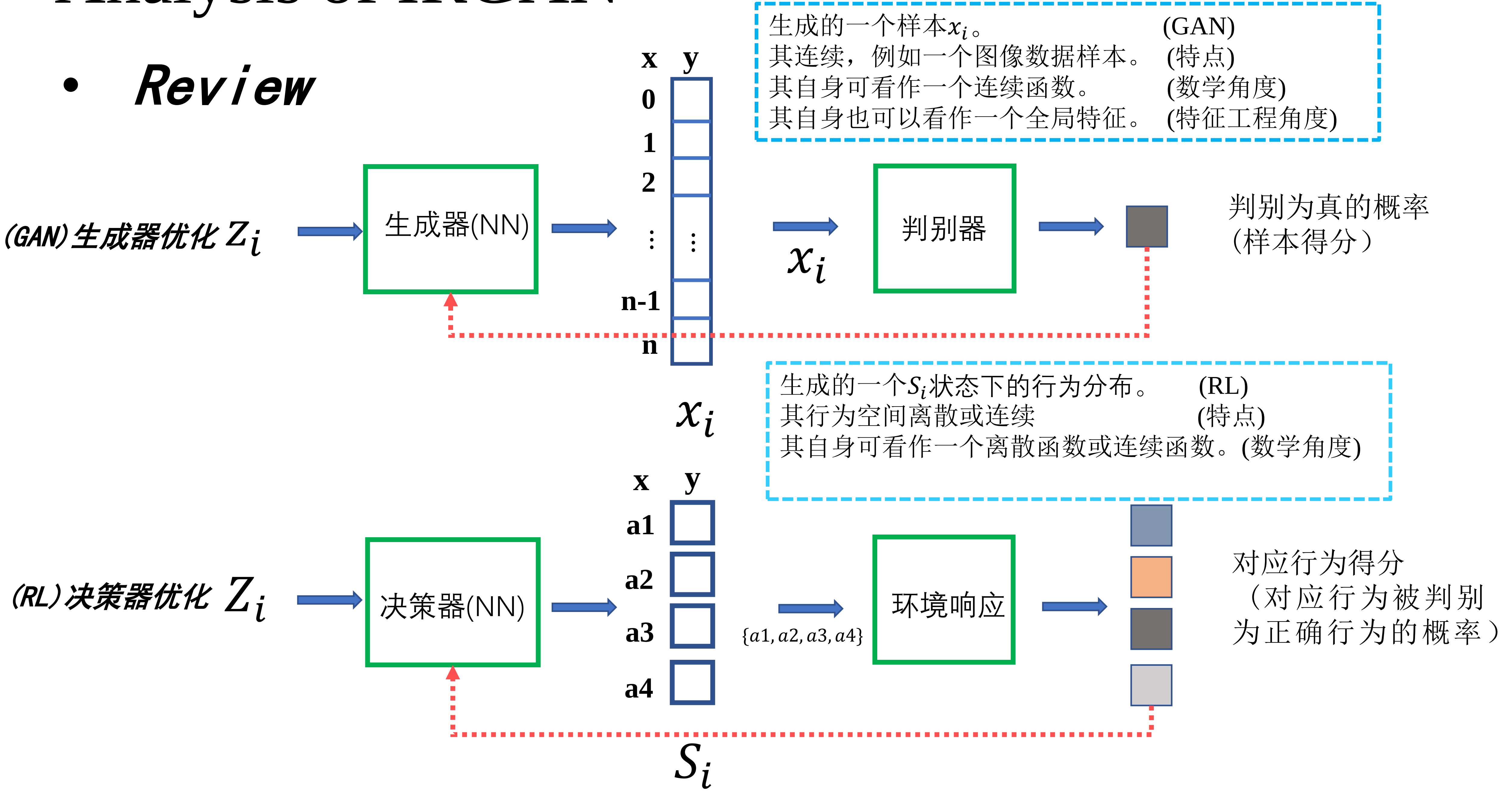
function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

Reinforcement Learning (RL)

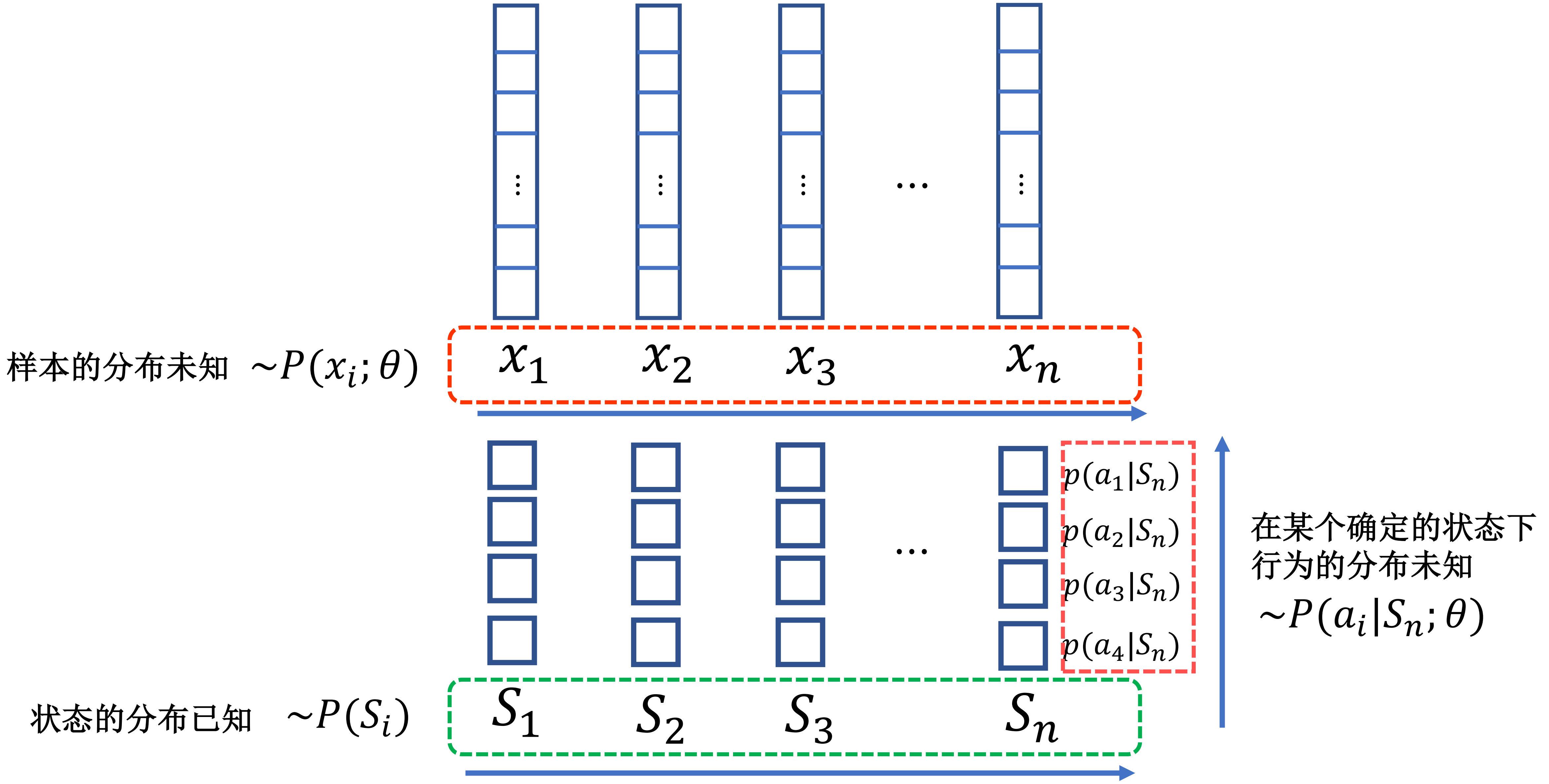


Analysis of IRGAN

- ***Review***



Analysis of IRGAN



Analysis of IRGAN

$$\hat{\theta} = argmin_{\theta_G} \sum_{i=1}^{m} P_G(x_i)\log(1 - D(Y = 1|x_i;\theta_D))$$

G含参数$\boldsymbol{\theta_G}$，通过优化$\boldsymbol{\theta_G}$来改变$\boldsymbol{x_i}$的分布，故$\boldsymbol{x_i}$的分布$\boldsymbol{P_G(x_i)}$受$\boldsymbol{\theta_G}$影响，**应写作$\boldsymbol{P_G(x_i;\theta_G)}$**

$$\hat{\theta} = argmin_{\theta_G} \sum_{i=1}^{m} P_G(x_i;\theta_G)\log(1 - D(Y = 1|x_i = G(z_i;\theta_G);\theta_D))$$

连续

离散化

离散

$$\hat{\theta} = argmin_{\theta_G} \sum_{i=1}^{m}\sum_{t=1}^{n} P_G(a_t|x_i;\theta_G)\log(1 - D\,(Y = 1|(a_t|x_i) = G(z_i;\theta_G);\theta_D))$$

$$\hat{\theta} = argmin_{\theta_G} \sum_{i=1}^{m}\sum_{t=1}^{n} \boxed{P_G(a_t|x_i;\theta_G)}\boxed{\log(1 - D\,(Y = 1|(a_t|x_i);\theta_D))}$$

Reward

行为分布

使用RL中 Policy Gradient 优化

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{t=1}^{n} P_G(a_t|x_i;\theta_G)\log(1 - D\,(Y = 1|(a_t|x_i);\theta_D)$$

Analysis of IRGAN

$\{q_1, q_2, \cdots q_n\}$ search keywords question

$\{d_1, d_2, \cdots d_n\}$ textual documents answers

Form: *Jun Wang*, **IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models**[Paper],SIGIR,*2017*

Analysis of IRGAN

$$J^{G^*,D^*} = \min_{\theta} \max_{\phi} \sum_{n=1}^{N} \Big(\mathbb{E}_{d \sim p_{\text{true}}(d|q_n,r)} \left[\log D(d|q_n) \right] + \tag{1}$$
$$\mathbb{E}_{d \sim p_{\theta}(d|q_n,r)} \left[\log(1 - D(d|q_n)) \right] \Big) ,$$

Analysis of IRGAN

$$D(d|q) = \sigma(f_\phi(d,q)) = \frac{\exp(f_\phi(d,q))}{1+\exp(f_\phi(d,q))} \,. \qquad (2)$$

$$\phi^* = \arg\max_{\phi} \sum_{n=1}^{N} \Big(\mathbb{E}_{d\sim p_{\text{true}}(d|q_n,r)} \left[\log(\sigma(f_\phi(d,q_n))\right] + \mathbb{E}_{d\sim p_{\theta^*}(d|q_n,r)} \left[\log(1-\sigma(f_\phi(d,q_n)))\right] \Big), \qquad (3)$$

Analysis of IRGAN

$$\theta^* = \arg\min_\theta \sum_{n=1}^{N} \Big(\mathbb{E}_{d\sim p_{\text{true}}(d|q_n,r)} \left[\log \sigma(f_\phi(d,q_n))\right] + \mathbb{E}_{d\sim p_\theta(d|q_n,r)} \left[\log(1-\sigma(f_\phi(d,q_n)))\right] \Big)$$
$$= \arg\max_\theta \sum_{n=1}^{N} \underbrace{\mathbb{E}_{d\sim p_\theta(d|q_n,r)} \left[\log(1+\exp(f_\phi(d,q_n)))\right]}_{\text{denoted as } J^G(q_n)}, \quad (4)$$

$$\begin{aligned}
&\nabla_\theta J^G(q_n) \\
&= \nabla_\theta \mathbb{E}_{d\sim p_\theta(d|q_n,r)} \left[\log(1+\exp(f_\phi(d,q_n)))\right] \\
&= \sum_{i=1}^{M} \nabla_\theta p_\theta(d_i|q_n,r) \log(1+\exp(f_\phi(d_i,q_n))) \\
&= \sum_{i=1}^{M} p_\theta(d_i|q_n,r) \nabla_\theta \log p_\theta(d_i|q_n,r) \log(1+\exp(f_\phi(d_i,q_n))) \\
&= \mathbb{E}_{d\sim p_\theta(d|q_n,r)} \left[\nabla_\theta \log p_\theta(d|q_n,r) \log(1+\exp(f_\phi(d,q_n)))\right] \\
&\simeq \frac{1}{K} \sum_{k=1}^{K} \nabla_\theta \log p_\theta(d_k|q_n,r) \log(1+\exp(f_\phi(d_k,q_n))), \qquad (5)
\end{aligned}$$

Analysis of IRGAN

Algorithm 1 Minimax Game for IR (a.k.a IRGAN)

Input: generator $p_\theta(d|q,r)$; discriminator $f_\phi(x_i^q)$;
training dataset $\mathcal{S} = \{x\}$

```
Initialise p_θ(d|q, r), f_φ(q, d) with random weights θ, φ.
Pre-train p_θ(d|q, r), f_φ(q, d) using S
repeat
    for g-steps do
        p_θ(d|q, r) generates K documents for each query q
        Update generator parameters via policy gradient Eq. (22)
    end for
    for d-steps do
        Use current p_θ(d|q, r) to generate negative examples and com-
        bine with given positive examples S
        Train discriminator f_φ(q, d) by Eq. (3)
    end for
until IRGAN converges
```