Notes: 人工智能导论

黄宇翔
清华大学计算机系

2023 年 6 月 17 日

1 搜索问题

1.1 DFS, BFS

DFS：优先扩展子结点（深度深的结点），可以使用可变深度限制（试探不同的深度），相较于其他方法节省内存（只需要储存从初始结点到当前结点的路径）

BFS：优先扩展兄弟结点

1.2 启发式图搜索

引入启发知识，保证找到最优解的情况下减少搜索范围。

启发知识：评估结点到达目标的距离。

定义 1.1 (评价函数)**.** *$f(n) = g(n) + h(n)$，$f(n)$ 是评价函数，$h(n)$ 启发函数。*

定义 1.2. *$g^*(n)$：从 s 到 n 的最短路径耗散值*

$h^*(n)$：从 n 到 g 的最短路径耗散值

$f^*(n) = g^*(n) + h^*(n)$：从 s 经过 n 到 g 的最短路径耗散值

$g(n), h(n), f(n)$ 是 $g^*(n), h^*(n), f^*(n)$ 的估计值

其中 $g(n)$ 可以直接用源点到当前结点的路径长度，只需要估计 $h(n)$ 即可。用 $f(n)$ 对待扩展结点进行评价，选择最小的 $f(n)$ 结点扩展。

算法 1.1 (A 算法)**.** $OPEN = (s), CLOSED = (), f(s) = g(s) + h(s)$

while $OPEN \neq \emptyset$ *do:*

begin

  $n = FIRST(OPEN)$

  if $GOAL(n)$ *return* n

  $REMOVE(n, OPEN), ADD(n, CLOSED)$

$EXPAND(n) \rightarrow \{m_i\}$ // 生成子结点 m_i

计算 $f(n,m_i) = g(n,m_i) + h(m_i)$

$ADD(m_j, OPEN)$, 标记 $m_j \rightarrow n$ 指针

if $f(n,m_k) < f(m_k)$: 标记 $m_k \rightarrow n$ 指针

if $f(n,m_l) < f(m_l)$: 标记 $m_k \rightarrow n$ 指针，将 m_l 从 *CLOSED* 移除加到 *OPEN* 中

end

return FAIL

Closed 表：已经被扩展过的结点集合。Open 表：未被扩展的结点集合。

在扩展结点 n 的时候，存在三种与 n 相连的结点。m_j : 由 n 产生的结点。m_k : 由之前扩展的结点生成出，但还未被扩展。m_l : 由之前扩展的结点生成出，已经被扩展。

m_j：其父结点是 n，只计算了一次 $g(m_j)$。m_k : 之前已经计算了 $g(m_k)$，需要计算当前由 n 到 m_k 是否能引入更小的 $g(m_k)$，如果可以，需要更新 $g(m_k)$。m_l: 需要更新它和它的孩子的 $g()$。

针对三种子结点的处理策略：第一次被生成的 m_j 直接加入到 OPEN 中；已经被生成但未被扩展的结点 m_k 已经在 OPEN 中了，只需要更新 $f(n)$ 和父亲指针；已经被扩展的 m_l 如果发现更短路径，则需要重新被考虑，因此把状态重新纳入 OPEN 中。

解路径的获得：从目标结点沿着父亲指针一路到源点。

算法 1.2 (A* 算法). *在 A 算法中，如果满足如下条件，则 A 算法称为 A^* 算法。*

$$h(n) \leq h^*(n)$$

实质：真实的最短路径不能短于估计的最短路径长度。

定理 1.1 (可采纳性定理). *若存在从初始结点 s 到目标节点 t 的路径，则 A^* 算法一定可以找到最优路径。*

证明：考虑目标节点 t 的两条解路径 A_1, A_2，假设 $|A_2| > |A_1|$。若路径 A_2 被当做了最优路径，算法的最后一步应当是从 OPEN 表中删除结点 t。

此时，由于 A_1 没有被当做最优路径，则一定存在 $n \in A_1, n \neq t, n \in$OPEN. 考察此时的权重 $f(\cdot)$，有

$$f(n) = g(n) + h(n) \leq g(n) + h^*(n) = f(t)$$

则算法最后一步会从 OPEN 表中删除 n 而并非 t，矛盾。因此，若 $h(n) \leq h^*(n), \forall n \in G$，A* 算法一定可以找到最优路径。

定理 1.2. *对于同一问题，定义两个 A^* 算法 A_1^*, A_2^*，其中启发函数分别为 $h_1(n), h_2(n)$，若除了目标节点均有*

$$h_2(n) > h_1(n)$$

则由 A_2 扩展的每个结点均由 A_1 扩展，即 A_1 扩展的结点数至少和 A_2 一样多。

解释：在 A* 算法中，$h(n) \le h^*(n)$ 恒成立，更大的 $h(n)$ 与 $h^*(n)$ 更接近，扩展的结点更少。

证明：先证明一个引理：若 $f(n) < f(t)$，则 n 一定会被扩展。

不妨设解路径长度有限，也就是 $|A| < +\infty$，由可采纳性定理可知，算法能在有限步停止，因此对步数归纳。

算法第 0 步，将 s 扩展，s 满足 $f(s) = g(s) + h(s) < 0 + h^*(s) = f(t)$。

归纳。若第 $k-1$ 步可达的所有 $f(n) < f(t)$ 的结点都被扩展，则对于第 k 步可达的所有这样的结点，其父结点一定被扩展过，也就是这样的结点 n 一定进入 OPEN 表。

下面采用反证法。如果 n 仅进入 OPEN 表而没有被扩展，则 n 将会在 OPEN 表中遗留直到算法结束。考察算法的最后一步，此时 t, n 同属 OPEN 表，且 $f(n) < f(t)$，则算法应当扩展 n 而不是 t，矛盾。因此，若所以第 k 步可达的所有这样的结点都会被扩展。

从引理可以看出，对于 $h^*(n) > h_2(n) > h_1(n)$，总有 $f_1(n) < f_2(n)$，所以算法 A_2^* 扩展的结点必然被 A_1^* 扩展，但反之不成立，因此 $|A_1| \ge |A_2|$.

注意：扩展的结点数 $\ne$ 扩展次数，多次被记录在同一个结点上。

对 h 的评价方法：平均分叉数

定义 1.3 (平均分叉数)**.** *设共扩展了 d 层结点，共搜索了 N 个结点，则*

$$N = \frac{1 - b^{*(d+1)}}{1 - b^*}$$

其中 b^* 称为平均分叉数

b^* 越小越好，是稳定常数。实质：将搜索树视作 b^* 叉树。

合理的 h 函数应当满足的条件：设 n_i 是 n_j 的父节点，满足 $h^*(n_j) - h(n_j) \le h^*(n_i) - h(n_i)$，越近的结点误差越小。此时如果有 n_j 在 $n_i \to t$ 的最短路径上，那么 $h(n_i) - h(n_j) \le h^*(n_i) - h^*(n_j) = C(n_i, n_j)$。当不在最短路径上，已经有不等关系 $h^*(n_i) < C(n_i, n_j) + h^*(n_j)$，仍然有 $h(n_i) - h(n_j) \le h^*(n_i) - h^*(n_j) < C(n_i, n_j)$。

定义 1.4 (单调性)**.** *若 h 满足*

$$h(n_i) - h(n_j) \le C(n_i, n_j), \forall n_i, n_j; h(t) = 0$$

其中 t 是目标结点，C 是最短路径长度，则称 h 是单调的。

定理 1.3. *若 h 是单调的，则 A^* 扩展了结点 n 后，就已经找到了到达结点 n 的最佳路径，即当 A^* 选 n 扩展时，总有*

$$g(n) = g^*(n).$$

推论 1.1. *若 $h(n)$ 满足单调条件，则一定满足 A^* 条件。*

证明：从目标结点向上归纳，略。

例子 1.1. 在八数码问题中，若 h 为不在位的将排数，则满足单调性条件。

引理 1.1. *OPEN* 表上任一具有 $f(n) < f^*(n)$ 的结点一定会被扩展。

引理 1.2. $A*$ 选作扩展的任一结点 n，一定有 $f(n) \le f^*(n)$。

引理 1.3. 当 $h(n) = 0$ 时，也满足单调性条件，此时算法退化成 *BFS*。

算法 1.3 (改进的 A* 算法)**.** $OPEN = (s), CLOSED = (), f(s) = g(s) + h(s), f_m = 0$

while $OPEN \neq \emptyset$ *do:*

begin

$NEST = \{n_i : f(n_i) < f_m, n_i \in OPEN\}$

if $NEST \neq \emptyset$*:* $n = NEST$ 中 g 最小的结点

else: $n = FIRST(OPEN), f_m = f(n)$

if $GOAL(n)$ *return* n

$REMOVE(n, OPEN), ADD(n, CLOSED)$

$EXPAND(n) \to \{m_i\}$ // 生成子结点 m_i

计算 $f(n, m_i) = g(n, m_i) + h(m_i)$

$ADD(m_j, OPEN)$, 标记 $m_j \to n$ 指针

if $f(n, m_k) < f(m_k)$*:* 标记 $m_k \to n$ 指针

if $f(n, m_l) < f(m_l)$*:* 标记 $m_k \to n$ 指针，将 m_l 从 *CLOSED* 移除加到 *OPEN* 中

end

return FAIL

核心思想：不扩展比上一轮中 f 值更大的结点，因为一定不会导致更优解。

对单调性导致结点单次扩展的证明：

单调性的实质：当 $h(n)$ 满足单调性时，将图上任意结点作为目标结点搜索，都满足 $h(n_i) - h(n) \le c(n_i, n)$，也就是都满足可采纳性条件。因此搜索到任意结点 n 的一条路径 A_n 都是从 s 到 n 的最优路径。

对任意结点 n，将其作为目标节点，由可采纳性证明得知，s 到 n 的最优路径优先于任何其他路径被发现。由 f 单调递增知，其他路径不会被扩展。因此单调性可以解决多次扩展的问题。

1.3 Viterbi 算法

将图划分阶段，在每个阶段中选择最优的一步。源点：w_0，走一步：$w_{11}, w_{12}, \dots$，走 n 步：$w_{n1}, w_{n2}, \dots$，目标结点是 w_{n+1}.

1.4 汉字识别后处理

O : 扫描之后的图像，S : 句子 S.

$$P(S|O) = \frac{P(O|S)P(S)}{P(O)}$$

问题：求解 $\hat{S} = argmax_S P(S|O) = argmax_S P(O|S)P(S)$.（因为 $P(O)$ 是常数）

其中 $P(S) = \prod_{i=1}^{n} P(w_i|w_1 \dots w_{i-1})$.（前 $i-1$ 个字出现时，第 i 个字出现的概率之积），$P(w_i|w_1 \dots w_{i-1})$ 需要依赖在语料库上进行贝叶斯估计或极大似然估计。

二元语法时，$P(S) = \prod_{i=1}^{n} P(w_i|w_{i-1})$

在图像识别中，$P(O|S)$ 可以用图像识别信度 $\prod_{i=1}^{n} CF(w_i)$ 代替。

问题变成求解

$$S = (w_1 \dots w_n) = argmax_{(w_1 \dots w_n)} \prod_{i=1}^{n} P(w_i|w_{i-1})CF(w_i)$$

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}w_i)}{count(w_{i-1})}$$

注意：不希望任何一项 $P(w_i|w_{i-1})$ 是 0，不然连乘会得到 0. 可以采用平滑的方法：

$$\lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i) \to P(w_i|w_{i-1})$$

在拼音输入法不考虑模糊音和多音字的情况下，不妨认为 $P(O|S) \approx 1$(满足拼音与汉字对应的约束条件). 此时对于一句满足约束条件的话，有

$$\hat{S} = argmax_S P(S|O) = argmax_S P(S)$$

$$= argmax_S \prod_{i=1}^{n} P(w_i|w_{i-1}) = argmin_S (-\sum_{i=1}^{n} \log P(w_i|w_{i-1}))$$

求解最小：视作求图上最短路径，使用 viterbi 算法。

2 神经网络与深度学习

2.1 神经元与非线性函数

定义 2.1 (感知器)**.**

$$net = w_1x_1 + \cdots + w_nx_n + b$$

$$y = sigmoid(net)$$

常用的激活函数

sigmoid 函数

$$\sigma(net) = \frac{1}{1+e^{-net}}$$

sgn 函数

$$sgn(net) = I_{net\geq 0}$$

双曲正切函数

$$\tanh(net) = \frac{e^{net}-e^{-net}}{e^{net}+e^{-net}}$$

ReLU 函数

$$ReLU(net) = \max(0, net)$$

Softmax 函数：用来做概率归一化，常用于分类问题的输出

$$\texttt{softmax}_k(net) = \frac{e^{net_k}}{\sum_{i=1}^n e^{net_i}}$$

2.2 学习过程

2.2.1 梯度下降法

$$w_i^{new} = w_i^{old} + \Delta w_i$$

其中 $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$

数据分批处理：随机梯度下降、小批量梯度下降、批量梯度下降

2.2.2 梯度计算

设神经网络是全连接神经网络，输出 $o \in \mathbb{R}^m$，ground truth $t \in \mathbb{R}^m$。设损失函数为

$$L = \frac{1}{2}\sum_{i=1}^m (t_i - o_i)^2$$

1. 输出层

设输出层的非线性函数为 `sigmoid` 函数 $\sigma(\cdot)$，首先先研究其导数性质。设 $y = \sigma(x) = 1/(1+e^{-x})$，则有

$$\frac{\mathrm{d}\sigma}{\mathrm{d}x} = \frac{-e^x}{(1+e^{-x})^2} = y(1-y)$$

记号如下。w_{ji}：最后一个隐藏层的第 i 个神经元到输出层第 j 个神经元的权重。x_{ji}：最后一个隐藏层第 i 个神经元的输出。net_j：输出层第 j 个神经元过非线性函数前的数；o_j：输出层第 j 个神经元过非线性函数后的输出。也就是

$$net_j = \sum_{k=1}^n w_{jk}x_{jk}$$

那么权重 w_{ji} 的梯度 $\partial E_d/\partial w_{ji}$ 为

$$
\begin{aligned}
\frac{\partial E_d}{\partial w_{ji}} &= \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} \\
&= \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}\cdot x_{ji}
\end{aligned}
$$

其中的

$$
\frac{\partial E_d}{\partial o_j} = -(t_j - o_j),\ \frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma}{\partial net_j} = o_j(1-o_j)
$$

最终，得到梯度

$$
\frac{\partial E_d}{\partial w_{ji}} = -(t_j - o_j)o_j(1-o_j)\cdot x_{ji} =: -\delta_j x_{ji}
$$

其中 $\delta_j = (t_j - o_j)o_j(1-o_j)$.

2. 隐藏层

记号如下。w_{ji}：上一层的第 i 个神经元到当前层第 j 个神经元的权重。x_{ji}：上一层第 i 个神经元的输出。设 $\delta'_k = \partial E_d/\partial net'_k$，其中 net'_k 是下一层第 k 个神经元非线性函数前的数。

那么权重 w_{ji} 的梯度 $\partial E_d/\partial w_{ji}$ 为

$$
\begin{aligned}
\frac{\partial E_d}{\partial w_{ji}} &= \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} \\
&= \sum_{k=1}^{n}\left(\frac{\partial E_d}{\partial net'_k}\frac{\partial net'_k}{\partial net_j}\right)\cdot x_{ji}
\end{aligned}
$$

其中的

$$
\frac{\partial E_d}{\partial net'_k} = -\delta'_k,\ \frac{\partial net'_k}{\partial net_j} = \frac{\partial net'_k}{\partial o_j}\cdot o_j(1-o_j) = w_{kj}o_j(1-o_j)
$$

最终，得到梯度

$$
\frac{\partial E_d}{\partial w_{ji}} = -o_j(1-o_j)\sum_{k=1}^{n}\delta'_k w_{kj}\cdot x_{ji} =: -\delta_j x_{ji}
$$

其中 $\delta_j = o_j(1-o_j)\sum_{k=1}^{n}\delta'_k w'_{kj}$.

2.2.3 反向传播算法

仍然以上述神经网络作为背景讨论。

为了区分不同层的神经元，设 x_i^k 表示第 k 层的第 i 个神经元输出，$k = 1, 2, \ldots, N$。特别地，x_i^0 表示输入的第 i 维，x_i^N 表示输出的第 i 维。第 k 层的 δ_i 记作 δ_i^k。第 k 层参数（从第 $k-1$ 层神经元映射到 k 层）中，第 i 个神经元映射到第 j 个神经元的权重为 w_{ji}^k.

1. 前向传播。在计算的过程中，每层结点记录下自己的 x_i^k
2. 计算输出层的 $\delta_j^N = (t_j - o_j)o_j(1 - o_j)$，以及偏导数 $\partial E_d / \partial w_{ji}^N = -\delta_k^N x_i^{N-1}$
3. 从后向前，每层按照如下方式计算，设当前层数为 k

$$\begin{aligned}
\delta_j^k &= x_j^k(1 - x_j^k) \sum_{t=1}^{n} \delta_t^{k+1} w_{tj}^{k+1} \\
\frac{\partial E_d}{\partial w_{ji}^k} &= -\delta_j^k x_i^{k-1}
\end{aligned}$$

最终，可以计算得到所有参数的梯度，然后更新梯度即可。

2.3 损失函数

1. 误差平方和：用于希望输出与给定值接近的情形，例如蒸馏

$$L_x = \frac{1}{2} \sum_{i=1}^{m} (t_i - o_i)^2, L = \frac{1}{n} \sum_{j=1}^{n} L_{x_j}$$

2. 交叉熵：用于分类问题，输出是概率的情形

$$L_x = -\sum_{i=1}^{m} t_i \log(o_i), L = \frac{1}{n} \sum_{j=1}^{n} L_{x_j}$$

2.4 卷积神经网络

卷积前后的尺寸计算：设原图片为 $n \times m \times k$ 的大小（共有 k 个通道），则卷积核的大小应当为 $a \times b \times k$。如果有 t 个卷积核，则卷积后的图片尺寸应当是

$$(n - a + 1) \times (m - b + 1) \times t$$

使用池化降维：设原图为 $n \times m \times k$ 的大小，进行 $a \times b$ 的池化，则生成图片大小为

$$\lceil \frac{n}{a} \rceil \times \lceil \frac{m}{b} \rceil \times k$$

2.5 梯度消失问题

1. 使用 ReLU
2. 增加中间输出（GoogLeNet），在某些层的输出处计算梯度可以照顾到前面层的梯度
3. 增加残差结构（ResNet）

2.6 过拟合问题

1. 使用训练集、验证集、测试集，当验证集上表现开始变差时不再继续训练
2. 增加正则化项 $||w||_2^2$
3. Dropout
4. 数据增广

2.7 NLP

2.7.1 Word Embedding: CBOW

Denote the tokens as w_i and word embeddings as $C(w_i)$.

Context: $C(w_{t-c}), \dots, C(w_{t-1}), C(w_{t+1}), \dots, C(w_{t+c})$

First layer:

$$x_w = \sum_{i=1}^{c} (C(w_{t-i}), C(w_{t+i})) \in \mathbb{R}^d$$

Second layer: project dim d to dim $\log_2(V)$, and output is denoted as $o \in \mathbb{R}^{\log_2(V)}$

Words should be organized into a huffman tree (we wish it to be balanced. If not, the dimension of second layer will increase). Use $o = (o_1, o_2, \dots)$ to select the word as infered output. o_i is calculated as $x_w^T \theta_i$.

The probability of choosing right: $p(R) = 1/(1 + e^{-x_w^T \theta_i})$

The probability of choosing left: $p(L) = 1 - 1/(1 + e^{-x_w^T \theta_i})$

The likelihood function:

$$\hat{L} = \prod_{i=2}^{l_2} [\sigma(w_x^T \theta_{i-1}^w)]^{1-o_i^w} [1 - \sigma(w_x^T \theta_{i-1}^w)]^{o_i^w}$$

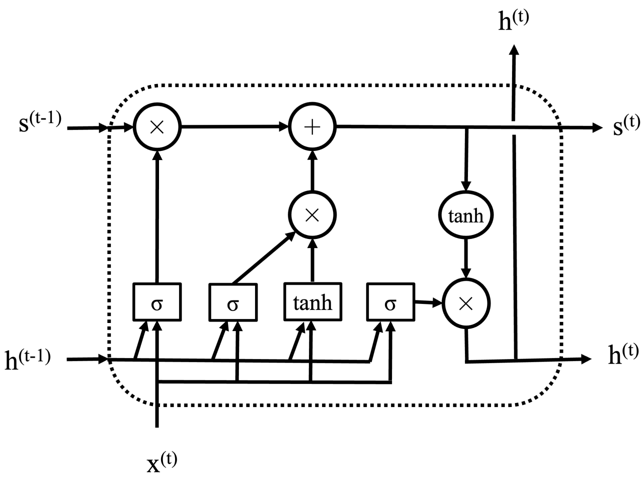
Loss function:

$$L = -\log \hat{L} = -\sum_{i=2}^{l_w} \{(1 - o_i^w) \log[\sigma(w_x^T \theta_{i-1}^w)] + (o_i^w) \log[1 - \sigma(w_x^T \theta_{i-1}^w)]\}$$

2.7.2 TextCNN

1. Convolution (one side of kernel has the same length of word vector)
2. Max pooling in the conved vector (thus the length is eliminated to 1)
3. Concatenate and Projection

2.7.3 LSTM



Gates: forget gate, input gate, output gate.

3 对抗搜索

3.1 极小极大模型

定义 3.1 (极大结点). 设结点权重为 $w(\cdot)$，若

$$w(v) = \max_{u \in child(v)} w(u)$$

则称 v 是极大节点

定义 3.2 (极大结点). 设结点权重为 $w(\cdot)$，若

$$w(v) = \min_{u \in child(v)} w(u)$$

则称 v 是极小节点

根节点是极大结点。

结点权值都是站在当前方考虑的。根节点选最大值：希望选择最有利于当前方的下法；第一层结点选择最小值：自己最小值就是对手的最大值，假设对方是理性的。

考虑搜索树从当前结点向下 $2n$ 层，每单数层获得下一层孩子中的极小值，每双数层获得下一层孩子的极大值。（当前节点是第 0 层。）（ppt CH3 pp.10）

例如向前看 4 步，则第 4 步：第 3 层从第 4 层获得极小值，第 3 步：第 2 层从第 3 层获得极大值，第 2 步：第 1 层从第 2 层获得极小值，第 1 步：当前节点从第 1 层获得极大值。

使用 DFS 在最大最小树上进行搜索。当回溯到当前结点时，更新当前结点权值。

问题：计算量巨大。

3.2 $\alpha - \beta$ 剪枝算法

定义 3.3 (α, β)**.** 设当前结点为 v，根节点为 r，$A = P(r, a)$，是根节点到 v 的路径。记 `max(A)` 表示 A 中的极大节点，`min(A)` 表示 A 中的极小结点。

$$\alpha = \max_{u \in \texttt{max(A)}} w(u)$$

$$\beta = \min_{u \in \texttt{min(A)}} w(u)$$

也就是

$$\forall u \in \texttt{max(A)}, \alpha \geq w(u)$$

$$\forall u \in \texttt{min(A)}, \beta \leq w(u)$$

算法 3.1 ($\alpha - \beta$ 剪枝)**.** 设当前结点为 v。在每次从子节点回溯，更新权值后，按照下面原则检查。

若当前结点为极大结点，且 $w(v) \geq \beta$，也就是

$$\exists u \in \texttt{min(A)}, w(v) \geq w(u)$$

则直接返回（剪枝）；

若当前结点为极小结点，且 $w(v) \geq \alpha$，也就是

$$\exists u \in \texttt{max(A)}, w(v) \leq w(u)$$

注：只考虑已经得到（暂时）权重的结点。未得到权重的祖先结点不考虑。

简记：极大 $\geq$ 极小；极小 $\leq$ 极大，则剪枝。

3.3 蒙特卡洛树搜索

将可能出现的状态转移过程用状态树表示。从初始状态开始重复抽样，逐步扩展树中结点，父节点用子节点的模拟结果提高效率。

蒙特卡洛方法：在棋盘上随机落点，一路下棋判断顺序，计算该点落棋的胜率。

蒙特卡洛搜索过程：选择、扩展、模拟、回传

选择被扩展的结点：多臂老虎机模型，信心上限算法（UCB1）

多臂老虎机模型：有 k 个独立的老虎机拉杆，每个拉杆的收益是独立的，如何选择最优的拉杆。

3.3.1 信心上限算法

对于每个拉杆 j，访问拉杆并记录收益。

当尚未达到访问次数限制时，计算每个拉杆的 UCB1 信心上界 I_j，访问信心上界最大的手臂。

信心上界的计算：

$$I_j = \bar{X}_j + \sqrt{\frac{2\ln(n)}{T_j(n)}}$$

其中 $\bar{X}_j$ 是获得回报的均值，n 是至今访问总次数，$T_j(n)$ 是拉杆 j 到目前为止的访问次数。

3.3.2 信心上限数算法 UCT

将 UCB1 应用到蒙特卡洛树搜索中，用于选择可落子点，每次选择子节点中信心上限值最大的结点，直到访问到结点，它的子节点没有被生成。

按照如下公式计算信心值。

$$I_j = \bar{X}_j + c\sqrt{\frac{2\ln(n)}{T_j(n)}}$$

一些需要注意的点：

1. 融合了极大极小搜索的观点，希望最大化最劣势时的胜率。因此当前结点胜率记录自己方的，当前节点下一代结点记录敌方胜率，再下一代结点记录我方胜率，以此类推。

2. 回传时，从新扩展出的叶子结点一路回传到当前结点。注意融合了极大极小搜索的观点。如果叶子结点代表敌方，敌方胜利，那么它的父节点代表我方，则我方失败。

每个结点记录了两个值：w_j, t_j 分别代表了**我方**胜利次数，总模拟次数。所以在选择获胜概率大的结点时，每次都选权重最大的结点即可。

蒙特卡洛搜索的问题：

1. 生成所有子节点

2. 模拟具有盲目性

3.4 AlphaGo 原理

整体想法：将神经网络与蒙特卡洛搜索结合，缩小搜索范围，提升模拟水平。

3.4.1 AlphaGo 的神经网络（策略网络、估值网络）

AlphaGo 用到的两类网络：策略网络（输入当前棋局，输出每个点的落子概率）、估值网络（输入棋局，输出胜率/权重）

策略网络：输入是 19*19 的棋盘，48 个通道，包含了执子颜色、壹平面、零平面、明智度等围棋概念。使用 13 层卷积神经网络实现，最终用 softmax 输出 19*19 的图。

策略网络的训练数据：16 万盘人类旗手的数据，等效为一个分类问题。若黑子获胜，认为黑子每一步都是正确的。损失函数用交叉熵 $L(w) = -t_a \log(p_a)$，$t_a = 1$ 意味着人在 a 处下棋，为 0 则代表不在此处下棋，p_a 是预测得到下在 a 处的概率。

估值网络：以上 48 个通道加上当前执子方（当前为黑棋全部为 1，为白棋则全部为 0）。使用 16 层卷积神经网络实现，最后一层是全连接层，通过 tanh 输出，让输出值在 $[-1, 1]$ 之间。

估值网络的训练数据：等效为一个回归问题，使用人类旗手的数据，使用 MSE loss $L(w) = (R - V(w))^2$ 作为损失函数，R 是棋盘胜负情况，胜为 1，$V(w)$ 是预测的胜率。

3.4.2 神经网络与蒙特卡洛树结合

设结点 s 第 i 次模拟的收益为

$$v_i(s) = \lambda value(s) + (1 - \lambda) rollout(s)$$

其中 $value(s), rollout(s)$ 分别代表估值网络的输出和一次模拟结果

平均收益：

$$Q(s_a) = \frac{1}{n} \sum_{i=1}^{n} v_i(s_a)$$

其中 s_a 是 s 在 a 处落子后的棋局

探索项：

$$u(s_a) = c \cdot p(s_a) \frac{\sqrt{N(s)}}{N(s_a) + 1}$$

其中 $N(\cdot)$ 为模拟次数，$p(s_a)$ 为策略网络输出的在 a 处下棋的概率，c 为加权系数。

选择过程：用 $Q(s_a) + u(s_a)$ 代替信心上限 J，有限选择 $Q(s_a) + u(s_a)$ 的子节点，遇到叶结点 s_l 结束，该结点被选中

生成过程：生成 s_l 的所有子节点，规定了最大的节点深度。

模拟过程：计算 $v_i(s)$，采用推演策略网络（一个更小的（被压缩的）策略网络，用于提升推理速度）计算 rollout，规定了总模拟次数

回传过程：结点给父结点回传带符号的权重（因为是极大极小搜索）

选择过程：选择根节点的子节点中被选择次数最多的结点作为最终的走步。

对比：

$$\text{MCTS: } I_j = \bar{X} + c \cdot \sqrt{\frac{2\ln(n)}{T_j(n)}}$$

$$\text{AlphaGo: } I_j = \frac{1}{n}\sum_{i=1}^{n}(\lambda value(s) + (1-\lambda)rollout(s)) + c \cdot p(s_a)\frac{\sqrt{N(s)}}{N(s_a)+1}$$

3.5 围棋中的强化学习方法

强化学习：试错和延迟收益。深度强化学习：使用神经网络方法。

将收益转化为标注：不能获得所有情况下既正确又有代表性的示例。

手段：将深度强化学习问题转化为神经网络训练问题。不同转换方法构成了不同的深度学习方法，关键是损失函数的定义。

3.5.1 基于策略梯度的强化学习

数据：自我博弈产生。数据格式：(s, a, p_a, t_a) 其中 s 是当前棋局，a 指在 a 处下棋，p_a 是策略网络生成的获胜概率，t_a 是胜负值，获胜为 1，没获胜为-1.

延迟收益：只有当一局棋下完后，才能对所有操作中的 t_a 标注。

损失函数：交叉熵 $L(w) = -t_a \log(p_a)$，假设获胜时的每一步棋都是正确的，负者都是不正确的。获负时对权重的修改量大小与获胜时一样，方向相反。

学习流程：设一个胜率比例 ε。对新的数据（或者是自行对弈）得到的新网络，与旧网络进行对弈，如果胜率更高（超过 ε）则更新网络。

注意点：强化学习过程中，每个样本只使用一次；基于策略梯度的强化学习方法学到的是在每个可落子点行棋的获胜概率；监督学习策略网络学到的是某个可落子行棋的概率。

3.5.2 基于价值评估的强化学习

价值评估网络：对一个行棋点的价值（收益）进行评估。输入：当前棋局和行棋点，输出 $[-1, 1]$ 之间的估值。

数据：自我博弈产生。数据格式为 $(s, a, V(s, a), R)$，$V(s, a)$ 表示 s 在 a 处落子时网络的输出。$R \in \{1, -1\}$，胜负值。

损失函数：$L(w) = (R - V(s, a))^2$

3.5.3 基于演员-评价方法的强化学习

两个网络：策略网络、估值网络。策略网络相当于演员，估值网络相当于教练。估值网络对策略网络的每一步进行评价。

收益增量：评价一步棋的好坏 $A = Q(s,a) - V(s)$，$V(s)$ 是预期收益，取值范围 $[-1,1]$；$Q(s,a)$ 是实际收益，取值范围 $[-1,1]$，$A \in [-2,2]$，A 越大说明这一步越好。实际计算时，$Q(s,a) := R$，指定为最终胜负值。

损失函数：两部分。

评价部分：$L_1(w) = (R - V(s))^2$，为了使评价尽可能的准，避免 $V(s)$ 训的过小

演员部分：$L_2(w) = -A\log(p_a)$，其中 A 是收益增量，p_a 是策略网络在 a 处下棋的概率

综合损失函数：$L(w) = L_1(w) + \lambda L_2(w)$

3.6 AlphaGo Zero

将策略网络、估值通道合并为一个双输出网络。

通道：17 个，16 个记录了到目前为止的 8 个棋局，每个棋局两个通道，分别记录黑、白棋位置；另一个通道记录执棋方。

输出：19*19 + 1，增加了放弃行为。

3.6.1 Zero 中的 MTCS

节点第 i 次模拟的收益：去掉了 rollout，直接令 $v_i(s) = value(s)$，其他与上述过程相同。

损失函数：估值网络 $L_1 = (z-v)^2$，z, v 分别代表实际胜负值和估值网络输出；策略网络 $L_2 = -\pi_1 \log(p_1) - \cdots - \pi_{362}\log(p_{362})$

总损失函数是 $L = L_1 + L_2 + ||\theta||_2^2$

引入多样性：人为对策略网络的输出增加噪声，防止走入错误的方向。使用狄利克雷分布。落子概率调整为 $\lambda p_a + (1-\lambda)p_d$，其中 p_d 是狄利克雷分布采样，p_a 是策略网络输出。

4 统计机器学习

4.1 线性可分支持向量机

定义 4.1 (LSVM). 给定线性可分训练集，其中

$$T = \{(x_1, y_1), \ldots, (x_N, y_N)\}, x_i \in X = \mathbb{R}^n, y = \{+1, -1\}$$

通过间隔最大化得到分类超平面 $w^* x + b^* = 0$，分类函数 $f(x) = sgn(w^* x + b)$ 称为线性可分支持向量机

定义 4.2 (函数间隔). 设训练集为 T，超平面 (w,b)，定义 (w,b) 与样本点 (x_i, y_i) 的函数间隔为

$$\hat{\gamma}_i = y_i(wx_i + b)$$

定义超平面关于 T 的函数间隔为

$$\hat{\gamma} = \min_i \hat{\gamma}_i$$

定义 4.3 (几何间隔). 设训练集为 T，超平面 (w,b)，定义 (w,b) 与样本点 (x_i,y_i) 的函数间隔为

$$\gamma_i = y_i(\frac{w}{||w||}x_i + \frac{b}{||b||})$$

定义超平面关于 T 的函数间隔为

$$\gamma = \min_i \gamma_i$$

问题：间隔最大化

$$\begin{aligned}&\max_{w,b} \gamma \\ &s.t. y_i(\frac{w}{||w||}x_i + \frac{b}{||b||}) \geq \gamma, i = 1,2,\dots,N\end{aligned}$$

这等价于

$$\begin{aligned}&\max_{w,b} \frac{\hat{\gamma}}{||w||} \\ &s.t. y_i(wx_i + b) \geq \hat{\gamma}, i = 1,2,\dots,N\end{aligned}$$

不妨令 $\hat{\gamma} = 1$，因为可以整体放缩，所以问题等价于

$$\begin{aligned}&\min_{w,b} \frac{1}{2}||w||^2 \\ &s.t. y_i(wx_i + b) \geq 1, i = 1,2,\dots,N\end{aligned}$$

使用 Lagrange 乘子法求解：Lagrange 函数为

$$L(w,b,\alpha) = \frac{1}{2}||w||^2 + \sum_{i=1}^{N} \alpha_i[1 - y_i(wx_i + b)]$$

注：对 L 求最大，则要求了 $\frac{1}{2}||w||^2$ 最大的同时，$\alpha_i[1 - y_i(wx_i + b)]$ 也最大。由 $\alpha_i[1 - y_i(wx_i + b)]$ 最大和约束条件知道，$y_i(wx_i + b) = 1$

所以

$$\max_\alpha L = \frac{1}{2}||w||^2$$

因此

$$\min_{w,b} \max_\alpha L = \min_{w,b} \frac{1}{2}||w||^2$$

考察对偶问题 $\max_\alpha \min_{w,b} L$，由于有关系

$$\min_{w,b} L \leq L \leq \max_\alpha L$$

所以有如下关系，当满足 KKT 条件时，等式成立。

$$\max_{\alpha} \min_{w,b} L \leq \min_{w,b} \max_{\alpha} L$$

KKT 条件:

$$\begin{aligned}
&\frac{\partial L}{\partial (w,b)} = 0 \\
&\alpha_i[1 - y_i(wx_i + b)] = 0 \\
&[1 - y_i(wx_i + b)] \leq 0 \\
&\alpha_i \geq 0 \\
&i = 1, 2, \ldots, N
\end{aligned}$$

由 KKT 条件求 w^*, b^*:

$$\frac{\partial L}{\partial w} = w^* - \sum_{i=1}^{N} \alpha_i^* y_i x_i = 0 \Rightarrow w^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i$$

$$\alpha_i[1 - y_i(wx_i + b)] = 0 \Rightarrow b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i x_i^T x_j$$

注：与 $\alpha_i > 0$ 对应的 x_i 就是支持向量。
最终，求解线性可分支持向量机转化为求解如下优化问题:

$$\begin{aligned}
&\min_{\alpha} \left(\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{N} \alpha_i \right) \\
&s.t. \sum_{i=1}^{N} \alpha_i y_i = 0, \forall \alpha_i \geq 0
\end{aligned}$$

4.2 线性支持向量机

增加松弛参数:

$$\begin{aligned}
&\min_{w,b} \frac{1}{2} ||w||^2 + C \sum_{i=1}^{N} \xi_i \\
&s.t. y_i(wx_i + b) \geq 1 - \xi_i, i = 1, 2, \ldots, N \\
&\xi_i \geq 0, i = 1, 2, \ldots, N
\end{aligned}$$

通过化简，得到求解问题为

$$\min_{\alpha}\left(\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j-\sum_{i=1}^{N}\alpha_i\right)$$

$$s.t.\sum_{i=1}^{N}\alpha_i y_i=0,\forall 0\leq\alpha_i\leq C$$

注：在计算 b^* 时候选择一个 $0<\alpha_j<C$ 计算

$$b^*=y_j-\sum_{i=1}^{N}y_i\alpha_i^* x_i^T x_j$$

4.3 非线性支持向量机算法

定义 4.4 (核函数)**.** 设 X 是输入空间，H 是特征空间，如果存在函数 $\Phi: X\to H$ 满足

$$K(x,z)=\Phi^T(x)\Phi(z)$$

则称 $K(\cdot,\cdot)$ 为核函数。

$$\min_{\alpha}\left(\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i,x_j)-\sum_{i=1}^{N}\alpha_i\right)$$

$$s.t.\sum_{i=1}^{N}\alpha_i y_i=0,\forall 0\leq\alpha_i\leq C$$

分类超平面：

$$\sum_{i=1}^{N}\alpha_i^* y_i K(x_i,x)+b^*=0$$

常用核函数：

1. 多项式核函数

$$K(x,z)=(x^T z+1)^p$$

2. 高斯核函数：

$$K(x,z)=\exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)$$

总结

1. 线性可分支持向量机

$$\min_{\alpha}\left(\frac{1}{2}\alpha_i\alpha_j y_i y_j x_i^T x_j-\sum_{i=1}^{N}\alpha_i\right)$$

$$s.t.\sum_{i=1}^{N}\alpha_i y_i=0,\forall\alpha_i\geq 0.$$

解得

$$w^* = \sum_{i=1}^{N} \alpha_i y_i x_i$$

挑选一个 $\alpha_i > 0$，通过下式解出 b^*

$$y_i(w^* x_i + b^*) = 1$$

2. 线性支持向量机

$$\min_{\alpha} \left(\frac{1}{2} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{N} \alpha_i \right)$$

$$s.t. \sum_{i=1}^{N} \alpha_i y_i = 0, \forall \alpha_i, 0 \leq \alpha_i \leq C.$$

解得

$$w^* = \sum_{i=1}^{N} \alpha_i y_i x_i$$

挑选一个 $0 < \alpha_i < C$，通过下式解出 b^*

$$y_i(w^* x_i + b^*) = 1$$

3. 非线性支持向量机

$$\min_{\alpha} \left(\frac{1}{2} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N} \alpha_i \right)$$

$$s.t. \sum_{i=1}^{N} \alpha_i y_i = 0, \forall \alpha_i, 0 \leq \alpha_i \leq C.$$

解得

$$w^* = \sum_{i=1}^{N} \alpha_i y_i \Phi(x_i)$$

挑选一个 $0 < \alpha_i < C$，通过下式解出 b^*

$$y_i(w^* \Phi(x_i) + b^*) = y_i(\sum_{j=1}^{N} \alpha_j y_i K(x_j, x_i) + b^*) = 1$$

4.4 决策树

随机变量的熵：$H(X) = -\sum_{i=1}^{n} p_i \log p_i$
当概率由数据集 D 得到时，记作 $H(D)$

条件熵表示已知 X 时 Y 的不确定性：

$$H(Y|X)=\sum_{i=1}^{n}p_iH(Y|X=x_i),p_i=P(X=x_i)$$

其中

$$H(Y|X=x_i)=-\sum_{j=1}^{N}P(Y=y_j|X=x_i)\log P(Y=y_j|X=x_i)=-\sum_{j=1}^{N}\frac{p_{ij}}{p_i}\log\frac{p_{ij}}{p_i}$$

信息增益：特征 A 对数据集 D 的信息增益定义为

$$g(D,A)=H(D)-H(D|A)$$

4.4.1 ID3

输入：训练集 D, 特征集 A，$\varepsilon>0$

输出：决策树 T

1. 若 D 中所有实例属于同一类 C_k，则 T 是单节点树，将 C_k 作为该结点的类标记，返回 T

2. 若 A 为空，则 T 是单节点数，将 D 中实例数最大的类 C_k 作为该结点的类标记，返回 T

3. 否则计算 A 中各特征对 D 的信息增益，选择信息增益最大的特征 A_g

4. 若 A_g 的信息增益小于阈值 ε，则令 T 为单节点树，将实例数中最大的类别 C_k 作为类标记，返回 T

5. 否则对 A_g 的每一个可能值 a_i，按照 $A_g=a_i$ 将 D 分割成若干子集作为 D 的子节点

6. 对于 D 的每个子节点 D_i，若 D_i 为空则选实例最大的类作为标记构建子节点，不然构建子节点 $A-\{A_g\}$，将子节点当成当前节点递归地进行下去

4.4.2 C4.5

定义 4.5 (信息增益比)**.**

$$g_R(D,A)=\frac{g(D,A)}{H_A(D)}$$

其中 $H_A(D)=-\sum_i\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}$，其中 $|D_i|$ 是特征 A 取值为 A_i 时的样本个数。

使用信息增益比选择特征，得到了 C4.5 算法。除此之外，增加了对连续值属性。对于连续值 A，找到属性值 a_0，将 $\leq a_1,>a_0$ 划分到左右子树。选择信息增益比最大的 a_0。

信息增益比的问题：倾向于分割不均匀的特征（处于不同取值的样本数量不均衡）。解决办法：先取 top n 个**信息增益**大的特征，再从其中选择**信息增益比**最大的特征。

4.4.3 决策树的剪枝

记号：树的叶结点个数表示为 $|T|$，t 是树的叶结点，该结点有 N_t 个样本，其中 k 类的样本点有 N_{tk} 个。

定义 4.6 (经验熵)**.**

$$H_t(T) = -\sum_{k}^{K} \frac{N_{tk}}{N_t} \log_2 \frac{N_{tk}}{N_t}$$

经验熵的大小可以衡量该叶结点分类纯度。

定义 4.7 (剪枝损失函数)**.**

$$C_a(T) = \sum_{i=1}^{|T|} N_i H_{t_i}(T) + a|T|$$

记 $C(T) := \sum_{i=1}^{|T|} N_i H_{t_i}(T)$ 反应了预测误差，$a|T|$ 反应树的大小，$C_a(T) = C(T) + a|T|$

剪枝方法：从下向上，当叶子结点剪枝后损失函数变小，则剪枝；若不减少则不剪枝。
另一种解决过拟合的问题：随机森林，通过投票改善决策树