1 搜索

1.1 搜索问题的概念

搜索问题的五个要素是指**状态空间**、**后继函数**、**初始状态**、**目标函数**和**路径耗散**。在状态图上，每个节点表示一个状态，一个有向边及其两个端点表示后继函数。当状态图中初始结点和终态结点在不连通的两个分量中，该搜索问题无解。对状态的定义应尽可能使所有状态是可达的，状态数目越少越好，并且最好使后继函数和搜索算法方便设计。终态可以是一个满足某些条件的状态集合。

对状态图的宽度优先遍历算法是求解搜索问题的基准算法。其他搜索算法与基准算法相比，相同点在于算法大框架基本一致、所有可达和不可达状态构成的状态图一致；不同点在于从状态图中选择和处理的状态子集不一样、状态子集构成的状态子图不一样、后继函数不一样。将状态图上算法的搜索过程涉及到的状态和搜索顺序抽取出来就得到了对应的搜索树。状态图中的状态可能被重复访问，体现在状态图上的同一节点会变成搜索树上的不同节点。

搜索树的边界是指那些已访问过但未扩展状态的节点集合。搜索树上的叶节点不一定是搜索树的边界节点。修改节点集合的优先性质就是在改变搜索的扩展策略。

评估算法性能时要考虑的三个性质是**完备性**、**最优性**和**复杂性**。一个搜索树的大小由初态、后继函数和搜索策略决定，其影响因素是分支因子、最浅目标状态的深度和路径的最大长度。时间复杂度是指访问过的节点数目，空间复杂度是指同时保存在内存中节点数目的最大值。

1.2 盲搜索

从搜索树的边界 FRINGE 这个角度来看，无信息搜索、盲搜索的特点是边界是无序的，而有信息搜索、启发式搜索的特点是将更有希望的节点放在边界的最前面。

基准算法是盲搜索算法，它单纯将新的节点插入 FRINGE 的末尾。

基准算法的两个重要参数是分支因子 b（后继函数返回的最大状态数目）和目标状态的最小深度 d。基准算法是完备的；在路径耗散相同的情况下是最优的；访问节点数是 $O(b^d)$。当问题无解时，如果状态空间无限大或者任意状态可以任意次重复访问，则基准算法不会停止。

1.2.1 盲搜索的改进

双向搜索算法 是指分别从初态和终态启动两个搜索算法。维护两个边界集合，当集合相交时算法结束。双向搜索的两个搜索算法可以不一样。双向搜索的评价：完备的；路径耗散相同时保证最优性；时空复杂度（假设两个方向的分支因子相同）是 $O(b^{d/2})$。

深度优先搜索 节点扩展时，新节点总是插入到边界集合的队头。深度优先的评价：若搜索树有限则完备；不一定最优；时间复杂度（最坏情形，假设 m 是叶结点的最大深度）$O(b^m)$，空间复杂度 $O(bm)$。

回溯法 节点扩展时，只插入一个新节点到边界集合的队头。评价：若搜索树有限则完备；不一定最优；时间复杂度（最坏情形，假设 m 是叶结点的最大深度）$O(b^m)$，空间复杂度 $O(m)$。

深度受限的深度优先搜索 设置扩展节点的深度阈值 k，当节点深度大于阈值时不再扩展。评价：若搜索树有限则完备；不一定最优；时间复杂度 $O(b^k)$，空间复杂度 $O(bk)$。

迭代深入搜索 使用不同的受限深度参数 $k = 0, 1, 2 \cdots$ 重复执行深度受限搜索。结合了 BFS 和 DFS 的长处，但要重复搜索节点。评价：当 $k = d$ 时，完备；单步路径耗散相同时最优，否则不一定；访问节点数（最坏情形）$O(b^d)$，空间复杂度 $O(bd)$。

代价一致搜索 每次在边界中选择让目前获得的路径耗散最小的节点扩展，当单步路径耗散相同时等价于基准算法。要求单步耗散有下界 $x_i \geq \varepsilon > 0$。评价：当搜索树有限且单步耗散有下界时完备；最优；时空复杂度为 $O(b^{\lceil C^*/\varepsilon \rceil})$。

1.2.2 避免状态重复访问

重复访问状态的产生原因是可逆的行动。可以设置一个标识数组记录已访问过的节点，若扩展结点得到的后继状态在标识数组中则直接丢弃。这一方法需要的内存比较大，而且如果要保证最优性需使用宽度优先搜索算法。例如在代价一致搜索中，用一个 CLOSED 表保存每个状态路径的最优路径耗散，要扩展节点 s 就检查在 CLOSED 表中已有的 s 路径代价，保存更小的路径代价并将 s 所有不在 CLOSED 表中的后继状态放入边界中等待扩展。

树搜索中不同的节点可能表示的是相同的状态，分别表示从初态出发到达同一状态的不同路径；而图搜索中不同的节点代表不同的状态。

若状态空间无限，或者有限但允许状态被任意重复访问则搜索一般是不完备的；若状态空间有限且访问时重复状态被抛弃则搜索是完备的但一般不是最优的。

1.3 启发式搜索

启发式搜索通过设计一个评估函数 f 将每一个边界中的节点映射为一个值，每次选择边界中值最小的节点进行扩展。

设 $g(N)$ 是已走过的路径耗散，$h(n)$ 是启发式函数估计的预期耗散。设计 $f(N) = g(N) + h(N)$ 对应 A* 算法，$f(N) = h(N)$ 对应贪婪算法。

1.3.1 启发式函数的设计要求

基本要求 $h(N) \geq 0, h(Goal) = 0$；离目标越近 $h(N)$ 越小。

可采纳性 $0 \leq h(N) \leq h^*(N)$，其中 $h^*(N)$ 是到目标的实际最优路径。这代表启发式函数必须乐观的估计将来的耗散。通常可以用一个松弛问题设计可采纳的启发式函数。

一致性/单调性 $h(N) \leq c(N, N') + h(N')$，其中 N' 是 N 的后继。注意一致的启发式函数都是可采纳的。这带来 $h(N) \leq h^*(N) \leq c(N, N') + h(N')$。

可以证明的是，一个可采纳的启发式函数使得 A* 算法具有完备性（不丢弃重复访问节点，而是保存 $g(N)$ 更小的节点）、（树搜索上的）最优性。若状态空间无限或允许状态重复访问，则 A* 算法不会终止。

如果一个启发式函数是一致的，A* 算法在图搜索上将获得最优性。此时面对重复节点，如果一个新节点已在 CLOSED 中则直接丢弃；如果在 FRINGE 中就保留 $f(N)$ 更小的节点。一个特例是 $f(N) = g(N)$，这时就是代价一致搜索。

如果对任何节点 $h_1(N) \leq h_2(N)$ 就说 h_1 更精确。更精确的启发式函数使得它扩展的节点更少。

1.3.2 IDA*

设置 f 的阈值，超过阈值的节点不再扩展；迭代执行以降低 A* 对内存的需求。每次更新阈值为 FRINGE 中 f 的最小值。IDA* 是完备的、最优的，避免了 FRINGE 集合的排序开销，但无法充分利用内存、两次迭代之间保留的信息太少，无法避免重复访问不在路径上的节点。SMA* 把内存耗尽，抛弃保留的高耗散最旧节点插入新节点。

1.4 优化问题

优化问题是指

$$\min f(x) \text{ s.t. } h(x) \leq 0$$

其中 $f(x)$ 是目标函数，$h(x)$ 是约束条件。这是一个搜索问题，初态是一个随机解，状态空间由任意解构成，目标状态是最小值的解，后继函数是由不同算法给定的。一个后继函数将把一个无序的状态空间建立某种序关系，在这一关系上每个解对应的函数值构成了优化问题的地貌图。

1.4.1 梯度下降法

对终止准则 T、步长序列 λ 和初态 X_0，迭代更新

$$X_{t+1} = X_t - \lambda_t \nabla_X g(X_s), \text{where } \nabla_X f = \left(\frac{\partial g}{\partial x_1}, \frac{\partial g}{\partial x_2}, \cdots, \frac{\partial g}{\partial x_n} \right)$$

不能保证获得全局最优，且终止准则和步长序列需要人工经验。

直线搜索 在一个预定的测试步长集合 $\{\cdots, -2c, -c, c, 2c, \cdots\}$ 中每轮选择其中使函数下降最快的那一个 $\lambda^* = \arg_\lambda \min g(X - \lambda \nabla_X g)$。

随机梯度下降 每次迭代的梯度信息都随机地挑选一条训练数据计算，降低时间开销。

1.4.2 牛顿法

通过求解函数一阶导数 $g'(x)$ 的零点来找到函数极值对应的 X。即每轮找 $g'(x)$ 在 X_n 的切线，以其与轴的交点作为新点 X_{n+1}。

离散的组合优化问题可用分支定界法解决。

1.4.3 爬山法

每一轮在 X_n 的邻域的表现更好的子集中找到策略确定的最优点作为下一轮的 X_{n+1}，这找到的是局部最优解。策略可以是最小值（离散化的梯度下降）、随机值（随机梯度下降），甚至可以直接在邻域中随机找。

局部剪枝搜索 初始时刻有 k 个种子，每一轮在各自产生的 l 个后继共 kl 个候选者中根据策略选择下一轮的 k 个解。

1.4.4 模拟退火

每一轮在 X_n 的邻域中随机选择一个，如果表现更好则选中；否则概率地选中这个新点。这个概率可以根据一个 schedule 随轮次逐渐下降，称为退火。

1.4.5 Tabu 搜索

保留一个长度有限的历史解列表，如果发现的新解在列表中则退而求其次的选择其他解，可以一定程度上防止局部最优。列表越长搜索域越广，列表短时的局域性比较好。

1.4.6 $(1+1)-EA$

每一轮在全状态空间中随机选择一个解，如果这个新解更好则转移。相当于是全局邻居算子的爬山法。

并行化 初始时刻有 k 个解，每一轮在各自产生的 l 个后继共 kl 个候选者中根据策略选择 k 个（称为 $(k,kl)-ES$）或者在 kl 个候选者和 k 个种子共 $k+kl$ 个解中选择 k 个（称为 $(k+kl)-ES$）。

后继函数的设计 可以从 k 个种子中每次挑选不少于 1 个进行后继的产生。比如两个二进制串的混杂、某些位的翻转等等，称为遗传算法。

1.5 约束满足问题

一个 CSP 问题是一个三元组 (X,D,C)，其中 X 是一组变量；D 是值域；C 是一个约束组，包含了 m 个约束 X 分量间取值关系的计算公式。根据解的数目是否有限可以分成有限和无限 CSP。一旦有一个变量的取值范围是无限的则 CSP 问题就是无限的。可以认为 CSP 是一个由满足了前 k 个分量的赋值构成状态空间，新增一维满足约束的分量为后继函数的搜索问题。

这产生的关键问题是每一次选择剩余的哪个分量进行赋值？赋可选择的哪个值？

1.5.1 前向检验

每进行一次赋值，就把未赋值的变量中现在不再能取的值从其候选值域中去掉。然后，每次从残留值域最小的变量中，选择在未满足约束中出现次数最多的变量，对它进行本轮赋值，选择残留值域中，使得其他未赋值变量的残留值域缩减最小的那个值。

2 不确定搜索

2.1 马尔可夫决策过程

现在一个后继函数将使得状态概率地行进到一系列状态。用 $T(s,a,s')$ 表示从 s 出发采取行动 a 导致结果 s' 的概率，这一过程获得的收益是 $Reward(s,a,s')$。我们假定有马尔可夫性，行动 a 的确定只和当前状态 s 有关。

很自然地，一个状态的后继状态集合地转移概率分布要满足归一性 $\sum_{s'\in\mathbf{S}}T(s,a,s')=1$。

对这样的状态图，我们称一个解是一个策略，它给出了对每一个状态 s 的最优行动 a^*。

在状态 s 下的每个行动 a，都自然的有其期望收益

$$\sum_{s'\in\mathbf{S}}T(s,a,s')Reward(s,a,s')$$

一个策略 π 导致的所有从初态到终态的路径的收益期望定义为策略的价值，或称策略收益 V_π。如果通过每条路径乘上每条路径的概率计算策略收益是过于困难的，因此需要更好的策略价值计算方式。

我们注意到策略 π 在状态 s 处获得的期望收益满足

$$V_\pi(s) = \sum_{s' \in \mathbf{S}} T(s, \pi(s), s')[Reward(s, \pi(s), s') + V_\pi(s')]$$

因此我们得到了 n 个状态对应的 n 个未知数的方程，可以联立求解。或者我们采用迭代更新的方法求策略在每个状态的收益，迭代式正是上式。

现在我们考虑到，对于 n 个状态的空间，共有 $\prod_{i=1}^{n} |Action(s_i)|$ 种策略。那么哪个策略是最优的呢？

2.1.1 Q-value

我们此前定义的 $V_\pi(s)$ 是说，在 s 处采用行动 $\pi(s)$ 后得到的预期价值。现在我们想要看看仅仅改变在 s 处采取的行动，我们得到的预期价值会发生什么变化？因此我们定义 $Q_\pi(s, a)$，它是说在 s 处采用行动 a（而不是 $\pi(s)$）但在后续过程中仍然采用策略 π 将获得的预期收益。因此我们有

$$Q_\pi(s, a) = \sum_{s' \in \mathbf{S}} T(s, a, s')[Reward(s, a, s') + V_\pi(s')]$$

我们如何利用 Q-value 来更新策略呢？每一轮，我们对任意状态 s，寻找一个动作 a，使得 $Q_\pi(s, a)$ 最大。即

$$\pi_{new}(s) = \arg\max_{a \in Action(s)} Q_\pi(s, a)$$

现在我们在每一轮首先评估策略得到 V_π，用这个评估值和 Q-value 去更新策略 π 直到满足终止条件。

然而麻烦在于我们每一轮都要评估策略，而策略评估本身也是一个迭代算法。我们进行的下一个改进是在每一轮中，只需对所有状态更新

$$V_\pi^t(s) = \max_{a \in Action(s)} \sum_{s' \in \mathbf{S}} T(s, a, s')[Reward(s, a, s') + V_\pi^{t-1}(s')]$$

在这个迭代式中我们把策略评估要求的迭代更新和策略改进要求的行动选择结合在了一起，同样保证了全局最优性。

2.1.2 折扣因子

现在我们认为未来路径上距离越远获得的收益越低。也即路径收益变为 $r_1 + \lambda r_2 + \lambda^2 r_3 + \cdots$。这会使得策略评估的递推方程变为

$$V_\pi(s) = \sum_{s' \in \mathbf{S}} T(s, \pi(s), s')[Reward(s, \pi(s), s') + \lambda V_\pi(s')]$$

2.2 强化学习

此前我们的计算中要求对全局信息 T 和 $Reward$ 的了解。现在我们假设我们丢失了这些信息，需要我们进行探索。问题变成了，我们知道一系列历史探索序列 $s_0, a_1, r_1, s_1, a_2, r_2, \cdots, a_n, r_n, s_n$，我们仍想得到一个最优策略。这一算法的框架是，在每一轮选择行动、收集奖励、转移状态、更新参数。然而选择行动的策略是什么？参数是什么？如何更新？

如果我们不知道具体的 $T(s, a, s')$，就用已知数据中 s, a, s' 的频率 $N(s, a, s')/N(s, a)$ 估计它。相应地，用 $\sum_{r \in (s,a,r,s')} r/N(s, a, s')$ 来估计 $Reward(s, a, s')$。这样的方法称为基于模型的蒙特卡洛方法，

其本质是去估计转移概率和奖励构成的模型。但是这样估计的问题在于很多状态行动序列可能没有历史数据，同时历史数据可能不满足独立性假设。

现在我们直接考虑对一个策略 π 的估计。我们假设采用该策略得到一个历史轨迹，那么就用轨迹中的 (s,a) 组带来的收益均值来估计 $Q_\pi(s,a)$。这样，我们每遇到一次 (s,a)，就去更新对应的 $Q_\pi(s,a)$

$$Q_\pi(s,a) = (1-\xi)Q_\pi(s,a) + \xi u_t$$

其中 $\xi = 1/((s,a)$此前更新次数 $+1)$，u_t 是本次的收益估计。对于遇到的轨迹 $s,a,r_t,s',a',r_{t+1},\cdots$ 要估计 u_t，我们可以仅从本轮轨迹得到 $u_t = r_t + \lambda r_{t+1} + \lambda^2 t_{t+2} + \cdots$；也可以采用 SARSA 法认为 $u_t = r_t + \lambda Q_\pi(s',a')$。

有了对策略的估计，我们可以每轮评估策略、改进策略来获得最优策略；或者直接对 Q 进行值迭代更新

$$Q_{opt}(s,a) = (1-\xi)Q_{opt}(s,a) + \xi(r + \lambda \max_{a' \in Action(s')} Q_{opt}(s',a'))$$

那么回到最开始的问题：我们现在如何选择行动？当遇到状态 s 时，我们有不同的策略选择行动：如果是贪婪策略，我们将选择 $\arg\max_{a \in Action(s)} Q_{opt}(s,a)$；或者我们随机游走。这两种策略对应着剥削策略（Exploitation）和探索策略（Exploration）。一般我们折中，递减概率 ε 地选择随机游走策略来进行。

现实应用中，如果有很多未经探索的 (s,a) 该如何估计？我们可能使用已经出现的 (s,a) 来回归估计它们。比如用已有数据训练一个特征提取函数 $\phi()$ 和线性分类函数 $\mathbf{W}$，来估计 $Q_{opt}(s,a) = \mathbf{W}\phi(s,a)$。现在问题是：$\phi()$ 如何设计，$\mathbf{W}$ 又如何确定呢？

3 对抗搜索

博弈的基本要素是**参与者、行动集**（策略集）和**收益**。**完美信息**是指每个参与者对博弈信息完全了解；**静态博弈**是指所有参与者同时选择策略并行动；**理性参与者**是指参与者均追求各自的利益最大化；**独立决策**是指参与者之间不协商。

要求解博弈过程，如果两人都有严格占优策略（该策略对其他参与者的所有策略都是严格最佳应对），则预计他们均采取它；如果只有一人有严格占优策略，则对手必然采取其最佳应对；如果二人均不存在严格占优策略，则寻找一个纳什均衡。一个策略组是纳什均衡，是指策略组中的所有策略互为最佳应对。此时，没有参与者会更换策略；没有参与者可以通过只改变自己的策略而得到额外好处，尽管可能存在一个对二者都更好的策略组。

零和博弈是指不存在纳什均衡的博弈。混合策略是指概率地选择不同策略。混合策略均衡是指在各自概率策略下，双方的收益期望互为最大。

更常见的动态博弈，是指各个参与者的行动非同时发生。完美信息的动态博弈是一种马尔可夫决策过程，但常见的下棋是一种强化学习，因为不知道转移概率和每个行动的奖励。

现在可以把博弈问题描述为搜索问题。状态是参与者的策略及其收益，后继函数是各种可能的新策略，终态时参与者无论如何调整策略收益都不再增加。注意，这个问题中中间状态的效用可能并

不重要，并且并不关心路径耗散。由于博弈具有对抗性，因此博弈评估包含对对手策略的假设；并且分支不会产生收益，它仅仅描述终态的收益在博弈树非叶节点上的聚集。

3.1 极小极大算法

给定一颗博弈树，最优策略可以通过检查每个结点的极小极大值决定，它对应该状态对于 MAX 而言的效用值。对于终止节点，其 MINIMAX 值为其效用值自身。对于 MAX 节点，其 MINIMAX 值就是所有可能的后继节点 MINIMAX 的最大值；对于 MIN 节点，其 MINIMAX 值就是所有可能的后继节点 MINIMAX 的最小值。极小极大算法递归计算每个后继的 MINIMAX 值，自上而下一直前进到叶节点，然后随着递归回溯把 MINIMAX 值回传。

$\alpha-\beta$ 剪枝 剪枝的一般原则是：考虑在树中某处的节点，如果对应的参与者在其父节点或者更上层的任何选择点有更好的选择，那么实际情况中永远不会到达这个节点。α 表示到目前为止路径上发现的 MAX 的最大值选择，β 表示 MIN 的最小值选择。算法不断更新 α 和 β 的值，并且当某个结点的值分别比目前的 MAX 的 α 或者比 MIN 的 β 更差的时候剪裁此节点剩下的分支。

3.2 蒙特卡洛树搜索

博弈树太大，搜索效率太低。采用随机模拟来估计每个分支的收益和转移概率。

3.2.1 上置信区间

在棋类游戏中，经常有这样的问题，我们发现在某种棋的状态下，有 2 个可选动作，第一个动作历史棋局中是 0 胜 1 负，第二个动作历史棋局中是 8 胜 10 负，那么我们应该选择哪个动作好呢？如果按 $\varepsilon-$ 贪婪策略，则第二个动作非常容易被选择到。但是其实虽然第一个动作胜利 0%，但是很可能是因为这个动作的历史棋局少，数据不够导致的，很可能该动作也是一个不错的动作。那么我们如何在最优策略和探索度达到一个选择平衡呢？$\varepsilon-$ 贪婪策略可以用，但是 UCT 是一个更不错的选择。

UCT 首先计算每一个可选动作节点对应的分数，这个分数考虑了历史最优策略和探索度：

$$UCT = \bar{X}_i + 2C_p\sqrt{\frac{2\ln n}{n_i}}$$

其中 $\bar{X}_i$ 是节点获得的平均奖励，n 是总的模拟次数，n_i 表示该节点被探索的次数。UCT 平衡了搜索的深度和广度。

3.2.2 蒙特卡洛树搜索

MCTS 的搜索需要走 4 步：

第一步是选择 (Selection), 这一步会从根节点开始，每次都选一个最佳子节点，即使用 UCT 选择分数最高的节点，直到来到一个“存在未扩展的子节点”的节点。之所以叫做“存在未扩展的子节点”，是指这个局面存在未走过的后续着法，也就是 MCTS 中没有后续的动作可以参考了。这个选择结点的过程称为树策略。这时进入第二步。

第二步是扩展 (Expansion), 在这个搜索到的存在未扩展的子节点，加上一个新的子节点，表示没有历史记录参考。这时我们进入第三步。

第三步是仿真 (Simulation)，从上面这个没有试过的着法开始，用一个简单策略比如快速走子策略（Rollout policy）走到终态，得到一个奖励。快速走子策略一般适合选择走子很快但可能不是很精确的策略。因为如果这个策略走得慢，结果虽然会更准确，但由于耗时多了，在单位时间内的模拟次数就少了，所以不一定会棋力更强，有可能会更弱。这也是为什么我们一般只模拟一次，因为如果模拟多次，虽然更准确，但更慢。这个仿真过程称为默认策略。

第四步是回溯 (Backpropagation), 将我们最后得到的奖励回溯加到 MCTS 树结构上。注意除了之前的 MCTS 树要回溯外，新加入的节点也要加上奖励。

MCTS 不需要特定领域的启发式算法，但如果结合启发式算法将可以很好地降低模拟次数，但可能陷入局部最优。

3.2.3 蒙特卡洛树搜索的改进

略。

4 机器学习

什么是机器学习？机器学习试图由一个不完整的数据表格得到完整的表格，或者获得一个描述长度较小的函数 f 来表示这个表格。或者形式化地说，对一个训练数据集 $\mathcal{D}_{train}$，学习器 $learner$ 试图学习一个函数 h，使得对一个输入 x，让 $h(x)$ 更接近真实函数的输出 $y = f(x)$。因此机器学习的对象和结果，是一个函数 h，它表示预定义在 h 中的知识用以压缩函数的描述长度；同时它应该对所有输入都定义有相应的输出。

确定 h 分为两步。首先是模型选择，是采用枚举模型还是利用预定义知识确定线性关系；然后是训练，即确定模型参数或是确定线性表达式的系数。

如何评价 h 的好坏？从**准确性**、**复杂性**和**完备性**来看。准确性指 h 和 f 之间的差异；复杂性指 h 的描述长度，它通常和计算的时空代价相关；完备性指 h 是否对每一个输入都定义了一个相应。

h 的准确性 常见的准确性定义有绝对误差、平方误差和误差计数：

$$e_1(h,f) = \sum_{i=1}^{|X|} |f(\mathbf{x}_i) - h(\mathbf{x}_i)|$$

$$e_2(h,f) = \sum_{i=1}^{|X|} (f(\mathbf{x}_i) - h(\mathbf{x}_i))^2$$

$$e_3(h,f) = \sum_{i=1}^{|X|} 1[f(\mathbf{x}_i) \neq h(\mathbf{x}_i)]$$

其中误差计数适用于离散值情形。然而这种计算方法要求的遍历使得时间耗费太长，而且其值与值域大小有关。因此常常将上述误差除以值域大小得到误差均值，以此来评估准确性。

在实际应用中，选择一部分行不参与训练而作为测试集，用 h 在测试集上的误差值评价 h 的准确性。

h 的复杂性 机器学习中用线性模型来近似 f，即 $y = \mathbf{W} \cdot \mathbf{x}$，其中的 $\cdot$ 表示点积。

现在我们来考虑如何获得线性模型中的 $\mathbf{W}$。

4.1 线性回归

我们现在要学习一个线性模型

$$h(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + b$$

用以近似 f。在此之前，我们先把输入变量 $\mathbf{x}$ 做一个简单的变换

$$\mathbf{x} = (x_1, x_2, \cdots, x_n)^T \to \mathbf{x}' = (x_1, x_2, \cdots, x_n, 1)^T$$

就可以消去常数项 b。或者更一般地，我们用一个特征提取函数 $\phi()$ 对 $\mathbf{x}$ 进行一个变换使得

$$\phi(\mathbf{x}) = (\phi_1(x), \phi_2(x), \cdots, \phi_n(x))$$

在这里我们不考虑初始向量 $\mathbf{x}$ 的维度而通过特征提取函数将它全部转换为 n 维。现在我们的模型变成了

$$h(\mathbf{x}) = \mathbf{W} \cdot \phi(\mathbf{x})$$

从准确性出发，我们期望模型的误差最小。也即对于每行数据，令下面的损失函数最小

$$loss_1(\mathbf{W}, f, \mathbf{x}) = |\mathbf{W} \cdot \phi(\mathbf{x}) - f(\mathbf{x})|$$

$$loss_2(\mathbf{W}, f, \mathbf{x}) = (\mathbf{W} \cdot \phi(\mathbf{x}) - f(\mathbf{x}))^2$$

$$loss_3(\mathbf{W}, f, \mathbf{x}) = 1[\mathbf{W} \cdot \phi(\mathbf{x}) \neq f(\mathbf{x})]$$

其中的 $\mathbf{W} \cdot \phi(\mathbf{x}) - f(\mathbf{x})$ 称为模型的残差，即预测值和真实值的差。请注意，这一先后顺序是不能改变的。

一个自然的想法是将所有行上的损失求和，以此来评估模型在整个数据集上的整体效果。但这个方法的计算时空复杂度太高，不够实用。因此我们退而求其次，计算所有训练集上的损失和

$$TrainLoss(\mathcal{D}_{train}, \mathbf{W}) = \frac{\sum_{(\mathbf{x},y)\in\mathcal{D}_{train}} loss(\mathbf{x}, y, \mathbf{W})}{|\mathcal{D}_{train}|}$$

称为训练误差。训练集上之所以会有损失，是因为模型太简单不足以描述 f，并且训练集上可能有噪声。

类似的，在测试集上也有对应的经验风险

$$TestLoss(\mathcal{D}_{test}, \mathbf{W}) = \frac{\sum_{(\mathbf{x},y)\in\mathcal{D}_{test}} loss(\mathbf{x}, y, \mathbf{W})}{|\mathcal{D}_{test}|}$$

通常以此来作为 h 的评价标准。但是测试集不能用于训练，因此我们只能最小化训练误差。

现在我们的问题变成了，选择一个恰当的损失函数（平方损失综合所有样本的影响，绝对损失抵抗离群点的影响，平方损失函数对应的即为最小二乘法），求解一个 $\mathbf{W}$ 使得训练误差最小。解决这个问题就可以采用前文的随机梯度下降法了。

4.2 线性分类

现在我们考虑分类问题。分类问题对应的 f 的输出域应是一个离散的类别标签集合，但我们的线性分类器 $h(\mathbf{x}) = \mathbf{W} \cdot \phi(\mathbf{x})$ 往往输出的是一个连续实数值。因此在分类问题中，我们认为线性分类

器给出一个评分 $score$，仍需一个离散化过程将得分转换为类别标签。一个最简单的离散器就是符号函数 $sign()$。

以二分类问题为例。分数 $score = \mathbf{W} \cdot \phi(\mathbf{x})$ 描述了模型对预测结果的自信程度，其绝对值越大表示模型越自信；间隔 $margin = \mathbf{W} \cdot \phi(\mathbf{x})y$ 表示预测结果的正确性有多好，负值表示预测错误。这一定义比符号函数保留了更多信息，它可以视作平面外点到平面距离的 $\|\mathbf{W}\|$ 倍。

注意到现在我们的问题已经变成了如何让 $margin$ 值变大。根据上一节的经验，我们应找到一个损失函数来描述误差。要想使用梯度下降法，我们应尽量使得损失函数具有可使用的误差信息（因此我们不使用计数误差或符号函数误差）。使用 logistic 损失函数的分类器故得名 logistic 分类：

$$loss_{logistic}(\mathbf{x}, y, \mathbf{W}) = \log(1 + e^{-\mathbf{W}\cdot\phi(\mathbf{x})y})$$

有时采用 hinge 损失函数：

$$loss_{hinge}(\mathbf{x}, y, \mathbf{W}) = \max\{1 - \mathbf{W} \cdot \phi(\mathbf{x})y, 0\}$$

注意到任何地方 hinge 损失都要大于符号函数损失，这意味着 hinge 损失更抵触误差；而且在 (0, 1) 区间仍有损失存在使得分类器尽可能让分类具有说服力。

4.2.1 SVM

然而 hinge 损失让我们注意到另一个事实，也就是该损失实际描述了样本点到分类平面的距离的某种关系。如果我们继续向几何结构思考，可以定义一种几何间隔，它直接地计算了数据点到分类平面的距离

$$g - margin = \frac{\mathbf{W} \cdot \phi(\mathbf{x})y}{\|\mathbf{W}\|}$$

我们回头看 hinge 损失的要求，它希望代数间隔 $margin$ 大于等于 1，这也就是让几何间隔大于等于 $1/\|\mathbf{W}\|$，即追求所有样本点到分类面的距离超过这一值。

现在我们的问题变成了，找到一个分类面使得任意数据点到分类面的几何距离大于等于 $1/\|\mathbf{W}\|$；而这个界应该越大越好，越大说明分类器的说服力越高。那么如何找到这个最优分类面呢？

我们知道不同分类面的这个距离取决于距离分类面最近的样本点，它们称为支持向量。而随着分类面发生变化，支持向量也会发生改变。所以非支持向量实际是一种约束，它使得模型在试图改变分类面时，面对新的支持向量考虑新的最大间隔距离。

现在可以描述这个优化问题了

$$\min \|\mathbf{W}\| \Leftrightarrow \min \frac{1}{2}\mathbf{W} \cdot \mathbf{W} \text{ s.t. } 1 - (\mathbf{W} \cdot \mathbf{x} + b)y \leq 0, \forall(\mathbf{x}, y) \in \mathcal{D}_{train}$$

该式的含义是：每个训练样本带来一个约束条件；每个约束条件表明所有训练数据都被正确分开、且分开的足够远以至于代数间隔均大于等于 1。这样的优化问题如何求解？

拉格朗日乘子 为了使优化问题的约束条件变成等式，我们对每个约束条件附加一个非负乘法因子 $\alpha_i \geq 0$ 使条件变成

$$[1 - (\mathbf{W} \cdot \mathbf{x}_i + b)y_i]\alpha_i = 0$$

注意到在支持向量上，由于 $1-(\mathbf{W}\cdot\mathbf{x}_i+b)y_i=0$ 因此其乘法因子可以任意取值；在非支持向量上则乘法因子必须取 0。当进行了乘法因子变换，我们就可以将所有约束条件相加得到一个约束等式

$$\sum_{(\mathbf{x}_i,y_i)\in\mathcal{D}_{train}}[1-(\mathbf{W}\cdot\mathbf{x}_i+b)y_i]\alpha_i=0$$

现在我们可以惊人地将原约束问题转化为一个无约束问题

$$\min\mathcal{L}(\mathbf{W},b,\boldsymbol{\alpha})=\frac{1}{2}\mathbf{W}\cdot\mathbf{W}+\sum_{(\mathbf{x}_i,y_i)\in\mathcal{D}_{train}}[1-(\mathbf{W}\cdot\mathbf{x}_i+b)y_i]\alpha_i$$

现在我们知道了，拉格朗日乘子法的核心就在于去掉约束条件将其加入到目标函数中。

然而这个函数仍然有点复杂。我们注意到该优化问题满足 KKT 条件，根据一些数学变换我们又能将问题转变为

$$\max_{\alpha}\sum_i\alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j\mathbf{x}_i\cdot\mathbf{x}_j$$
$$\text{s.t. }\sum_i\alpha_i y_i=0,\forall\alpha_i>0$$

注意到仅非支持向量的乘法因子非 0，因此该函数实际仅涉及非支持向量的内积。那么这个优化问题该如何计算呢？

SMO SMO 算法每次固定 $|X|-2$ 个乘法因子并让剩余的两个自由变化而寻找最佳值；循环迭代调整固定不同的 $|X|-2$ 个乘法因子直至收敛。在固定这些乘法因子的时候，优化问题退化为含等式约束的二元二次函数极值问题，再利用约束等式就可以把可变参数变为一个；在确定可变的两个乘法因子时，可以采用随机次序、固定次序或者用一个启发式函数确定次序。

线性 SVM 的求解的空间需求是和数据集大小有关的线性需求；而时间代价较难估计，一般是数据集大小的若干倍。

然而有一个重大的问题我们一直避而不谈：如果数据集非线性可分该怎么办？

4.3 非线性处理

有时我们会遇到，当数据集线性不可分或者不同类别的支持向量离得太近。这可能是由于数据噪声引起的，如果我们的分类平面过于看重这些噪声，将使得分类的泛化能力变差。

4.3.1 松弛变量法

仍然记得在 SVM 中我们将代数间隔作为约束要求。现在我们希望能够有一个参数使得一些代数间隔过小的噪声数据仍然能够通过约束要求，也就是对代数间隔做一个正的补偿。当然，我们希望在整个训练集上各个训练点的补偿和尽可能小：

$$\min_{\mathbf{W},\boldsymbol{\xi}}\left(\frac{1}{2}\mathbf{W}\cdot\mathbf{W}+C\sum_i\xi_i\right)\text{ s.t. }1-[(\mathbf{W}\cdot\mathbf{x}_i+b)y_i+\xi_i]\le 0,\forall(\mathbf{x}_i,y_i)\in\mathcal{D}_{train}$$

现在我们实际有两类支持向量：补偿因子为 0 的真实支持向量，他们的误差也是 0；补偿因子为正的噪音支持向量，他们的误差即使相应补偿因子的值。注意到当获得最优解时必有 $\xi_i=1-(\mathbf{W}\cdot\mathbf{x}_i+b)y_i$，

也即样本的 hinge 损失，因此我们又可以把优化问题的目标函数写成

$$\min_{\mathbf{W},\boldsymbol{\xi}} \left(\sum_i loss_{hinge} + \frac{1}{2}\mathbf{W}\cdot\mathbf{W} \right)$$

这时最初的优化目标 $\frac{1}{2}\mathbf{W}\cdot\mathbf{W}$ 被称为正则化项，训练集上的 hinge 损失被称为训练误差最小化项。现在的优化目标通过误差项提升模型的准确性，又通过正则项控制模型的复杂性。

4.3.2 非线性可分

有时数据在低维就是线性不可分的。现在我们寄希望于找到一个特征提取函数将数据映射到高维空间，以求在高维空间存在线性可分的分类面。这样的特征提取函数可能把线性不可分的数据集变成线性可分，但必然会增加时空代价。

我们注意到线性 SVM 分类器得到的 $h()$ 函数是

$$h(\mathbf{x}) = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b, \text{ where } (\mathbf{x}_i, y_i) \text{ is a supporting vector}$$

如果我们利用一个特征提取函数作用到所有的数据点上，该分类器变成

$$h(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + b, \text{ where } (\mathbf{x}_i, y_i) \text{ is a supporting vector}$$

也就是对于一条新数据，我们要计算它的特征向量与所有支持向量的特征向量的内积。现在为了避免学习器需要分别计算数据点的特征向量，我们直接定义一个函数，将两个数据点直接映射到它们的特征向量内积上。

核函数 这样的函数我们称之为核函数 $K(\mathbf{x}_i, \mathbf{x}_j)$。此时我们实际上认为这个核函数映射直接表征了高维空间的内积属性，而置低维空间到高维空间的具体变换过程于不顾。核函数从未直接给出到高维空间的映射，它只给出了某个高维特征空间下的内积计算公式。或者，也可以把核函数看作两个数据点的某种相似性度量。

当然，核函数不保证在对应的高维空间中数据是完全线性可分的，而这条性质是否满足同样也很难检验。因此，我们常常结合松弛变量方法使用核函数。一些常见的核函数包括：

$$K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}\cdot\mathbf{y})^2$$

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}\cdot\mathbf{y}$$

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}}$$

4.3.3 K 近邻方法

在线性分类中我们始终认为数据集存在一个单一的分类面，然而这种假设在 K 近邻看来太过严格。K 近邻通过某种方式用一系列分界面组合在一起进行决策。

K 近邻的训练过程就是储存所有样本。在某种距离意义下，K 近邻计算一个预测点最近的 K 个邻居的类别作为预测标签。

K 近邻几乎不需要训练过程，而且可解释性也比较好；但是算法在预测时的复杂度可能很高，而且在训练数据不足时容易过拟合、泛化能力不足。

参数方法与非参数方法 所谓参数，可以认为是决策边界的细节。对于类似 K 近邻这类的非参数方法，可以认为参数的数量随着样本数目的增加而增加。而线性分类器的方法，不论样本多少，权向量 $\mathbf{W}$ 的维数也不会改变，因此参数固定。这种方法称为参数方法。一般对于参数方法，样本越多、参数设置得会更合理。

KNN 算法的扩展 我们假设有这样一种 KNN 算法的扩展。它的训练过程与 KNN 相同，但预测方法是：如果要预测一个新样本，就随机返回一个标签，或直接返回不知道。这一算法在训练集误差意义上是最优的。然而我们在决策边界的复杂性上相比线性分类器复杂太多，这实际造成了对训练数据的过拟合，其泛化能力不够。所谓泛化能力，是指训练集损失和测试集损失相当。这一评价标准提示我们，不能一味追求训练集损失而使模型过于复杂。在线性分类中，我们通过加入正则化项的方式来加强模型的泛化能力。

现在再来思考另一个扩展：它对任何样本返回训练集中出现最多次的类别标签。这个算法可以认为是当 K 取训练集大小时的 KNN 特例。尽管该算法没有复杂的分类边界、也没有过拟合训练数据，但由于其模型过于简单导致它的预测能力太差。

KNN 算法对复杂性的控制是通过随着样本增加分类边界的复杂性增加而实现的。在线性分类器训练中，正则化项，可以认为是认为设定的模型中的预定义知识，被用来控制模型的复杂性。最大描述复杂性的模型就是完整映射表。

KNN 与 SVM 现在我们回头看，SVM 只是用核函数定义了一种新的距离计算公式，压缩了 KNN 需要保留的大量训练样本，只保留了关键数据样本（即支持向量）用于构建距离计算公式。定义合适的距离计算公式是 KNN 设计的关键要素之一，而 SVM 是通过核函数和支持向量解决的。

4.3.4 决策树

决策树是一种树形结构，每个节点是一个简单的线性决策器 $h_i(x) = A_i$，依据属性 Ai 的取值不同划分为不同的类。决策树中任意一个非叶节点有两个特点：拥有一个训练数据集的一个子集 Di，对应数据的一个列/属性（称为分割属性）。决策树中任何一层中所有节点的训练数据子集互不相交，且并集为完整的训练数据集。叶节点不包括任何数据，只有一个标记父节点中数据所属的类别号。

决策树的训练过程就是从根节点出发确定每个非叶节点分割属性的过程。在预测时，沿从树根开始到叶节点的一条路径，依次使用路过节点的线性分类器。

现在的问题是：如何确定每个非叶节点的分割属性，使得分割效果最好？我们通常考虑数据集合的不纯度作为评价标准：所谓纯度越高，是指分割完后的子集异类的数据越少越好。然而如何量化所谓异类数据的多少呢？

熵不纯度 给定节点 N，定义其熵不纯度为

$$i(N) = H(N) = -\sum_i p_i \log p_i$$

其中 p_i 是节点的数据集中第 i 类数据在其中的占比。

在使用熵不纯度确定某个结点的划分属性时，我们对节点的所有属性 A_i，根据 A_i 划分出的数个子节点 N_i 计算各个子节点熵不纯度的加权和，认定原节点的的熵不纯度与加权和的差是这个属性

带来的信息增益：

$$IG(N|A_i) = H(N) - \sum_i \frac{|N_i|}{|N|} H(N_i)$$

这样所有属性中信息增益最大的那个属性作为该节点的分割属性。有时为了避免信息增益倾向选择分类多的属性而使用信息增益率：

$$\frac{IG(N|A_i)}{\sum_i \frac{|N_i|}{|N|} H(N_i)}$$

误差不纯度和基尼指数 给定节点 N，定义其误差不纯度为

$$i(N) = H(N) = 1 - \max_i p_i$$

其中 p_i 是节点的数据集中第 i 类数据在其中的占比。误差不纯度实质就是把所有非占比最大属性全看作误差，求误差的占比。

从误差不纯度引申得到的是基尼指数，它是将来自集合中的某种结果随机应用于集合中某一数据项的预期误差率：

$$i(N) = H(N) = \sum_{i \neq j} p_i p_j = 1 - \sum_i p_i^2$$

当使用基尼指数确定划分属性是，本质还是找到一个属性，使得在它分割数据集后剩余的基尼指数最小。

要应用基尼指数，要不停地二分节点 N 为两个子节点，对于任意一个多值属性需要一个“值域二分”过程，即把值域分成两个子集，以此来划分数据集。在各种值域二分后的结果上都要计算最优的基尼指数。

不纯度度量的选择 信息增益偏向选择值多的属性；信息增益率解决了 ID3/信息增益偏向值多属性的弊端，但是会造成不平衡分割，偏向于分割为大小相差大的各个子集；基尼系数偏向于值多的属性，标签类多的时候存在困难，偏向于子集大小大致相等、纯度大致相等的分割。但不纯度度量的选择往往不是决策树设计时最重要的因素，大量的实践表明，不同的不纯度，结果基本相当。

决策树的过拟合问题 决策树有时会出现过拟合问题。有时决策树的分支太多，有些靠近叶的分支可能因为噪声，异常点等造成过拟合，导致泛化能力太差。可以通过先剪枝和后剪枝解决。

先剪枝是指，设置信息增益的阈值，如果某个节点无论如何分割都无法获得超过阈值的信息增益，那么分割就停止，在此节点上保留占优类或者类分布/比例。

后剪枝是指，生成决策树之后，再去掉某些分支。比如，训练时保留一部分训练数据（剪枝集）不用，用来把一个完全训练好的决策树剪枝。

决策树的改进 在面对连续性数据时，可以采用离散化区间进行离散化处理。面对缺失值，可以采用中心趋势度量给每一个取值赋予一个取值概率。面对无法一次性在内存中放下所有训练数据的情况，可以采用 RainForest 算法载入一次数据，完成对所有候选分割属性的不纯度计算；或者随机采样获得可以全部放入内存的训练数据子集、在获得的子集上构造决策树，再重复采样、构造多棵决策树，然后用集成学习的方法综合多棵决策树的结果获得最终判决。

决策树本质上是一个综合多个线性判决的方法。它思想简单、可解释性好，而且是完备的。一般用决策树的节点数目度量其复杂性。

4.3.5 神经网络

我们知道特征提取函数可以用来实现对某些非线性情况的处理。我们接下来要面对的问题是如何寻找通用的特征提取函数的构造方法。

对特征提取函数的评价 我们知道在线性分类中，分数 $score$ 是由权向量 $\mathbf{W}$ 和特征提取函数共同确定的。我们已经有评价 $h()$ 的方法（即各种损失函数），现在我们试图屏蔽不同权向量的影响而分析特征提取函数。为了达到这个目的，我们考虑在所有可能的权向量中，使得其与我们要考察的特征提取函数构成的线性分类器效果最好的一个，以此代表该特征提取函数的性能。也即考虑假设类

$$\mathcal{H}_\phi = \{h_{\mathbf{W}} : \mathbf{W} \in \mathbb{R}^d\}$$

描述的所有权向量和 $\phi()$ 构成的线性分类集合当中性能最好的一个。显然，不同的 $\phi()$ 将描述不同的假设类空间，他们各自都是所有可能假设的子集。此前的学习算法就是在给定某个 $\phi()$ 的假设类中搜索最优的权向量 $\mathbf{W}$。这就让我们想到，如果一个 $\phi()$ 确定的假设类空间本身就很糟糕，那么学习算法无论如何也找不到一个好的假设 h。相反，我们不关心 $\phi()$ 构成的假设类中有多少不好的假设，我们只关心该假设类中表现最好的假设性能如何。在先前提到的学习过程中，正则化的目的就是为了控制假设类中 $\mathbf{W}$ 取值的范围，减小假设类的大小，优化学习效率。

增加一个特征函数的维度，就是增加表达能力，同时扩大了假设类的范围、增加了搜索最优权向量的难度。

如何选择特征是困难的。特征和输出之间的关系可能是非单调的，可能是非线性的，而且特征之间有可能是相关的。我们试图找到一系列基本构造因子来提取特征，但如何确定这些基本构造因子呢？我们一般用输入的单项式构成的向量作为特征。但这些基本构造因子的最高次数如何确定？如果次数过高会导致过拟合，过低又无法处理复杂的非线性决策边界。

非线性问题的线性化处理 在神经网络，我们把一个非线性问题用多个线性子问题近似。我们仍然记得曾使用过 logistic 函数代替没有梯度信息的符号函数。现在，我们管这类拟合符号函数的非线性映射称作激活函数，它们均是为了增加特征提取函数的非线性处理能力设计的。常见的三个激活函数是 Sigmoid, Tanh 和 ReLu。

我们把激活函数应用到线性模型中，现在我们得到的 h 函数变成

$$h(\mathbf{x}) = \sigma(score) = \sigma(\mathbf{W} \cdot \phi(\mathbf{x}))$$

如果我们把激活函数当成一个可以任意组合使用的元素，我们就得到了神经网络。

神经网络 神经网络的网络结构由输入层、隐层和输出层构成。每一层的输出按某种要学习的权值传递给下一层。那么如何学习这个权值呢？

我们知道每个函数对其自变量都有导数。对任何一种损失函数我们都可以通过链式法则求其自变量的偏导，利用这个偏导信息进行梯度下降优化。因此我们可以在函数的导数关系树上，先将数据

代入叶子节点向上传递获得输出，此过程称为前向过程；然后将误差值从根节点向下反馈，此过程即是后向传播。

深度学习 神经网络的一个问题是，在做预测/分类之前，神经网络的训练代价大，收敛速度慢（获得收敛的权值）；当隐层的层数和节点的数据增加时，该问题尤为突出；以前，通常应用中都只用一个隐层。深度神经网络设计了很多隐层，但并不用 BP 算法训练权值。

在深度学习中，每个隐层独立训练，训练一个三层 BP 网络。对任意一个三层 BP 网络，输入层是前一个隐层的输出（或原始输入数据），输出层与输入层完全一样。训练该三层 BP 网络并丢弃输出层及其关联边。重复上述过程，进行多次，得到多个连续的隐层；这个循环过程构建了一系列的自编码器。最后，利用“监督信号”训练最后一个输出层连边的权值。

现实中，训练数据/类标签通常难以获得；有大量的数据没有标签，该方法的特征提取过程可以充分利用大量没添加标签的数据。虽然每个三层 BP 网络的训练还是比较费时间，相对于多层的 BP，时间代价已经降低了。

在深度学习中，任何一个隐层，实现了对原始输入数据的一种特征提取，找到了数据的“规律”，实现了一种“数据压缩”。一般认为：随着层次的增加，后续的特征越来越抽象。但是这种抽象是否真正有利于分类仍需要调试考察。

总的来说，深度学习带来两个改变。观点的改变：神经网络用于特征提取，隐层是一个特征提取函数；权值学习方法的改变：自编码器，实现隐层代表的特征提取函数。

CNN CNN 将隐层的每个节点视为一个神经元，能感知局部外界输入信息（和输入层相连），处理一部分输入层的信息。将同一隐层的每个节点都看作是一样的，假设其对输入的加权方式和处理方式都是同样的，即功能完全一样的神经元，这样的一个神经元称作一个卷积核。然后我们训练不同的卷积核，对应一类不同的图像特征 (或者叫特征提取方法)；每个卷积核作用在输入图像上，就得到一个特征映射 (feature map)；特征映射可视为图像的一种变换。取多个不同的卷积核，提取多种特征，多角度描述图像；每个特征形成一个“平面图像”。参数的个数和卷积核的结构大小，卷积核的个数相关，与输入层节点数、隐层节点数无直接关系。卷积层的层数越高，提取到的特征就越全局化。

有时会对特征进行聚合、降维，这一过程称为池化，以达到减少运算量的目的。

RNN 与 LSTM 略。

5 贝叶斯网络

概率质量函数是离散随机变量在各特定取值上的概率。给定随机变量，其概率质量函数可以用一个完整表格表示。我们现在期望简化表示这个表格。可以使用方差、期望、均值、信息熵等方法来描述这个概率分布。

现在我们考虑随机向量而非随机变量，即多个相关或不相关的随机变量构成的向量。其联合概率质量/密度函数是指随机向量的任何一个取值（每个随机分量取一个值构成一个值向量）都对应一个概率，离散时称联合概率质量函数，连续时称联合概率密度函数。联合概率质量函数的表格形式将

有 $\prod_i |X_i|$ 行，显然是太大了。我们该如何学习这个表格？另外，我们可以利用这个表格进行的条件概率推理，在学习模型中又如何实现？

贝叶斯网络通过网络结构、若干边缘分布和若干条件概率质量函数得到完整的联合概率质量函数。对任意联合概率（也就是联合概率质量函数的任何一行），我们都可以先用网络结构得到联合概率的计算表达式，然后根据边缘分布和条件概率质量函数进行计算；因此，我们从 Bayes 网络可以恢复出整个联合概率质量函数。

根据这样的思想，出现了朴素贝叶斯模型和隐马尔科夫模型。

5.1 贝叶斯网络的学习

我们假定已知数据集，试图学习贝叶斯网络的网络结构和各种概率函数。

5.1.1 网络结构的确定

确定网络结构一般有三类方法。依靠专家建模，手工给出网络结构；从数据集中利用卡方/互信息等做相关性测试，或者基于搜索-评分的方法自动学习网络结构；混合前两者的方法。

5.1.2 概率函数的确定

在给定数据集合网络结构的情况下，利用蒙特卡洛逼近。将数据集视为“粒子”集，然后计算网络结构需要的各种边缘/条件概率质量函数/表。即用频度近似概率值。在多变量网络结构中由于已知数据的稀缺性导致估计的条件概率质量函数中出现了大量的 0，这是非常不合理/不准确地对概率值的估计。此时若在统计每个值出现次数时，计数器的初始值设置为非零，这就是所谓的“拉普拉斯平滑”。

5.2 隐变量的学习

在诸如隐马尔科夫模型、聚类模型问题中，数据集中有隐变量存在，即没有任何的观测到的分量数据。此时将条件概率质量函数视为参数/变量，在这些参数/变量的取值中找一组最好的参数，使得出现观测到输出变量（已知数据集）的概率最大。

解决这一问题的 EM 算法是这样做的：隐变量 H 和条件概率质量函数都是未知变量，在算法依次/轮流更新；随机设置条件概率质量函数的初值, 不妨设为 cpt_0；

E-step 对所有 $H=h$，计算 $p(E=e|h,cpt_{i-1})$，给定第 $i-1$ 次迭代获得的各个“小表”cpt_{i-1}，我们可以计算该条件概率值。比较不同 h 的概率值 $p(E=e|h,cpt_i)$，取最大概率值对应的 $h*$ 来构建随机向量 (H,E) 的取值 $(h*,e)$，这是概率中的“极大似然估计”，也就是说隐藏的嫌疑犯 H 就是最有可能造成观测到的证据 $E=e$ 出现的人。

M-step 获得了样本数据集 (H,E)，用它来计算/更新 Bayes 网络的各种边缘/条件概率质量函数，即 cpt_i。

以上两步迭代循环直至满足收敛条件。EM 算法的想法是，假设我们想估计知道 A 和 B 两个参数，在开始状态下二者都是未知的，但如果知道了 A 的信息就可以得到 B 的信息，反过来知道了 B 也就得到了 A。可以考虑首先赋予 A 某种初值，以此得到 B 的估计值，然后从 B 的当前值出发，重新估计 A 的取值，这个过程一直持续到收敛为止。

5.3 Bayes 网络的概率推理

假定我们已知 Bayes 网络，想要考虑在观测现象 E 时某个原因 Q 的条件概率。首先在各“小表”中选择 $E = e$ 的行留下（消去 E）；所有的“小表”通过“连接”操作，组合成一张“大表”，制作一个新的概率值列，即联合概率质量函数；再将不关心的因素 M 去掉（消去 M），即有多行满足 $Q = q$，这些行的 M 值不一样，若这些行的概率值之和为 p，则产生新的行 (q, p)；实际上我们并没有组合出完整的联合概率质量函数，因此，依次消去 M 中的某个分量时，通常只需要访问一部分“小表”即可，减少了时间代价。最终得到新表 (Q, P) 即是所求。

这一概率推理的空间复杂度，当网络中某个节点的（入）度为 k 时，会产生一个 $k+1$ 列的条件概率“小表”，表的行数是 k 的指数函数，因此 Bayes 网络描述联合概率质量函数并非完美解决了存储代价问题。时间复杂度方面，遍历所有的“小表”是推理的基本操作之一，因此推理的时间复杂度和各个表的大小相关；从网络结构的角度看，单连通网络（任何两个节点之间的路径只有一条）通常能线性时间解决；非单连通网络需要指数时间复杂度。

5.4 蒙特卡洛逼近

在 Bayes 网络中由于网络太复杂导致精确计算一个概率值常常不可能。当计算概率值的时间代价是指数的复杂度时，采用计算概率近似值的方法。蒙特卡洛逼近是说，利用 Bayes 网络，产生各个随机分量，最终获得一个完整的“粒子”。重复这个采样过程，得到大量粒子的集合。统计/计数粒子集合，用各种比例代表各种对应的概率值。

抽样程序可以使用马尔可夫蒙特卡罗方法从任意概率质量函数 $p(x)$ 中抽样。利用马尔可夫链，每个样本是随机过程的一个状态，大量的样本/状态的分布服从随机过程的稳态分布 $p(x)$，即我们要抽样的分布。从当前状态（粒子），以一定的概率转移到下一个状态（粒子），并不一定每次都转移成功，有一个转移接受率。以此进行抽样。

也可以使用 Gibbs 抽样，下一个粒子只改变当前粒子的一个分量的值，改变的方法为对应的条件概率用一次随机抽样。