第二章

2.1 命题演算Propositional Calculus

命题演算的符号包括：

- 命题符号：P，Q，R，S...大写的字母等
- 真值符号：true，false
- 连接词：$\wedge, \vee, \neg, \rightarrow, \equiv$等

命题符号表示命题对世界的陈述，可能为真可能为假。

- 每个命题符号和真值符号都是一条语句

> 合法的语句被称为合式公式
>
> 合取项：$P \wedge Q$
> 析取项：$P \vee Q$
> 非：$\neg Q$
> 蕴含：$P \rightarrow Q$(P是前提,Q是结论)
> 等价：$P \wedge Q \equiv R$

等价公式

换质换位定律	$(P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$
德摩根定律	$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$；$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$
交换律	$(P \wedge Q) \equiv (Q \wedge P)$；$(P \vee Q) \equiv (Q \vee P)$
结合律	$((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$；$((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$
分配律	$P \wedge (Q \vee R) \equiv (P \vee Q) \wedge (P \vee R)$；$P \vee (Q \wedge R) \equiv (P \wedge Q) \vee (P \wedge R)$

2.2 谓词演算Predicate Calculus

谓词演算符号：用以构成谓词演算符号的字符表：

- 真值符号：true false
- 常量符号constant：小写字母开头的符号表达式
- 变量符号variable：大写字母开头的符号表达式
- 函数符号function：小写字母开头的符号表达式，且具有附带的元数用以指出被映射到值域的元素的定义域的元素个数

原子语句 （原子表达式、原子或命题）：谓词演算语言中的最基本单位，式元数为n的谓词，后用括号为包围且逗号分隔的n个参数项

谓词演算语句（=连结符+原子语句）：每一条原子语句都是一条谓词演算语句

2.3.1 推理规则

- 满足：在解释下一个特定的变量赋值使表达式为真，则解释满足表达式，只允许一种解释
- 模型：对所有的变量赋值来说都满足表达式，则说解释是表达式的模型
- 有效：对所有可能的解释来说表达式的值都是真，则表达式有效
- 不一致：表达式集合不可满足

逻辑派生：满足表达式集合S的所有解释也满足另一个表达式X，则X逻辑派生自S

可靠：表达式集合S产生的所有表达式也派生自表达式集合S，则推理规则可靠

完备：表达式集合S，一个规则可以推理出逻辑派生自S的所有表达式，则推理规则完备

- 拒式假言推理
- 取式假言推理
- 与消除
- 与引入
- 全称例化

2.3.2 合一

合一：判断什么样的替换可以产生的两个谓词演算表达式的算法

替换：f(X)-->f(a)就是把X替换为a

> 斯科伦化skolemization：把每个存在量化变量用一个函数替换

> 常量不可被替换

复合：两个替换集合进行复核，其中一个集合的元素应用到另一个集合上，再把结果加到本身

最一般合一most general unifier：如果s是表达式E的任意一个合一式，g式这个表达式最一般合一式，则应用到E的s，存在另一个合一式s'使Es=Egs'，期中Es和Egs'是应用到表达式E的合一式的复合

合一树图示

第三章

3.1 图搜索结构与技术

状态图搜索

- 图是结点(状态)的集合和连接这些节点的弧的集合。
- 带标签的图：用一个或多个标签附加在每个结点上
- 弧也可以有标签：
  - 说明弧代表的是一种命名关系
  - 把权值附加在弧上
- 有向图：弧具有方向性，具有两个方向的弧可以带两个箭头
- 路径：通过多个相继的弧把一系列结点连接起来，可以用有序链表表示
- 有根图：具有一个特殊的结点，这个结点到任意一个结点都有一条路径
- 树：两个结点之间至多有一条路径的图

有限状态图：

状态转换表：

与或树

博弈树

3.2 搜索策略

数据驱动

也称正向追索，问题求解器从问题的给定事实和改变状态的合法移动或规则的集合入手。把规则应用到事实产生新的事实，以此下推，直到产生满足目标条件的一条路径。

> 从问题的事实入手，应用规则或合法移动产生通往目标的新的事实

适合问题：

- 问题的初始陈述给出了所有或大部分数据
- 潜在目标的数量非常庞大，但使用特定问题实例的给定信息和事实的方式很有限

- 难以组成目标或假设

目标驱动

从想要求解的目标入手，先分析怎样使用规划或合法移动来产生这个目标并求出想要应用这些规则或移动必须具有的条件。条件则成为新的要搜索的目标或者称为子目标，然后反向追溯相继的子目标，直到返回到问题中的事实。

> 把焦点集中再目标上，寻找可以产生这个目标的规则，并环环相扣地反向追索相继的规则和子目标直到到达问题的给定事实

适合问题：

- 目标或假设再问题陈述中已经给出，或很容易形式化
- 与问题事实匹配的规则数量非常多，产生的结论或目标越来越多
- 问题数据不是给定的，要由问题求解器来获取

宽度优先搜索

一层一层地搜索空间，只有给定层上不存在要检索的状态时才转移到下一层

尝试用open和closed跟踪穿越空间的过程：

- open视为队列，先入先出，后继状态从右端插入

> 可以找到从起始结点到目标结点的最短路径

深度优先搜索

尽可能的向搜索空间的更深层前进，只有找不到状态的后代才会考虑他的兄弟

- open被视作堆栈，先入后出，后继状态从左端插入

迭代加深的深度优先

3.3 实例应用

逻辑系统的状态空间

八数码

一字棋

TSP旅行商问题

第四章

采用启发式的情况：

- 在问题陈述或现有数据中存在固有的模糊性，所以问题可能没有精确解
- 问题可能有精确解，但是找到这个解的运算开销可能让人难以接受

启发式搜索-->启发度量+使用度量进行空间搜索的算法

4.2 最好优先算法

具有更好的灵活性，使用优先级队列。

> 用列表维护状态：
> open记录搜索的当前搜索带
> closed列表记录已经访问过的状态
> 新增：对open中的状态进行排序，依据是状态与目标的接近程度的某种启发性估计
>
> 把open维护成为有序列表，因此常称为优先级队列

启发估价函数：F(n)=g(n)+h(n)

- g(n)是从任意状态n到起始状态的实际路径长度
- h(n)是对状态n到目标的距离的启发性估计

4.3.1 可采纳性admissibility

只要存在到达一个目标的最短路径，启发就可以找到这条最短路径，则启发可采纳

A算法：

> 考虑估价函数 f(n)=g(n)+h(n)

> n是搜索中遇到的任意状态
> g(n)是从起始状态到n的代价
> h(n)是对n到目标状态代价的启发式设计

A^*算法：

> A算法中使用的估价函数满足 ::: $h(n) \leq n$到目标的最短路径代价

特点：所有的A^*算法都是可采纳的

4.3.2 单调性monotonicity

是否存在“局部可采纳”的启发，即是否总是可以找到到达搜索中遇到的每个状态的最短路径

> 启发函数h单调的条件：
>
> - 对于所有状态n_i和n_j,其中n_j是n_i的后继
> - 目标状态的启发值为零,即h(Goal)=0

启发的可采纳性是否包含了单调性?

4.3.3 信息度Informedness

> 对于两个A_*启发h_1和h_2，如果对搜索空间中的所有状态n都满足$h_1(n) \leq h_2(n)$，就说明$h_2比h_1具有更高的信息都

4.4.1 极小极大过程

基于假定(假定对手具有相同的关于状态空间的知识，也应用这一知识以一致的方式赢得比赛)

极大极小搜索根据规则沿连续的父节点向上传播值：

- 父节点状态是MAX结点，则把孩子的最大值赋值给他
- 父节点状态是MIN结点，把孩子的最小值赋值给他

4.4.2 n层预判-固定层深的极小极大过程

> 由于子图的节点不是最终状态，所以需要根据某个启发评估函数给每个节点赋值，向上传播的值并不表示是否可以胜利，只代表从当前起始节点通过n次移动可以达到的最佳状态的启发值

需要对搜索空间进行两遍分析

1. 向下降到预判层并在那里应用启发评估
2. 沿树向上传播评估值

4.4.3 $\alpha-\beta$过程

提高双人博弈的搜索效率

> 不搜索预判深度的整个空间，而是以深度优先的方式前进，产生两个值为α和β，α与MAX结点的关联从不减小，β与MIN结点相关联从不增大
>
> α是下界，β是上界
>
> 即：
>
> - MAX结点下，发现一个α值大于或等于任意一个MIN祖先结点的β值，就终止搜索
> - MIN结点下，发现一个β值小于或等于任意一个MAN祖先结点的α值，就终止搜索

第六章

递归Recursion Search

> 递归步骤：过程调用自身完成一系列动作
>
> 使递归过程从无限递归中停止的条件

模式驱动Module Driver

产生式系统production system

> 为控制问题求解过程提供了一种面向模式的手段

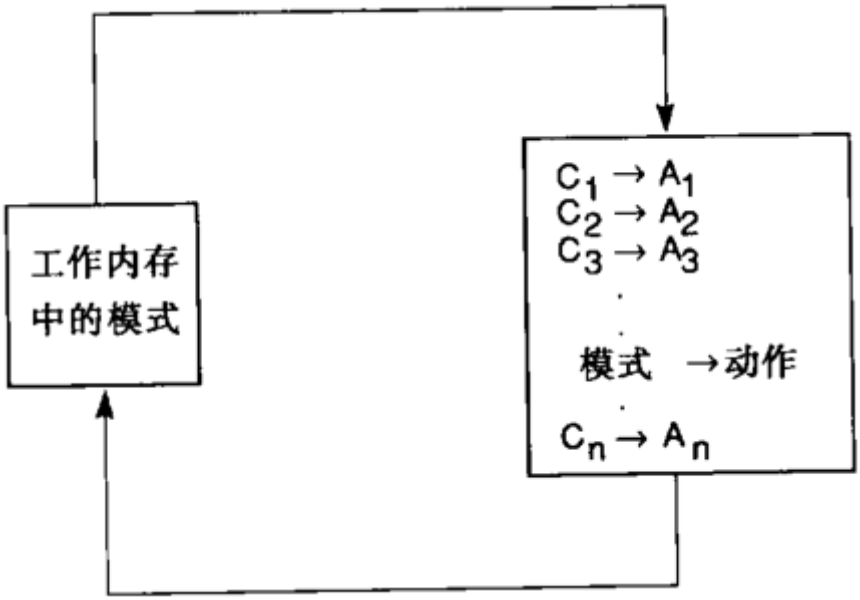
产生式系统的构成：

- 产生式规则集合the set of production rules：产生式就是一个“条件-动作”对，定义了求解问题的一个知识块。
  - **条件部分**是一种模式，用来确定何时可以把这个规则应用到问题实例上
  - **动作部分**：定义相关联的求解问题步骤
- 工作内存working memory：包含了推理过程中对世界当前状态的描述
- 识别-动作循环the recognize-act cycle：
  - 先使用问题描述初始化工作内存，并把求解问题的当前状态维护为工作内存中的模式集合；
  - 再把这些模式与产生式规则的条件进行匹配；
  - 条件和工作内存中的模式相匹配的规则形成一个子集，称为**冲突集合**

- 冲突集合中选择一个产生式(称为使能的)，并激发它（激发一个规则就是执行他的动作）；激发后对修改的工作内存重复这个控制循环，直到内存内容没有和规则匹配的内容
- 冲突消解conflict resolution：从冲突集合中选取一个要激发的规则

> 纯粹的产生式模型不包含任何从搜索死端恢复的机制，只是不断执行识别-动作循环直到不再有任何已经使能的产生式便停止

产生式系统示意图：



产生式集合:

1. ba → ab
2. ca → ac
3. cb → bc

产生式执行过程：

迭代	工作内存	冲突集合	激发的规则
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	Ø	Halt

图 6-2　一个简单产生式系统的执行过程

Production Rules：if...then... P->Q

搜索控制

数据驱动搜索

从问题的描述开始，从数据中推理出新知识

> 首先都世界的当前描述应用推理规则，在博弈中选取合法移动
> 或应用其他的“状态产生”操作，把得到的结果加入到问题的描述中

目标驱动控制搜索

规则结构控制搜索

冲突消解控制搜索

知识表示

语义网处理系统

一个与一阶谓词演算具有同样能力的定理证明器

概念依赖关系

脚本

脚本：一种结构化的表示，用来描述特定上下文中固定不变的事件序列

组成：

- 进入条件：要调用这个脚本必须满足的世界描述
- 结果：脚本一旦终止就成立的事实
- 道具：支持脚本内容的东西
- 角色任务：各个参与者所执行的动作
- 场次：把脚本分解成一系列场次

框架

显示组织的数据结构中捕获问题域中隐含的信息连接

> 可以把框架看作数据结构，包含很对典型实体相关的信息

包含信息：

- 框架标识信息
- 与其他框架之间的关系
- 特征描述
- 描述结构用法的过程信息
- 默认信息
- 新实例信息

概念图

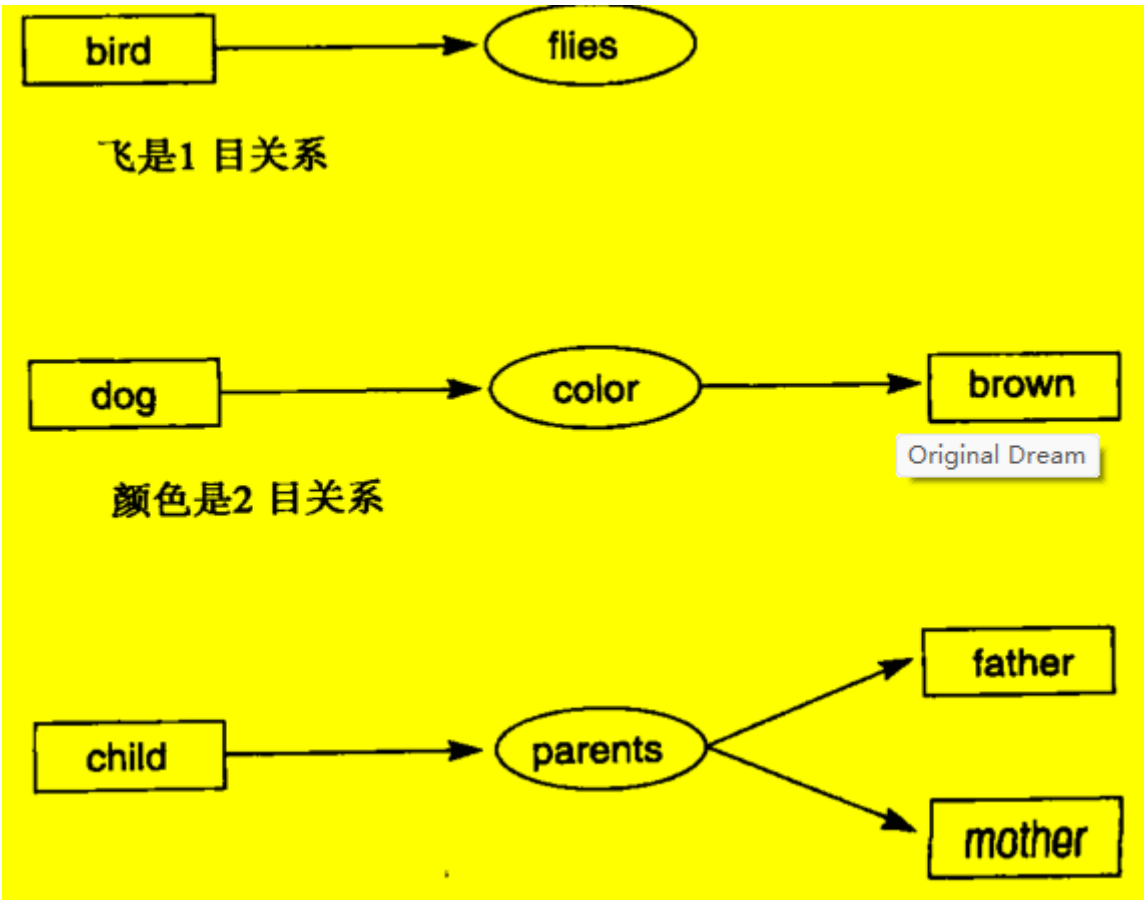
> - 概念图是一种有限、连接、二部图。图的结点要么是概念，要么是概念关系
> - 概念图用概念关系结点表示概念之间的关系
> - 概念结点要么表示具体对象，要么表示抽象对象

概念图结点表示：

- 主语---agent
- 术语宾语--object
- 地点--location
- 时间--time
- 对象--object

概念关系

一目关系、二目关系、三目关系



类型、个体和名字

类型标签：指出这个节点所表示的个体的类型或类

个体标签：标出每个概念所代表的个体

名字：特定个体

- 类型和个体用“：”
- 个体和名字用“#数字”
- 未指定个体用“*”

类型层次

一种用“≤”表示的部分有序关系，若t≤s，则称t是s的子类型，s是t的超类型

> 一个类型可以由一个或多个超类型，也可由一个或多个子类型

通用类型：所有类型的超累型“T”

荒谬类型：所有类型的子类型“倒T”

运算关系

泛化-复制：

特化-限定：

特化-联合：

泛化-简化：

命题结点

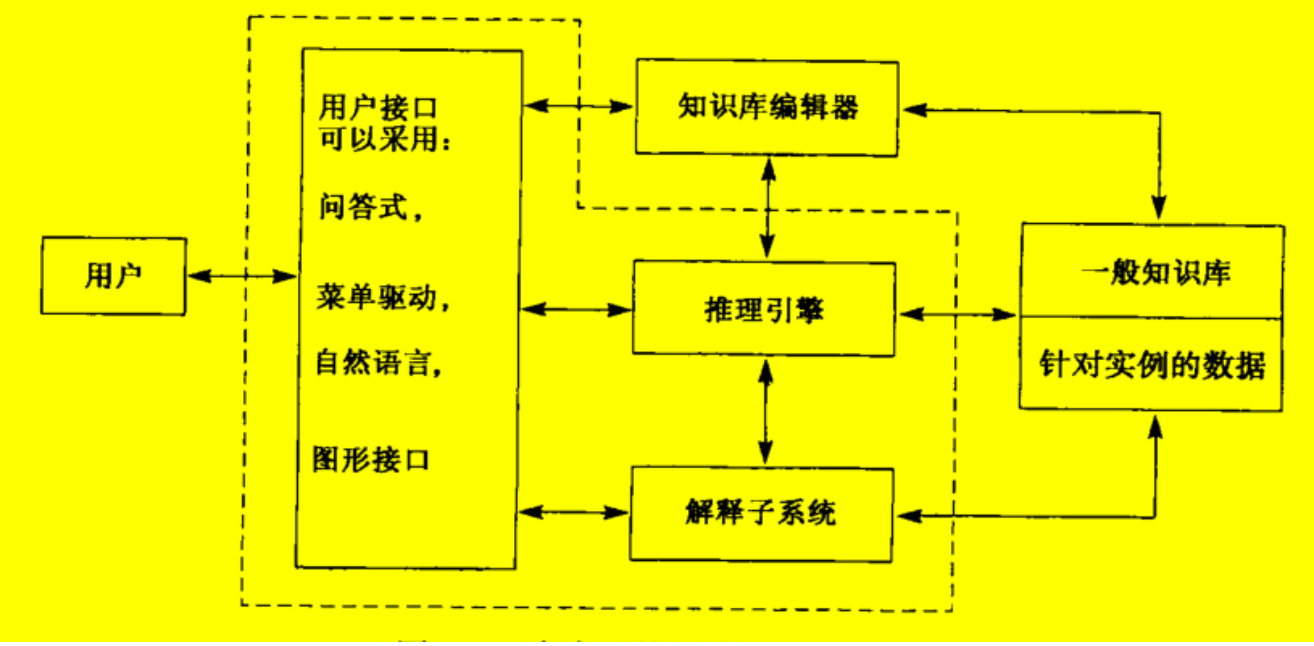
概念图和逻辑

概念图表示谓词演算表达式：

1. 为g中的n个一般概念中每一个分配一个唯一变量
2. 为每一个个体概念分配一个唯一常量，表示概念指向的名字或标志
3. 用一元谓词表示概念结点
4. n元概念关系表示为n元谓词
5. 第3，4步的所有原子语句进行合取，得到表达式主体

专家系统强方法

专家系统典型体系结构



推理引擎：知识库的解释器，执行“识别-动作”循环

> 知识库和推理引擎需要分开：
>
> 1. 分离使我们更自然地表示知识
> 2. 知识库和程序的底层控制结构时分离的
> 3. 分离可以使修改部分知识库不会对其他部分产生影响
> 4. 使一套控制和接口软件可以用于多个系统

知识库编辑器：帮助程序员定位和修改程序的执行错误；帮助增加新的知识、维护和纠正规则语法、对更新后的知识库进行一致性检查

知识库：专家系统的核心，包含一般知识和针对实例的信息

用户接口：用户与系统之间交互的通道，简化通信过程并隐藏复杂细节，人机接口

解释子系统：向用户对推理进行解释，包括系统如何做出结论、系统为什么需要特定数据以及系统动作的有价值的辅导性解释或深层的理论依据

建立专家系统关键人员

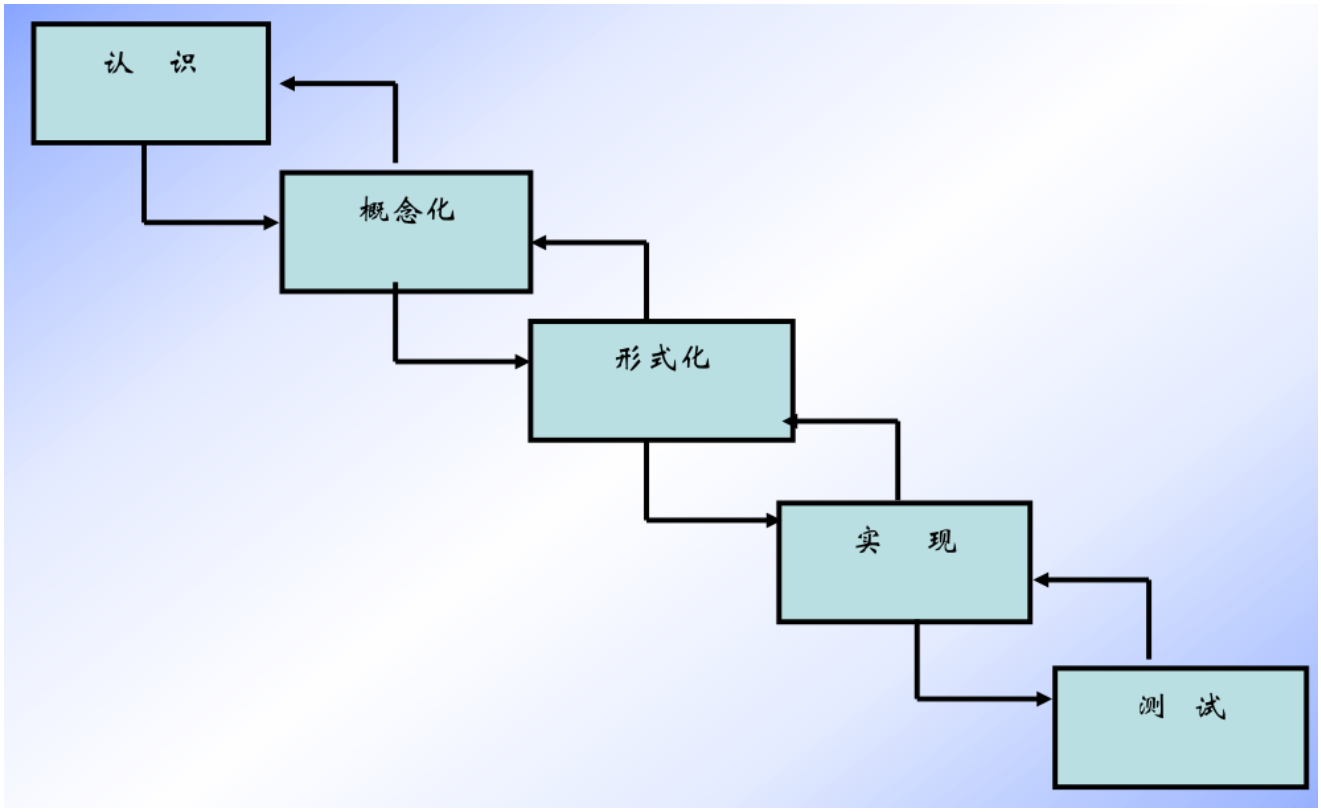
知识工程师

领域专家

最终用户

开发循环

瀑布模型开发过程



知识获取

基于规则推理

基于规则例如“如果..则..”

目标驱动推理反向使用规则

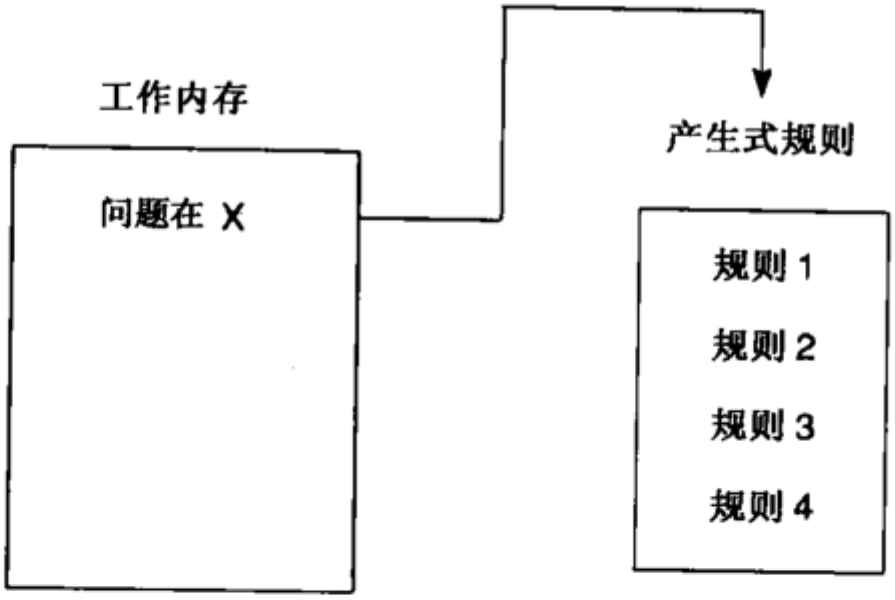


图 8-5　用来诊断汽车故障的产生式系统的初始状态

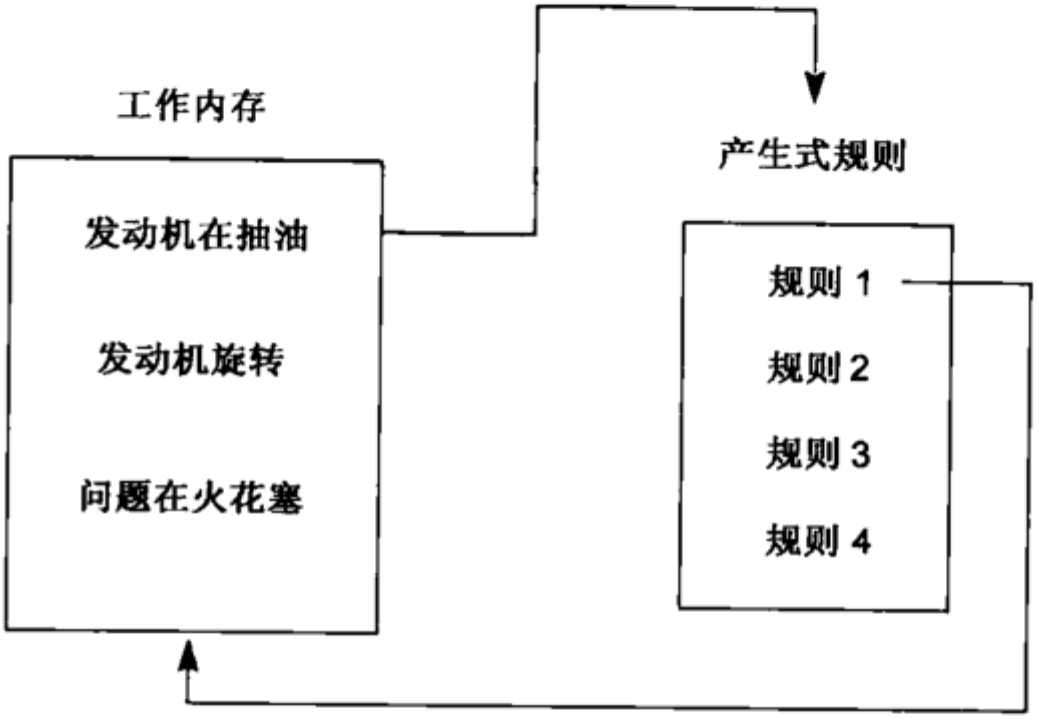
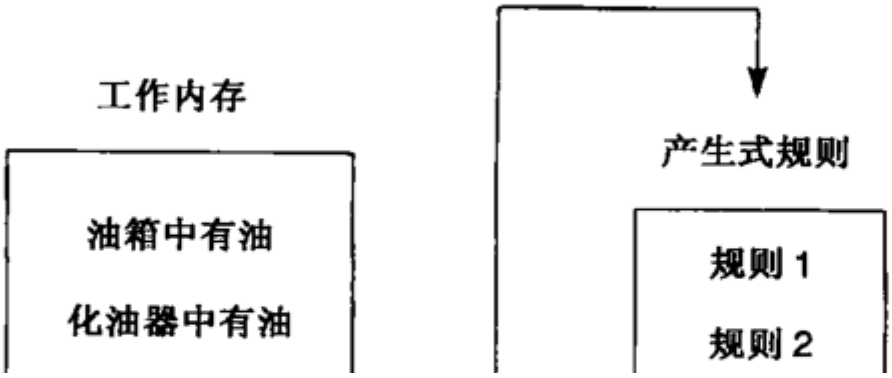


图 8-6　图 8-5 中的产生式系统在规则 1 被激发后的状态



搜索与或图：

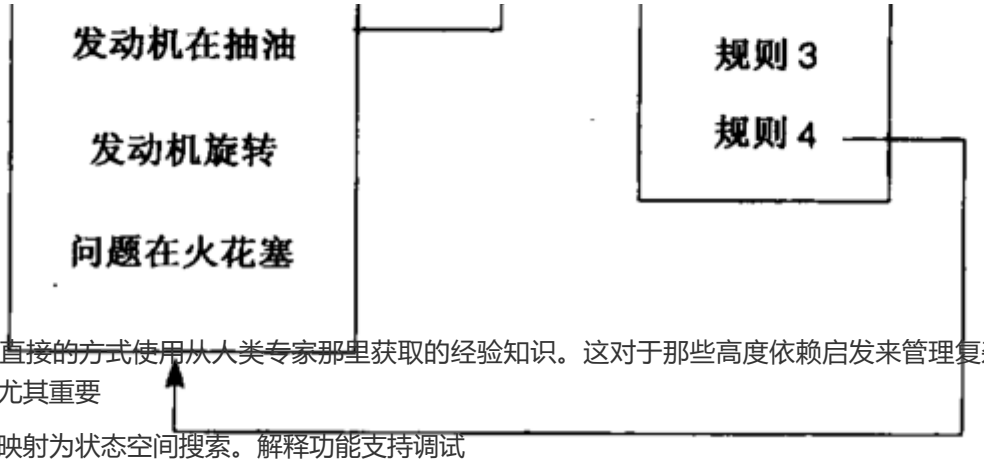


图 8-7 规则 4 被激发后的状态

解释性：

透明性：

优点：

- 能够以非常直接的方式使用从人类专家那里获取的经验知识。这对于那些高度依赖启发来管理复杂性和/或残缺信息的领域尤其重要
- 可以把规则映射为状态空间搜索。解释功能支持调试
- 知识和控制的分离简化了专家系统的开发过程，可以循环地组织开发过程，即不断重复获取、实现和检验规则这一过程。
- 在有限的领域内可以得到很好的性能。因为智能问题求解所需的知识数量极其庞大，所以专家系统的应用仅限于很窄的一些领域。不过，已经证明在很多领域中都已经取得了非常好的效果(只要设计得当)
- 解释机制很好。尽管基本的基于规则框架支持灵活的针对问题的解释，但是必须注意到这些解释的最终质量依赖于规则的结构和内容。数据驱动系统和目标驱动系统中的解释机制差异很大

缺点：

- 很多时候从人类专家那里获取的规则具有很强的启发性，没有捕获到领域中的功能性一基于模型的知识
- 启发性规则往往比较“脆弱”，不能处理有残缺的信息或意外的数据值
- 规则脆弱性的另一表现就是在领域知识的边缘附近其性能迅速退化。和人类不同，基于规则系统通常不能在遇到新奇问题时回退到基本的推理原则
- 解释功能仅是描述层的，无法做出理论性解释。这是由以下事实所导致的:启发式规则的强大性主要来源于把问题症状和解答直接联系起来，并不需要更深层的推理
- 知识往往具有很强的任务依赖性。形式化的领域知识往往就是直接针对当前应用的。日前的知识表示语言还无法接近人类推理的灵活性。

基于案例推理

优点

- 直接编码历史知识的能力。在很多领域中，可以从存在的案例历史、修理记录或其他资料中获取案例，这缓解了从人类专家那里获取大量知识的压力
- 使快捷推理成为可能。如果可以找到恰当的案例，那么可以非常快地求解问题，比根据规则或模型产生解要快得多
- 它使系统可以避免过去的错误但使用过去的成功之处。基于案例推理提供了一个很好的学习模型，不仅有理论研究价值，而且有足够的实践价值用来求解复杂的问题
- 不需要对领域知识进行广泛的分析。在基于规则的系统中，知识工程师必须预料到规则的相互作用，与此不同，CBR可以在知识获取中采用简单的相加模型。这需要恰当的案例表示、有效的索引检索以及合理的案例适应策略
- 恰当的索引策略增加了系统的洞察力和问题求解能力。区别目标问题的差异并选择恰当案例的能力是基于案例推理程序的关键，很多时候索引算法可以自动产生这种效果

缺点

- 很多时候案例不能包含更深层的领域知识。这影响了解释功能，而且在很多情况中可能造成错误应用案例，导致错误的或质量很差的建议
- 庞大的案例库可能受到存储/计算平衡等问题的困扰
- 难以确定好的索引和匹配案例标准。目前，单词检索和相似匹配算法必须经过仔细的手工处理;这大大抵消了 CBR 所具有的知识获取优点

基于模型推理

优点

- 在问题求解中使用功能性/结构性领域知识的能力。这提高了推理程序处理不同问题的能力，包括那些系统设计者可能未预料到的问题
- 基于模型推理往往非常鲁棒。和人类经常在遇到新问题时重复基本原则的原因相同，基于模型推理程序往往具有非常好的鲁棒性和灵活性
- 某些知识可以在任务之间转移。基于模型推理程序经常是使用科学的理论性知识建立的。科学所力求的是广泛适用的理论，这种一般性也为基于模型推理程序提供了优势。
- 很多时候，基于模型推理程序可以提供因果关系解释。这可以向用户表达对故障的更深层理解，因而能起到很重要的辅导作用

缺点

-)缺少领域的经验性(描述性)知识。基于规则系统使用的启发方法反映了经验知识是一项有价值的技能
- 它需要一个显式的领域模型。很多领域(例如电子电路的诊断)有很强的理论基础，可以很好地支持基于模型方法。不过，很多领域(例如某些医疗专业、大多数设计问题或者很多财经应用)缺乏完善定义的科学理论。基于模型方法不能用在这些领域中
- 复杂度高。基于模型推理通常是在非常复杂的层次运行，这也是人类专家把启发放在第位的主要原因之一
- 意外情况。例如，异常的环境、桥接故障或电子部件中多个故障的相互影响可能会以事先难以预测到的方式影响系统功能。

混合设计

目前研究和应用的一个重要热点就是把不同的推理模型组合在一起。在混合体系结构中，两种或更多种模式被集成到一起，以得到一种协作效果:用一种策略的优点来弥补其他策略的不足。通过组合，我们可以弥补前面讨论所提到的各种不足

例如，基于规则系统和基于案例系统的组合可以:

- 提供一种自然的解决方法:先检查已知案例，然后再进行基于规则推理和有关的搜索

- 通过规则库中保存的内容提供解的样例和异常
- 把基于搜索得到的结果记录为案例供将来使用。通过保存合适的案例，推理程序可以避免重复代价很高的搜索

architecture of a typical ES

exploratory development cycle

domain experts

knowledge engineer

end users

开发最快--快速原型法

规则/逻辑系统/状态空间

不确定下推理

斯坦福stanford 信任度

离散的马尔可夫模型

基于符号的机器学习

> 概念学习的目的，就是让你从训练样本集D中来去归纳演绎出一般规律，来覆盖假设空间H。从一般（general）到特殊（special）是概念学习的精髓所在

<ϕ,ϕ,ϕ,ϕ,ϕ,ϕ>最特殊序列
< ?,?,?,?,?,? > 是最一般序列

假设空间

设置归纳偏置，对应两假设存在偏序关系：$h_j(x) \geq_g h_k(x)$，那么我们就称j比k更一般

变形空间

用训练样本集D去逼近假设空间H

假设$h \in H$与训练样本集D一致，当且仅当对于任意样例$< x, c(x) > \in D$都有$h(x) = c(x)$

如果我们对于任意的$h \in H$都能保证与D 一致，那么，我们也就能说样本集D（一致）逼近到了H上。我们将上述一致形成的集合空间表示出来,形成了变形空间

变形空间覆盖了 **目标概念c(x)** 中所有合理的变形，也就是说它包含了**最佳假设**，$VS_{H,D} \equiv \{h \in H | h\text{一致于}D\}$

> 边界的约束就取决于H**与**D相一致的极大一般集合和极大特殊集合(可以理解成上下确界)，也就是VS必须包含D中所有正例，而抑制D中所有反例

Find-s

只通过正例来寻求**极大特殊集合**，**反例我可以直接无视

- step1 将h初始化为H中最特殊假设
- step2 使用每个正例，将h中属性替换为更一般的约束
- step3 输出h

候选消除算法

改进Find-s算法，寻找**极大一般边界**---考虑反例如何从空间中剔除

> 正例反例是合取关系：
> **正例一定得到的是最特殊的，反例剔除后得到的一定是最一般的**

> 将G集合初始化为H中极大一般假设
> 将S集合初始化为H中极大特殊假设

对每个训练例d，进行以下操作：

- 如果d是一正例
  - 从G中移去所有与d不一致的假设
  - 对S中每个与d不一致的假设s，将s替换成一般的“?”
  - 排除S中比S的其他假设更一般的假设
  - 排除S中比G中更一般的假设
- 如果d是一个反例
  - 从S中移去所有d一致的假设
  - 对匹配d的G中g，用不匹配d的最一般特化?来替换g
  - 排除G中比G其他假设更特殊的假设
  - 排除G中比S中更特殊的假设

ID3算法

决策树

一种树形结构，其组成包括结点(node)和有向边(directed edge)。而结点有两种类型，分别是内部结点(internal node)和叶结点(leaf node)。其中每个内部结点表示一个属性（特征），每个叶节点表示一个类别

熵值计算

熵：$H = -\Sigma_{i=1}^{n} P(x_i) log_2 P(x_i)$

经验熵：$H(D) = -\Sigma_{k=1}^{K} \frac{|C_k|}{|D|} log_2 \frac{|C_k|}{|D|}$

条件熵：$H(Y|X) = \Sigma_{i=1}^{n} p_i H(Y|X = x_i)$

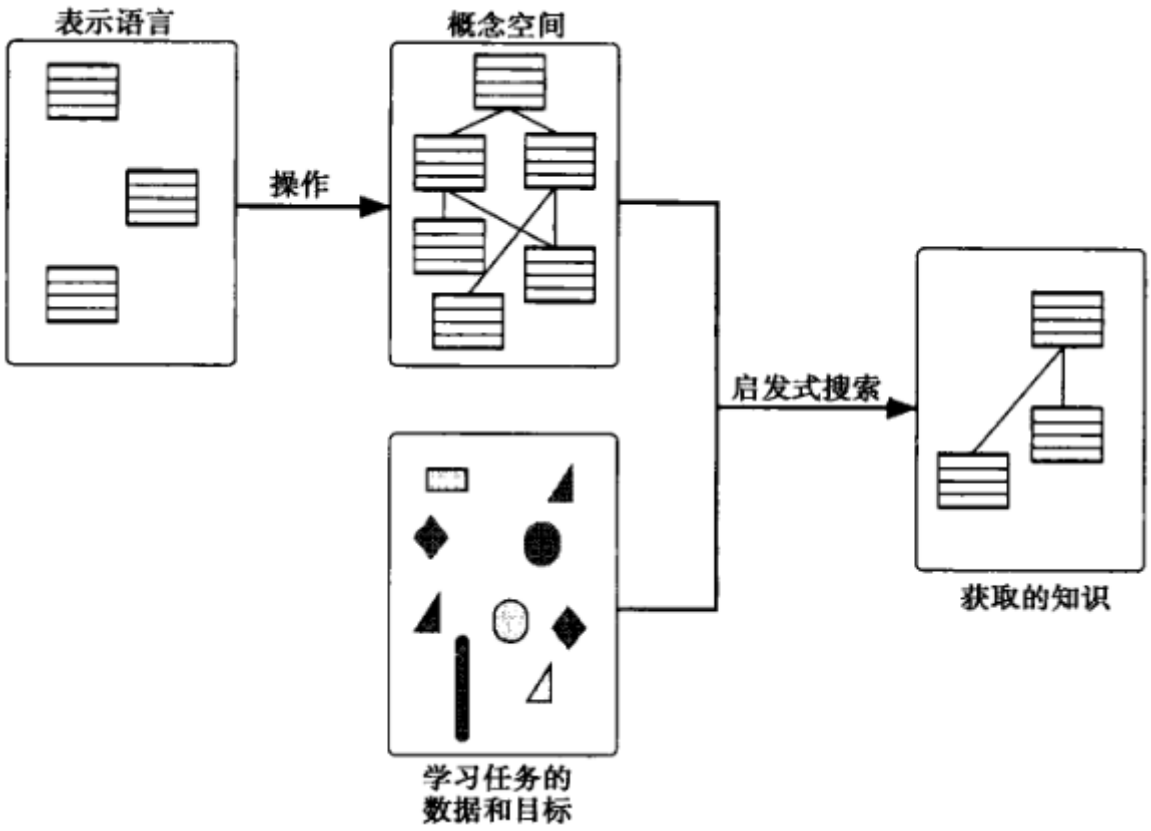
信息增益：集合D的经验熵H(D)与特征A给定条件下D的经验条件熵H(D|A)之差
$g(D, A) = H(D) - H(D|A)$

执行过程

1. 收集数据：可以使用任何方法
2. 准备数据：树构造算法只适用于标称型数据，因此数值型数据必须离散化
3. 分析数据：可以使用任何方法，构造树完成之后，我们应该检查图形是否符合预期
4. 训练算法：构造树的数据结构
5. 测试算法：使用经验树计算错误率

6. 使用算法：此步骤可以适用于任何监督学习算法，而使用决策树可以更好地理解数据的内在含义

基于符号学习的系统空间架构



1. 学习任务的数据和目标
2. 所学知识的表示
3. 操作的集合
4. 概念空间
5. 启发式搜索

连接机制

MP感知机

只有一层神经网络，输入输出值都是1或-1，权重和=$\Sigma x_i w_i$

如果权重和大于等于阈值函数，输出1；反之输出-1
因此期望值和实际值的差值为0或2

权值分量$\Delta(w_i) = c(d - sign(\Sigma x_i w_i))x_i$(d是期望输出，c是常数--学习率)
权值向量分量：

- 期望==输出，权值不变
- 期望是1而输出-1，对第i个分量增加$2cx_i$
- 期望是-1而输出1，对第i个分量减少$2cx_i$

这个过程不断地减少训练集和的平均误差

> 感知机不能解决非线性可分问题

过程：

- 从可能的问题空间中选择原始数据，然后变换成新的数据/模式空间
- 在这新的模式空间中，空间特征被确定
- 对这些特征所表示的实体分类

*BP*神经网络

1.激活函数

激活函数（Activation Function）是在人工神经网络的神经元上运行的函数，负责将神经元的输入映射到输出端。激活函数对于人工神经网络模型去学习、理解复杂的非线性函数，具有十分重要的作用。

如果不使用激活函数，每一层输出都是上一层输入的线性运算，无论神经网络有多少层，最终的输出只是输入的线性组合，相当于感知机。如果使用了激活函数，将非线性因素引入到网络中，使得神经网络可以任意逼近任何非线性函数，能够应用到更多的非线性模型。

$sigmoid$ 函数

$Sigmoid$函数是一个在生物学中常见的S型函数，也称为S型生长曲线。在信息科学中，由于其单增以及反函数单增等性质，Sigmoid函数常被用作神经网络的阈值函数，将变量映射到0,1之间，公式如下：

$$f(x)=\frac{1}{1+e^{(-x)}}$$

$ReLU$函数

$Relu$激活函数（The Rectified Linear Unit），用于隐藏层的神经元输出。公式如下：

$$f(x)=max(0,x)$$

$Tanh$ 函数

$Tanh$ 是双曲函数中的一个，$Tanh()$ 为双曲正切。在数学中，双曲正切“$Tanh$”是由基本双曲函数双曲正弦和双曲余弦推导而来。公式如下：

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

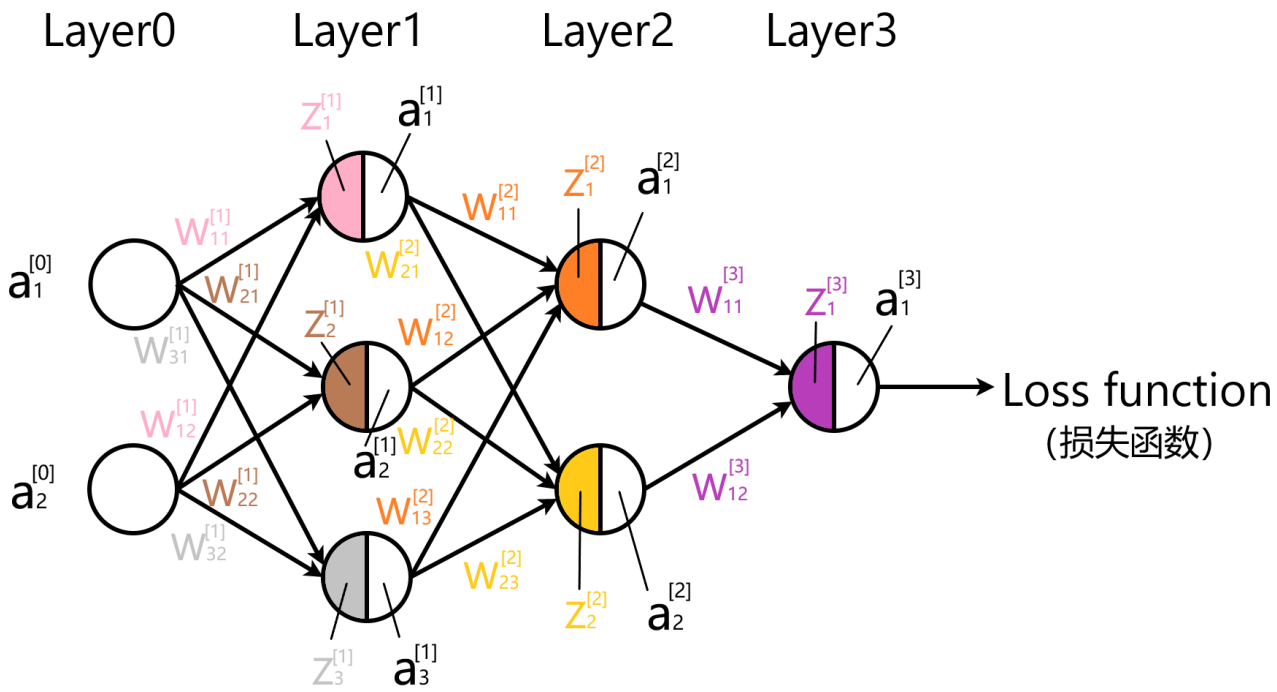
$softmax$ 函数

$softmax$ 函数用于输出层。假设输出层共有 n 个神经元，计算第 k 个神经元的输出 y_k。$softmax$ 函数的分子是输入信号 a_k 的指数函数，分母是所有输入信号的指数函数的和。$softmax$ 函数公式如下：

$$y_k = \frac{e^{a_k}}{\sum_{i=1}^{n} e^{a_i}}$$

2.神经网络结构

第0层是输入层（2个神经元），第1层是隐含层（3个神经元），第2层是隐含层（2个神经元），第3层是输出层。



符号约定

$w_{jk}^{[l]}$表示从网络第$(l-1)^{th}$ 层第k^{th} 个神经元指向第 l^{th} 层第 j^{th} 个神经元的连接权重，同时也是第 l 层权重矩阵第 j 行第 k 列的元素。
例如，上图中 $w_{21}^{[1]}$，第0层第1个神经元指向第1层第2个神经元的权重（褐色），也就是第 1 层权重矩阵第 2 行第 1 列的元素。
同理，使用 $b_j^{[l]}$ 表示第 l^{th} 层第 j^{th} 个神经元的偏置，同时也是第 l 层偏置向量的第 j 个元素。
使用 $z_j^{[l]}$ 表示第 l^{th} 层第 j^{th} 个神经元的线性结果，使用 $a_j^{[l]}$ 来表示第 l^{th} 层第 j^{th} 个神经元的激活函数输出。
其中，激活函数使用符号σ表示，第 l^{th} 层中第 j^{th} 个神经元的激活为:

$$a_j^{[l]} = \sigma(z_j^{[l]}) = \sigma\left(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}\right)$$

$w^{[l]}$ 表示第 l 层的权重矩阵，$b^{[l]}$ 表示第 l 层的偏置向量，$a^{[l]}$ 表示第 l 层的神经元向量，结合上图讲述：

$$w^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \end{bmatrix} \quad w^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} & w_{23}^{[2]} \end{bmatrix}$$

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix} \quad b^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \end{bmatrix}$$

进行线性矩阵运算。

$$z^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \end{bmatrix} \cdot \begin{bmatrix} a_1^{[0]} \\ a_2^{[0]} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[1]} a_1^{[0]} + w_{12}^{[1]} a_2^{[0]} + b_1^{[1]} \\ w_{21}^{[1]} a_1^{[0]} + w_{22}^{[1]} a_2^{[0]} + b_2^{[1]} \\ w_{31}^{[1]} a_1^{[0]} + w_{32}^{[1]} a_2^{[0]} + b_3^{[1]} \end{bmatrix}$$

矩阵形状　(3,2)　(2,1) (3,1) (3,1)

$$z^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} & w_{23}^{[2]} \end{bmatrix} \cdot \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} + \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[2]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + w_{13}^{[2]} a_3^{[1]} + b_1^{[2]} \\ w_{21}^{[2]} a_1^{[1]} + w_{22}^{[2]} a_2^{[1]} + w_{23}^{[2]} a_3^{[1]} + b_2^{[2]} \end{bmatrix}$$

矩阵形状(2,3) (3,1) (2,1) (2,1)

那么，前向传播过程可以表示为：

$$a^{[l]} = \sigma\left(w^{[l]} a^{[l-1]} + b^{[l]}\right)$$

上述讲述的前向传播过程，输入层只有1个列向量，也就是只有一个输入样本。对于多个样本，输入不再是1个列向量，而是m个列向量，每1列表示一个输入样本。m个$a^{[l-1]}$列向量组成一个m列的矩阵$A^{[l-1]}$。

$$A^{[l-1]} = \begin{bmatrix} | & | & \cdots & | \\ a^{[l-1](1)} & a^{[l-1](2)} & \cdots & a^{[l-1](m)} \\ | & | & \cdots & | \end{bmatrix}$$

多样本输入的前向传播过程可以表示为：

$$Z^{[l]} = w^{[l]} \cdot A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = \sigma\left(Z^{[l]}\right)$$

与单样本输入相比，多样本$w^{[l]}$和$b^{[l]}$的定义是完全一样的，不同的只是$Z^{[l]}$和$A^{[l]}$从1列变成m列，每1列表示一个样本的计算结果。

3.损失函数

在有监督的机器学习算法中，我们希望在学习过程中最小化每个训练样例的误差。通过梯度下降等优化策略完成的，而这个误差来自损失函数。

损失函数用于单个训练样本，而**成本函数**是多个训练样本的平均损失。优化策略旨在最小化成本函数。下面例举几个常用的损失函数。

回归问题

1. 绝对值损失函数(L_1损失函数)：

$L(\hat{y}, y) = |y - \hat{y}|$

y 表示真实值或期望值，$\hat{y}$ 表示预测值

2. 平方损失函数(L_2损失函数)：

$L(\hat{y}, y) = (y - \hat{y})^2$

y 表示真实值或期望值，$\hat{y}$ 表示预测值

分类问题

1. 交叉熵损失：

$L(\hat{y}, y) = -y \log(\hat{y}) - (1-y) \log(1-\hat{y})$

y 表示真实值或期望值，$\hat{y}$ 表示预测值

4.反向传播

反向传播的基本思想：通过计算输出层与期望值之间的误差来调整网络参数，使得误差变小(最小化损失函数或成本函数)。反向传播基于**四个基础等式**，非常简洁优美，但想要理解透彻还是挺烧脑的。

求解梯度矩阵

假设函数 $f: R^{n\times 1} \to R$ 将输入的列向量（shape: $n \times 1$ ）映射为一个实数。那么，函数 f 的梯度定义为：

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

同理，假设函数 $f: R^{m\times n} \to R$ 将输入的矩阵（shape: $m \times n$ ）映射为一个实数。函数 f 的梯度定义为：

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{13}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

可以简化为：

$(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}}$

注意：梯度求解的前提是函数 f 返回的必须是一个实数，如果函数返回的是一个矩阵或者向量，是没有办法求解梯度的。例如，函数 $f(A) = \sum_{i=0}^{m}\sum_{j=0}^{n} A_{ij}^2$，函数返回一个实数，可以求解梯度矩阵。如果 $f(x) = Ax \left(A \in R^{m\times n}, x \in R^{n\times 1}\right)$, 函数返回一个m行的列向量，就不能对 f 求解梯度矩阵。

矩阵相乘

矩阵 $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$，矩阵 $B = \begin{bmatrix} -1 & -2 \\ -3 & -4 \end{bmatrix}$

$$AB = \begin{bmatrix} 1\times -1 + 2 \times -3 & 1 \times -2 + 2 \times -4 \\ 3 \times -1 + 4 \times -3 & 3 \times -2 + 4 \times -4 \end{bmatrix} = \begin{bmatrix} -7 & -10 \\ -15 & -22 \end{bmatrix}$$

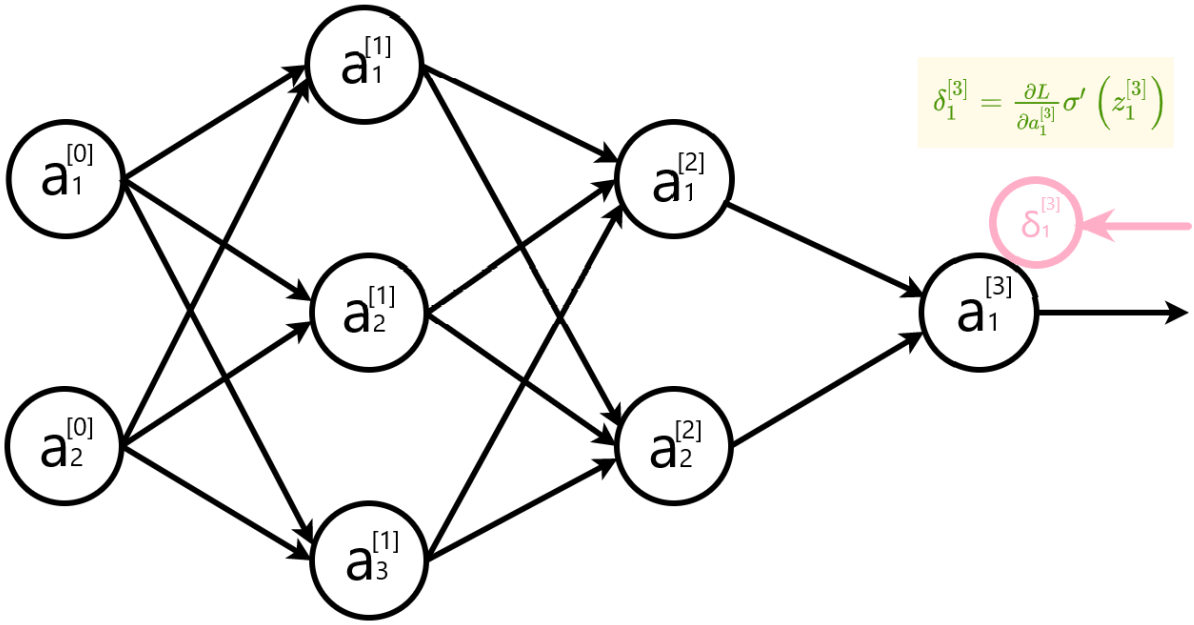
矩阵对应元素相乘

使用符号⊙表示：

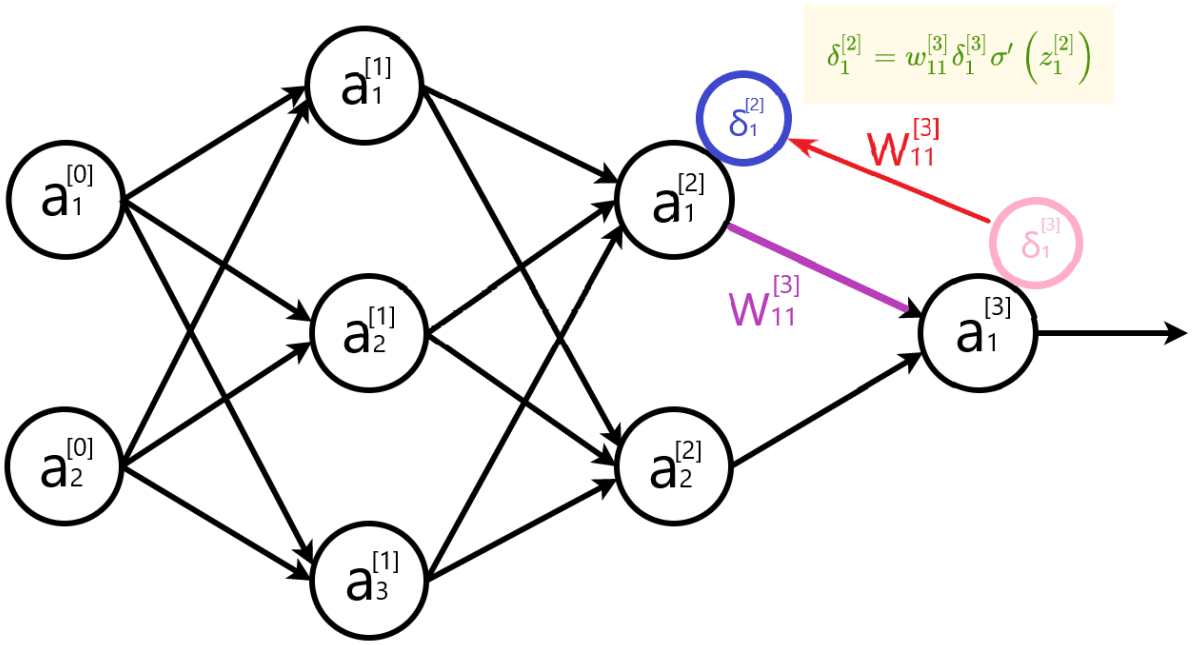
$$A \odot B = \begin{bmatrix} 1 \times -1 & 2 \times -2 \\ 3 \times -3 & 4 \times -4 \end{bmatrix} = \begin{bmatrix} -1 & -4 \\ -9 & -16 \end{bmatrix}$$

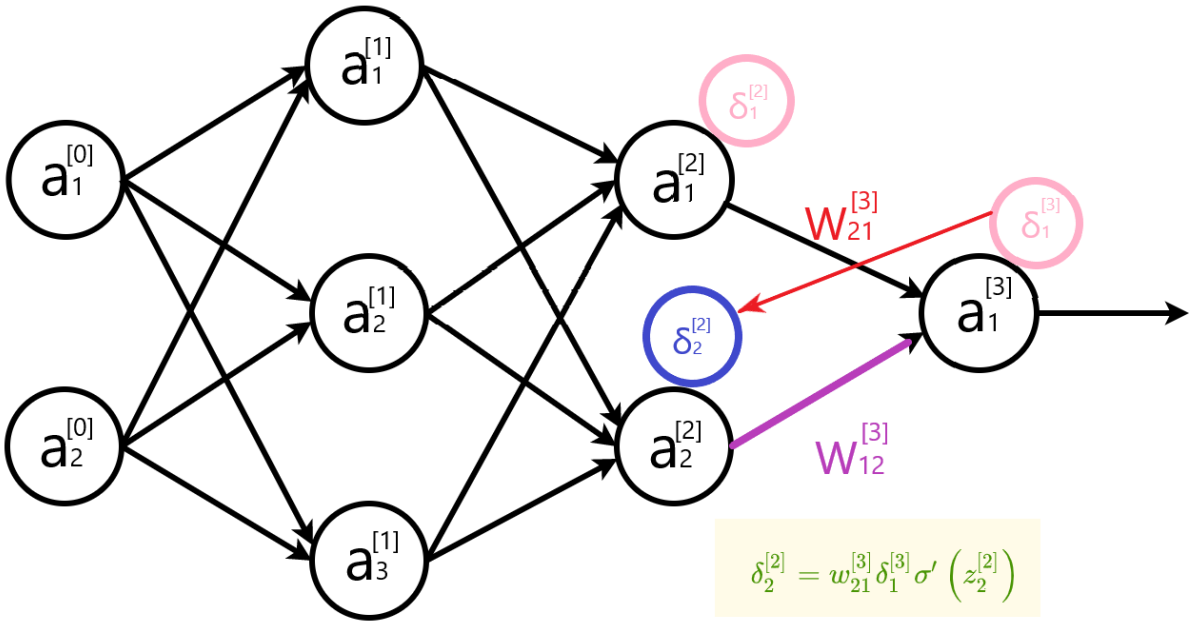
反向传播图解

计算输出层误差

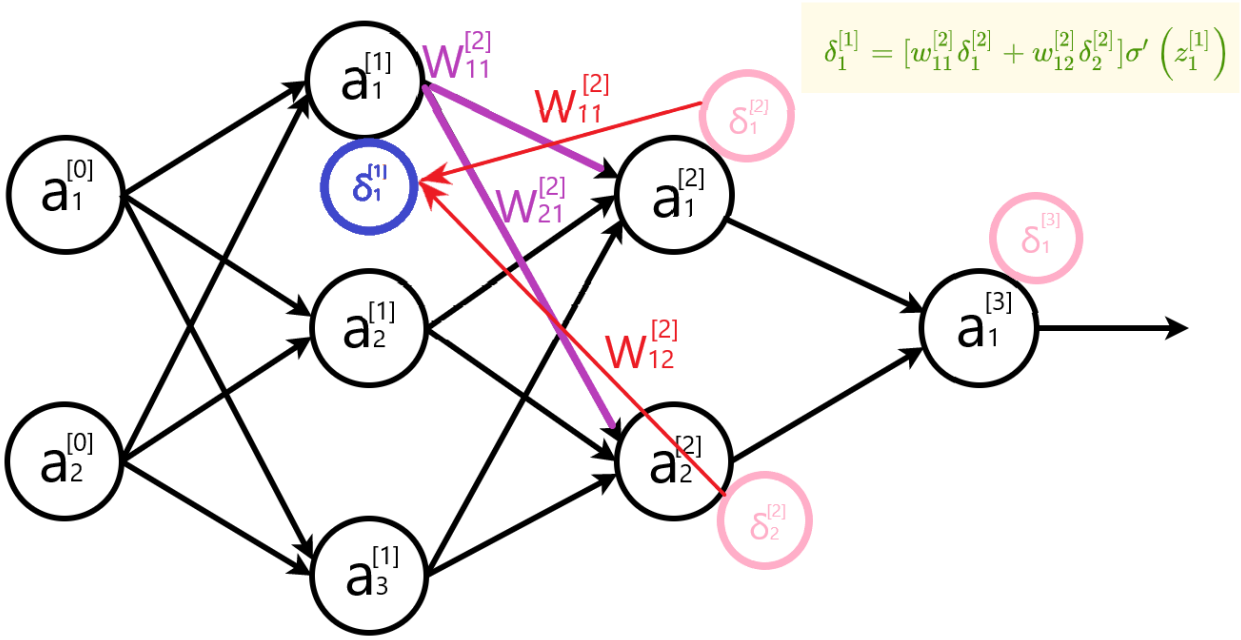


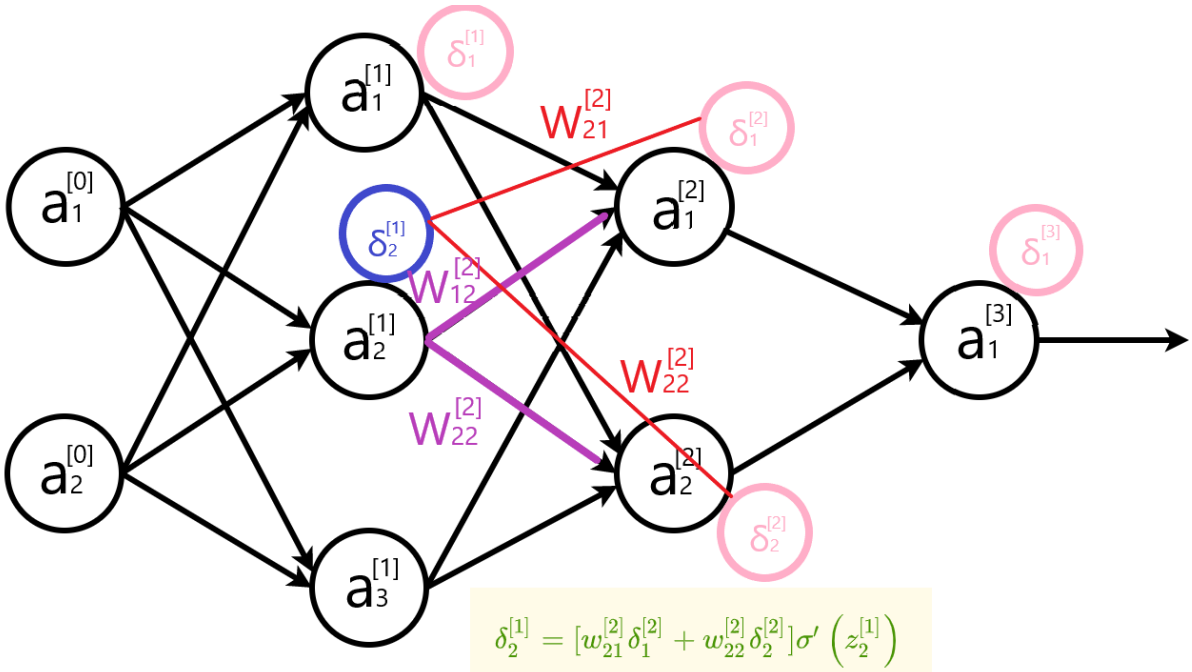
计算隐藏层误差



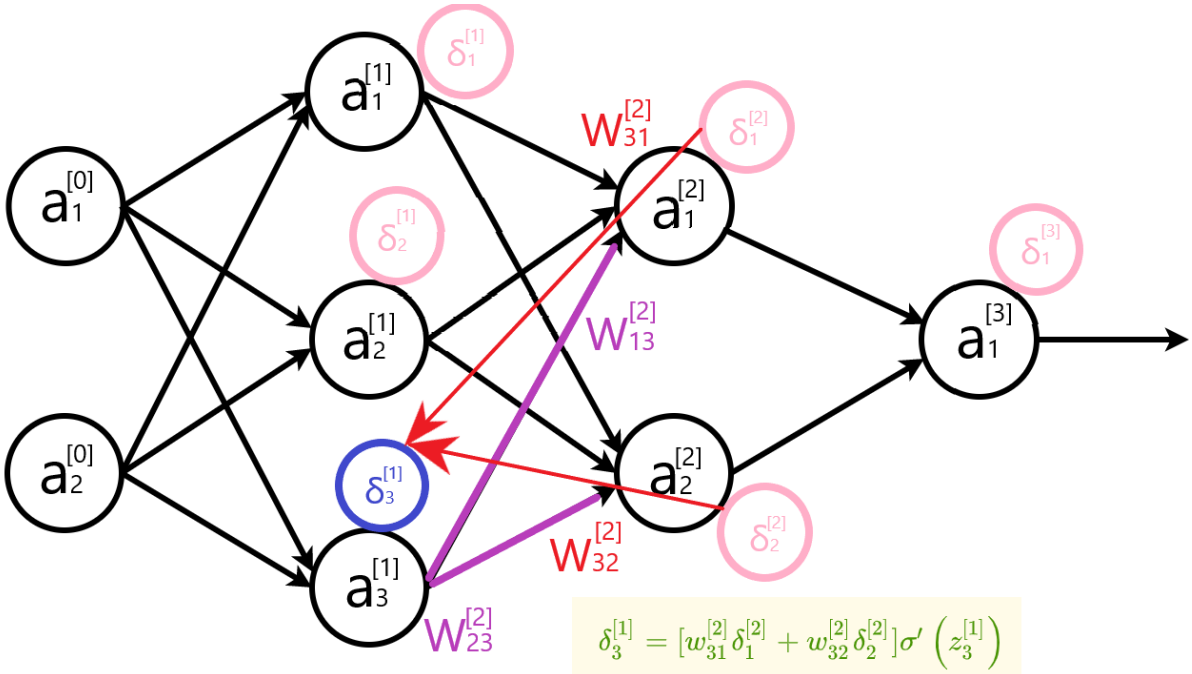


隐藏层误差公式写成矩阵形式 $\delta^{[l]} = \left[w^{[l+1]^T}\delta^{[l+1]}\right] \odot \sigma'\left(z^{[l]}\right)$ 时， 权重矩阵需要转置。上面两幅图，直观地解释了转置的原因。



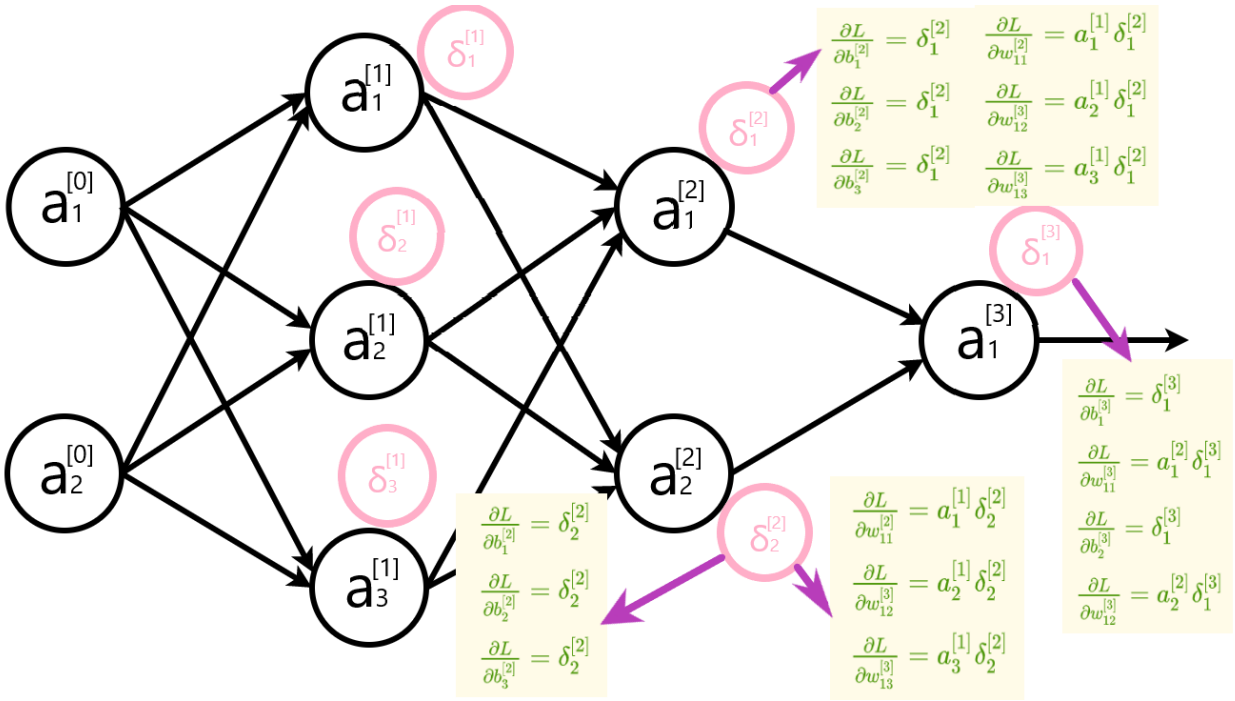


$$\delta_2^{[1]} = [w_{21}^{[2]}\delta_1^{[2]} + w_{22}^{[2]}\delta_2^{[2]}]\sigma'\left(z_2^{[1]}\right)$$



$$\delta_3^{[1]} = [w_{31}^{[2]}\delta_1^{[2]} + w_{32}^{[2]}\delta_2^{[2]}]\sigma'\left(z_3^{[1]}\right)$$

计算参数变化率



最后更新每层的参数。

反向传播公式总结

单样本输入公式表

说明	公式
输出层误差	$\delta^{[L]} = \nabla_a L \odot \sigma'\left(z^{[L]}\right)$
隐含层误差	$\delta^{[l]} = \left[w^{[l+1]^T} \delta^{[l+1]}\right] \odot \sigma'\left(z^{[l]}\right)$
参数变化率	$\frac{\partial L}{\partial b^{[l]}} = \delta^{[l]}$ $\frac{\partial L}{\partial w^{[l]}} = \delta^{[l]} a^{[l-1]T}$
参数更新	$b^{[l]} \leftarrow b^{[l]} - \alpha \frac{\partial L}{\partial b^{[l]}}$ $w^{[l]} \leftarrow w^{[l]} - \alpha \frac{\partial L}{\partial w^{[l]}}$

多样本输入公式表

成本函数

多样本输入使用的成本函数与单样本不同。假设单样本的成本函数是交叉熵损失函数。

$L(a, y) = -[y \cdot \log(a) + (1-y) \cdot \log(1-a)]$

那么，对于m个样本输入，成本函数是每个样本的成本总和的平均值。

$C(A, y) = -\frac{1}{m} \sum_{i=0}^{m} \left(y^{(i)} \cdot \log\left(a^{(i)}\right) + \left(1-y^{(i)}\right) \cdot \log\left(1-a^{(i)}\right)\right)$

误差

单样本输入的每一层的误差是一个列向量

$$\delta^{[l]} = \begin{bmatrix} \delta_1^{[l]} \\ \delta_2^{[l]} \\ \vdots \\ \delta_n^{[l]} \end{bmatrix}$$

而多样本输入的每一层的误差不再是一个列向量，变成一个m列的矩阵，每一列对应一个样本的向量。那么多样本的误差定义为：

$$dZ^{[l]} = \left[\delta^{[l](1)}\delta^{[l](2)}\dots\delta^{[l](m)}\right] = \begin{bmatrix} \delta_1^{[l](1)} & \delta_1^{[l](2)} & \dots & \delta_1^{[l](m)} \\ \delta_2^{[l](1)} & \delta_2^{[l](2)} & \dots & \delta_2^{[l](m)} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_n^{[l](1)} & \delta_n^{[l](2)} & \dots & \delta_n^{[l](m)} \end{bmatrix}$$

$dZ^{[l]}$的维度是 $n \times m$ ， n 表示第 l 层神经元的个数， m 表示样本数量。

参数变换率

因为$dZ^{[l]}$的维度是 $j \times m$ ，更新 $b^{[l]}$ 的时候需要对每行求平均值，使得维度变为 $j \times 1$，再乘以$\frac{1}{m}$。

$dZ^{[l]}$的维度是 $j \times m$ ， $A^{[l-1]T}$ 的维度是 $m \times k$，矩阵相乘得到的维度是 $j \times k$ ，与$w^{[l]}$ 本身的维度相同。因此更新 $w^{[l]}$ 时只需乘以 $\frac{1}{m}$ 求平均值。

说明	公式
输出层误差	$dZ^{[L]} = \nabla_A C \odot \sigma'\left(Z^{[L]}\right)$
隐含层误差	$dZ^{[l]} = \left[w^{[l+1]T} dZ^{[l+1]}\right] \odot \sigma'\left(Z^{[l]}\right)$
参数变化率	$db^{[l]} = \frac{\partial C}{\partial b^{[l]}} = \frac{1}{m} meanOfEachRow\left(dZ^{[l]}\right)$ $dw^{[l]} = \frac{\partial C}{\partial w^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$
参数更新	$b^{[l]} \leftarrow b^{[l]} - \alpha \frac{\partial C}{\partial b^{[l]}}$ $w^{[l]} \leftarrow w^{[l]} - \alpha \frac{\partial C}{\partial w^{[l]}}$

卷积神经网络CNN

- 卷积
- 数据

遗传性和涌现性

遗传算法

基本过程--怎么用，解决什么问题

概率理论

马尔科夫模型

- 基本关系
- 什么是***

自动推理

归结反驳证明

命题演算或谓词演算中的定理证明

归结反驳证明步骤：

```
将前提或公理转化为子句形式
    去蕴含
    化简表达式
    重命名
    所有量词都移到左部
    消除存在两次，使用斯克轮化过程消除
    去掉全称量词，直接去掉即可
    打开括号，进行结合律、分配律
    根据合取拆成不同子句
    对两个子句进行标准化，给子句每个变量取不同的名字
将要证明的命题取反，转化为子句形式加入公理集合
归结这些子句，生成可以从逻辑上推到出的新子句
通过生成空子句得出矛盾
用来得出空子句的代换是哪些使得取反后的目标的反为真的代换
```

二元归结证明

> 归结反驳证明过程通过将子句集简化出一个矛盾来回答问题或推导出新结果

$a \vee b$和$\neg b \vee c$的归结式是$a \vee c$，因为$b \vee \neg b$永真

证明求解