

2 Logic and reasoning

2.1 propositional logic

表2.2 五种主要联结词构成的复合命题的真值表

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

2.2 predicate logic

定理 2.3 在约束变元相同的情况下，量词的运算满足分配律。

设 $A(x)$ 和 $B(x)$ 是包含变元 x 的谓词公式，则存在如下关系：

$$(\forall x)(A(x) \vee B(x)) \equiv (\forall x)A(x) \vee (\forall x)B(x) \quad (\text{不成立})$$

$$(\forall x)(A(x) \wedge B(x)) \equiv (\forall x)A(x) \wedge (\forall x)B(x)$$

$$(\exists x)(A(x) \vee B(x)) \equiv (\exists x)A(x) \vee (\exists x)B(x)$$

$$(\exists x)(A(x) \wedge B(x)) \equiv (\exists x)A(x) \wedge (\exists x)B(x) \quad (\text{不成立})$$

定理 2.4 当公式中存在多个量词时，若多个量词都是全称量词或者都是存在量词，则量词的位置可以互换；若多个量词中既有全称量词又有存在量词，则量词的位置不可以随意互换。

设 $A(x, y)$ 是包含变元 x, y 的谓词公式，则如下关系成立：

$$(\forall x)(\forall y)A(x, y) \Leftrightarrow (\forall y)(\forall x)A(x, y)$$

$$(\exists x)(\exists y)A(x, y) \Leftrightarrow (\exists y)(\exists x)A(x, y)$$

$$(\forall x)(\forall y)A(x, y) \Rightarrow (\exists y)(\forall x)A(x, y)$$

$$(\forall x)(\forall y)A(x, y) \Rightarrow (\exists x)(\forall y)A(x, y)$$

$$(\exists y)(\forall x)A(x, y) \Leftrightarrow (\forall x)(\exists y)A(x, y)$$

$$(\exists x)(\forall y)A(x, y) \Leftrightarrow (\forall y)(\exists x)A(x, y)$$

$$(\forall x)(\exists y)A(x, y) \Rightarrow (\exists y)(\exists x)A(x, y)$$

$$(\forall y)(\exists x)A(x, y) \Rightarrow (\exists x)(\exists y)A(x, y)$$

2.3 knowledge map reasoning

FOIL推理得到问号的部分： $(\forall x)(\forall y)(\dots)(?? \wedge ?? \wedge \dots \rightarrow P(x, y))$

知识图谱推理：FOIL (First Order Inductive Learner)

$$(\forall x)(\forall y)(\forall z)(Mother(z, y) \wedge Couple(x, z) \rightarrow Father(x, y))$$



前提约束谓词
(学习得到)



目标谓词
(已知)

哪些谓词好呢？可以作为目标谓词的前提约束谓词？

FOIL中信息增益值
(information gain)

FOIL信息增益值计算方法如下：

$$FOIL_Gain = \widehat{m}_+ \cdot \left(\log_2 \frac{\widehat{m}_+}{\widehat{m}_+ + \widehat{m}_-} - \log_2 \frac{m_+}{m_+ + m_-} \right)$$

其中， \widehat{m}_+ 和 \widehat{m}_- 是增加前提约束谓词后所得新推理规则覆盖的正例和反例的数量， m_+ 和 m_- 是原推理规则所覆盖的正例和反例数量。

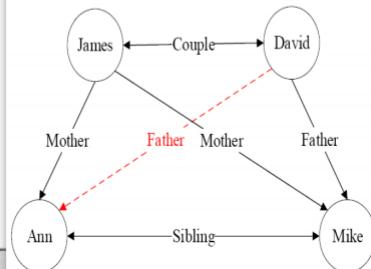
知识图谱推理：FOIL (First Order Inductive Learner)

推理规则		推理规则涵盖的正例和反例数		FOIL信息增益值
目标谓词	前提约束谓词	正例	反例	信息增益值
$Father(x, y) \leftarrow$	空集	$m_+ = 1$	$m_- = 4$	$FOIL_Gain$
	$Mother(x, y)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 2$	NA
	$Mother(x, z)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 2$	NA
	$Mother(y, x)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 1$	NA
	$Mother(y, z)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 1$	NA
	$Mother(z, x)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 1$	NA
	$Mother(z, y)$	$\widehat{m}_+ = 1$	$\widehat{m}_- = 3$	0.32
	$Sibling(x, y)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 1$	NA
	$Sibling(x, z)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 1$	NA
	$Sibling(y, x)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 0$	NA
	$Sibling(y, z)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 0$	NA
	$Sibling(z, x)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 0$	NA
	$Sibling(z, y)$	$\widehat{m}_+ = 1$	$\widehat{m}_- = 2$	0.74
	$Couple(x, y)$	$\widehat{m}_+ = 0$	$\widehat{m}_- = 1$	NA
	$Couple(x, z)$	$\widehat{m}_+ = 1$	$\widehat{m}_- = 1$	1.32

$Mother(x, y) \rightarrow Father(x, y)$

覆盖正例和反例数量分别为0和2，即 $\widehat{m}_+ = 0$, $\widehat{m}_- = 2$

由于 $\widehat{m}_+ = 0$ ，代入 $FOIL_Gain$ 公式时会出现负无穷的情况，此时 $FOIL_Gain$ 记为NA (Not Available)



PRA推理：给出路径组成的向量和标签，依赖分类器来得到规则（结果）

知识图谱推理：路径排序推

例 2.16 对图2.1中的知识图谱采用路径排序算法。

(1) 目标关系: *Father*

(2) 对于目标关系*Father*, 生成四组训练样例, 一个为正例、三个为负例:

正例: (David, Mike)

负例: (David, James), (James, Ann), (James, Mike)

(3) 从知识图谱采样得到路径, 每一路径链接上述每个训练样例中两个实体:

(David, Mike)对应路径: *Couple* → *Mother*

(David, James)对应路径: *Father* → *Mother*⁻¹ (*Mother*⁻¹与 *Mother*为相反关系)

(James, Ann)对应路径: *Mother* → *Sibling*

(James, Mike)对应路径: *Couple* → *Father*

(4) 对于每一个正例/负例, 判断上述四条路径可否链接其包含的两个实体, 将可链接(记为1)和不可链接(记为0)作为特征, 于是每一个正例/负例得到一个四维特征向量:

(David, Mike): {[1, 0, 0, 0], 1}

(David, James): {[0, 1, 0, 0], -1} 向量表示四条路径

最后一个±1表示正/负例

(James, Ann): {[0, 0, 1, 0], -1}

(James, Mike): {[0, 0, 1, 1], -1} (*Mother* → *Sibling* 和 *Couple* → *Father* 两条路径均包含了James和Mike)

(5) 依据(4)中的训练样本, 训练分类器M。

(6) 预测。对于图2.1中形如(David, Ann)的样例, 得到其特征值为[1, 0, 0, 0] (*Couple* → *Mother* 这一条路径包含了David和Ann), 将特征向量输入到分类器M中, 如果分类器M给出分类结果为1, 则 *Father*(David, Ann)成立。

3 Search

- 图搜索维护闭表
- 树搜索不维护闭表, 在搜索到终点之前允许结点重复出现

3.2 heuristic search

- 贪婪
 - 直接令评价函数等于启发函数
- A*

$$f(n) = \underbrace{g(n)}_{\substack{\text{评价函数} \\ \text{起始结点到结点 } n \text{ 代价} \\ (\text{当前最小代价})}} + \underbrace{h(n)}_{\substack{\text{结点 } n \text{ 到目标结点代价} \\ (\text{后续估计最小代价})}}$$

辅助信息	所求解问题之外、与所求解问题相关的特定信息或知识。	
评价函数 (evaluation function) $f(n)$	从当前结点 n 出发, 根据评价函数来选择后续结点。	下一个结点是谁?
启发函数 (heuristic function) $h(n)$	计算从结点 n 到目标结点之间所形成路径的最小代价, 这里将两点之间的直线距离作为启发函数。	完成任务还需要多少代价?

- 可容性admissible 启发函数不会高估实际代价
- 一致性consistency $h(n) \leq c(n, a, n') + h(n')$
- 一致性比可容性强, 如果一个启发函数满足一致性且终止节点的启发函数值均为0, 则必定可容

3.3 adversarial search

- method: minmax search
 - 改进
 - alpha-beta剪枝：剪去其实不需要遍历的节点
 - alpha初始化为 $-\infty$, 会越来越大; beta初始化为 $+\infty$, 会越来越小
 - alpha和beta继承自父节点
 - MAX层更新alpha, MIN层更新beta (MAX层需要记录已知的最大收益, MIN层相反, 更新值来自子节点)
 - 如果一个节点满足 $\text{alpha} \geq \text{beta}$, 其未被访问的子节点将被剪去
- method: Monte-Carlo tree search
 - 选择 (selection) : 选择指算法从搜索树的根节点开始, 向下递归选择子节点, 直至到达叶子节点或者到达具有还未被扩展过的子节点的节点L。这个向下递归选择过程可由UCB1算法来实现, 在递归选择过程中记录下每个节点被选择次数和每个节点得到的奖励均值。
 - 扩展 (expansion) : 如果节点L不是一个终止节点 (或对抗搜索的终局节点),
 - 则随机扩展它的一个未被扩展过的后继边缘节点M。
 - 模拟 (simulation) : 从节点M出发, 模拟扩展搜索树, 直到找到一个终止节点。模拟过程使用的策略和采用UCB1算法实现的选择过程并不相同, 前者通常会使用比较简单的策略, 例如使用随机策略。
 - 反向传播 (Back Propagation) : 用模拟所得结果 (终止节点的代价或游戏终局分数) 回溯更新模拟路径中M以上 (含M) 节点的奖励均值和被访问次数。
 - UCB(upper bound):
$$\frac{\text{self.reward}}{\text{self.visited}} + C \sqrt{\frac{2 \ln \text{parent.visited}}{\text{self.visited}}}$$
 - MAX层的分数为负数 (相反数)

4 Supervised learning

4.2 regression

- 一元线性回归
 - 求导求解
- 多元线性回归
 - 求导求解
- Logistic regression
 - 目的: 对付离群点
 - 使用梯度下降求解

$$P(y=1|x) = h_\theta(x) = \frac{1}{1+e^{-(w^T x + b)}}$$

o

$$h_\theta(x) = \frac{e^{-(w^T x + b)}}{1+e^{-(w^T x + b)}}。 \theta \text{ 表示模型参数 } (\theta = \{w, b\})。 \text{ 于是有:}$$

$$\text{logit}(P(y=1|x)) = \log\left(\frac{P(y=1|x)}{P(y=0|x)}\right) = \log\left(\frac{P}{1-P}\right) = w^T x + b$$

4.3 decision tree

- 根据信息把大样本集合划分成多个小样本集合，递归进行

- $E(D) = - \sum_{k=1}^K p_k \log_2 p_k$
- $\text{Gain}(D, A) = Ent(D) - \sum_{i=1}^n \frac{|D_i|}{|D|} Ent(D_i)$

4.4 LDA

- 有标签分类、降维
- 原则：“类内方差小，类间间隔大”，表现为类内的协方差小，类的均值间隔大

对线性判别分析的降维步骤描述如下：

- (1) 计算数据样本集中每个类别样本的均值；
- (2) 计算类内散度矩阵 S_w 和类间散度矩阵 S_b ；
- (3) 根据 $S_w^{-1} S_b W = \lambda W$ 来求解 $S_w^{-1} S_b$ 所对应前 r 个最大特征值所对应特征向量 (w_1, w_2, \dots, w_r) ，构成矩阵 W ；
- (4) 通过矩阵 W 将每个样本映射到低维空间，实现特征降维。

需要注意的是，通过LDA对原始 d 维数据进行降维后，所得维度 r 最大取值为 $\min(K-1, d)$ ，这是因为 S_b 的秩为 $\min(K-1, d)$ 。这也说明了在二分类问题中，原始高维数据只能被投影到一维空间中（无论其原始维度是多少）。

4.5 Ada Boosting

- 将弱分类器线性组合成强分类器
- 弱分类器：错误率略低于随机分类的算法
- 训练过程见教材P137
 - 若某轮训练的弱分类器错误分类了某个样本，该样本的惩罚会被增加

5 Unsupervised learning

5.1 K-means (分类)

- steps
 1. 初始化聚类质心
 2. 根据预定的相似度 / 距离函数（通常为欧氏距离）对数据进行聚类
 3. 更新聚类质心
 4. 重复2和3直到收敛
- 迭代终止
 - 到达了次数上限
 - 相邻两次迭代的质心不变
- 对数据的尺度（单位/坐标空间）敏感
- 找到局部最优，不确保找到全局最优

5.2 PCA (降维)

- 无标签降维
- 根据方差实现降维
- 其中 tr 表示矩阵的迹 (trace)，即一个方阵主对角线（从左上方到右下方的对角线）上各个元素的总和。降维前 n 个 d 维样本数据 X 的协方差矩阵记为：

$$\Sigma = \frac{1}{n-1} X^T X$$

主成分分析的优化求解目标函数为：

$$\max_w \text{tr}(W^T \Sigma W)$$

该优化求解目标需要满足如下约束条件

$$w_i^T w_i = 1 \quad i \in \{1, 2, \dots, l\}$$

这是带约束的最优化问题求解，可以通过拉格朗日乘子法将上述转化为无约束的最优化问题，拉格朗日函数如下：

$$L(W, \lambda) = \text{tr}(W^T \Sigma W) - \sum_{i=1}^l \lambda_i (w_i^T w_i - 1)$$

算法 5.2 主成分分析

输入： n 个 d 维样本数据所构成的矩阵 X ，降维后的维数 l 。
输出： 映射矩阵 $W = \{w_1, w_2, \dots, w_l\}$ 。

算法步骤：

1. 对于每个样本数据 x_i 进行中心化处理： $x_i = x_i - \mu$, $\mu = \frac{1}{n} \sum_{j=1}^n x_j$;
2. 计算原始样本数据的协方差矩阵： $\Sigma = \frac{1}{n-1} X^T X$;
3. 对协方差矩阵 Σ 特征值分解，对所得特征根进行排序 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$;
4. 取前 l 个最大特征根所对应特征向量 w_1, w_2, \dots, w_l 组成映射矩阵。

- 其他常用降维方法：

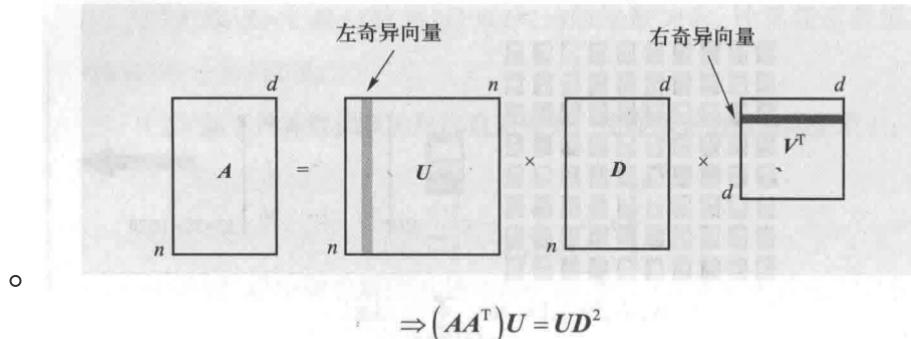
- 非负矩阵分解
- 多维尺度法
- 局部线性嵌入

5.3 Eigenface (降维)

- 奇异值分解

- $A = UDV^T$

- 特征维度较高时，PCA暴力求特征向量比较耗时，因此使用奇异值分解来进行降维



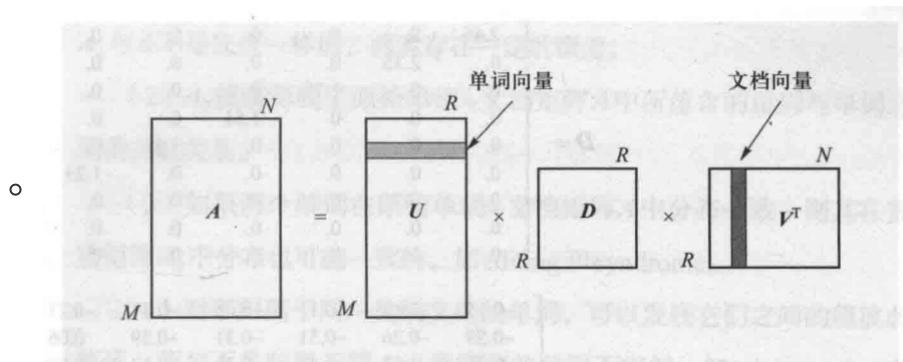
所以矩阵 U 即为矩阵 AA^T 所有特征向量构成的矩阵，同理矩阵 V 即为矩阵 A^TA 所有特征向量构成的矩阵。同时，矩阵 D 就等于矩阵 AA^T 或矩阵 A^TA 的所有特征值开根号后组成的对角矩阵。或者也可以使用矩阵 V 和矩阵 U ，通过式子 $U^TAV = D$ 得到矩阵 D 。

- 特征人脸方法

- 本质是使用一组特征向量线性组合来表示原始人脸，进而实现人脸识别
- 每张特征人脸表示为 $1 \times d^2$ 的列向量，将其转化为 n 张特征人脸的 $1 \times n$ 向量表示（特征人脸空间大小为 $d^2 \times n$ ）

5.4 latent semantic analysis (降维)

- 奇异值分解



矩阵 D 是一个 $R \times R$ 的对角矩阵，其对角线上值按照从大到小进行排序。

$$D = \text{diag}(\sigma_1, \dots, \sigma_R), \sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_k \geq \dots \geq \sigma_R$$

矩阵 U 是一个 $M \times R$ 的矩阵， U 中的每一个行向量被称为 LSI 单词向量 (LSI term vectors)。每个 M 维大小的文档可以被单词向量嵌入到 R 维空间。

V 是一个 $N \times R$ 的矩阵， V^T 是一个 $R \times N$ 的矩阵， V 中的每一个行向量被称为 LSI 文档向量 (LSI document vectors)。

可以认为，原始文档和单词内嵌在了一个隐性空间中，这个隐性空间中单词向量和文档向量的表示蕴含了单词和单词之间、文档和文档之间的关联关系，正交矩阵 U 、正交矩阵 V 和对角矩阵 D 联合起来将原始单词-文档矩阵表示成了线性无关的向量或因素值 (factor values)。和特征人脸中类似，如果取前 k 个奇异值 (这里的 k 即前述的 R)，以及它们对应的维度为 k 的单词向量和文档向量，就可以重建一个维度为 $M \times N$ 的矩阵 A_k ， $A_k = \sum_{i=1}^k u_i \sigma_i v_i^T$ 。很显然，重建矩阵 A_k 与原始矩阵 A 不完全相同，是对原始矩阵 A 的逼近。

5.5 expectation maximization algorithm (概率)

- 解决含有隐变量问题的参数估计方法

- 在上面的表示基础上，优化目标为求解合适的 Θ 和 Z 使得对数似然函数最大：

$$(\Theta, Z) = \underset{\Theta, Z}{\operatorname{argmax}} L(\Theta, Z) = \underset{\Theta, Z}{\operatorname{argmax}} \sum_{i=1}^n \log \sum_{z_i} P(x_i, z_i | \Theta)$$

但是，优化求解含有未观测数据 Z 的对数似然函数 $L(\Theta, Z)$ 十分困难，EM 算法不断构造对数似然函数 $L(\Theta, Z)$ 的一个下界 (E 步骤)，然后最大化这个下界 (M 步骤)，以迭代方式逼近模型参数所能取得极大似然值。

$$\begin{aligned} \sum_{i=1}^n \log \sum_{z_i} P(x_i, z_i | \Theta) &= \sum_{i=1}^n \log \sum_{z_i} Q_i(z_i) \frac{P(x_i, z_i | \Theta)}{Q_i(z_i)} \\ &\geq \underbrace{\sum_{i=1}^n \sum_{z_i} Q_i(z_i) \log \frac{P(x_i, z_i | \Theta)}{Q_i(z_i)}}_{\text{对数似然函数下界}} \end{aligned}$$

算法 5.3 EM 算法

输入：观测所得样本数据 X 及其对应的隐变量 Z (即无法观测数据)、联合分布 $P(X, Z | \Theta)$ 。
输出：模型参数 Θ 。

算法步骤：

- 初始化参数取值 Θ^0 。
- 求取期望步骤 (E 步骤)：计算 $Q(\Theta | \Theta^t) = \sum_{i=1}^n \sum_{z_i} P(z_i | x_i, \Theta) \log P(x_i, z_i | \Theta)$ 。

其含义是对数似然函数 $\log P(x_i, z_i | \Theta)$ 在已观测数据 X 和当前参数 Θ^t 下去估计隐变量 Z 的条件概率分布 $P(z_i | x_i, \Theta)$ 。

- 期望最大化步骤 (M 步骤)： $\Theta^{t+1} = \underset{\Theta}{\operatorname{argmax}} Q(\Theta | \Theta^t)$ 。
- 重复第 2 步和第 3 步，直到收敛。

- 不准确理解：

- 隐变量的期望：首先计算隐变量的期望 (消去隐变量)

- 最大化参数：迭代更新参数与隐变量期望

6 Deep learning

6.2 feedforward neural network

- 每层神经元只和相邻层神经元相连

- 激活函数

- sigmoid $\frac{1}{1+e^{-x}}$

- 梯度消失

- tanh

- 梯度消失

- ReLU

- softmax $y_i = \text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}}$

- 损失函数

- 均方误差 $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$

- 交叉熵 $H(p, q) = -\sum_x p(x) \cdot \log q(x)$

- 参数学习

- 梯度下降

- 反向传播

6.3 CNN

- convolution

- correspond to receptive field

- down sampling (without padding)

- weights can be learned

- padding & striding

- pooling

- down sampling

- commonly used methods

- max pooling

- average pooling

- K-max pooling

- regularization

- 为了提高泛化能力

- Dropout

- batch normalization

- L1 & L2 regularization

- 如果网络参数很大（权重很大），往往代表着过拟合。损失函数加入了考虑权重大小的正则项，使得模型优化时倾向于向参数更小、更平滑、泛化能力方向更好的方向迭代

6.4 RNN

6.4.1 RNN

- 时刻 t 所得到的隐式编码 h_t , 是由上一时刻隐式编码 h_{t-1} , 和当前输入 x_t 共同参与生成的, 这可认为隐式编码 h_{t-1} 已经“记忆”了 t 时刻之前的时序信息, 或者说前序时刻信息影响了后续时刻信息的处理。与前馈神经网络和卷积神经网络在处理时需要将所有数据一次性输入不同, 循环神经网络在每一时刻都有数据输入且结合前一时刻所得结果进行计算, 这体现了循环神经网络可刻画序列数据时序依赖这一重要特点

- $$\begin{aligned} h_t &= \Phi(U \times x_t + W \times h_{t-1}) = \Phi(U \times x_t + W \times \Phi(U \times x_{t-1} + W \times h_{t-2})) \\ &= \Phi\left(U \times \underbrace{x_t}_{t\text{时刻输入}} + W \times \Phi\left(U \times \underbrace{x_{t-1}}_{t-1\text{时刻输入}} + W \times \Phi\left(U \times \underbrace{x_{t-2}}_{t-2\text{时刻输入}} + \dots\right)\right)\right) \end{aligned}$$

- 使用BPTT (backpropagation through time)求解
- 句子的向量编码
 - 最后一个单词的输出
 - 所有单词隐式编码的加权平均
- 有梯度消失、梯度爆炸、无法记忆远距离依赖关系等问题

6.4.2 long short-term memory network

表6.3 长短时记忆网络中所用符号及其含义

符号	内容描述或操作
x_t	时刻 t 的输入数据
i_t	输入门的输出: $i_t = \text{sigmoid}(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$ (W_{xi} 、 W_{hi} 和 b_i 为输入门的参数)
f_t	遗忘门的输出: $f_t = \text{sigmoid}(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$ (W_{xf} 、 W_{hf} 和 b_f 为遗忘门的参数)
o_t	输出门的输出: $o_t = \text{sigmoid}(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$ (W_{xo} 、 W_{ho} 和 b_o 为输出门的参数)
c_t	内部记忆单元的输出: $c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$ (W_{xc} 、 W_{hc} 和 b_c 为记忆单元的参数。输入门 i_t 控制有多少信息流入当前时刻内部记忆单元 c_t 、遗忘门控制上一时刻内部记忆单元 c_{t-1} 中有多少信息可累积到当前时刻内部记忆单元 c_t)
h_t	时刻 t 输入数据的隐式编码: $h_t = o_t \odot \tanh(c_t)$ $= o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c))$ (注: 输入门、遗忘门和输出门的信息 i_t 、 f_t 、 o_t 一起参与得到 h_t)
\odot	两个向量中对应元素按位相乘 ((element-wise product)。 如: $[10 \ 6 \ 3 \ 7] \odot [0.2 \ 0.1 \ 0.8 \ 0.5] = [2 \ 0.6 \ 2.4 \ 3.5]$

- 将上一时刻的记忆遗忘一部分后, 再接受当下输入的信息, 得到当下的记忆
- 当下的输入、上一时刻的隐式编码和当下的记忆共同得到当下的隐式编码
 - 由于 sigmoid 函数的值域在(0, 1)之间, 输入门、遗忘门和输出门三种门结构采用 sigmoid 这一非线性映射函数而起到了信息“控制门”的作用。在内部记忆单元和隐式编码中, 使用了 tanh 函数而没有继续使用 sigmoid 函数, 其原因在于 tanh 函数的值域为(-1, 1), 使得其在进行信息整合时可起到信息“增(为正)”或“减(为负)”的效果。

对于 $c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$, 有如下求导结果的
存在:

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t + \frac{\partial f_t}{\partial c_{t-1}} \times c_{t-1} + \dots$$

- 可见, $\frac{\partial c_t}{\partial c_{t-1}}$ 求导的结果至少大于等于 f_t , 即遗忘门的输出结果。如果遗忘门选择保留旧状态, 则这一求导结果就接近 1 (或者接近向量 1), 使得梯度是存在的, 从而避免了梯度消失问题。也就是说, LSTM 通过引入门结构, 在从 t 到 $t+1$ 过程中引入加法来进行信息更新, 避免了梯度消失问题。
- 内部记忆单元信息 c_t 好比长时记忆, 隐式编码 h_t 好比短时记忆

6.4.3 gated recurrent unit

- GRU 不再使用记忆单元来传递信息, 仅使用隐藏状态来进行信息的传递, 计算速度更快
- 简化过的LSTM

给定当前时刻输入数据 x_t 和前一时刻隐式编码 h_{t-1} , GRU 利用更新门和重置门来输出当前时刻隐式编码, 并且通过隐式编码将信息向后传递。记更新门和重置门信息为 z_t 和 r_t , GRU 的信息更新如下。

更新门信息输出: $z_t = \text{sigmoid}(W_z x_t + U_z h_{t-1} + b_z)$

重置门信息输出: $r_t = \text{sigmoid}(W_r x_t + U_r h_{t-1} + b_r)$

隐式编码输出: $h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \tanh(W_h x_t + U_h (r_t \otimes h_{t-1}) + b_h)$

与循环神经网络一样, 这里的 W_z 、 U_z 、 b_z 、 W_r 、 U_r 、 b_r 、 W_h 、 U_h 和 b_h 都是模型参数, 需要通过训练优化。

6.6 Natural language & CV

6.6.1 Word2Vec

- 用深度学习算法生成每个单词的向量表达, 可以保留更多信息
- 用权重作为词向量而不是输出

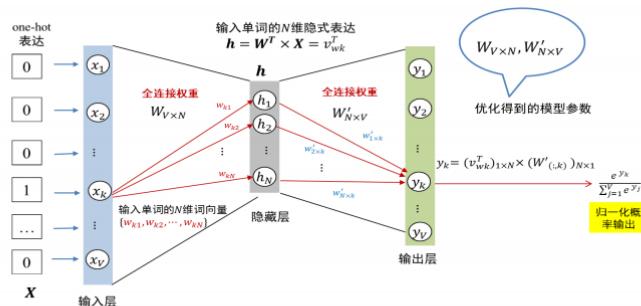


图6.5.2 单个单词的词向量(WordVec)生成示意图

在图6.5.2中, 输入层中 x_k 与隐藏层连接权重为 $\{w_{k1}, w_{k2}, \dots, w_{kN}\}$, 输入单词的 N 维隐式表达 $h = W^T \times X = v_{wk}^T$ 。所得得到的隐藏层结果与输出层是全连接, 全连接权重矩阵为 $W'_{N \times V}$ 。从图中可知, 第 k 个单词对应的输出为 $y_k = (v_{wk}^T)_{1 \times N} \times (W'_{(:,k)})_{N \times 1}$ 。这里 $W'_{(:,k)}$ 指的是从连接权重矩阵 $W'_{N \times V}$ 中取出第 k 列的向量。接着对 y_k 通过 Softmax 函数进行归一化, 得到第 k 个单词对应的归一化概率输出。显然, 第 k 个单词归一化的概率值应该远远大于输出层中其他位置所对应的归一化概率值。

这里 $W_{V \times N}$ 和 $W'_{N \times V}$ 为模型参数。一旦训练得到了图6.5.2的神经网络, 就可以将输入层中 x_k 与隐藏层连接权重为 $\{w_{k1}, w_{k2}, \dots, w_{kN}\}$ 作为第 k 个单词 N 维词向量(word vector)。

- CBOW

- 用上下文预测单词
- 用句子的所有单词的one-hot编码均值作为输入
- 取目标单词的连接权重作为向量
- skip-gram
 - 用单词预测上下文

6.6.2 image classification and object localization

- bounding box
- multi-task learning: 同时完成回归问题（位置判断）和分类问题（物体识别）
- CNN将处理后得到的结果并行分发给两个任务，两个任务的误差作为总误差

7 Reinforcement learning

7.1 problem definition

- 强化学习求解的是探索性和交互性的问题
- 同时包含了学习过程

表7.1 三种学习方式特点对比

	有监督学习	无监督学习	强化学习
学习依据	基于监督信息	基于对数据结构的假设	基于评估
数据来源	一次给定	一次给定	在时序交互中产生
决策过程	单步决策 (如分类和识别等)	无	序贯决策 (如棋类博弈)
学习目标	样本到语义标签的映射	数据的分布模式	选择能够获取最大收益 的状态到动作的映射

- Markov

一个随机过程实际上是一列随时间变化的随机变量。当时间是离散量时，一个随机过程可以表示为 $\{X_t\}_{t=0,1,2,\dots}$ ，这里每个 X_t 都是一个随机变量，这被称为离散随机过程。为了方便分析和求解，通常要求通过合理的问题定义使得一个随机过程满足马尔可夫性（Markov property），

- 即满足如下性质：

$$P(X_{t+1} = x_{t+1} | X_0 = x_0, X_1 = x_1, \dots, X_t = x_t) = P(X_{t+1} = x_{t+1} | X_t = x_t) \quad (\text{式 7.1})$$

这个公式的直观解释为：下一刻的状态 X_{t+1} 只由当前状态 X_t 决定（而与更早的所有状态均无关）。满足马尔可夫性的离散随机过程被称为马尔可夫链（Markov chain）。

至此有了足以描述智能体与环境交互过程的所有要素，可如下定义马尔可夫决策过程（Markov decision process, MDP）^[4, 36] 为 $MDP = (S, A, P, R, \gamma)$ 。其中的参数解释如下。

- 状态集合 S : 所求解问题中所有可能出现的状态所构成的集合，这个集合可能是一个有限的集合，也可能是一个无限的集合。
- 动作集合 A : 所求解问题中智能体能够采取的所有动作所构成集合，这个集合同样可以是有限的，也可以是无限的，本章所述内容中均假设该集合是一个有限集合。
- 状态转移概率 $P(S_{t+1}|S_t, A_t)$: 表示在当前状态 S_t 下采取了动作 A_t 后进入下一时刻状态 S_{t+1} 的概率。显然，状态转移概率满足马尔可夫性。状态转移可以是概率性的（stochastic），也可以是确定的（deterministic）。确定的状态转移指在给定状态 S_t 下采取了动作 A_t 后，转移到某一状态的概率为 1。
- 奖励函数 $R(S_t, A_t, S_{t+1})$: 在状态 S_t 执行了动作 A_t 后到达状态 S_{t+1} 时，智能体能够得到的奖励。
- 折扣因子 γ : 后续时刻奖励对当前动作的价值系数， $\gamma \in [0, 1]$ 。
- 奖励reward: 每个状态有一个对应的奖励
- 回报return: 每个时刻有一个对应的回报，反映了该时刻之后的奖励累积

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$
 - 强化学习中的奖励机制更加注重短期收益，而非长期收益
- 一个好的策略函数应该能够使得智能体在采取了一系列行动后可得到最佳奖励，即最大化每一时刻的回报值 $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$ 。
从 G_t 的定义可知， G_t 要根据一次包含了终止状态的轨迹序列来计算，因此定义如下函数。
 - 价值函数 (value function): $V: S \mapsto \mathbb{R}$ ，其中 $V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$ 表示智能体在时刻 t 处于状态 s 时，按照策略 π 采取行动时所获得回报的期望。价值函数衡量了某个状态的好坏程度，反映了智能体从当前状态出发后在将来还能获得多少好处。
 - 动作-价值函数(action-value function): $q: S \times A \mapsto \mathbb{R}$ ，其中 $q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$ ，表示智能体在时刻 t 处于状态 s 时，选择了动作 a 后，在 t 时刻后根据策略 π 采取行动所获得回报的期望。
- Bellman Equation

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\
&= \mathbb{E}_{a \sim \pi(s)} \left[\mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s, A_t = a \right] \right] \\
&= \sum_{a \in A} \underbrace{\pi(s, a)}_{\text{采取动作 } a \text{ 的概率}} \times \underbrace{q_\pi(s, a)}_{\text{采取动作 } a \text{ 后带来的回报期望}} \\
&= \sum_{a \in A} \pi(s, a) q_\pi(s, a)
\end{aligned} \tag{式 7.3}$$

可见，可以用动作-价值函数来表达价值函数，即从状态 s 出发，采用策略 π 完成任务所得回报的期望，可如下计算：在状态 s 下可采取的每

$$\begin{aligned}
q_\pi(s, a) &= \mathbb{E}_\pi [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s, A_t = a] \\
&= \mathbb{E}_{s' \sim P(\cdot|s, a)} [R(s, a, s') + \gamma \mathbb{E}_\pi [R_{t+2} + \gamma R_{t+3} + \dots | S_{t+1} = s']] \\
&= \sum_{s' \in S} \underbrace{P(s'|s, a)}_{\text{在状态 } s \text{ 采取行动 } a \text{ 进入状态 } s' \text{ 的概率}} \times \left[\underbrace{R(s, a, s')}_{\text{在 } s \text{ 采取 } a \text{ 进入 } s' \text{ 得到的回报}} + \gamma \times \underbrace{V_\pi(s')}_{\text{在 } s' \text{ 获得的回报期望}} \right] \\
\circ &= \sum_{s' \in S} P(s'|s, a) [R(s, a, s') + \gamma V_\pi(s')] \quad (\text{式 7.4})
\end{aligned}$$

- 于是，也可用价值函数来表示动作-价值函数，即在状态 s 下采取动作 a 所取得的价值，可如下计算：采取某个具体动作 a 进入状态 s' 的概率，乘以进入后续状态 s' 所得回报 $R(s, a, s')$ 与后续状态 s' 价值函数的折扣值之和，然后对在状态 s 采取动作 a 后所进入全部状态的如上取值进行累加。再加以替换可以看出，两个价值函数都与时间无关。

7.2 value based

7.2.1-7.2.3 generalized policy iteration, GPI: policy evaluation + policy improvement

- policy improvement:

- 如何改进我的策略？

可以证明，给定任意给定状态 $s \in S$ ，如果两个策略 π 和 π' 满足如下条件：

$$q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s))$$

那么对于该任意给定状态 $s \in S$ ，有

$$V_{\pi'}(s) \geq V_\pi(s)$$

即策略 π' 不比策略 π 差。这个结论称为策略优化定理 (policy improvement theorem)^[5]。注意 $q_\pi(s, \pi'(s))$ 的含义并不是在当前状态 s

- policy evaluation

- 如何判定我的策略在某个状态/动作下有多少价值？
 - dynamic programming

- 严格地实际走一遍，记录反馈

算法 7.1 基于动态规划的策略评估

函数：DPPolicyEvaluation()	
输入：策略 π ，状态转移函数 P ，奖励函数 R	
输出：价值函数 V_π	
1	随机初始化 V_π
2	repeat
3	foreach $s \in S$ do
4	$V_\pi(s) \leftarrow \sum_{a \in A} \sum_{s' \in S} P(s' s, a) [R(s, a, s') + \gamma V_\pi(s')]$
5	end
6	until V_π 收敛

- 要求状态转移概率函数 P 作为输入，即要求算法了解环境的运作机理，不现实
 - 状态集合很大时，效率低
 - monte-carlo
 - 随机地实际走一遍，记录整个片段的反馈

算法 7.2 基于蒙特卡洛方法的策略评估

函数: MCPolicyEvaluation()
输入: 策略 π
输出: 价值函数 V_π
■ 1 随机初始化 V_π
■ 2 对任意 $s \in S$, 初始化 $s.\text{returns}$ 为一个空列表
■ 3 for $i \leftarrow 1$ to MaxIter do
■ 4 根据 π 产生片段 D
■ 5 foreach $s \in S$ do
■ 6 $G \leftarrow s$ 在 D 中初次出现时的反馈
■ 7 append ($s.\text{returns}, G$)
■ 8 $V_\pi(s) \leftarrow \text{average}(s.\text{returns})$
■ 9 end
■ 10 end

- 对价值函数的估计难以准确
- 往往只有到达终止状态才能获得回报值, 导致采样序列长, 影响效率

- temporal difference

- combine the above methods
- 随机地走一步, 记录下一状态和当下动作的反馈

算法 7.3 基于时序差分法的策略评估

函数: TemporalDifference()
输入: 策略 π
输出: 价值函数 V_π
1 随机初始化 V_π
2 repeat
3 $s \leftarrow$ 初始状态
4 repeat
5 $a \sim \pi(s, \cdot)$
6 执行动作 a , 观察奖励 R 和下一个状态 s'
7 $V_\pi(s) \leftarrow V_\pi(s) + \alpha [R + \gamma V_\pi(s') - V_\pi(s)]$
8 $s \leftarrow s'$
9 until s 是终止状态
10 until V_π 收敛

- 边采样边更新价值函数, 用蒙特卡洛来回避dp需要状态转移概率的问题
- 时序差分法并非使用一个片段中的终止状态所提供的实际回报值来估计价值函数, 而是根据下一个状态的价值函数来估计

7.2.4 value-based reinforcement learning

- 注意书上本节讨论的是一种简单的策略确定的情形 (状态价值被确定的同时, 在这个状态采取什么动作也确定了, 因此不需要考虑转移概率)
- 为了方便配合策略优化定理, 这里的状态价值与动作价值无异
-

```

1 随机初始化  $\pi$ 
2 repeat
3    $q_s \leftarrow \text{PolicyEvaluation}(\pi)$ 
4   foreach  $s \in S$  do
5      $\pi(s) \leftarrow \text{argmax}_a q_s(s, a)$ 
6   end
7 until  $\pi$  收敛

```

算法 7.4 存在的主要问题是，策略评估函数本身也是迭代求解的，如果每次都要等待策略评估函数收敛，必然会影响算法执行的效率。另一方面，随着策略评估迭代次数的增多，动作-价值函数的变化也会越来越小，有时候甚至小到不足以影响新策略的取值，这种情况下继续对价值函数进行迭代计算的意义就不大了。

- 实际上，以更细的粒度进行策略评估和策略优化的迭代通常并不会影响算法的收敛性，而且往往有着更高的效率。例如，在策略评估动态规划法的基础上，每次迭代中只对一个状态进行策略评估和策略优化，就可以得到算法 7.5，这个算法被称为价值迭代（value iteration）算法^[20]。

算法 7.5 价值迭代算法

函数: ValueIteration ()
输入: 马尔可夫决策过程 $MDP = (S, A, P, R, \gamma)$
输出: 策略 π
<pre> 1 随机初始化 V_π 2 repeat 3 foreach $s \in S$ do 4 $V_\pi(s) \leftarrow \max_a \sum_{s'} P(s' s, a) [R(s, a, s') + \gamma V_\pi(s')]$ 5 end 6 until V_π 收敛 7 $\pi(s) := \arg \max_a \sum_{s'} P(s' s, a) [R(s, a, s') + \gamma V_\pi(s')]$ </pre>

- 经典但依赖状态转移函数，且计算所有的状态导致效率低下
- 克服以上缺点，基于时序差分的Q-learning

算法 7.6 Q 学习算法

函数: QLearning ()
输入: 马尔可夫决策过程 $MDP = (S, A, P, R, \gamma)$
输出: 策略 π
<pre> 1 随机初始化 q_s 2 repeat 3 $s \leftarrow$ 初始状态 4 repeat 5 $a \leftarrow \text{argmax}_a q_\pi(s, a)$ 6 执行动作 a，观察奖励 R 和下一个状态 s' 7 $q_\pi(s, a) \leftarrow q_\pi(s, a) + \alpha [R + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s, a)]$ 8 $s \leftarrow s'$ 9 until s 是终止状态 10 until q_s 收敛 11 $\pi(s) := \arg \max_a q(s, a)$ </pre>

- 直接记录动作价值函数，因此一个状态可以保存多个动作的价值

7.2.5 exploration & exploitation

- 为了更好地探索，加入贪心
 - 时序差分的Q-learning总是选择具有最大价值的动作，会存在探索不足的问题
 - 实际运用中，这个贪心率会衰减（前期探索已经探索出了最优策略，后面则没必要保持相同的探索热情）

•	<table border="1"><tr><td>函数: EpsGreedy ()</td></tr><tr><td>输入: 状态 s, 动作-价值函数 q_π, 参数 ϵ</td></tr><tr><td>输出: 动作 a</td></tr><tr><td>1 $n \sim \text{Uniform}(0, 1)$</td></tr><tr><td>2 if $n < \epsilon$ then</td></tr><tr><td>3 $a \leftarrow$ 从 A 中随机选择</td></tr><tr><td>4 else</td></tr><tr><td>5 $a \leftarrow \text{argmax}_{a'} q_\pi(s, a')$</td></tr><tr><td>6 end</td></tr><tr><td> </td></tr><tr><td>函数: QLearning ()</td></tr><tr><td>输入: 马尔可夫决策过程 $MDP = (S, A, P, R, \gamma)$</td></tr><tr><td>输出: 策略 π</td></tr><tr><td>1 随机初始化 q_π</td></tr><tr><td>2 repeat</td></tr><tr><td>3 $s \leftarrow$ 初始状态</td></tr><tr><td>4 repeat</td></tr><tr><td>5 $a \leftarrow \text{EpsGreedy}(s, q_\pi, \epsilon)$</td></tr><tr><td>6 执行动作 a, 观察奖励 R 和下一个状态 s'</td></tr><tr><td>7 $q_\pi(s, a) \leftarrow q_\pi(s, a) + \alpha [R + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s, a)]$</td></tr><tr><td>8 $s \leftarrow s'$</td></tr><tr><td>9 until s 是终止状态</td></tr><tr><td>10 until q_π 收敛</td></tr><tr><td>11 $\pi(s) := \text{argmax}_a q(s, a)$</td></tr></table>	函数: EpsGreedy ()	输入: 状态 s , 动作-价值函数 q_π , 参数 ϵ	输出: 动作 a	1 $n \sim \text{Uniform}(0, 1)$	2 if $n < \epsilon$ then	3 $a \leftarrow$ 从 A 中随机选择	4 else	5 $a \leftarrow \text{argmax}_{a'} q_\pi(s, a')$	6 end		函数: QLearning ()	输入: 马尔可夫决策过程 $MDP = (S, A, P, R, \gamma)$	输出: 策略 π	1 随机初始化 q_π	2 repeat	3 $s \leftarrow$ 初始状态	4 repeat	5 $a \leftarrow \text{EpsGreedy}(s, q_\pi, \epsilon)$	6 执行动作 a , 观察奖励 R 和下一个状态 s'	7 $q_\pi(s, a) \leftarrow q_\pi(s, a) + \alpha [R + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s, a)]$	8 $s \leftarrow s'$	9 until s 是终止状态	10 until q_π 收敛	11 $\pi(s) := \text{argmax}_a q(s, a)$
函数: EpsGreedy ()																									
输入: 状态 s , 动作-价值函数 q_π , 参数 ϵ																									
输出: 动作 a																									
1 $n \sim \text{Uniform}(0, 1)$																									
2 if $n < \epsilon$ then																									
3 $a \leftarrow$ 从 A 中随机选择																									
4 else																									
5 $a \leftarrow \text{argmax}_{a'} q_\pi(s, a')$																									
6 end																									
函数: QLearning ()																									
输入: 马尔可夫决策过程 $MDP = (S, A, P, R, \gamma)$																									
输出: 策略 π																									
1 随机初始化 q_π																									
2 repeat																									
3 $s \leftarrow$ 初始状态																									
4 repeat																									
5 $a \leftarrow \text{EpsGreedy}(s, q_\pi, \epsilon)$																									
6 执行动作 a , 观察奖励 R 和下一个状态 s'																									
7 $q_\pi(s, a) \leftarrow q_\pi(s, a) + \alpha [R + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s, a)]$																									
8 $s \leftarrow s'$																									
9 until s 是终止状态																									
10 until q_π 收敛																									
11 $\pi(s) := \text{argmax}_a q(s, a)$																									

7.2.6 parametrize & Deep RL

- 状态数量过多
 - 算法很难用一个数组来存储动作-价值函数的值
 - 有些状态的访问次数可能很少甚至根本没有被访问过，这些状态的价值估计是不可靠的
- 一种解决方案是将动作-价值函数参数化，即用一个回归模型来拟合 q_π 函数。如果回归模型是一个深度神经网络，那么这样的算法就被称为深度强化学习算法
 - 动作-价值函数的参数化可能导致算法不稳定或产生较大偏差
 - 参数用于估计两个动作价值函数值，优化目标不明确

算法 7.8 参数化的 Q 学习算法

函数: DeepQLearning ()
输入: 马尔可夫决策过程 $MDP = (S, A, P, R, \gamma)$
输出: 策略 π
1 随机初始化 q_π 的参数 θ
2 repeat
3 $s \leftarrow$ 初始状态
4 repeat
5 $a \leftarrow \text{EpsGreedy}(s, \theta, \epsilon)$
6 执行动作 a , 观察奖励 R 和下一个状态 s'
7 $L(\theta) \leftarrow \frac{1}{2} [R + \gamma \max_{a'} q_\pi(s', a'; \theta) - q_\pi(s, a; \theta)]^2$
8 $\theta \leftarrow \theta - \eta \frac{\partial L(\theta)}{\partial \theta}$
9 $s \leftarrow s'$
10 until s 是终止状态
11 until q_π 收敛
12 $\pi(s) := \text{argmax}_a q(s, a)$

- 改进: DQN

对于使用同样的参数来计算动作-价值函数的当前值和目标值的问题, DQN 使用目标网络的方法来应对。所谓目标网络, 即用另一组参数

- θ^- 来计算动作-价值函数的预测值。具体来说, 对损失函数进行了如下修改:

$$L(\theta) = \frac{1}{2} \left[R + \gamma \max_{a'} q_\pi(s', a'; \theta^-) - q_\pi(s, a; \theta) \right]^2$$

7.3 policy based

- 直接参数化策略函数
- REINFORCE方法
 - monte-carlo
- Actor-Critic算法
 - temporal difference