人工智能

搜索算法

对于搜索算法，评价指标为：完备性、最优性、时间复杂度和空间复杂度。

开表（open list）： 在树搜索算法中，集合F用来保存搜索树中可用于下一步探索的所有候选节点，这个集合被称为 **边缘（fringe）集合**，有时也称为开表。

深度优先搜索不具备完备性，在有环路的情况下，算法不会停止。

启发式搜索

启发式搜索指搜索过程中利用与所求解问题相关的**辅助信息**，其代表算法为贪婪最佳优先搜索，A*搜索。

辅助信息

- 评价函数evaluation function, f(n)。从当前节点出发，根据评价函数选择后续节点。
- 启发函数heuristic function, h(n)。计算从节点n到目标节点之间所形成路径的最小代价值。一般将两点之间的欧式距离作为启发函数。

贪婪最佳优先搜索

评价函数f(n)=启发函数h(n)，不保证最优性。

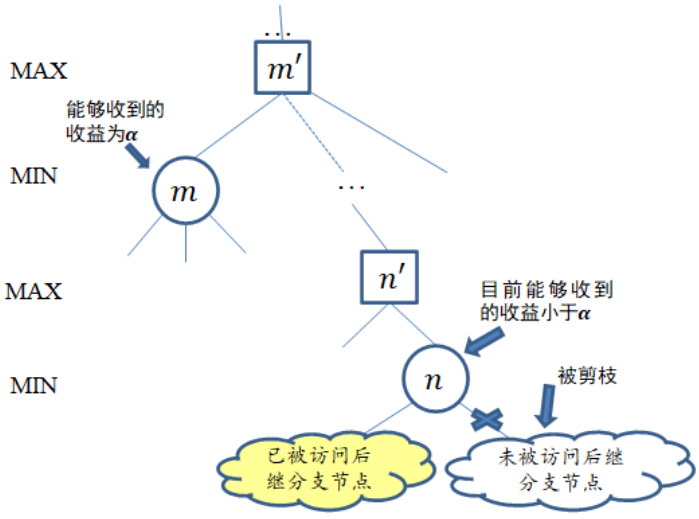
A*算法

评价函数f(n)=g(n)+h(n)，g(n)表示从起始点到节点n的开销。

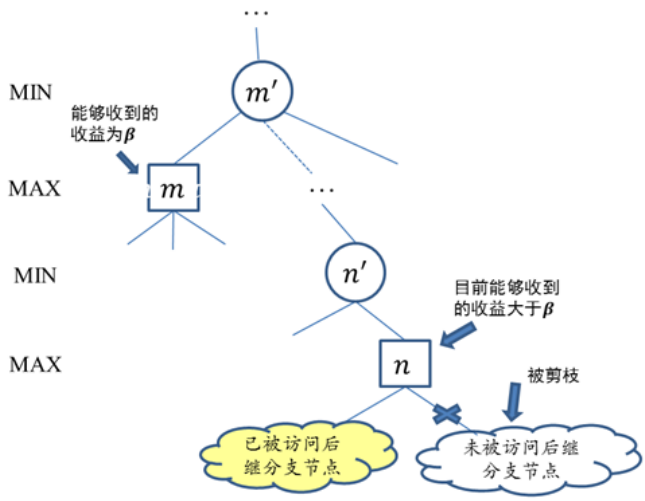
对抗搜索

α-β剪枝

α-β剪枝搜索是基础MAX-MIN搜索的剪枝策略。



假设一个位于MIN层的节点m，已知其可以收到的收益为α。其兄弟节点的后代节点n的后代节点被访问一部分后，知道节点n能够向上一层MAX节点反馈收益小于α，则节点n的后续后代节点不需要继续扩展。



反之，对于MAX层节点m可以接收到收益为β，其兄弟节点的后代节点n的扩展节点可以获得的收益大于β，则其后续子节点不需要继续扩展。

αβ值更新： 对于MAX节点，如果其孩子节点MIN的收益大于当前的α值，则将α值更新为该收益；对于MIN节点，如果其孩子节点MAX的收益小于当前的β值，则将β值更新为该收益。根节点MAX的α和β值分别初始化为-∞和+∞。

随着搜索算法不断执行，每个节点的α值和β值不断更新，大体来说，α值逐渐增加，β值逐渐减小。如果一个节点的α>β值，则该节点尚未被访问的后续节点就会被剪枝，因而不会被智能体访问。

蒙特卡洛树搜索

AlphaGo Zero的基础思想，基本思路是，对于当前的一种状态，做以下几步：

1. 向前思考一步，即自己行动，对于每个当前可选的行动进行评估，选取最优的行动（如果有行动没有采取过，那么这个行动最优）。
2. 如果选取的行动从来没有采取过，那么采取后进行随机行动直到结束，然后根据结果调整从初始状态到这次行动的一系列树点。
3. 再向前思考一步，选取1步中最优行动的下一行动，（围棋中是对手行动），此时重复1步骤。（如果是博弈，那么此时最优的行动就是收益最小的行动）。
4. 重复k次步骤2，即向前思考k步。然后回到1的情况，选取当前情况下最优的一步。

对于代码而言，树的结点保存四个信息：

1. 当前行动（对于棋类，就是落子的位置）。
2. 收益。
3. 这个状态经历的总次数。
4. 孩子节点。

第一次遍历时，root就是当前状态，它的孩子就是目前可以落子的所有位置。然后一开始所有的点的收益和经历次数都是0。蒙特卡洛树上长出来的所有点都是未来的预测状态，所以都是0/0.

> 误区一：对于当前的一个状态，这个状态可能出现在之前的状态的蒙特卡洛树上，但是这两者没有关系。按这个状态为根节点，重新进行蒙特卡洛树搜索过程。

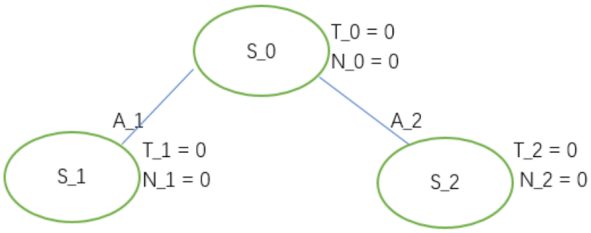
对于当前可选择行动的评估，一般使用UCB1算法，即

$$UCB1(S_i) = V_i + c\sqrt{\frac{log(N)}{n_i}}, c = 2.$$

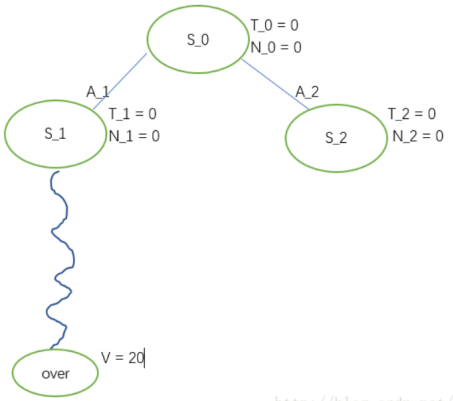
N就是收益，n就是经历总次数。

> 例：

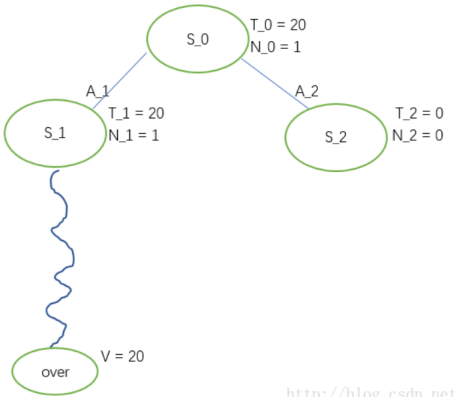
初始状态：



向前看1步：

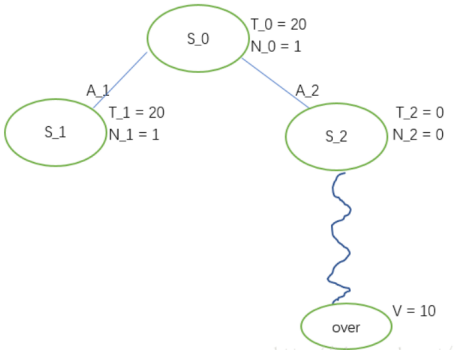


然后进行反向传播修改经历的状态，用来表示选择这一个行为带来的收益影响：

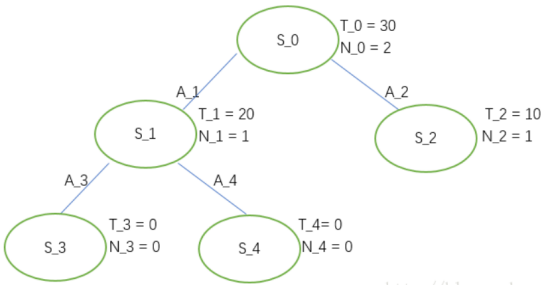


然后删除随机探索的这一部分。

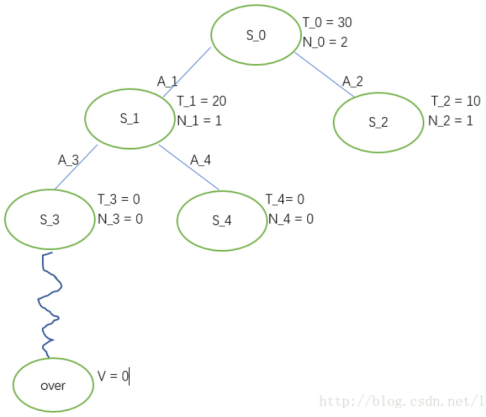
进行第二次迭代：



进行第三次迭代：

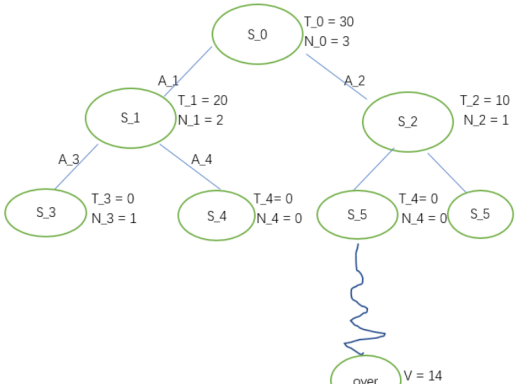


然后进行探索：



然后反向传播影响。

然后进行下一次迭代：



然后反向传播影响。

当迭代停止，计算S_1和S_2的UCB1值来判断选取哪个最优。

> 其实蒙特卡洛树搜索就是不断向前看n步，然后再具体判断对于当前情况，哪一种行动最优。这比直接贪心选择当前最优要强，但是问题是，这棵树的宽度会非常大，即有时候对于当前情况的选择太多的。受到计算资源的限制，我们需要牺牲树的高度，即我们总是不能看到太远的情况。

机器学习

机器学习的基础目标为，预测值与正确值（标签）的差距最小化，分为以下三种：

- 经验风险最小化：$\min \sum Loss(y_i, f(x_i))$.
- 期望风险最小化：$\min \int Loss(y, f(x))P(x, y)dxdy$.
- 结构风险最小化：为了防止过拟合，在以上两种最小化中加入模型复杂度的正则化项或者惩罚项(penalty term)：$\min \sum Loss(y_i, f(x_i)) + \lambda w^T w$.

监督学习两种方法：

- 判别模型。直接学习判别函数f(x)或者条件概率分布P(Y|X)作为预测的模型。
- 生成模型。从数据中学习联合概率分布P(X,Y)，通过似然概率P(X|Y)和P(Y)的乘积来求。

损失函数：

- 均方误差损失函数

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

- 交叉熵损失函数

  假定p和q是数据x的两个概率分布，通过q来表示p的交叉熵可如下计算：

$$H(p,q) = -\sum_x p(x) * log(q(x))$$

  交叉熵越小，两个概率分布p和q越接近。

  假设y为数据x的真实分布，$\hat{y}$表示模型预测的分布，则对于数据x，交叉熵损失函数定义为：

  cross entropy = $-y * log(\hat{y})$.

  如果将softmax和交叉熵损失函数相结合，会对偏导的计算带来极大便利。假设所预测的值经过softmax层后为(0.34,0.46,0.20)，如果选择交叉熵损失函数来优化模型，则这一层的偏导值为(0.34-1,0.46,0.20)=(-0.66,0.46,0.20).

梯度下降

梯度下降计算所有样本的误差，使得损失函数最小化。

随机梯度下降： 由于梯度下降法需要计算所有样本，消耗大，随机梯度下降旨在利用一份或者少样本计算误差来最小化损失函数。这样计算会很快。

决策树

- **信息熵entropy**

  假设样本集合D有K个样本，发生的概率分别为p_k，则K个信息的信息熵为：
  $E(D) = -\sum_{k=1}^{K} p_k log_2 p_k$。

  信息熵越小，说明D包含的信息越确定，即D的纯度越高。

决策树的思想就是，随着树不断分支，该分支下的样本信息熵越来越小，最终包含相同类别。

那么决策树具体按照什么标志来决定接下来划分什么属性呢？信息增益。

信息增益通过信息熵来计算，$Gain(D,A) = E(D) - \sum_{i=1}^{n}\frac{|D_i|}{|D|}E(D_i)$得到各个属性的信息熵，增益大的先进行分支。

Boosting

思想：将若干弱分类器组合形成一个强分类器。

- 强可学习模型：模型能够以较高精度对绝大多数样本完成识别分类。
- 弱可学习模型：模型仅能对若干部分样本完成识别与分类，精度略高于随机。

这两种学习模型是等价了，如果存在一个弱可学习模型，可以将其提升(Boosting)为一个强可学习模型。

Ada Boosting算法训练一系列弱分类器，将这些弱分类器线性组合得到强分类器。其中，对某数据的识别分类有这些弱分类器进行线性投票组合，结果服从多数弱分类器的分类结果。

如果某个样本无法被第m个弱分类器分类成功，则需要增大该样本的权重，否则减少该样本的权重。这样，被错误分类的样本会在训练第m+1个弱分类器时重点关注。

即在每一轮学习过程中，Ada Boosting算法均在 **重视当前尚未被正确分类的样本。**

降维

- **LDA线性判别分析**

  Fisher提出。LDA利用类别信息，将一组高维数据投影到一个低维空间上，在低维空间中使得同一类别样本尽可能靠近，不同类别样本尽可能远离。即"类内方差小，类间间隔大"。

  LDA降维步骤如下：

  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$S_w^{-1}S_bW = \lambda W$来求解前r个最大特征值所对应的特征向量，构成矩阵W。
  4. 通过W将原样本映射到r维。

- **PCA主成分分析**

  主成分分析的思想是将n维特征数据映射到m维空间，去除原始数据之间的一些冗余性。

  我们对原样本求协方差矩阵，然后求其特征值，求解对应的特征向量，构成矩阵W。

	线性判别分析	主成分分析
是否需要样本标签	监督学习	无监督学习
降维方法	优化寻找特征向量$\boldsymbol{w}$	优化寻找特征向量$\boldsymbol{w}$
目标	类内方差小、类间距离大	寻找投影后数据之间方差最大的投影方向
维度	LDA降维后所得到维度是与数据样本的类别个数K有关	PCA对高维数据降维后的维数是与原始数据特征维度相关

- **NMF非负矩阵分解**

  该方法将非负大矩阵分解成两个非负小矩阵。

  主成分分析方法可以实现这一点，不过主成分分析不要求原矩阵非负。到那时在很多实际情况中如图像像素和单词-文档矩阵中自然不存在为负数的元素，因此，对非负矩阵的分解很有用。

- **MDS多维尺度法**

  MDS保持原始数据之间两两距离不变。MDS计算原始数据两两之间的距离，形成一个距离矩阵。不过MDS需要知道这样的距离矩阵，所以无法对新数据集直接进行降维，这被称为"out-of-sample"问题。

- **LLE局部线性嵌入**

  PCAMDS都属于线性降维方法，LLE是一种非线性降维方法。LLE的基本假设是：一个流形的局部可以近似于一个欧氏空间，每个样本均可以利用其邻居进行线性重构。即假设数据是局部线性的（即使数据的原始高维空间是非线性流形嵌入）。LLE使用局部线性来逼近全局非线性。

模型参数估计

- 最大似然估计

  假设n个数据样本从参数为θ的某个模型中以一定概率独立采样得到，那么最大似然估计就是求θ使得这个参数得到的模型出现这些样本的概率最大。$\hat{\Theta} = argmax_{\Theta}P(D|\Theta)$.

- 最大后验估计

  $\hat{\Theta} = argmax_{\Theta}P(\Theta|D)$，对其取对数，得到$argmax(logP(D|\Theta)) + logP(\Theta)$，可见，最大后验估计与最大似然估计相比，多了一项先验概率$P(\Theta)$。

- 期望最大化EM

EM算法是一种重要的用于解决含有隐变量（latent variable）问题的参数估计方法。分为求取期望E步骤和期望最大化M步骤。

1. E步：先假设模型参数初始值，估计隐变量取值。
2. M步：基于假设的数值，最大化“拟合”样本数据，更新模型参数。
3. 不断迭代M步骤，直到更新参数收敛。

深度学习

卷积神经网络

循环神经网络

- RNN

  在每一时刻t，循环神经网络单元会读取当前输入数据x_t和前一时刻输入数据对应的隐式编码结果h_{t-1}，一起生成t时刻的隐式编码结果h_t。

  按照时间将循环神经网络展开后，可以得到一个和前馈神经网络类似的网络结构。这个网络可以利用反向传播来优化参数。称为“沿时间反向传播（BPTT）”。

  由于循环神经网络每个时刻都有一个输出，所以在计算循环神经网络的损失时，通常需要将所有时刻上的损失累加。

$$h_t = \Phi(\mathrm{U} \times x_t + \mathrm{W} \times h_{t-1}) = \Phi\big(\mathrm{U} \times x_t + \mathrm{W} \times \Phi(\mathrm{U} \times x_{t-1} + \mathrm{W} \times h_{t-2})\big)$$

$$= \Phi\left(\mathrm{U} \times \underbrace{x_t}_{t\text{时刻输入}} + \mathrm{W} \times \Phi\left(\mathrm{U} \times \underbrace{x_{t-1}}_{t-1\text{时刻输入}} + \mathrm{W} \times \Phi(\mathrm{U} \times \underbrace{x_{t-2}}_{t-2\text{时刻输入}} + \cdots)\right)\right)$$

  求偏导时，需要将前面时刻的W依次求导，然后再将求导结果进行累加：

$$\frac{\partial E_t}{\partial W_x} = \sum_{i=1}^{t} \frac{\partial E_t}{\partial O_t} \frac{\partial O_t}{\partial h_t} \Big(\prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}\Big) \frac{\partial h_i}{\partial W_x}$$

$$\text{其中} \prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=i+1}^{t} tanh' \times W_h$$

  令$t=3$，则有：

$$\frac{\partial E_3}{\partial W_x} = \frac{\partial E_3}{\partial O_3}\frac{\partial O_3}{\partial h_3}\frac{\partial h_3}{\partial W_x} + \frac{\partial E_3}{\partial O_3}\frac{\partial O_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_x} + \frac{\partial E_3}{\partial O_3}\frac{\partial O_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_x}$$

  对于长序列，这样的累积会使得参数求导结果很小，引发梯度消失问题。

- LSTM

  为了解决传统RNN的梯度消失，提出了LSTM长短时记忆模型。它引入了 **内部记忆单元** 和 **门** 两种结构。这里，内部记忆单元可视为“历史信息”的累积。而门则一般有 **输入门input gate，遗忘门forget gate，输出门output gate。**

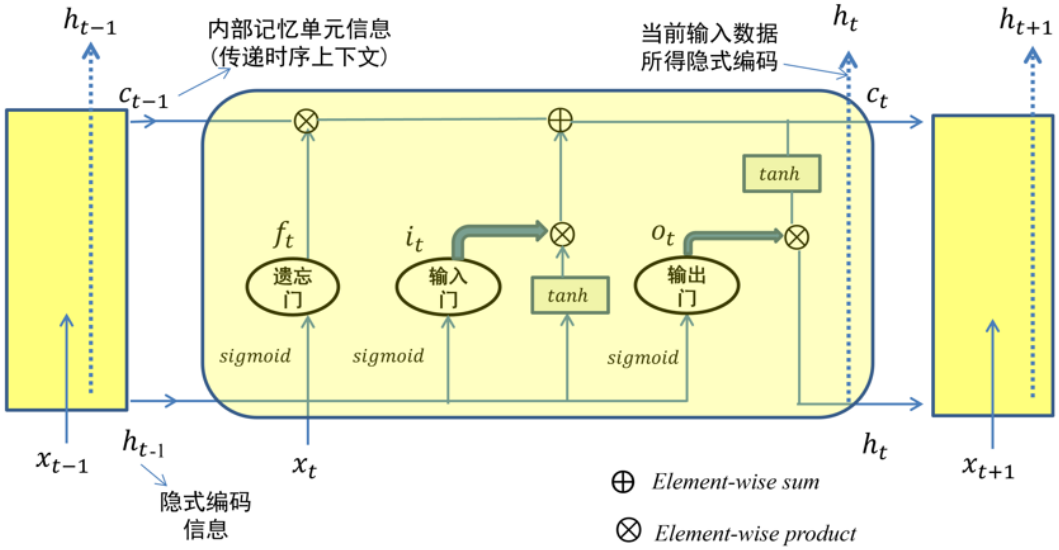
  输入门信息输出：$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$.

  遗忘门信息输出：$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$.

  输出门信息输出：$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$.

  内部记忆单元信息输出：$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$.

  隐式编码输出：$h_t = o_t \odot tanh(c_t)$.



对于内部记忆单元，存在加法，遗忘门的求导结果至少为f_t，如果遗忘门选择保留就状态，则导数等于1，使得梯度不为0，避免了梯度消失。

从整体来看，内部记忆单元c类似长时记忆，而隐式编码h类似短时记忆。

强化学习

- 马尔可夫奖励过程

  $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$。

  γ为折扣系数，范围[0,1]。即认为，越是遥远的未来的奖励对现在的反馈贡献越小。

  对于R，一般而言，如果此时到达终点，则获取回报，否则为0。

- 策略学习

  策略函数表示处于某个状态采取某种行动以获得最大回报值。

  - **价值函数（value function）**：$V_\pi(s) = E_\pi[G_t|S_t = s]$，即在第t步状态为s时，按照策略$\pi$行动后在未来所获得的回报的期望。
  - **动作-价值函数（action-value function）**：$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a]$表示在第t步状态为s时，按照策略$\pi$采取动作a后在未来获得的回报值。

  这样，策略学习转换为一个优化问题：寻找最优策略π^*,使得对任意状态，$V_{\pi^*}(S)$最大。

- **贝尔曼方程**

  也称动态规划方程。

  $V_\pi(s) = \sum_a \pi(s,a) q_\pi(s,a)$，即在状态s采取动作a的概率×采取动作a后带来的回报。

  $q_\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_\pi(s')]$，即在状态s采取动作a的概率×（采取a进入s'得到的回报+处于s'可以得到的回报）。

策略评估方法

- 动态规划

  我们将贝尔曼方程中的V的算式中的q替换为其算式，就得到了动态规划方法：

初始化V_π函数
循环
　　枚举$s \in S$
　　$V_\pi(s) \leftarrow \sum_{a \in A} \pi(s,a) \sum_{s' \in S} Pr(s'|s,a)\,[R(s,a,s') + \gamma V_\pi(s')]$　　（$q_\pi(s,a)$）
直到V_π收敛

缺点是需要提前知道状态转移概率。同时无法处理状态集合无限大的情况，比如状态连续。

- 蒙特卡洛采样

选择不同的起始状态，按照当前策略 π 采样若干轨迹，记它们的集合为 D
枚举 $s \in S$
    计算 D 中 s 每次出现时对应的反馈 $G_1, G_2, \cdots, G_k$
    $V_\pi(s) \leftarrow \frac{1}{k}\sum_{i=1}^{k} G_i$

缺点是：状态集合比较大时，状态可能非常稀疏，不利于估计期望，同时获取回报可能需要的周期很长。

- 时序差分

初始化 V_π 函数
循环
    初始化s为初始状态
    循环
        $a \sim \pi(s,\cdot)$
        执行动作 a，观察奖励 R 和下一个状态 s'
        更新 $V_\pi(s) \leftarrow V_\pi(s) + \alpha[R(s,a,s') + \gamma V_\pi(s') - V_\pi(s)]$
        $s \leftarrow s'$
    直到 s 是终止状态
直到 V_π 收敛

其更新V值的方式为：$V_\pi(s) \leftarrow (1-\alpha)V_\pi(s) + \alpha[R(s,a,s') + \gamma V_\pi(s')]$，表示原来的价值函数与学习得到的价值函数值共同更新价值函数。

Q-learning

初始化 q_π 函数
循环
    初始化s为初始状态
    循环
        $a = \text{argmax}_{a'} q_\pi(s,a')$
        执行动作 a，观察奖励 R 和下一个状态 s'
        更新 $q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s',a') - q_\pi(s,a)\right]$
        $s \leftarrow s'$
    直到 s 是终止状态
直到 q_π 收敛

s_d	s_7	s_8	s_9
	s_4	s_5	s_6
	s_1	s_2	s_3

设定，从s1到s9，如果下一状态为s9，则回报R为1，若为sd，则回报R为-1，其余为0。

首先要初始化q函数，我们用a/b表示q(s,上)=a,q(s,右)=b。

为了防止初始值设置不合理导致不断重复执行同一策略没有提升，我们使用e贪心策略，每次以概率e选取随机动作。

Deep Q-learning

Q-learning由于使用Q-table无法处理状态连续或无穷多的情况，所以我们使用神经网络来代替Q-table做策略。

初始化q_π函数的参数θ
循环
    初始化s为初始状态
    循环
        采样$a \sim \epsilon\text{-}greedy_\pi(s;\theta)$
        执行动作a，观察奖励R和下一个状态s'
        损失函数$\mathbf{L}(\theta) = \frac{1}{2}\left[R + \gamma \max_{a'} q_\pi(s', a'; \theta) - q_\pi(s, a; \theta)\right]^2$
        根据梯度$\partial L(\theta)/\partial\theta$更新参数$\theta$
        $s \leftarrow s'$
    直到s是终止状态
直到q_π收敛