人工智能课程开卷考试资料

> 说明：本文件由各章节考场资料合并生成，英文关键词用于匹配英文题目，中文解释用于作答。

01 Introduction - 考场资料

来源：`01Introduction.pdf`。原则：覆盖课件中的可答题信息；非考点性联系信息不收录。

0. 本章总览

```
AI = Representation + Reasoning + Learning + Search/Optimization + Agent
```

模块	本章对应内容	后续章节
Representation	逻辑、谓词、规则、知识库	Knowledge representation, Logic, Prolog, CLIPS
Reasoning	syllogism, modus ponens, Tarski semantic, inference engine	Logic, Prolog, CLIPS
Learning	artificial neural systems	Connectionist, Deep learning
Search/Optimization	genetic algorithm	Genetic algorithm
Agent	multi-agent systems, situated/interactional agents	Reinforcement learning

1. 参考书与引入材料 p3, p5

Books p3

可作为开放考试中补充引用，不是核心考点。

方向	参考书
综合 AI	George F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*
深度学习	Goodfellow, Bengio, Courville, *Deep Learning*
模式识别	Bishop, *Pattern Recognition and Machine Learning*
统计学习	Hastie, Tibshirani, Friedman, *The Elements of Statistical Learning*
专家系统	Giarratano, Riley, *Expert Systems: Principles and Programming*

Greek mythology: Prometheus p5

课件用 Prometheus 引入 intelligence：

```
Prometheus 不只是给人类带来 fire，也象征给人类带来 intelligence/enlightenment。
```

可用答法：

```
课件用 Prometheus 神话说明 intelligence 被看作启蒙人类、扩展人类能力的关键。
```

2. AI 起点与历史线索 p4-p9, p13, p33

Dartmouth Workshop p4

```
1956 Dartmouth Conference: The Founding Fathers of AI
```

常见答法：

```
Dartmouth Workshop 通常被视为人工智能作为独立研究领域的起点。它标志着“机器智能”开始成为一个明确的研究主题。
```

课件图中人物：

```
John McCarthy, Marvin Minsky, Claude Shannon, Ray Solomonoff,
Alan Newell, Herbert Simon, Arthur Samuel, Oliver Selfridge,
Nathaniel Rochester, Trenchard More
```

Historical foundations p6-p9, p13, p33

人物	关键词	与 AI 的关系
Aristotle	syllogism, modus ponens	早期形式推理；从前提出结论
Copernicus	Copernican revolution	世界的真实结构可能不同于表象
Galileo	scientific observations, mathematics	用数学描述世界
Descartes	cognitive introspection, Cogito ergo sum	从心智/认知出发讨论现实基础
Hobbes	Reasoning is but reckoning	推理可理解为计算
Leibnitz	formal logic, automated calculation	形式逻辑和自动化计算
Boole	Boolean algebra	逻辑规律的数学化，是计算机科学基础
Frege	foundations of arithmetic, formal language	用形式语言精确定义算术基础
Russell & Whitehead	axioms, formal operations	试图用公理和形式操作推出整个数学
Tarski	semantic theory of truth	公式能精确指向/解释现实世界
Turing	computing machinery and intelligence	机器能否思考、智能是否可计算
Newell & Simon	production rules, general problem solver	人类问题求解可表达为 IF-THEN 规则

3. 逻辑推理与符号操作 p6, p10-p16

Syllogism / Modus ponens p6, p11-p12

课件例子：

```
All men are mortal.
Socrates is a man.
Therefore, Socrates is mortal.
```

对应规则：

```
P -> Q
P
therefore Q
```

一阶谓词表达 p12：

```
forall x (man(x) -> mortal(x))
man(Socrates)
therefore mortal(Socrates)
```

考场答法：

```
三段论/Modus ponens 说明推理可以由形式规则完成：如果一般规则和具体事实成立，机器可按规则推出结论。
```

String manipulation p10

课件核心：

```
Any mathematic or logic system is simply a set of rules specifying how to change one string of symbols into another set of
symbols.
```

课件例子：

```
person has fever AND fever is less than 39 -> take aspirin

alpha ^ beta -> gamma
alpha AND beta -> gamma
```

考场答法：

```
形式逻辑可以看成符号串变换系统。只要给定合法符号和变换规则，推理就能被机械地执行。
```

Syntax vs Semantics

概念	关注点	答题句
Syntax	公式形式是否合法	判断一个表达式是不是合式公式，即“写法是否正确”。
Semantics	公式在模型中含义和真假	判断公式在某个解释/model 下是什么意思、是真是假。

Tarski's semantic p14-p16

题目：

```
Given: ((A v C) ^ (B v not C)) is true.
Question: is (A v B) true?
```

真值表：

A	B	C	A v C	B v not C	Premise	A v B
0	0	0	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	0	0	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

结论：

```
Premise 为真的行是 models: 011, 100, 110, 111。
在这些 models 中，A v B 都为真。
所以 ((A v C) ^ (B v not C)) entails (A v B)。
```

答题模板：

```
判断 semantic entailment 时，先列出所有使 premise 为真的 model；如果 consequence 在所有这些 model 中也为真，则 premise entails consequence。
```

4. Cognitive science 与产生式规则 p17-p18

Newell & Simon p17

课件要点：

```
Much of human problem solving or cognition can be expressed by IF-THEN type production rules.
```

认知结构：

课件术语	含义
Long-term memory	rules
Short-term memory / working memory	当前事实/状态
Cognitive processor	inference engine
General problem solver	Newell & Simon 的通用问题求解思想

Production rule p18

模板：

```
IF <conditional elements / patterns>
THEN <actions / conclusions>
```

课件例子：

```
IF the car doesn't run AND the fuel gauge reads empty
THEN fill the gas tank
```

术语：

术语	解释
Antecedent	条件部分
LHS	left-hand-side，规则左部
Conditional element / pattern	可与事实匹配的条件元素
Consequent	结论/动作部分
RHS	right-hand-side，规则右部

标准答法：

```
Production rule 是 IF-THEN 形式的规则，LHS/antecedent 表示条件或模式，RHS/consequent 表示动作或结论。它适合表示专家经验，并可由推理机根据工作记忆中的事实触发。
```

5. Rule-based expert system p19-p20

专家系统结构 p19

课件结构图：

```
Knowledge Base (Rules) <-> Inference Engine <-> Working Memory (Facts)
                                |
                             Agenda
                                |
          Explanation Facility, Knowledge Acquisition Facility
                                |
                          User Interface
```

核心公式：

```
Expert System = Knowledge Base + Working Memory + Inference Engine
```

部件	作用
Knowledge Base (Rules)	存放领域知识/规则
Working Memory (Facts)	存放当前事实和中间结论
Inference Engine	匹配事实与规则，执行推理
Agenda	保存当前可触发规则，决定执行顺序
Explanation Facility	解释为什么得出某结论
Knowledge Acquisition Facility	帮助获取/维护专家知识
User Interface	用户输入事实、查看结论

课件列出的优点：

```
Modular nature.
Explanation facilities.
Similarity to human cognitive process.
```

标准答法：

```
规则专家系统由知识库、工作记忆和推理机构成。知识库存放规则，工作记忆存放事实，推理机将事实与规则匹配并触发规则，agenda 管理可执行规则，解释设施说明推理过程。其优点是模块化、可解释，并且与人类基于规则的问题求解方式相似。
```

Expert systems applications p20

系统	时间/机构	功能	关键点
MYCIN	mid-1970s, Stanford	诊断并建议治疗脑膜炎和血液细菌感染	expert medical knowledge
PROSPECTOR	1979, Duda	分析地质数据寻找矿藏	discovered mineral deposit worth $100m
XCON	1981, Carnegie-Mellon and DEC	配置计算机系统	saved DEC millions of dollars a year

6. Artificial neural systems p21

本页是连接主义/神经网络的预告。

```
Artificial neural systems 受生物神经系统启发，用由许多简单处理单元连接而成的网络表示和学习复杂函数，是连接主义和深度学习的基础。
```

7. Evolution 与 Genetic algorithm p22-p31

Evolution p22

Darwin 引文用于说明：复杂适应性可通过渐进演化产生。

考场答法：

```
遗传算法受生物进化启发，通过选择、交叉、变异等操作，使候选解群体逐步适应目标函数。
```

例题：maximize f(x)=x^2, x in [1,31] p23-p24, p30

Representation p23

```
x in [1,31], use 5-bit binary representation: x in {0,1}^5
```

Initialization p23

Individual	Decimal interpretation	Fitness f(x)=x^2
01101	13	169
11000	24	576
01000	8	64
10011	19	361

Total fitness:

```
169 + 576 + 64 + 361 = 1170
```

Selection p24

Individual	Fitness	Selection probability
01101	169	0.14
11000	576	0.49
01000	64	0.06
10011	361	0.31

Selection result:

```
01101, 11000, 11000, 10011
```

Russian roulette p25-p29

含义:

```
Roulette-wheel selection 指按适应度比例选择个体。
适应度越大，在轮盘中占的扇区越大，被选中的概率越高。
```

本例概率:

```
01101: 14%
11000: 49%
01000: 6%
10011: 31%
```

Crossover p30

课件展示两个父代交叉生成后代:

```
01101 + 11000 -> 01100 + 11001
11000 + 10011 -> 11011 + 10000
```

答题句:

```
Crossover 将被选中的父代字符串片段重新组合，生成后代，使较好的局部结构有机会被保留和重组。
```

Mutation p30

课件例子:

```
01100 -> 11100
```

答题句:

```
Mutation 随机改变个体中的某些位，用于维持种群多样性，并帮助避免过早收敛。
```

Genetic algorithm pseudocode p31

```
begin
    set time t = 0
    initialize the population P(t)
    while the termination condition is not met do
    begin
        evaluate fitness of each member of population P(t)
        select members from P(t) based on fitness
        produce offspring using genetic operators
        replace candidates of P(t) with these offspring
        set time t = t + 1
    end
end
```

步骤速查:

```
encoding -> initialization -> fitness evaluation -> selection -> crossover/mutation -> replacement -> termination
```

8. Multi-agent systems p32

课件特征：

```
Agents are autonomous or semi-autonomous.
Agents are situated.
Agents are interactional.
The society of agents is structured.
The phenomenon of intelligence in this environment is emergent.
```

标准答法：

```
Multi-agent systems 由多个 autonomous 或 semi-autonomous agents 组成。每个 agent situated in an environment，并与其他 agent interaction；多个 agent 形成 structured society，整体智能可能从局部交互中 emergent。
```

中文答法：

```
多智能体系统由多个自主或半自主智能体组成。智能体处在环境中，能够与环境和其他智能体交互，并形成结构化社会；整体智能可能由局部交互涌现。
```

9. Overview of AI application areas p34

课件列出应用方向：

```
Game playing
Automated reasoning
Expert systems
Natural language understanding
Planning and robotics
Machine learning
```

答题句：

```
AI 的应用领域包括 game playing、automated reasoning、expert systems、natural language understanding、planning and robotics、machine learning。
```

10. 高频题模板

说明：考试题目可能是英文，因此标题保留英文题眼；作答用中文。

Q1. Why is formal logic important for AI?

```
形式逻辑把 reasoning 表示为基于显式规则的符号操作。因此，机器可以检查 well-formed formulas，应用 inference rules，并推出 conclusions。它为 knowledge representation、automated reasoning、Prolog 和 rule-based expert systems 提供基础。
```

Q2. Explain Tarski's semantic with the example in course material.

```
Tarski's semantic 说明形式公式可以在 model 中解释，并根据 model 判断真假。对课件中的 premise `((A v C) ^ (B v not C))`，先列 truth table，找出所有 premise 为真的 models。若在这些 models 中 `A v B` 都为真，则说明 premise semantically entails `A v B`。
```

Q3. What is a production rule?

```
Production rule 是 IF-THEN 形式的规则。LHS/antecedent 包含 conditions 或 patterns，RHS/consequent 包含 actions 或 conclusions。它用于表示 expert knowledge；当条件与 working memory facts 匹配时，可由 inference engine 触发执行。
```

Q4. Describe the structure of a rule-based expert system.

```
Rule-based expert system 主要包含 knowledge base、working memory 和 inference engine。Knowledge base 存放 rules，working memory 存放 current facts，inference engine 将 facts 与 rules 匹配并触发可用规则。此外系统还可包含 agenda、explanation facility、knowledge acquisition facility 和 user interface。
```

Q5. Expert system vs traditional program.

```
Traditional program 主要是 algorithms + data structures；expert system 主要是 knowledge + inference。专家系统把 domain knowledge 以 rules 的形式显式存储，并与 inference mechanism 分离，因此更模块化、更容易解释，也更便于维护知识。
```

Q6. Describe genetic algorithm steps.

```
Genetic algorithm 先对 candidate solutions 编码并初始化 population，然后计算每个个体的 fitness，按 fitness 选择个体，通过 crossover 和 mutation 产生 offspring，用后代更新 population，并重复直到满足 termination condition。
```

Q7. What is roulette-wheel selection?

Roulette-wheel selection 是按 fitness 比例选择个体的方法。Fitness 越高的个体在轮盘中占据越大的扇区，因此被选中的概率越高。

Q8. What are multi-agent systems?

Multi-agent systems 由多个 autonomous 或 semi-autonomous agents 组成。Agents 处于环境中，彼此 interaction，并形成 structured society；整体 intelligent behavior 可能从这些交互中 emergent。

02 Connectionist - 考场资料

来源：`02Connectionist.pdf`，41 页。定位：连接主义/神经网络基础。已按页面图像核对，重点不是新闻背景，而是人工神经元、感知机、梯度下降、BP 和 winner-take-all。

英文题目识别词

English in exam	中文定位	看到后去找
connectionist / connectionism	连接主义	本章整体
artificial neuron	人工神经元	输入、权重、加权和、激活函数
activation level / activation function	激活值/激活函数	人工神经元、sigmoid
threshold function / hard limiting threshold	阈值函数/硬阈值	McCulloch-Pitts, perceptron
bias	偏置	McCulloch-Pitts, perceptron
McCulloch-Pitts neuron	McCulloch-Pitts 神经元	AND/OR 阈值单元
perceptron	感知机	输出公式、权重更新、线性可分
learning constant / learning rate	学习常数/学习率	权重更新、梯度下降
desired output	期望输出	感知机、MSE、BP
actual output	实际输出	感知机、MSE、BP
linearly separable	线性可分	感知机局限
exclusive-or / XOR	异或问题	单层感知机不能解决
nonlinear	非线性	XOR、多层网络
sigmoidal function	sigmoid 函数	公式和导数
gradient	梯度	负梯度方向
gradient descent	梯度下降	误差最小化
error surface	误差曲面	梯度下降
local minimum	局部最小值	梯度下降
mean squared network error	网络均方误差	MSE
backpropagation / BP	反向传播	链式法则、权重更新
chain rule	链式法则	BP 推导
hidden layer	隐藏层	BP、多层网络、XOR
output layer	输出层	BP 输出层公式
winner-take-all	胜者全得	无监督竞争学习
Kohonen based learning network	Kohonen 学习网络	winner-take-all
Euclidean distance	欧氏距离	WTA 计算
unsupervised learning	无监督学习	WTA/Kohonen
prototype	原型向量	WTA 聚类结果
transducer	转换器	完整分类系统
feature extractor	特征提取器	完整分类系统
classifier	分类器	完整分类系统

考点索引

主题	可能题型	必须会写
Artificial neuron 人工神经元	定义/结构题	输入、权重、加权和、激活函数、阈值
McCulloch-Pitts neuron	简答/例题	用阈值单元实现 AND/OR
Perceptron 感知机	计算/简答	输出公式、权重更新、线性可分
XOR / exclusive-or 异或	证明/解释	单层感知机不能解决非线性可分问题
Sigmoid / sigmoidal function	公式题	$f(\sigma)=1/(1+e^{-\sigma})$, $f'=f(1-f)$
Gradient descent 梯度下降	简答/计算	沿负梯度方向更新以减小误差
Backpropagation / BP 反向传播	推导/公式	输出层和隐藏层权重更新
Winner-take-all 胜者全得	计算/简答	欧氏距离、winner、原型向输入靠近
Classification system 分类系统	简答	Transducer -> Feature extractor -> Classifier

深度学习背景

深度学习定义/动机

课件关键句：

```
Deep learning moves machine learning systems towards the discovery of multiple levels of representation.
```

考场答法：

```
深度学习通过多层网络自动发现多层表示。低层表示可组合成高层表示，因此适合图像、语音、自然语言等复杂任务。
```

课件点名例子

例子	要点
ImageNet	大规模图像分类，1.2M images, 1K categories, top-5 error
Image captioning / Show and Tell	根据图像生成自然语言描述
LeNet-5	Yann LeCun，支票/手写数字识别，CNN 早期代表
DeepMind Atari	deep reinforcement learning，输入图像映射到 action-value functions
Go	深度网络结合搜索在围棋中取得突破

深度学习相关人物

人物	课件信息
Yann LeCun	New York University；LeNet-5
Geoffrey Hinton	University of Toronto / Google；DNNresearch
Yoshua Bengio	University of Montreal
Andrew Ng	Stanford
Ilya Sutskever	Hinton 的 PhD student；DNNresearch
Alex Krizhevsky	Hinton 的 PhD student；DNNresearch

产业/新闻背景

信息	课件内容
公司投入	Google, Microsoft, Apple, IBM, Baidu 投资 deep learning
早期产品	面向消费者的 speech recognition
2012 新闻	New York Times front-page articles
Kaggle 药物发现	deep learning 用于 drug discovery competition
Google 收购	Google acqui-hired DNNresearch

ImageNet 数据

项目	课件信息
任务	ImageNet Challenge, top-5 classification
数据规模	1.2M images into 1K categories
图中类别例子	mite, container ship, motor scooter, leopard, grille, mushroom, cherry, Madagascar cat

Accuracy of ImageNet Top-5 Prediction：

年份	Classic CV	DNN
2010	71.8	-
2011	74.2	-
2012	73.8	83.6
2013	74.8	88.3
2014	-	93.3

补充：

```
GoogLeNet top-5 error: 6.7%
Human error: about 5%
```

LeNet-5 / US checks

课件用途：说明 CNN 可用于手写字符/支票识别。

LeNet-5 结构图：

```
Input: 32 x 32
C1: feature maps 6 @ 28 x 28      convolution
S2: feature maps 6 @ 14 x 14      subsampling
C3: feature maps 16 @ 10 x 10     convolution
S4: feature maps 16 @ 5 x 5       subsampling
C5: layer 120
F6: layer 84
Output: 10
```

点名信息：

```
LeNet-5 was invented by Prof. Yann LeCun from NYU-Courant.
```

Eric Richard Kandel

课件用途：从生物神经系统引出人工神经系统。

信息	内容
身份	American neuropsychiatrist
奖项	2000 Nobel Prize in Physiology or Medicine
研究	physiological basis of memory storage in neurons
方法	reductionist approach, one cell at a time
实验对象	sea slug Aplysia californica
原因	large nerve cells, amenable to experimental manipulation

人工神经元

结构

$$\textbf{inputs}: x_1, x_2, \ldots, x_n$$

$$\textbf{weights}: w_1, w_2, \ldots, w_n$$

$$\sigma = \sum_i w_i x_i$$

$$y = f(\sigma)$$

部件	作用
Input signals	输入特征
Weights	表示每个输入的重要性
Weighted sum	$\sigma = \sum_i w_i x_i$
Activation/threshold function	把加权和映射为输出
Bias/threshold	调整决策边界位置

标准答法

人工神经元接收多个输入信号，对每个输入乘以对应权重并求和，再通过激活函数或阈值函数产生输出。权重决定输入对输出的影响，`bias/threshold` 决定激活条件。

McCulloch-Pitts Neuron

定位：早期二值阈值神经元，用固定权重和 bias/threshold 实现逻辑函数。

课件图示

AND:

$$\textbf{net} = x + y - 2, \quad \textbf{output} = 1 \iff \textbf{net} \geq 0$$

OR:

$$\textbf{net} = x + y - 1, \quad \textbf{output} = 1 \iff \textbf{net} \geq 0$$

AND 的真值：

x	y	x+y-2	output
1	1	0	1
1	0	-1	-1
0	1	-1	-1
0	0	-2	-1

OR 的真值：

x	y	x+y-1	output
1	1	1	1
1	0	0	1
0	1	0	1
0	0	-1	-1

可用答题句

`McCulloch-Pitts 神经元说明 AND、OR 等布尔逻辑可以由带 bias 的阈值单元实现，因此神经元模型可以表达简单逻辑运算。`

Perceptron

输出公式

课件使用 bipolar 输出：

$$y = \begin{cases} 1, & \sum_i w_i x_i \geq t \\ -1, & \sum_i w_i x_i < t \end{cases}$$

等价写法：把 threshold 吸收到 bias 中：

$$y = \mathbf{sign}\left(\sum_i w_i x_i\right)$$

权重更新

$$\Delta w_i = r\left(d - \mathbf{sign}\left(\sum_i w_i x_i\right)\right) x_i$$

也可写成：

$$\Delta w_i = r(d - y) x_i$$

符号	含义
`r`	learning constant / learning rate
`d`	desired output
`y`	actual output, 即 $\mathbf{sign}(\sum_i w_i x_i)$
`xi`	第 i 个输入

三种情况

情况	更新
`d = y`	不更新
`y = -1`, `d = 1`	$\Delta w_i = 2 r x_i$，权重增加
`y = 1`, `d = -1`	$\Delta w_i = -2 r x_i$，权重减少

计算题模板

1. 计算 $\sigma = \sum_i w_i x_i$。
2. 由 $y = \mathbf{sign}(\sigma)$ 得到输出。
3. 比较 y 和 d。
4. 用 $\Delta \mathbf{w} = r(d - y)\mathbf{x}$ 更新。
5. 写出新权重 $\mathbf{w}_{new} = \mathbf{w}_{old} + \Delta \mathbf{w}$。

课件例子格式

感知机分类数据集：

x1	x2	Output
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1

初始权重：

$$\mathbf{w} = [0.75, 0.5, -0.6], \quad r = 0.2$$

若样本 `x = [9.4, 6.4, 1.0]`，`d = -1`：

$$\sigma = 0.75 \times 9.4 + 0.5 \times 6.4 - 0.6 \times 1 = 9.65$$

$$y = f(\sigma) = 1$$

$$\Delta \mathbf{w} = 0.2(-1 - 1)\mathbf{x} = -0.4\mathbf{x}$$

$$\mathbf{w}_{new} = [0.75, 0.5, -0.6] - 0.4[9.4, 6.4, 1.0] = [-3.01, -2.06, -1.00]$$

Perceptron 的几何意义

二输入感知机的决策边界是直线：

$$w_1 x_1 + w_2 x_2 + b = 0$$

课件例子收敛后的边界：

$$-1.3x_1 - 1.1x_2 + 10.9 = 0$$

标准答法：

> 单层感知机本质上学习一个线性决策边界，因此只能解决线性可分问题。

XOR 局限

XOR 真值表

x1	x2	XOR
1	1	0
1	0	1
0	1	1
0	0	0

不可解原因

XOR 的正负样本不能被一条直线分开，因此单层感知机无法表示。

课件式不等式证明

若输出 1 要满足 `w1 x1 + w2 x2 > t`，输出 0 要满足 `< t`：

$$w_1 + w_2 < t \quad \text{from } (1,1) \to 0$$

$$w_1 > t \quad \text{from } (1,0) \to 1$$

$$w_2 > t \quad \text{from } (0,1) \to 1$$

$$0 < t \quad \text{from } (0,0) \to 0$$

由 `0 < t`、`w1 > t`、`w2 > t` 得 `w1 + w2 > 2t > t`，与 `w1+w2 < t` 矛盾。

标准答法

> XOR 是非线性可分问题，单层感知机只能形成线性决策边界，因此不能解决 XOR。要解决 XOR，需要引入隐藏层或非线性变换。

Sigmoid Function

公式

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

导数

$$f'(\sigma) = f(\sigma)(1 - f(\sigma))$$

如果 $O = f(\sigma)$:

$$\frac{dO}{d\sigma} = O(1 - O)$$

为什么重要

`sigmoid` 是连续可导函数，可用于梯度下降和反向传播；硬阈值函数不可导，不适合用梯度法训练多层网络。

Gradient Descent

核心思想

为了最小化误差，权重应沿误差函数负梯度方向更新。

图中要点

```
Error surface: 误差随权重变化形成的曲面/曲线
Old weights -> New weights: 按学习常数移动
Learning constant: 控制每次更新步长
Local minimum: 梯度下降可能停在局部最小值
```

Gradient 含义

梯度是由各变量偏导数组成的向量，指向函数增长最快的方向。
最小化误差时沿负梯度方向更新。

模板

$$w_{\text{new}} = w_{\text{old}} - r\frac{\partial Error}{\partial w}$$

符号	含义
`Error`	误差函数
`r`	learning rate
$\partial Error/\partial w$	权重方向上的梯度

Mean Squared Error

误差函数

$$Error = \frac{1}{2}\sum_i (d_i - O_i)^2$$

符号	含义
`d_i`	第 i 个输出节点的期望输出
`O_i`	第 i 个输出节点的实际输出

对输出求导

$$\frac{\partial Error}{\partial O_i} = -(d_i - O_i)$$

Backpropagation

网络记号

课件三层结构:

记号	含义
k	input layer 节点
j	hidden layer 节点
i	output layer 节点
w_{kj}	input k -> hidden j
w_{ji}	hidden j -> output i

注意：不同教材下标方向可能相反，考试按课件公式写即可。

输出层权重更新

从 hidden node `j` 到 output node `i`：

$$\Delta w_{ji} = r(d_i - O_i)O_i(1 - O_i)O_j$$

记忆形式：

$$\Delta w_{ji} = r \cdot \text{output\_error}_i \cdot \text{sigmoid\_derivative}_i \cdot O_j$$

其中：

$$\text{output\_error}_i = d_i - O_i$$

$$\text{sigmoid\_derivative}_i = O_i(1 - O_i)$$

输出层推导链式法则

$$\frac{\partial Error}{\partial w_j} = \frac{\partial Error}{\partial O_i} \cdot \frac{\partial O_i}{\partial w_j}$$

$$\frac{\partial Error}{\partial w_j} = -(d_i - O_i)O_i(1 - O_i)O_j$$

负梯度更新：

$$\Delta w_j = r(d_i - O_i)O_i(1 - O_i)O_j$$

隐藏层权重更新

从 input node `k` 到 hidden node `j`：

$$\Delta w_{kj} = rO_j(1 - O_j)O_k \sum_i [w_{ji}O_i(1 - O_i)(d_i - O_i)]$$

记忆形式：

$$\Delta w_{kj} = r \cdot \text{hidden\_derivative}_j \cdot O_k \cdot \text{weighted\_sum\_of\_output\_errors}$$

```
z1 = W1 x + b1
h = activation(z1)

z2 = W2 h + b2
y_hat = activation(z2)

L = loss(y_hat, y)
则有：
δ2 = ∂L/∂z2  (∂L/∂z2 = ∂L/∂y_hat * ∂y_hat/∂z2)
∂L/∂W2 = δ2 * h
∂L/∂b2 = δ2

往前传到隐藏层：
希望求：δ1 = ∂L/∂z1 = ∂L/∂h * ∂h/∂z1
∂L/∂h = W2^T δ2
因此：
δ1 = W2^T δ2 * activation'(z1)
```

BP 标准步骤

```
1. Forward pass: 输入样本，逐层计算所有节点输出。
2. Compute output error: 用 d_i - O_i 计算输出误差。
3. Update output weights: 用输出层公式更新 hidden->output 权重。
4. Backpropagate error: 将输出层误差按连接权重传回隐藏层。
5. Update hidden weights: 用隐藏层公式更新 input->hidden 权重。
6. Repeat until error small enough or max iterations reached.
```

BP 标准答法

反向传播通过链式法则计算误差函数对各层权重的偏导。先前向传播得到输出，再计算输出误差；输出层权重直接根据输出误差更新，隐藏层误差由后一层误差按连接权重反传得到。权重沿负梯度方向调整，从而逐步减小网络误差。

NETtalk

课件例子：输入字母窗口，输出发音特征。

部分	数量/含义
Inputs	`7 x 29` inputs
Hidden	`80` hidden units
Outputs	`26` output units
输入编码	26 letters + 3 punctuation/spaces
输出编码	articulation features, stress, syllable boundaries

答题用途：说明神经网络可用于 sequence/text-to-speech 类任务。

用隐藏层解决 XOR

单层感知机不能解决 XOR，多层网络可以。

隐藏层可以组合多个线性边界，形成非线性决策区域，因此多层感知机可以表示 XOR。

Winner-take-all / Kohonen Learning

基本思想

Winner-take-all 是无监督学习：对输入向量 `X`，计算每个竞争节点的激活或距离，选出 winner，只更新 winner 的权重，使其更接近输入。

欧氏距离

课件公式写的是欧氏距离；例题计算时实际比较平方距离，因开根号不改变大小，分类 winner 相同。

$$\|X-W\|=\sqrt{\sum_i (x_i-w_i)^2}$$

比较 winner 时可直接比较平方距离：

$$\sum_i (x_i-w_i)^2$$

更新公式

$$W_{\text{new}}=W_{\text{old}}+c(X-W_{\text{old}})$$

等价：

$$\Delta W=c(X-W)$$

符号	含义
`X`	输入向量
`W`	winner 节点的权重/原型向量
`c`	learning constant

计算题模板

1. 对每个节点 A,B,... 计算 $\|X-W_A\|$, $\|X-W_B\|$。
2. 距离最小者为 winner。
3. 只更新 winner：$W_{new}=W_{old}+c(X-W_{old})$。
4. 非 winner 权重不变。

课件例子

无监督数据集：

x1	x2	Output
1.0	1.0	unlabeled
9.4	6.4	unlabeled
2.5	2.1	unlabeled
8.0	7.7	unlabeled
0.5	2.2	unlabeled
7.9	8.4	unlabeled
7.0	7.0	unlabeled
2.8	0.8	unlabeled
1.2	3.0	unlabeled
7.8	6.1	unlabeled

Kohonen network 初始权重：

$$w_{1A}=7,\quad w_{2A}=2\quad\Rightarrow\quad W_A=(7,2)$$
$$w_{1B}=2,\quad w_{2B}=9\quad\Rightarrow\quad W_B=(2,9)$$

初始：

$$W_A=(7,2),\quad W_B=(2,9),\quad c=0.5,\quad X=(1,1)$$

距离，按课件例题比较平方距离：

$$\|X-W_A\|^2=(1-7)^2+(1-2)^2=37$$
$$\|X-W_B\|^2=(1-2)^2+(1-9)^2=65$$

所以 A 是 winner：

$$W_{A,\text{new}}=(7,2)+0.5((1,1)-(7,2))=(4,1.5)$$

第二个点：

$$X=(9.4,6.4),\quad W_A=(4,1.5),\quad W_B=(2,9)$$
$$\|X-W_A\|^2=(9.4-4)^2+(6.4-1.5)^2=53.17$$
$$\|X-W_B\|^2=(9.4-2)^2+(6.4-9)^2=60.15$$

所以 A 是 winner：

$$W_{A,\text{new}}=(4,1.5)+0.5((9.4,6.4)-(4,1.5))=(6.7,4)$$

图中结论：

```
After 10 iterations, two prototypes move toward two clusters of data.
```

标准答法

`Winner-take-all` 学习是无监督的，因为没有给定类别标签。系统根据最大激活或最小距离自动选择 `winner`，并将 `winner` 的权重向当前输入移动，从而形成表示数据分布的原型。

完整分类系统

课件结构：

```
Raw data -> Transducer -> Feature Extractor -> Classifier -> Class
```

阶段	作用
Raw data space	原始数据
Transducer	把外部信号转为可处理数据
Feature extractor	提取特征，形成 pattern/features space
Classifier	根据特征输出类别

神经网络任务类型

课件列出：

```
Classification
Pattern recognition
Memory recall
Prediction
Optimization
Noise filtering
```

标准答法：

> 神经网络可用于分类、模式识别、记忆召回、预测、优化和噪声过滤等任务，因为它能从样本中学习输入与输出或输入结构之间的关系。

常见考题答题模板

Q1: 人工神经元由哪些部分组成？

人工神经元由输入、权重、加权求和和激活/阈值函数组成。输入 x_i 与权重 w_i 相乘并求和得到 $\sigma = \sum_i w_i x_i$，再经过函数 $f(\sigma)$ 得到输出。bias 或 threshold 用于调整激活边界。

Q2: 感知机如何学习？

感知机先计算 $y = \mathbf{sign}(\sum_i w_i x_i)$，再与期望输出 d 比较。如果 $d = y$，则不更新；否则按 $\Delta w_i = r(d-y)x_i$ 调整权重。该规则使错误分类样本推动决策边界向正确方向移动。

Q3: 为什么单层感知机不能解决 XOR？

> 单层感知机只能学习线性决策边界，而 XOR 的正负样本在线性空间中不可被一条直线分开，因此不存在一组权重和阈值能实现 XOR。引入隐藏层后可组合多个线性边界，从而表示非线性分类。

Q4: sigmoid 为什么适合 BP？

sigmoid 连续可导，且导数可写为 $f'(\sigma) = f(\sigma)(1-f(\sigma))$，便于使用链式法则计算梯度。硬阈值函数不可导，因此不适合梯度下降训练多层网络。

Q5: BP 的核心思想是什么？

> BP 使用链式法则将输出误差逐层反向传播，计算误差函数对每个权重的偏导，并沿负梯度方向更新权重，从而最小化网络误差。

Q6: Winner-take-all 为什么是无监督学习？

> `Winner-take-all` 没有使用类别标签。它根据输入与各节点权重的相似度或距离自动选出 `winner`，并只把 `winner` 的权重向输入方向移动，因此属于无监督竞争学习。

易错点

易错点	正确写法
感知机输出	本课件用 `1/-1` bipolar 输出，不是 `1/0`
阈值	可写为 threshold `t`，也可吸收到 bias 中
XOR	不是“感知机完全没用”，而是单层感知机不能解非线性可分问题
sigmoid 导数	$f'(\sigma) = f(\sigma)(1-f(\sigma))$
MSE 前面的 `1/2`	为了求导时抵消平方项的 2
BP 更新方向	沿负梯度方向减小误差
Winner-take-all	只更新 winner，非 winner 不更新
欧氏距离	比较距离时可用平方和，不一定开根号

03 Deep Learning - 考场资料

来源：`03Deep learning.pdf`，148 页。定位：深度学习主章，内容横跨表示学习、目标函数、CNN、中文分词、HMM/CRF、词向量、RNN/LSTM/GRU、Transformer、生成模型、应用、SNN。已按页面图像核对。

英文题目识别词

English in exam	中文定位	看到后去找
shallow architecture	浅层结构	Shallow vs Deep
deep architecture	深层结构	Shallow vs Deep, feature hierarchy
multiple levels of representation	多层表示	深度学习动机
representation learning	表示学习	Representation, embeddings
objective function / loss function	目标函数/损失函数	Loss, cross entropy, softmax
squared error	平方误差	Loss functions
cross entropy	交叉熵	binary/logistic/softmax
logistic classification	逻辑分类	sigmoid, logit
logit / log odds	对数几率	logistic classification
softmax regression	softmax 回归	多分类概率
weight decay	权重衰减	L2 regularization
regularization	正则化	防止过拟合
basis expansion	基函数扩展	特征变换
convolution	卷积	CNN
feature map	特征图	CNN
pooling	池化	CNN
max pooling / average pooling / k-max	最大/平均/k-max 池化	pooling layers
mini-batch	小批量训练	training tips
Chinese word segmentation	中文分词	CWS/HMM/CRF/deep CWS
word boundary ambiguity	分词歧义	combination/overlap/global ambiguity
unknown words	未登录词	CWS 难点
character-based tagging	基于字的标注	B/E/I/S
HMM / Hidden Markov Model	隐马尔可夫模型	HMM 五元组、三大问题
forward algorithm	前向算法	evaluation
backward procedure	后向算法	parameter estimation
Viterbi algorithm	维特比算法	decoding
EM / expectation maximization	期望最大化	HMM 参数估计
CRF / Conditional Random Fields	条件随机场	HMM vs CRF, feature templates
label bias problem	标注偏置问题	CRF 动机
feature template	特征模板	CRF/CWS
pre-training	预训练	unsupervised pre-training, embeddings
word embeddings	词向量	Word2Vec, CBOW, Skip-gram
CBOW / Skip-gram	连续词袋/跳字模型	Word2Vec
Huffman tree	哈夫曼树	hierarchical softmax
RNN / recurrent neural network	循环神经网络	sequence modeling
BPTT / RTRL	通过时间反传/实时循环学习	RNN training
vanishing gradient / blow up	梯度消失/爆炸	RNN 问题
LSTM	长短期记忆网络	gates, cell state
GRU	门控循环单元	update gate/reset gate
contextualized word representation	上下文化词表示	ELMo
attention	注意力机制	Transformer
Transformer / GPT / BERT	Transformer/GPT/BERT	attention, MLM
masked language model	掩码语言模型	BERT
prompt-based learning	基于提示的学习	prompt/template/verbalizer
mixture-of-experts	专家混合	gating/routing
VAE / variational autoencoder	变分自编码器	ELBO, reparameterization
diffusion model	扩散模型	denoise/generative process
GAN / generative adversarial net	生成对抗网络	generator vs discriminator
adversarial example	对抗样本	computer vision vulnerability
SNN / spiking neural network	脉冲神经网络	event-driven, energy-efficient

English in exam	中文定位	看到后去找
LIF / Leaky Integrate-and-Fire	泄露积分发放神经元	SNN
IBM TrueNorth	IBM TrueNorth 芯片	neurosynaptic cores, low energy
K-means	K 均值聚类	最后一页

章节主线

```
为什么需要深度学习
-> 表示学习和深层特征
-> 目标函数与 softmax
-> 正则化与模型复杂度
-> CNN
-> 中文分词
-> HMM/CRF 序列标注
-> 深度中文分词与词向量
-> RNN/LSTM/GRU/Transformer
-> 生成模型和多模态应用
-> SNN 与神经形态硬件
```

深度学习动机

Shallow vs. Deep Architecture

类型	英文关键词	中文答法
Shallow architectures	1 or 2 layers, SVM	层数少，依赖人工特征或简单投影，难以表示复杂层级结构。
Deep architectures	multiple-level feature projections	多层结构能逐层组合低层特征形成高层特征，实现更经济的表示。

标准答法：

深度学习通过多层非线性变换学习特征层级。低层特征可以组合成中层特征，高层特征再组合成更抽象的表示，因此深层网络比浅层网络更适合图像、语音、自然语言等复杂任务。

Support Vector Machine (SVM) 图示补充

`Support vector machine / SVM` 是课件用来代表 shallow architecture 的典型分类器。图里有两个核心词：

英文词	中文含义	考场理解
`optimal hyperplane`	最优分隔超平面	在特征空间中把两类样本分开的一条线/一个平面。
`maximum margin`	最大间隔	分隔面到两边最近训练样本的距离尽量大，使分类边界更稳健。
`support vectors`	支持向量	离分隔边界最近、真正决定边界位置的样本点。
`kernel`	核函数	当原空间线性不可分时，把样本隐式映射到更高维空间，使非线性边界可由线性方法表示。

一句话答法：SVM 通过寻找最大间隔分隔超平面进行分类，分类边界主要由 support vectors 决定；它可以用 kernel 处理非线性分类，但仍通常依赖人工特征和较浅的表示，因此课件把它放在 deep learning 的对照背景里。

为什么 representation 很重要

课件原意：

```
Representation is very important.
If the representation is robust enough, the contribution of classifier is limited.
```

中文答法：

表示决定了后续分类器能否轻松完成任务。如果表示足够鲁棒，不同类别在表示空间中已经容易分开，那么分类器本身的贡献会变小；深度学习的优势正是自动学习任务相关的表示。

DBN 与 Autoencoder

模型	英文	重点
Deep Belief Network	DBN	生成式图模型/深层神经网络，由多层 latent variables/hidden units 组成；层内无连接；与 Hinton、RBM 相关。
Autoencoder	autoassociator / Diabolo network	学习压缩、分布式表示，常用于 dimensionality reduction；结构为 encode -> hidden -> decode；与 Bengio 相关。

三个核心组成

课件列出：

```
Three core components of a DL framework:
1. Objective Functions
2. Learning rules
3. Architectures
```

答题句：

一个深度学习框架通常包括目标函数、学习规则和网络结构。目标函数定义优化目标，学习规则决定如何更新参数，网络结构决定信息如何在层间传播和表示。

Loss Functions

Squared Error

$$Error = \frac{1}{2}\sum_{i=1}^{n}(t_i - y(x_i))^2$$

用途：常用于回归或连续输出任务。

Cross Entropy

课件给出形式：

$$Error = -\sum_{i=1}^{n} t_i \ln y(x_i)$$

二分类更完整形式：

$$-\sum_{i=1}^{n}[t_i \ln y(x_i) + (1-t_i)\ln(1-y(x_i))]$$

答题句：

交叉熵来自最大似然估计。对分类问题，最小化负对数似然等价于最大化正确类别的概率。

Binary / Logistic Classification

Binary likelihood

对数据集 x_i, t_i，$t_i \in 0, 1$：

$$\prod_{i=1}^{n} y(x_i)^{t_i}(1-y(x_i))^{1-t_i}$$

取对数：

$$\sum_{i=1}^{n}[t_i \ln y(x_i) + (1-t_i)\ln(1-y(x_i))]$$

训练时最小化负对数似然，即 cross entropy。

Logistic hypothesis

sigmoid:确保线性函数任意范围的输出映射到 0-1 的概率

$$y(x_i) = \frac{1}{1+\exp(-f_\theta(x_i))}$$

通常线性函数的输出：

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$$

Logistic cost 损失函数，二元交叉熵

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[t_i \ln y(x_i) + (1-t_i)\ln(1-y(x_i))]$$

Logit / Log Odds

$$g(y(x_i)) = \ln\frac{y(x_i)}{1-y(x_i)} = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$$

答题句：

逻辑回归并不是直接用线性函数输出概率，而是用线性函数建模 log odds，再通过 sigmoid 映射为 $[0,1]$ 概率。

Softmax Regression

多分类概率

给定输入 x_i，softmax 输出 k 维概率向量：

$$y(x_i) = \begin{bmatrix} p(t_i = 1|x_i;\theta) \\ p(t_i = 2|x_i;\theta) \\ \vdots \\ p(t_i = k|x_i;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} \exp(f_{\theta_j}(x_i))} \begin{bmatrix} \exp(f_{\theta_1}(x_i)) \\ \exp(f_{\theta_2}(x_i)) \\ \vdots \\ \exp(f_{\theta_k}(x_i)) \end{bmatrix}$$

即：

$$p(t_i = j|x_i;\theta) = \frac{\exp(f_{\theta_j}(x_i))}{\sum_{\ell=1}^{k} \exp(f_{\theta_\ell}(x_i))}$$

Softmax loss

I{ti=j}是指示函数，i 的真实类别为 j 时取 1，其他情况为 0，保证样本i真实类别的项j贡献损失

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} I\{t_i = j\} \ln \frac{\exp(f_{\theta_j}(x_i))}{\sum_{\ell=1}^{k} \exp(f_{\theta_\ell}(x_i))}$$

Gradient

只有真实类别的项产生梯度

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} x_i \left[I\{t_i = j\} - p(t_i = j|x_i;\theta) \right]$$

答题句：

softmax regression 是 logistic regression 的多分类扩展。它把每个类别的 score 指数化并归一化，得到和为 1 的类别概率。

Weight Decay / Regularization

Weight decay loss

加一个正则项，防止参数太大，增强泛化

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} I\{t_i = j\} \ln \frac{\exp(f_{\theta_j}(x_i))}{\sum_{\ell=1}^{k} \exp(f_{\theta_\ell}(x_i))} + \frac{\lambda}{2} \sum_{i=1}^{n} \sum_{j=1}^{k} \theta_{ij}^2$$

Derivative

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} x_i \left[I\{t_i = j\} - p(t_i = j|x_i;\theta) \right] + \lambda \theta_j$$

答题句：

weight decay 通过在损失函数中加入参数平方惩罚项，抑制过大的权重，从而降低模型复杂度和过拟合风险。

Polynomial Curve Fitting / Regularization

多项式模型

$$y(x, w) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

误差

最小二乘损失，M 为多项式阶数

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2$$

正则化误差

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2 + \frac{\lambda}{2} \|w\|^2$$

图页	考点
various orders M	M 太小欠拟合，M 太大过拟合
coefficients	高阶多项式可能出现很大的参数
various λ	正则化强度影响模型复杂度

Bayesian Probabilities

Bayes theorem:

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)}$$

记忆:

$$posterior \propto likelihood \times prior$$

答题句:

贝叶斯方法通过先验约束参数，用观测数据的似然更新得到后验分布；在曲线拟合中可把正则化理解为对参数的先验约束，相当于偏好小参数

CNN / Convolution / Pooling

Convolution

场景	图示含义
image	卷积窗口在二维图像上 shift
speech/language	一维窗口沿 time 方向 shift

答题句:

卷积通过共享参数的局部窗口在输入上滑动，提取局部模式；在不同位置使用同一组参数，因此能减少参数量并增强平移不变性。

Feature Maps

feature map 是某个卷积核在所有位置上的响应图。多个卷积核会产生多个 feature maps，用于表示不同类型的局部特征。

Pooling Layers

课件列出:

```
Pooling methods:
Max (normally better)
Min
Average (generally bad choice)
k-max
```

答题句:

pooling 将局部区域压缩成较小表示，降低空间/时间分辨率和参数量，并增强对小位移的鲁棒性。课件中特别指出 max pooling 通常更好，average pooling 一般不是好选择。

Training Tips

项	课件建议
Weights	You can't use 'macro parameter character #' in math mode 或 You can't use 'macro parameter character #' in math mode
Bias	last layer: $[-0.2, +0.2]$；otherwise: $[0, -1]$
Learning rate	0.05 是好起点；参数越多，learning rate 越小
Hidden units	足够大时，hidden units 数量对性能影响有限
Update	mini-batch 更新网络参数

LeNet-5

```
Input 32x32
C1: 6 feature maps @ 28x28
S2: 6 feature maps @ 14x14
C3: 16 feature maps @ 10x10
S4: 16 feature maps @ 5x5
C5: 120
F6: 84
Output: 10
```

Chinese Word Segmentation

为什么难

课件要点：

中文分词虽然已有进展，但仍是困难问题；没有方法能在不经过大量训练数据微调的情况下可靠处理陌生文本，更不用说混合不同主题、体裁和地域的开放文本。

Word Boundary Ambiguities

英文	中文	例子
Combination Ambiguity	组合歧义	以/我/个人/的名义 vs 他/一/个/人/在家
Overlap Ambiguity	交集歧义	从/小学/到/中学 vs 从小/学/计算机
Global Ambiguity	全局歧义	美国/会/采取行动 vs 美/国会/采取行动

Unknown Words

类型：

类型	例子
数量、时间	2012年5月22日、一市斤、356克
人名、机构名、地名	李维汉、阿凡提、上海浦东张江镇
外文翻译及缩写	阿诺德-施瓦辛格、萨默维尔、FDA
专用名称缩写	复旦、中航、央行
新词	安家费、三通、菜鸟、八卦

Word Formation

类型	例子
重叠词	AA: 爸爸、宝宝；AABB: 大大咧咧
派生词	前缀+词根: 老虎；词根+后缀: 椅子；中缀: 对得起
词转化为短语	鞠躬 -> 鞠个躬
短语转化为词	土地改革 -> 土改；地下铁道 -> 地铁
截段/综合简缩	复旦大学 -> 复旦；安全理事会 -> 安理会

Character-based Tagging

课件例子：

```
Sentence: 从 小 学 计 算 机 。
Tags:     B  E  S  B  I  E  S
```

标签含义：

Tag	Meaning	中文
B	Begin	词首
I	Inside	词中
E	End	词尾
S	Single	单字词

答题句：

character-based tagging 把分词转化为序列标注问题，对已知词和未知词统一处理，不需要额外的未登录词例外机制。

HMM - 考场可用版

HMM 这块不要在考场重新推一大堆公式。先按题型定位：

英文题目触发词	中文任务	你该写什么
general form / five-tuple	HMM 组成	写 (S, K, Π, A, B) 和每个符号含义
evaluation / probability of observation / $P(O \mu)$		求观测序列概率
decoding / best state sequence	求最可能隐藏状态序列	写 Viterbi：把 forward 的求和换成 max，并记录路径
learning / parameter estimation / EM	学参数	写用期望次数重新估计 Π, A, B
HMM for speech recognition	语音识别	写 MFCC/谱向量 + HMM 状态序列 + Viterbi

HMM 一句话

Hidden Markov Model 是一个序列概率模型：真实状态 X 看不见，只能看到观测 O；状态按 Markov 过程转移，每一步状态产生一个观测。

中文答题句：

HMM 用隐藏状态序列解释观测序列。模型假设当前状态主要依赖前一状态，观测由隐藏状态产生，因此适合语音识别、词性标注、中文分词等序列建模任务。

Five-tuple

$$\mu = (S, K, \Pi, A, B)$$

符号	英文	中文	记忆
S	states	隐藏状态集合	系统内部真实状态
K	output alphabet	观测符号集合	你能看到的东西
Π	initial probabilities	初始状态概率	一开始在哪个状态
A	transition probabilities	状态转移概率	状态怎么跳
B	emission probabilities	发射概率	状态怎么产生观测

课件的发射概率写法：

$$P(O_t = k | X_t = s_i, X_{t+1} = s_j) = b_{ijk}$$

基础解释：B 表示隐藏状态/转移产生观测符号的概率。若题目给出课件这种 b_{ijk} 写法，就按“从 s_i 到 s_j 时产生观测 k”理解。

三个基本问题

问题	英文	输入	输出	算法
Evaluation	How likely is observation?	模型 μ + 观测 O	$P(O	\mu)$
Decoding	Best state sequence	模型 μ + 观测 O	最可能状态序列 $\hat{X}$	Viterbi
Learning	Best model parameters	观测 O	Π, A, B	EM / Baum-Welch

标准答法：

HMM 有三个基本问题：evaluation 计算观测序列在给定模型下的概率；decoding 寻找最能解释观测的隐藏状态序列；learning 根据观测数据估计模型参数。它们分别对应 forward/backward、Viterbi 和 EM/Baum-Welch。

Forward Algorithm：求概率，核心是“加起来”

用途：计算 $P(O|\mu)$。

直觉：到达某个状态的概率 = 从所有前一状态转移过来的概率总和。

变量含义：

符号	含义
$\alpha_j(t)$	看完前 t 个观测后，当前在状态 s_j 的总概率
a_{ij}	从状态 s_i 转移到 s_j 的概率
b_{ijo_t}	从 s_i 到 s_j 时产生观测 o_t 的概率

完整答题公式：

$$\alpha_i(1) = \pi_i$$

$$\alpha_j(t+1) = \sum_i \alpha_i(t) a_{ij} b_{ijo_t}$$

$$P(O|\mu) = \sum_i \alpha_i(T)$$

人话解释：$\sum_i$ 表示“所有可能前驱状态都贡献概率”，所以 forward 算的是观测序列的总概率，不关心哪条路径最好。

考场步骤：

1. 初始化：$\alpha_i(1) = \pi_i$。
2. 递推：每个时间步对所有前一状态求和。
3. 结束：把最后一列所有状态概率相加得到 $P(O|\mu)$。

必须记住：

```
Forward = sum over previous states
```

Backward Procedure：从后往前算概率

用途：也可计算 $P(O|\mu)$，并且常用于参数估计。

直觉：站在当前状态，看未来观测还能出现的概率。

变量含义：

符号	含义
$\beta_i(t)$	已经处在状态 s_i，从 t 之后还能生成剩余观测的概率

完整答题公式：

$$\beta_i(T) = 1$$

$$\beta_i(t) = \sum_j a_{ij} b_{ij o_t} \beta_j(t+1)$$

人话解释：backward 和 forward 方向相反，但本质仍然是“把未来所有可能路径的概率加起来”。

考场步骤：

1. 初始化：最后时刻之后设为 1。
2. 从后往前递推。
3. 可与 forward 结合计算某状态被访问的后验概率。

Viterbi Algorithm：求最佳路径，核心是“取最大”

用途：decoding，求最可能状态序列。

直觉：到达每个状态时，只保留目前最好的那条路径，同时记下它从哪里来。

变量含义：

符号	含义
$\delta_j(t)$	到第 t 步并停在 s_j 时，最佳路径的概率
$\psi_j(t)$	最佳路径到达 s_j 时来自哪个前驱状态

完整答题公式：

$$\delta_i(1) = \pi_i$$

$$\delta_j(t+1) = \max_i \delta_i(t) a_{ij} b_{ij o_t}$$

回溯指针：

$$\psi_j(t+1) = \arg\max_i \delta_i(t) a_{ij} b_{ij o_t}$$

$$\hat{X} = \text{backtrace}\left(\arg\max_i \delta_i(T)\right)$$

人话解释：Viterbi 不把所有路径加起来，只保留每个状态当前最好的路径；最后通过 ψ 回溯出整条隐藏状态序列。

考场步骤：

1. 初始化每个状态概率。
2. 对每个新位置，计算从所有前一状态过来的分数。
3. 只保留最大值，并记录 backpointer。
4. 结束时找最大终点，沿 backpointer 回溯出完整路径。

必须记住：

```
Forward = sum
Viterbi = max + backpointer
```

Soft Drink Machine 例子怎么用

这个例子可以按两层使用：概念题先识别 HMM 三个元素；计算题再套 forward 或 Viterbi 的递推表。

HMM 元素	例子
hidden states	CP = Cola Pref, IP = Iced Tea Pref
observations	Cola, Iced tea, Lemonade
initial probabilities	$\pi(CP) = 0.4$, $\pi(IP) = 0.6$
transition probabilities	CP/IP 偏好状态之间转移
emission probabilities	某偏好状态下买某饮料的概率

观测序列：

$$O = (cola,\ iced\ tea,\ lemonade)$$

Forward 问题：求这个观测序列出现的总概率。

步骤	该做什么	例子里的数值线索
初始	初始概率乘第一个观测概率	到 CP 为 0.24；到 IP 为 0.06
递推	对每个目标状态，把所有前驱路径加起来	第二步到 CP 约 0.019；到 IP 约 0.074
结束	最后一列两个状态相加	约 $\mathbf{0.0148 + 0.008 = 0.023}$

Viterbi 问题：求最可能的偏好状态序列。

步骤	该做什么	例子里要看的点
初始	每个状态只保留当前最大路径概率	CP 起点更大
递推	对每个目标状态取最大前驱	每一格记录 max 和 backpointer
回溯	从最后最大状态沿 backpointer 回去	得到一条最可能路径

考场提醒：如果题目问 probability / likelihood，用 forward，把路径加起来；如果题目问 best path / most likely state sequence，用 Viterbi，取最大并回溯。

Parameter Estimation：学参数，核心是“期望次数 / 总次数”

考试如果问 parameter estimation，不要展开长公式，抓住这三句话：

1. 用 forward-backward 计算每个状态在每个时间出现的后验概率。
2. 用这些后验概率得到“期望初始次数”“期望转移次数”“期望发射次数”。
3. 用期望次数重新估计 $\mathbf{\Pi}, \boldsymbol{A}, \boldsymbol{B}$，这就是 EM(expectation maximization)/Baum-Welch 思想。

核心后验概率：

$$\gamma_i(t) = P(X_t = i|O, \mu)$$

$$p_t(i, j) = P(X_t = i, X_{t+1} = j|O, \mu)$$

其中 $\gamma_i(t)$ 表示“第 t 个位置属于状态 i 的软计数”，$p_t(i, j)$ 表示“第 t 步从 i 转到 j 的软计数”。

常见更新公式：

$$\hat{\pi}_i = \gamma_i(1)$$

$$\hat{a}_{ij} = \frac{\sum_t p_t(i, j)}{\sum_t \gamma_i(t)}$$

$$\hat{b}_{ijk} = \frac{\sum_{t:o_t=k} p_t(i, j)}{\sum_t p_t(i, j)}$$

参数更新记忆：

参数	中文解释
$\hat{\pi}_i$	序列一开始处在状态 i 的期望概率
$\hat{a}_{ij}$	从 i 到 j 的期望转移次数 / 从 i 出发的期望总次数
$\hat{b}_{ijk}$	从 i 到 j 且观测到 k 的期望次数 / 从 i 到 j 的期望总次数

一句话：

EM 通过当前模型估计隐藏状态的期望分布，再用这些期望统计量更新模型参数，使 $P(O|\mu)$ 不下降。

HMM for Speech Recognition

答题框架：

1. 语音信号先经过 Fourier transform / MFCC 等方法变成 spectral vectors。
2. HMM 的隐藏状态可以是 phoneme、syllable 或 word。
3. 观测是声学特征序列，隐藏状态是语言单元序列。
4. 用 Viterbi 找最可能的词/音素路径。

标准答法：

语音识别中，连续语音被转换为 MFCC 等声学特征序列。HMM 用隐藏状态表示音素、音节或词，用发射概率描述状态产生声学观测的可能性，用转移概率描述语言单元之间的连接。识别时通常用 Viterbi 算法寻找最可能的隐藏状态序列。

连续语音识别补充：

英文词	中文含义	怎么写进答案
`connected word recognition`	连续词识别	输入不是一个孤立单词，而是一串连续发音的词。
`word boundary`	词边界	语音里词与词之间的边界通常不可直接观察，需要模型推断。
`spectral vectors`	频谱特征向量	waveform 经过 Fourier/MFCC 等处理后得到的观测序列。

HMM 处理连续词的思路：把词或音素的 HMM 按语言模型/词典连接成候选路径，观测序列是连续的声学特征，Viterbi 在所有候选隐藏状态路径中找概率最大的路径；这个路径同时给出识别出的词序列和隐含的边界位置。

CRF - 考场可用版

CRF 部分重点不是公式推导，而是：它为什么比 HMM 更适合序列标注，怎样用 feature templates 做中文分词。

CRF 一句话

Conditional Random Fields 是判别式序列标注模型，直接建模给定观测序列时标签序列的条件概率。

中文答题句：

CRF 直接建模 $P(Y|X)$，可以灵活利用观测上下文、标签转移和人工设计特征，因此常用于中文分词、词性标注、命名实体识别等序列标注任务。

核心公式：

$$P(Y|X) = \frac{1}{Z(X)} \exp\left(\sum_k \lambda_k f_k(Y, X)\right)$$

$$Z(X) = \sum_{Y'} \exp\left(\sum_k \lambda_k f_k(Y', X)\right)$$

变量解释：

符号	含义
X	观测序列，例如一句话的字序列
Y	标签序列，例如 B/I/E/S
$f_k(Y, X)$	第 k 个特征函数
λ_k	第 k 个特征的权重
$Z(X)$	normalization term，对所有可能标签序列求和

人话解释：CRF 给每条候选标签路径打分，再用 $Z(X)$ 做全局归一化，因此比较的是整条路径，而不是每一步局部决策。

HMM vs CRF

对比点	HMM	CRF
英文	Hidden Markov Model	Conditional Random Fields
模型类型	generative	discriminative
建模对象	$P(X, Y)$ 或观测生成过程	$P(Y
观测特征	受 emission probability 限制	可使用 arbitrary features
标签依赖	Markov transition	可建模相邻标签和丰富上下文
优点	结构简单，可生成观测	特征灵活，适合序列标注
局限	独立性假设强，表达能力有限	特征选择 critical and hard

标准答法：

HMM 是生成式模型，描述隐藏状态如何转移以及如何产生观测；CRF 是判别式模型，直接建模给定观测条件下的标签序列概率。CRF 可以加入任意上下文特征和领域知识，表达能力更强，适合中文分词等序列标注任务；但 CRF 的特征选择非常关键，也比较困难。

Label Bias Problem

可答要点：

label bias problem 指局部归一化序列模型容易偏向转移选择较少的状态，因为每个状态出去的概率都被局部归一化，导致模型不能充分利用后续观测信息。CRF 使用全局归一化，可缓解这一问题。

CRF Feature Templates

中文分词里，CRF 用窗口内的字和标签组合生成特征。

常见窗口：

```
C_-2, C_-1, C_0, C_1, C_2
Predict tag of C_0
```

常见标签：

Tag	English	中文
B	Begin	词首
I	Inside	词中
E	End	词尾
S	Single	单字词

特征模板怎么读：

Template	中文含义
C_0	当前字
C_{-1}	前一个字
C_1	后一个字
$C_{-1}C_0$	前一个字 + 当前字
C_0C_1	当前字 + 后一个字
$S_{-1}S_0$	前一标签 + 当前标签

Unigram / Bigram：

类型	例子	用途
Unigram	C_0S_0, $C_{-1}S_0$, C_1S_0	当前标签与观测特征关系
Bigram	$C_0S_{-1}S_0$, $C_{-1}S_{-1}S_0$	相邻标签转移和观测共同作用

标准答法：

feature template 定义了从句子窗口中抽取哪些观测和标签组合。CRF 根据这些模板生成大量二值特征，用特征权重评价某个标签序列的好坏。

特征函数例子：

$$f_k(Y,X,i)=\begin{cases}1, & C_0=\text{“中”}\ \text{且}\ y_i=B\\ 0, & otherwise\end{cases}$$

这类公式的意思不是让你背“中”这个字，而是说明 CRF 特征通常是二值指示函数：某个观测模板和标签条件同时满足，就取 1，否则取 0。

CRF Training

先掌握训练目标；如果题目问 gradient / derivative / feature expectation，再写后面的梯度形式。

1. 对正确标签序列给高分。
2. 对错误标签序列给低分。
3. 最大化训练集中正确标签序列的条件概率。

简化公式：

$$\log p(t|c,\theta)=s(c,t,\theta)-\log\sum_{t'}\exp(s(c,t',\theta))$$

中文解释：

第一项是正确路径得分，第二项是所有可能路径的归一化项。训练的目标是让正确路径相对于其他路径得分更高。

如果题目问 objective function，可以写成：

$$\theta^*=\arg\max_{\theta}\sum_{(c,t)}\log p(t|c,\theta)$$

也就是最大化训练集中正确标签序列的条件对数似然。

如果题目问梯度，核心结论是：

$$\frac{\partial\log p(t|c,\theta)}{\partial\theta_k}=f_k(c,t)-\sum_{t'}p(t'|c,\theta)f_k(c,t')$$

中文解释：

梯度 = 正确标签路径上的特征次数 - 模型在所有可能标签路径下的期望特征次数。训练会提高正确路径中出现的特征权重，降低模型过度预测的特征权重。

Deep Learning for Chinese Word Segmentation

这一部分是从“人工特征模板”转向“自动学习表示”。

Previous Methods vs Deep Learning

Previous methods	Deep Learning
依赖人工 feature templates	自动发现任务相关特征
特征选择很难	减少 task-specific feature engineering
字符先变成 symbolic ID / one-hot	学 distributed character representations
需要人工设计上下文组合	可用大规模无标签语料 pre-train 表示

Deep CWS Architecture

考场按流程写即可：

```
input window
-> lookup table / character embeddings
-> concatenate
-> linear layer
-> sigmoid
-> linear layer
-> tag inference
```

各层含义：

部分	作用
input window	取当前字附近上下文
lookup table	把字 ID 转为 dense embedding
concatenate	拼接窗口内字向量
hidden layer	学习非线性组合特征
tag inference	结合转移分数，找最佳 B/E/I/S 标签序列

网络层公式可以这样写：

$$x_i = [e(c_{i-w}); \dots ; e(c_i); \dots ; e(c_{i+w})]$$

$$h_i = \sigma(W x_i + b)$$

$$f_\theta(t|i) = U_t h_i + d_t$$

变量解释：

符号	含义
$e(c_i)$	字符 c_i 的 embedding
x_i	当前窗口内字符向量拼接
h_i	hidden layer 表示
$f_\theta(t	i)$

Tag Inference

记住一句话：

标签路径得分 = 标签转移分数 + 神经网络给每个位置标签的分数。

路径打分公式：

$$s(c, t, \theta) = \sum_i (A_{t_{i-1} t_i} + f_\theta(t_i|i))$$

预测：

$$t^* = \arg\max_{t'} s(c, t', \theta)$$

标准答法：

深度中文分词模型先用 lookup table 得到字符分布式表示，再通过神经网络计算每个位置的标签分数，最后结合标签转移矩阵，用类似 Viterbi 的动态规划寻找最高分标签序列。

Perceptron-style Algorithm

记忆版本：

```
correct path score up
predicted wrong path score down
```

更新公式：

$$\theta \leftarrow \theta + \Phi(c, t) - \Phi(c, \hat{t})$$

符号	含义
t	正确标签序列
$\hat{t}$	模型预测出的错误标签序列
$\Phi(c, t)$	句子 c 与标签路径 t 对应的整体特征

中文答法：

perceptron-style algorithm 比较正确标签序列和模型预测序列。如果预测错误，就增加正确路径上的特征权重，减少错误路径上的特征权重，使下一次正确路径更容易获得高分。

Unsupervised Pre-training / Embeddings

为什么需要 unsupervised pre-training

中文答法：

无监督预训练利用大量无标签数据先学习有用的表示或参数初始化，再用有标签数据进行监督微调。它可以引导优化进入泛化更好的区域，也可看作一种正则化。

Word Embeddings

词向量的核心作用：

```
symbolic word/character ID -> dense continuous vector
```

优点：

1. 能表达词之间的相似性。
2. 能作为神经网络输入。
3. 可由大规模无标签语料学习。

课件英文例子：

```
King – Man + Woman close to Queen
Madrid – Spain + France close to Paris
Russia + River close to Volga River
```

注意：课件指出这些现象在训练得到的 Chinese character embeddings 中没有发现。

Word2Vec

模型	英文	中文解释
CBOW	Continuous Bag-of-Words	用上下文预测中心词
Skip-gram	Skip-gram	用中心词预测上下文
Huffman Tree	hierarchical softmax	用树结构加速大词表概率计算

常见目标：

$$\textbf{CBOW: } \max P(w_t | w_{t-m}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+m})$$

$$\textbf{Skip-gram: } \max \prod_{-m \le j \le m, j \neq 0} P(w_{t+j} | w_t)$$

人话解释：CBOW 是“看上下文猜中心词”；Skip-gram 是“看中心词猜上下文”。

RNN / LSTM / GRU

RNN

定义：

Recurrent Neural Network 在每个时间步共享同一组参数，用当前输入 x_t 和上一时刻隐藏状态 h_{t-1} 计算当前隐藏状态 h_t。隐藏状态把过去的信息传到后面，因此 RNN 适合建模 sequence data。

基本公式：

$$h_t = \tanh(V h_{t-1} + U x_t + b)$$

如果还有输出层：

$$y_t = g(W h_t + c)$$

变量解释：

符号	含义
x_t	第 t 个时间步的输入
h_{t-1}	上一时刻隐藏状态，包含历史信息
h_t	当前隐藏状态
U	input-to-hidden 权重
V	hidden-to-hidden / recurrent 权重
W	hidden-to-output 权重
$\tanh$	非线性激活函数

机制解释：

RNN 在每个时间步接收当前输入 x_t 和前一隐藏状态 h_{t-1}，产生当前隐藏状态 h_t。由于隐藏状态在时间上传递，RNN 能建模序列中的上下文依赖。

训练算法：

英文	中文
BPTT	Back-Propagation Through Time，通过时间展开后反向传播
RTRL	Real-Time Recurrent Learning，实时循环学习

BPTT 怎么理解：

把 RNN 沿时间展开成一个很深的前馈网络，然后用反向传播计算所有时间步共享参数的梯度。因为同一个 U, V, W 在每个时间步复用，所以梯度要把多个时间步的贡献累加。

主要问题：

英文	中文解释
vanishing gradient	长序列反传时梯度变得很小，难学长期依赖
blow up / exploding gradient	梯度变得很大，训练不稳定

为什么会有 vanishing gradient：

长序列反向传播时，梯度会反复乘以 recurrent weight 和激活函数导数。如果这些乘积的模小于 1，梯度会指数级变小；如果大于 1，可能 exploding。

LSTM

定义：

Long Short-Term Memory 是带门控机制的 RNN。它增加 cell state c_t 作为较稳定的记忆通道，并用 gates 控制旧信息保留、新信息写入和当前输出，从而缓解普通 RNN 的 vanishing gradient / long-term dependency 问题。

组成：

部分	作用
cell state c_t	长期记忆通道
forget gate f_t	决定旧记忆保留多少
input gate i_t	决定新信息写入多少
output gate o_t	决定输出多少记忆
candidate memory $\tilde{c}_t$	当前输入和旧隐藏状态生成的新候选信息

完整计算流程：

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \tanh(c_t)$$

变量解释：

符号	含义
σ	sigmoid，输出在 0 到 1 之间，可看作“门开多少”
$\circ$	element-wise product，逐元素相乘
f_t	forget gate，控制 c_{t-1} 保留比例
i_t	input gate，控制 $\tilde{c}_t$ 写入比例
o_t	output gate，控制 cell state 暴露到 hidden state 的比例
c_t	cell state，长期记忆
h_t	hidden state，当前时间步输出给后续层/时间步的信息

公式怎么读：

公式部分	中文解释
$f_t \circ c_{t-1}$	旧记忆保留多少
$i_t \circ \tilde{c}_t$	新候选信息写入多少
$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$	新 cell state = 保留的旧记忆 + 写入的新信息
$h_t = o_t \circ \tanh(c_t)$	输出门决定当前记忆有多少变成 hidden state

为什么 LSTM 能缓解长期依赖问题：

LSTM 的 cell state 提供一条相对直接的信息通路。forget gate 可以让重要信息在多个时间步中保留下来，input gate 可以避免无关信息频繁覆盖记忆，因此梯度更容易沿 c_t 传播到较早时间步。

答题句：

LSTM 通过门控机制控制信息流。forget gate 控制旧 cell state 的保留，input gate 控制新信息写入，output gate 控制当前输出。这样可以在较长时间范围内保存有用信息，缓解 vanishing gradient。

GRU

定义：

Gated Recurrent Unit 是 LSTM 的简化门控变体。GRU 没有单独的 cell state 和 output gate，而是用 hidden state 同时承担记忆和输出，并用 update gate、reset gate 控制状态更新。

组成：

gate	中文	作用
update gate z_t	更新门	控制旧状态保留多少、新状态写入多少
reset gate r_t	重置门	控制计算候选状态时忽略多少过去信息

完整计算流程：

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h[r_t \circ h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$$

变量解释：

符号	含义
z_t	update gate，决定从旧状态和候选状态各取多少
r_t	reset gate，决定计算候选状态时保留多少过去信息
$\tilde{h}_t$	candidate hidden state，候选新状态
h_t	当前隐藏状态

公式怎么读：

公式部分	中文解释
$r_t \circ h_{t-1}$	reset gate 先筛选过去信息，再生成候选状态
$(1 - z_t) \circ h_{t-1}$	保留旧 hidden state 的部分
$z_t \circ \tilde{h}_t$	写入候选 hidden state 的部分
$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t$	新状态是旧状态和候选状态的加权混合

答题句：

GRU 使用 update gate 和 reset gate 控制隐藏状态更新。它没有 LSTM 的 output gate 和独立 cell state，因此参数更少，但在一些时间序列任务上性能接近 LSTM。

LSTM vs GRU

维度	LSTM	GRU
gates	input, forget, output	update, reset
memory	独立 cell state	hidden state 同时承担记忆
参数量	较多	较少
优点	表达能力强，适合长期依赖	更简单，训练更快

英文题目定位：

触发词	该写什么
recurrent neural network / sequence model	写 $\boldsymbol{h_t = \tanh(Vh_{t-1} + Ux_t + b)}$，说明 hidden state 传递历史
BPTT	写时间展开、共享参数、反向传播累加梯度
vanishing gradient	写长链式乘法导致梯度变小，难学 long-term dependency
LSTM gates	写 forget/input/output gate 和 $\boldsymbol{c_t}$ 更新公式
GRU gates	写 update/reset gate 和 $\boldsymbol{h_t}$ 更新公式
compare LSTM and GRU	写 LSTM 有 cell state + 3 gates；GRU 更简单、2 gates、参数少

Transformer / GPT / BERT / Prompt

Contextualized Word Representations

中文答法：

contextualized representation 指同一个词在不同上下文中有不同表示。比如 Apple 在“吃苹果”和“iPhone”语境中应有不同语义向量。

Attention

一句话：

attention 让模型在处理当前位置时，根据相关性选择性关注序列中其他位置的信息。

标准答法：

attention mechanism 为上下文中的不同 token 分配不同权重，将重要 token 的信息聚合到当前表示中，因此能建模长距离依赖，是 Transformer 的核心。

Transformer / GPT / BERT

模型	中文定位	关键词
Transformer	基于 attention 的序列模型	self-attention, encoder-decoder
GPT	生成式语言模型	decoder, left-to-right, generation
BERT	双向预训练模型	encoder, masked language model

Masked Language Model

答题句：

Masked Language Model 在预训练时随机遮盖输入中的部分 token，让模型根据上下文预测被遮盖的 token。BERT 使用这种任务学习双向上下文表示。

Prompt-based Learning

考场答法：

prompt-based learning 把下游任务改写成语言模型熟悉的文本补全或预测问题。通过 template 构造 prompt，再把语言模型输出映射到任务标签。

例子：

```
Input: I love this movie.
Template: Overall, it was a [MASK] movie.
Verbalizer: fantastic -> positive, boring -> negative
```

Mixture-of-Experts

核心问题

课件里的问题是：

```
Can we design a big model while keeping bounded computation overhead?
```

也就是：算法工程师想要 larger model 提高 accuracy，但系统工程师担心 GPU latency 太高。MoE 的答案是 leverage sparsity：参数总量很大，但每个输入只激活一部分 experts。

概念	中文解释
expert	专家子网络
gating strategy	门控网络根据输入选择/加权专家
routing strategy	把 token 或样本路由到少数专家
sparse model	只激活部分参数，降低计算开销

Dense Model vs Sparse MoE

模型	计算方式	优缺点
Dense model	每个输入经过所有权重/模块	简单稳定，但模型大时计算开销高
Sparse MoE	gating network 选择少数 expert	参数容量大，但每次计算只走部分专家

常见公式：

$$y = \sum_{i=1}^{N} g_i(x) E_i(x)$$

若使用 top-k routing：

$$g_i(x) = 0 \quad \text{for experts not selected}$$

变量解释：

符号	含义
$E_i(x)$	第 i 个 expert 的输出
$g_i(x)$	gate / router 给 expert i 的权重
top-k	每个 token 只选择得分最高的 k 个 experts

Gating Strategy

图中含义：

1. 输入 x 同时送给 experts 和 gate。
2. gate 输出 scalar weights，决定哪些专家更重要。
3. expert 输出是 vector，gate 权重是 scalar。
4. 可以有一个 gate 负责所有输出，也可以不同 tower 使用不同 gate。

答题句：

gating strategy 用门控网络为 experts 分配权重，然后把专家输出加权组合。它更像“软选择”，因为多个 expert 可以同时贡献输出。

Routing Strategy

图中结构：

```
self-attention
-> add + normalize
-> switching FFN layer / router
-> selected FFNs
-> add + normalize
```

关键点：

英文	中文
router	根据 token 表示选择 FFN expert
switching FFN layer	把普通 FFN 层换成多个可选择 FFN
more parameters	总参数更多
bounded computation	每个 token 只走少数 expert，计算受控

答题句：

routing strategy 通常在 Transformer 的 FFN 层中加入 router，把不同 token 分配给不同 FFN expert。这样可以增加模型总参数量，但每个 token 只激活少数 expert，因此计算开销不随总参数线性增长。

Generative Models

总览

课件把 representation learning 和 generative modeling 对比：

方向	形式	含义
Representation learning	$x \to z_x$	encoder 把 data space 中的样本映射到 representation space
Generative modeling	$z \to x$	generator 从 latent/code 生成 data space 中的样本

生成模型总表：

模型	英文关键词	核心思想
Autoencoder	encoder, decoder, bottleneck	学压缩表示 z 并重构 $\hat{x}$
VAE	latent variable, ELBO, KL	学概率潜变量分布并可采样生成
Diffusion	forward noising, reverse denoising	正向逐步加噪，反向逐步去噪
GAN	generator, discriminator, adversarial	生成器和判别器对抗训练

Autoencoder

结构：

```
x -> encoder -> z -> decoder -> x_hat
```

目标：

$$\min_{\theta} L(x, \hat{x})$$

含义：encoder 把输入压缩为 latent representation z，decoder 尽量从 z 重构原输入 $\hat{x}$。如果去掉 encoder，从先验分布中采样 z 再经过 decoder，就得到 generative model 的形式。

VAE

VAE 是 probabilistic autoencoder。它不是把 x 编码成一个固定向量，而是编码成一个 latent distribution。

```
encoder: x -> latent distribution q(z|x)
sample: z
decoder: z -> reconstructed x
loss: reconstruction term + KL term
```

基础概率关系：

$$p(x) = \int p(x, z)\, dz$$

$$p(x) = \frac{p(x, z)}{p(z|x)}$$

VAE 引入近似后验：

$$q_\phi(z|x) \approx p(z|x)$$

通常设：

$$q_\phi(z|x) = N(z; \mu_\phi(x), \sigma^2_\phi(x) I)$$

$$p(z) = N(z; 0, I)$$

VAE / ELBO 推导

课件推导的核心是把 $\log p(x)$ 写成“可优化下界 + 非负 KL 项”：

$$\log p(x) = E_{q_\phi(z|x)}[\log p(x)]$$

$$= E_{q_\phi(z|x)}\left[\log \frac{p(x, z) q_\phi(z|x)}{p(z|x) q_\phi(z|x)}\right]$$

$$= E_{q_\phi(z|x)}\left[\log \frac{p(x, z)}{q_\phi(z|x)}\right] + D_{KL}(q_\phi(z|x) \| p(z|x))$$

因为 KL divergence 非负：

$$\log p(x) \geq E_{q_\phi(z|x)}\left[\log \frac{p(x, z)}{q_\phi(z|x)}\right]$$

这个下界就是 ELBO。

目标函数常见写法：

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x) \| p(z))$$

也可写成课件里的两项：

$$E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)\|p(z))$$

项	英文	中文含义
$E_{q_\phi(z	x)}[\log p_\theta(x	z)]$
$D_{KL}(q_\phi(z	x)\Vert p(z))$	prior matching term

答题句：

VAE 是带概率潜变量的自编码器。encoder 学习近似后验 $q_\phi(z|x)$，decoder 建模 $p_\theta(x|z)$。训练最大化 ELBO，即提高重构质量，同时用 KL divergence 约束潜变量分布接近先验 $p(z)$。

Reparameterization trick：

$$z = \mu + \sigma\epsilon, \quad \epsilon \sim N(0, I)$$

作用：把从 $q_\phi(z|x)$ 中采样改写成“确定性函数 + 外部噪声”。这样梯度可以通过 $\mu_\phi(x)$ 和 $\sigma_\phi(x)$ 反向传播。

Diffusion Models

图中两个方向：

方向	公式	含义
forward diffusion / noising	$q(x_t	x_{t-1})$
reverse process / denoising	$p(x_{t-1}	x_t)$

课件图示：

$$x_0 \to x_1 \to \cdots \to x_t \to \cdots \to x_T$$

$$q(x_t|x_{t-1}) \quad \textbf{adds noise}$$

$$p(x_{t-1}|x_t) \quad \textbf{removes noise}$$

常见 forward 形式：

$$q(x_t|x_{t-1}) = N(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

答题句：

diffusion model 的前向过程逐步向真实数据加入噪声，直到接近纯噪声；反向过程学习逐步去噪，从随机噪声恢复出数据样本。

GAN

结构：

```
z/noise -> Generator G -> generated fake samples
real samples + fake samples -> Discriminator D -> real/fake decision
```

课件类比：

角色	类比	目标
Generator	counterfeiters	生成足以欺骗判别器的假样本
Discriminator	police	区分真实样本和生成样本

常见目标函数：

$$\min_G \max_D E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p_z}[\log(1 - D(G(z)))]$$

怎么读：

部分	含义
$D(x)$	判别器认为真实样本为真的概率
$G(z)$	生成器把噪声 z 映射成假样本
$\max_D$	判别器尽量分清 real / fake
$\min_G$	生成器尽量让 $D(G(z))$ 接近 real

标准答法：

GAN 包含 generator 和 discriminator。generator 从 latent noise 生成假样本，discriminator 判断样本是真实数据还是生成数据。二者进行 minimax adversarial training：判别器提高真假区分能力，生成器提高欺骗判别器的能力，最终推动生成样本接近真实数据分布。

Applications / Practical Issues

ImageNet Story

课件规则：

```
# training data > 10 x # parameters
computer vision network may have about 100M parameters
unlabeled data are everywhere
labeling is the bottleneck and expensive
```

中文答法：

深度模型通常参数很多，因此需要大量训练数据。无标签数据容易获得，但人工标注昂贵且是瓶颈，这也是预训练、半监督和自监督学习重要的原因。

Practical Tricks

课件列出的 practical tricks：

Trick	中文	作用
Initialization (weights)	权重初始化	避免一开始激活/梯度异常
Input pre-processing	输入预处理	让输入尺度更稳定
Order of samples	样本顺序	影响 SGD 训练轨迹
Mini-batch	小批量训练	平衡梯度稳定性和计算效率
Learning rate	学习率	控制参数更新步长
Normalization	归一化	稳定分布，加快训练
Dropout	随机失活	正则化，减少过拟合
Optimizer	优化器	如 SGD/Adam，影响收敛
Pre-training if possible	可行时预训练	利用无标签或大规模数据获得更好初始化

Adversarial Examples

图中形式：

$$image + 0.007 \times noise = adversarial\ image$$

答题句：

adversarial examples 是人为构造的小扰动输入，视觉上可能几乎不变，却能诱导神经网络产生错误预测，说明深度模型虽然强大但存在脆弱性。

课件例子：

例子	含义
panda + small noise -> wrong class	小扰动导致错误分类
look-alike different objects	外观相似但类别不同，模型可能误判

典型应用

Application	中文
text to image synthesis	文生图
visual translation	视觉翻译
speech recognition	语音识别
multi-modal systems	多模态系统
sentiment analysis	情感分析
dependency parsing	依存句法分析

补充定位：

页面主题	可写要点
text to image synthesis	根据文本描述生成图像，可用 GAN 等生成模型
visual translation	location + recognition + translation/rendering，甚至可离线运行
speech recognition	从 HMM/GMM 到 DNN-HMM，声学特征映射到状态/词序列
multi-modal systems	audio/video 输入映射到 shared representation
sentiment analysis	改变词语会改变情感分类概率
dependency parsing	预测句子中词与词之间的依存弧

Spiking Neural Networks / TrueNorth

Why SNNs

答题句：

SNN 使用离散 spike 进行事件驱动计算，更接近生物神经系统。由于只在 spike 事件发生时计算，SNN 更硬件友好、能效更高。

课件要点：

原因	内容
brain inspiration	大脑是高效神经网络灵感来源
low power	人脑约 20W，普通计算机识别任务可约 250W
spike timing	脉冲精确时间可携带更多信息
hardware friendly	SNN 更适合低功耗硬件

关键词解释：

英文	中文
rate coding	用 spike 频率编码强度
latency coding	用 spike 出现早晚编码强度
delta modulation	输入变化量触发 spike
event-driven	只有事件/spike 出现时计算

SNN Input / Output Coding

图中三种输入编码：

编码	高强度怎么表示	低强度怎么表示
Rate coded inputs	单位时间 spike 更多	spike 更少
Latency coded inputs	spike 更早出现	spike 更晚出现
Delta modulation	输入变化触发 spike	变化少则 spike 少

输出也可 rate coded 或 latency coded。分类时，输出 spike 模式对应 predicted class。

LIF Neuron

Leaky Integrate-and-Fire 的图中要点：

```
integrate input
leak over time
fire when threshold reached
reset after spike
```

电路类比：

元件/符号	含义
I_{in}	输入电流
$U(t)$	membrane potential / 膜电位
R	leak resistance
C	membrane capacitance
ϑ	threshold / 发放阈值

常见动态方程可写为：

$$\tau_m \frac{dU(t)}{dt} = -U(t) + RI_{in}(t)$$

发放规则：

$$U(t) \geq \vartheta \Rightarrow \textbf{fire spike and reset}$$

人话解释：输入 spike 使膜电位上升；没有输入时膜电位泄漏下降；膜电位超过阈值就发放 spike，然后重置。

Computational Graph of SNN

图中把 LIF 展开到时间：

量	含义
$X[t]$	第 t 步输入
$I_{in}[t]$	输入电流
$U[t]$	membrane potential
$S_{out}[t]$	输出 spike
W	输入权重
β	implicit recurrence / 漏电后的历史保留
$-\theta$	spike 后重置对下一状态的影响

递推可概括为：

$$I_{in}[t] = WX[t]$$

$$U[t] = \beta U[t-1] + I_{in}[t] - \theta S_{out}[t-1]$$

$$S_{out}[t] = H(U[t] - \theta)$$

其中 H 是 threshold / step function。

IBM TrueNorth

可记数据：

```
4096 neurosynaptic cores
1M programmable spiking neurons
256M synapses
1ms time step -> 1KHz clock
event-driven -> low energy
```

局限：

```
does not learn online
```

TrueNorth Classification Model

课件图中结构：

```
local feature extraction
-> pooling
-> classification
```

模型特征：

项	内容
biologically plausible	比 ImageNet 模型更接近生物结构
recurrent/lateral connections	比普通前馈网络多循环和侧向连接
parameters/synapses	2012: 62M/1B；2014: 130M/15B
limitation	does not learn online

Classifier Performance

课件结论：

数据/任务	结论
MNIST	过去模型错误率从约 2% 降到约 0.2%，约 10 倍提升
more complex datasets	最好模型在复杂真实低分辨率图像上仍可差一个数量级
3D EM image / circuit reconstruction	仍明显差于人类

AI / ML / Representation Learning / DL

层级关系：

```
AI
-> Machine Learning
-> Representation Learning
-> Deep Learning
```

例子：

层级	Example
AI	knowledge-based systems
ML	logistic regression
Representation Learning	shallow autoencoder
Deep Learning	MLPs

Conclusion

考试总结句：

深度学习的核心是 learning feature hierarchies。高层特征由低层特征组合而来，深层网络中的特征通过 backpropagation 自动训练为与任务相关。2006 年 DBN 和 stacked autoencoders 推动了深层结构的有效训练，其共同策略是 greedy layer-wise unsupervised pre-training followed by supervised fine-tuning。

K-means Algorithm

标准步骤：

1. 随机初始化 K 个中心。
2. Assignment：把每个样本分给最近中心。
3. Update：用簇内样本均值更新中心。
4. 重复直到中心或分配不再变化。

目标：

$$\min \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

PCA / Principal Component Analysis

课件图示的是二维散点云和两条正交方向：

英文词	中文含义	图示含义
`principal component`	主成分	数据变化最大的方向。
`largest principal component`	最大主成分/第一主成分	沿散点云拉伸最长的方向，保留最多方差信息。
`smallest principal component`	最小主成分	与第一主成分正交，数据变化较小，常可作为被压缩掉的信息。
`projection`	投影	把样本点投到选定主成分方向上，得到低维表示。

一句话：PCA 是线性降维方法，寻找数据方差最大的正交方向作为新坐标轴；保留前几个主成分可以在尽量保留主要变化信息的同时降低维度。

公式理解：

$$z = W^T(x - \mu)$$

其中 x 是原始样本，μ 是均值，W 是选出的主成分方向矩阵，z 是投影后的低维特征。考题若问 PCA 和 deep learning 的关系，可写：PCA 只能做线性表示变换，而深度网络通过多层非线性变换学习更复杂的 representation。

常见考题答题模板

Q1: Why do we need deep learning?

深度学习能够自动学习多层表示。低层特征可以组合形成高层特征，高层特征更抽象、更接近任务目标，因此深层网络适合处理图像、语音、自然语言等复杂数据，并减少人工特征工程。

Q2: Explain cross entropy for binary classification.

二分类中，模型输出 $y(x_i)$ 表示样本属于正类的概率。最大化所有样本标签的似然等价于最小化负对数似然，即交叉熵：

$$-\sum_i [t_i \ln y(x_i) + (1 - t_i) \ln(1 - y(x_i))]$$

Q3: What are the three fundamental HMM questions?

HMM 三个基本问题是 evaluation、decoding 和 learning。Evaluation 计算观测序列概率 $P(O|\mu)$；decoding 寻找最可能隐藏状态序列；learning 根据观测数据估计模型参数。对应方法分别是 forward/backward、Viterbi 和 EM/Baum-Welch。

Q4: Forward algorithm 和 Viterbi algorithm 区别是什么?

Forward 用于计算观测序列总概率，对所有可能路径求和；Viterbi 用于寻找最可能路径，对所有前驱状态取最大值并记录 backpointer。简记：forward = sum，Viterbi = max + backpointer。

Q5: Compare HMM and CRF.

HMM 是生成式模型，描述隐藏状态如何转移以及如何产生观测；CRF 是判别式模型，直接建模给定观测条件下的标签序列概率。CRF 可以灵活加入任意上下文特征和领域知识，通常更适合中文分词等序列标注任务。

Q6: Explain feature templates in CRF.

feature template 定义从输入窗口中抽取哪些观测和标签组合。例如 C_0 表示当前字，$C_{-1}C_0$ 表示前一个字和当前字组合，$S_{-1}S_0$ 表示相邻标签转移。CRF 根据模板生成特征并学习权重。

Q7: Why is deep learning useful for Chinese word segmentation?

传统方法依赖人工 feature templates，特征选择困难。深度学习把字符映射为 dense embeddings，并通过神经网络自动学习上下文特征，减少人工特征工程；还可利用大规模无标签语料进行预训练。

Q8: Compare LSTM and GRU.

LSTM 使用 input gate、forget gate、output gate 和 cell state 控制信息保存与输出；GRU 使用 update gate 和 reset gate，没有 output gate，参数更少。二者都用于缓解普通 RNN 的长期依赖和梯度问题。

Q9: Explain GAN.

GAN 包含 generator 和 discriminator。generator 生成假样本试图欺骗 discriminator，discriminator 判断样本真假。二者对抗训练，使生成样本逐渐接近真实数据分布。

Q10: Why are SNNs energy-efficient?

SNN 使用 spike 事件驱动计算，只在脉冲事件发生时传递和处理信息，减少无效计算；同时更接近生物神经系统，因此适合低功耗硬件实现。

易错点

易错点	正确写法
HMM 三个问题	evaluation、decoding、learning，不要混在一起
forward vs Viterbi	forward 求和，Viterbi 取最大并回溯
parameter estimation	用期望次数重新估计参数，不是直接数真实状态
HMM vs CRF	HMM 生成式；CRF 判别式，直接建模 $P(Y
CRF feature template	模板定义特征来源，不是具体参数值
Deep CWS	核心是 embedding + neural features + tag inference
RNN	适合序列，但有 vanishing/exploding gradient
LSTM	有 input/forget/output gates 和 cell state
GRU	有 update/reset gates，无 output gate
VAE	reconstruction + KL；reparameterization 支持反传
GAN	generator 和 discriminator 对抗训练
SNN	spike/event-driven，能效高

Chapter 03b - Linear to Neural

定位：这份补充材料不是单独讲一个新模型，而是在说明“线性模型为什么容易求解、神经网络为什么表达能力更强但优化更复杂”。考场上常见问法是：写出 least squares / normal equation，解释 projection view，说明 MLP 结构和非凸优化问题。

英文题目识别词

English trigger	中文定位	看到题目后先想什么
linear model	线性模型	输出是输入特征的加权和
basis function / $\phi(x)$	基函数	先把原始输入变成特征，再线性组合
RSS / residual sum of squares	残差平方和	衡量预测值和真实值的平方误差
gradient of RSS	RSS 梯度	对 w 求导，令梯度为 0
normal equation	正规方程	$\Phi^T\Phi w = \Phi^T Y$
design matrix / Φ	设计矩阵	每一行是一个样本的基函数值
projection / hat matrix	投影 / 帽子矩阵	预测值是 Y 在特征子空间上的投影
MLP / multilayer perceptron	多层感知机	输入层-隐藏层-输出层
hidden layer	隐层	学习中间表示 h
local minimum	局部极小	附近没有更低点
global minimum	全局极小	整个空间最低点
convexity	凸性	线性模型好优化，MLP 通常不凸

一句话总览

线性模型把数据映射到一个特征子空间，在这个子空间里找最接近真实标签 Y 的预测 F，所以有正规方程闭式解；MLP 在中间加入隐藏层和非线性激活，表达能力更强，但损失函数通常不是凸函数，训练只能依靠梯度方法寻找较好的参数。

Linear Model：从“加权和”到“基函数线性组合”

1. 原始线性模型

公式：

$$f(x, w) = w_0 + x_1 w_1 + \cdots + x_D w_D = w_0 + \sum_{j=1}^{D} x_j w_j$$

这句话在说什么：

模型接收一个输入 $x = (x_1, \ldots, x_D)$，把每个输入维度乘上对应权重，再加上偏置 w_0，得到预测值。

符号	中文含义
x_j	第 j 个输入特征
w_j	第 j 个特征的权重
w_0	bias / intercept，截距项
D	输入特征维度

考场写法：

线性模型假设输出可以表示为输入特征的加权和。每个权重表示对应特征对预测结果的影响大小，偏置项表示整体平移。

2. Basis Function 形式

公式：

$$f(x, w) = \sum_{j=0}^{M-1} \phi_j(x) w_j = \phi(x)^T w$$

这一步为什么要引入 $\phi(x)$：

原始输入 x 不一定直接线性可分或线性可拟合。可以先用 basis functions 把 x 变成新的特征 $\phi(x)$，然后仍然对新特征做线性组合。

符号	中文含义
$\phi_j(x)$	第 j 个基函数，也就是从 x 变换出的一个新特征
$\phi(x)$	所有基函数值组成的向量
M	基函数/新特征的个数
$\phi(x)^T w$	新特征与权重的内积

中文答题句：

基函数形式说明：模型对参数 w 仍是线性的，但可以通过 $\phi(x)$ 对输入做非线性特征变换。因此它比直接使用原始特征的线性模型更灵活。

RSS：为什么用残差平方和

1. 单样本误差

对第 i 个样本：

$$\hat{y}_i = \phi(x_i)^T w$$

$$e_i = y_i - \hat{y}_i = y_i - \phi(x_i)^T w$$

含义：

符号	中文含义
y_i	第 i 个样本真实标签
$\hat{y}_i$	第 i 个样本预测值
e_i	residual / 残差，即真实值减预测值

2. 所有样本误差合起来

公式：

$$RSS(w) \triangleq \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \phi(x_i)^T w\right)^2$$

为什么平方：

1. 正负误差不会互相抵消。
2. 大误差会受到更大惩罚。
3. 平方函数可导，方便求最优参数。

为什么有 $\frac{1}{2}$：

求导时平方会带出系数 2，前面的 $\frac{1}{2}$ 可以抵消它，让梯度更简洁。它不改变最小值位置。

中文答题句：

RSS 衡量所有训练样本预测误差的平方和。最小化 RSS 等价于寻找一组参数，使模型预测整体上最接近真实标签。

RSS 梯度：正规方程从哪里来

1. 梯度公式

课件公式：

$$\nabla RSS(w) = -\sum_{i=1}^{N} \left(y_i - \phi(x_i)^T w\right) \phi(x_i)$$

这句话怎么读：

每个样本都贡献一个梯度项。若某个样本预测误差大，它对参数更新影响也大；$\phi(x_i)$ 决定这个误差沿哪些特征方向修正。

2. 最优点条件

为了最小化 RSS，令梯度为 0：

$$-\sum_{i=1}^{N} \left(y_i - \phi(x_i)^T w\right) \phi(x_i) = 0$$

移项后得到：

$$\left(\sum_{i=1}^{N} \phi(x_i) \phi(x_i)^T\right) w = \sum_{i=1}^{N} y_i \phi(x_i)$$

这就是正规方程的展开形式。

Design Matrix：把求和写成矩阵

1. 设计矩阵 Φ

课件定义：

$$\Phi = \begin{bmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_{M-1}(x_N) \end{bmatrix}$$

怎么读这个矩阵：

位置	含义
第 i 行	第 i 个样本的所有基函数值
第 j 列	所有样本在第 j 个基函数上的取值
Φw	所有样本的预测值向量

标签向量：

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

2. 矩阵正规方程

展开式可以写成：

$$\Phi^T \Phi w = \Phi^T Y$$

若 $\Phi^T\Phi$ 可逆：

$$w = (\Phi^T\Phi)^{-1}\Phi^T Y$$

中文解释：

$\Phi^T\Phi$ 记录基函数之间的相关性，$\Phi^T Y$ 记录每个基函数与目标值 Y 的相关性。正规方程就是在所有基函数方向上让残差达到平衡。

考场推导步骤：

1. 写 RSS：$\frac{1}{2}(Y-\Phi w)^T(Y-\Phi w)$。
2. 对 w 求导：$\Phi^T\Phi w - \Phi^T Y$。
3. 令导数为 0：$\Phi^T\Phi w = \Phi^T Y$。
4. 若可逆，写 $w = (\Phi^T\Phi)^{-1}\Phi^T Y$。

Projection View：图里的子空间是什么意思

课件公式：

$$F = \Phi w = \Phi(\Phi^T\Phi)^{-1}\Phi^T Y = HY$$

其中：

$$H = \Phi(\Phi^T\Phi)^{-1}\Phi^T$$

图中含义：

图中对象	中文解释
子空间 S	由 Φ 的列向量张成的预测空间
Y	真实标签向量，可能不在子空间里
F	模型能给出的最佳预测，位于子空间 S
$Y-F$	残差向量
直角符号	残差 $Y-F$ 与子空间正交

为什么是投影：

线性模型的预测只能落在 Φ 的列空间中。如果真实 Y 不在这个空间里，就找空间里离 Y 最近的点 F。这个最近点就是正交投影。

中文答题句：

最小二乘的几何意义是：在由基函数张成的子空间中寻找离真实标签向量 Y 最近的预测向量 F。最优时残差 $Y-F$ 与该子空间正交，因此 F 是 Y 在该子空间上的投影。

Multilayer Perceptron：图和公式怎么读

1. 网络结构

图中有三层：

```
输入 x -> 隐层 h -> 输出 y
```

层	图中符号	作用
input layer	$x_1, \dots, x_D$	原始输入特征
hidden layer	$h_1, \dots, h_M$	学到的中间表示
output layer	$y_1, \dots, y_K$	输出预测

2. 输入到隐层

课件公式：

$$h_j = \sigma\left(\sum_{i=1}^{D} w_{ji} x_i + b_j\right)$$

怎么读：

第 j 个隐藏单元先把所有输入 x_i 加权求和，再加偏置 b_j，最后经过非线性激活函数 σ。

符号	中文含义
w_{ji}	输入 x_i 到隐藏单元 h_j 的权重
b_j	第 j 个隐藏单元的偏置
σ	非线性激活函数
h_j	第 j 个隐藏单元的输出

3. 隐层到输出

课件公式：

$$y_k = \sigma\left(\sum_{j=1}^{M} w_{kj} h_j + b_k\right)$$

怎么读：

第 k 个输出单元把所有隐藏单元 h_j 再加权求和，经过激活函数得到输出 y_k。

符号	中文含义
w_{kj}	隐藏单元 h_j 到输出 y_k 的权重
b_k	第 k 个输出单元偏置
M	隐藏单元数量
K	输出单元数量

4. 矩阵形式

课件写法：

$$y_i = \sigma\left(W_2 \sigma(W_1 x_i + b_1) + b_2\right)$$

含义：

部分	中文解释
$W_1 x_i + b_1$	输入到隐层的线性变换
$\sigma(W_1 x_i + b_1)$	隐层非线性表示
$W_2(\cdot) + b_2$	隐层到输出的线性变换
外层 σ	输出层激活

5. MLP 损失函数

课件公式：

$$\mathcal{L}(\theta) = \frac{1}{2} \sum_{i=1}^{N} (y_i - t_i)^2$$

这里：

符号	中文含义
θ	网络所有参数，包括 W_1, W_2, b_1, b_2
y_i	网络对第 i 个样本的输出
t_i	第 i 个样本目标值

中文答题句：

MLP 和线性模型一样可以用平方误差作为训练目标，但 MLP 的预测 y_i 是多层非线性函数，因此 $\mathcal{L}(\theta)$ 对参数通常不是凸函数。

MLP 为什么比线性模型更强

线性模型只是在固定的 $\phi(x)$ 空间里做线性组合；MLP 的隐藏层本身会学习新的中间表示 h，而且激活函数 σ 引入非线性。因此 MLP 可以表达比线性模型复杂得多的函数。

答题模板：

MLP 在输入和输出之间加入隐藏层。隐藏单元先对输入做加权求和，再经过非线性激活，形成 learned representation。输出层再组合这些隐藏表示。因此 MLP 不只是在线性特征空间中拟合，而是能够学习非线性映射。

MLP 优化：局部极小和凸性的推导逻辑

这几页的作用是说明一个关键事实：线性最小二乘有清楚的闭式解，而 MLP 的损失曲面更复杂，可能存在局部极小，通常不能像线性模型那样简单求全局解。

1. 参数更新

课件公式：

$$\theta^{(t+1)} = \theta^{(t)} + \eta \Delta\theta^{(t)}$$

含义：

符号	中文
$\theta^{(t)}$	第 t 次迭代的参数
η	learning rate / 学习率
$\Delta\theta^{(t)}$	更新方向

中文解释：

训练神经网络时通常没有正规方程闭式解，而是从当前参数出发，沿某个方向小步更新。

2. 局部泰勒展开

课件公式：

$$\mathcal{L}(\theta + \Delta\theta) = \mathcal{L}(\theta) + \nabla\mathcal{L}(\theta)\Delta\theta + \frac{1}{2}\Delta\theta^T H \Delta\theta + o(\|\Delta\theta\|^2)$$

在局部极小点附近：

$$\nabla\mathcal{L}(\theta) = 0$$

且 Hessian 半正定：

$$\Delta\theta^T H \Delta\theta \geq 0$$

所以：

$$\mathcal{L}(\theta) \leq \mathcal{L}(\theta + \Delta\theta)$$

这句话在说什么：

如果 θ 是局部极小点，那么在它附近小范围移动，损失不会变得更小。

3. 凸函数的性质

课件公式：

$$\mathcal{L}((1-\lambda)\theta_1 + \lambda\theta_2) \leq (1-\lambda)\mathcal{L}(\theta_1) + \lambda\mathcal{L}(\theta_2)$$

中文解释：

如果损失函数是凸的，那么连接两个参数点的线段上，函数值不会高于两端函数值的线性插值。直观上，凸函数没有“坏的局部极小”：局部极小就是全局极小。

4. 反证思路：局部极小和更优全局点不能同时出现在凸函数中

课件设：

$$\mathcal{L}(\theta^*) < \mathcal{L}(\theta)$$

其中 θ 是某个局部极小点，θ^* 是损失更低的全局点。

取两者之间的点：

$$\bar{\theta} = (1-\lambda)\theta + \lambda\theta^*$$

若 $\mathcal{L}$ 是凸函数，则：

$$\mathcal{L}(\bar{\theta}) \leq (1-\lambda)\mathcal{L}(\theta) + \lambda\mathcal{L}(\theta^*)$$

因为 $\mathcal{L}(\theta^*) < \mathcal{L}(\theta)$，所以：

$$\mathcal{L}(\bar{\theta}) < \mathcal{L}(\theta)$$

也就是说，从 θ 朝 θ^* 走一点，损失会变小。

5. 但局部极小要求附近不能更小

课件又取：

$$\lambda < \frac{\epsilon}{\|\theta^* - \theta\|}$$

则：

$$\bar{\theta} = \theta + \lambda(\theta^* - \theta)$$

$$\|\bar{\theta} - \theta\| = \lambda\|\theta^* - \theta\| < \epsilon$$

这说明 $\bar{\theta}$ 在 θ 的局部邻域内。

如果 θ 是局部极小点，则应该有：

$$\mathcal{L}(\bar{\theta}) \geq \mathcal{L}(\theta)$$

但凸性加上更优全局点又推出：

$$\mathcal{L}(\bar{\theta}) < \mathcal{L}(\theta)$$

二者矛盾。

结论：

如果一个损失函数存在“不是全局最优的局部极小”，它就不是凸函数。MLP 的损失函数通常会出现这种复杂形态，因此训练比线性最小二乘困难。

中文答题句：

线性最小二乘的目标函数是凸的，可以通过正规方程得到全局最优解。MLP 的损失函数由于多层非线性组合，通常不是凸函数，可能存在局部极小或复杂平坦区域，因此训练依赖梯度下降等迭代优化方法，不能保证像线性模型一样直接得到闭式全局解。

Linear Model vs MLP

维度	Linear Model	MLP
表达形式	对特征做线性组合	多层线性变换 + 非线性激活
特征来源	依赖人工/固定 basis functions	隐层自动学习表示
目标函数	RSS 通常是凸二次函数	通常非凸
求解方式	正规方程或梯度法	反向传播 + 梯度下降
几何解释	投影到特征子空间	参数空间损失曲面复杂
优点	可解释、可闭式求解	表达能力强
难点	表达能力受特征限制	优化难、可能局部极小

常见考题答题模板

Q1: 推导 linear least squares 的 normal equation

中文作答：

线性模型对所有样本的预测可以写成 Φw，真实标签向量为 Y。残差平方和为：

$$RSS(w) = \frac{1}{2}(Y - \Phi w)^T(Y - \Phi w)$$

对 w 求导：

$$\nabla RSS(w) = \Phi^T\Phi w - \Phi^T Y$$

令梯度为 0：

$$\Phi^T\Phi w = \Phi^T Y$$

若 $\boldsymbol{\Phi^T\Phi}$ 可逆：

$$\boldsymbol{w = (\Phi^T\Phi)^{-1}\Phi^T Y}$$

这就是正规方程解。

Q2: 解释 least squares 的 projection view

中文作答：

设计矩阵 $\boldsymbol{\Phi}$ 的列向量张成一个特征子空间。线性模型的预测 $\boldsymbol{F = \Phi w}$ 必须落在这个子空间中，而真实标签 $\boldsymbol{Y}$ 不一定在其中。最小二乘就是在该子空间中寻找离 $\boldsymbol{Y}$ 最近的点 $\boldsymbol{F}$。最优时残差 $\boldsymbol{Y - F}$ 与子空间正交，因此 $\boldsymbol{F = HY}$ 是 $\boldsymbol{Y}$ 的投影。

Q3: 说明 MLP 的结构和公式

中文作答：

MLP 由输入层、隐藏层和输出层组成。第 $\boldsymbol{j}$ 个隐藏单元为：

$$\boldsymbol{h_j = \sigma\left(\sum_i w_{ji}x_i + b_j\right)}$$

第 $\boldsymbol{k}$ 个输出为：

$$\boldsymbol{y_k = \sigma\left(\sum_j w_{kj}h_j + b_k\right)}$$

隐藏层通过非线性激活学习中间表示，使模型能够表达复杂非线性函数。

Q4: 为什么 MLP 的优化比线性模型困难

中文作答：

线性最小二乘的 RSS 是关于 $\boldsymbol{w}$ 的凸二次函数，满足正规方程，可得到闭式全局最优。MLP 的输出是多层参数和非线性激活的复合函数，损失函数通常非凸，可能存在局部极小或复杂损失曲面，因此需要用梯度下降和反向传播迭代训练，不能保证像线性模型一样直接得到全局闭式解。

易错点

易错点	正确理解
看到公式 $\boldsymbol{\phi(x)^T w}$ 就以为模型一定线性	它对参数 $\boldsymbol{w}$ 线性，但 $\boldsymbol{\phi(x)}$ 可以是非线性特征
把 $\boldsymbol{\Phi}$ 当成原始输入矩阵	$\boldsymbol{\Phi}$ 是基函数展开后的 design matrix
不理解 RSS 前面的 $\frac{1}{2}$	只是为了求导简洁，不改变最优解
normal equation 漏写条件	$\boldsymbol{w = (\Phi^T\Phi)^{-1}\Phi^T Y}$ 需要 $\boldsymbol{\Phi^T\Phi}$ 可逆
projection view 只背公式	要说明 $\boldsymbol{F}$ 是 $\boldsymbol{Y}$ 在 $\boldsymbol{\Phi}$ 列空间上的正交投影
MLP 只写“多层”	关键是隐藏层 + 非线性激活学习表示
把局部极小证明当成无意义公式	它在说明凸函数不会有坏局部极小，而 MLP 通常非凸

04 Knowledge Representation 考场资料

本章要抓住一条主线：人工智能系统不能只靠算法，还要把领域知识用机器可处理的形式存起来，并能在这些表示上做推理。因此本章反复出现两个关键词：`knowledge representation` 表示知识，`inference` 利用知识推出新结论。

0. 英文题目触发词速查

English trigger	中文定位	作答时要写出的核心
`expert system`	专家系统	专家系统 = 知识 + 推理；区别于普通程序只强调算法和数据结构
`knowledge representation`	知识表示	把现实世界知识编码成符号结构，使计算机可以存储、检索、推理
`semantic network`	语义网络	用结点表示对象/概念/属性，用边表示关系；典型推理是继承
`is-a`, `instance-of`, `property inheritance`	类属/实例/属性继承	下位概念或实例可继承上位概念的属性，特殊属性可覆盖一般属性
`frame`, `slot`, `filler`, `default`	框架、槽、槽值、默认值	用结构化槽描述一个概念或对象；槽可继承、可填默认值、可附过程
`conceptual graph`	概念图	矩形概念结点 + 椭圆关系结点；能表达实体、关系、命题嵌套
`restriction`, `join`, `simplify`	概念图操作	限制使概念更具体；合并把共享概念连起来；化简去冗余
`knowledge graph`, `RDF triple`	知识图谱/三元组	用 `(subject, predicate, object)` 表示事实，可查询、链接、推理
`natural language understanding`	自然语言理解	从句法分析到语义表示，再加入上下文/世界知识

1. 为什么需要 Knowledge Representation

普通程序常被概括为：

$$\textbf{Programs} = \textbf{Algorithms} + \textbf{Data Structures}$$

专家系统更强调：

$$\textbf{Expert Systems} = \textbf{Knowledge} + \textbf{Inference}$$

含义是：专家系统的能力不只来自求解步骤，还来自它拥有的领域知识，以及能用这些知识推出结论的机制。例如医学诊断系统需要知道症状、疾病、检验结果之间的关系；法律或设备诊断系统需要知道规则、例外和对象结构。

知识处理可以分成三个相关任务：

1. `representation of knowledge`：知识如何表示。也就是选什么结构存储事实、概念、关系、规则和默认信息。
2. `acquirement of knowledge`：知识如何获得。可以来自专家、文本、数据、学习过程等。
3. `use of knowledge`：知识如何使用。包括查询、匹配、继承、解释、诊断、规划等推理任务。

考场回答“为什么知识表示重要”时可以写：知识表示是知识获取和知识使用的基础。如果知识没有被表示成计算机可处理的结构，就无法系统地检索、组合和推理。

2. Semantic Network 语义网络

2.1 基本定义

语义网络是一种图结构：

- `node` 结点：表示对象、概念、类别、属性值等。
- `arc/link` 边：表示结点之间的关系，如 `is-a`、`has`、`can`、`color`、`owns`。
- 主要思想：概念之间有层级关系，属性可以沿着层级继承。

常见关系：

关系	含义	例子
`is-a`	一个类属于另一个更一般的类	`CANARY is-a BIRD`
`instance-of`	个体属于某个类	`TWEETY instance-of CANARY`
`has-a` / `part-of`	整体与部分	`BIRD has WINGS`
`property`	属性或能力	`BIRD can FLY`
普通二元关系	对象之间的关系	`SYLVESTER owns TWEETY`

2.2 典型层级：动物、鸟、金丝雀、鸵鸟

下面把层级关系和属性直接写成文字事实，考试时不需要另看示意图。

类别层级：

- `CANARY is-a BIRD`：金丝雀是一种鸟。
- `OSTRICH is-a BIRD`：鸵鸟是一种鸟。
- `BIRD is-a ANIMAL`：鸟是一种动物。
- `FISH is-a ANIMAL`：鱼是一种动物。

动物层属性：

- `ANIMAL can breathe`：动物能呼吸。
- `ANIMAL has skin`：动物有皮肤。
- `ANIMAL can move`：动物能运动。

鸟类层属性：

- `BIRD can fly`：鸟一般能飞。
- `BIRD has wings`：鸟有翅膀。
- `BIRD has feathers`：鸟有羽毛。

金丝雀层属性：

- `CANARY can sing`：金丝雀会唱。
- `CANARY color yellow`：金丝雀是黄色的。
- `CANARY can fly`：金丝雀能飞。

鸵鸟层属性：

- `OSTRICH is tall`：鸵鸟很高。
- `OSTRICH cannot fly`：鸵鸟不能飞。

这里体现了两个考点：

1. 继承：`CANARY` 继承 `BIRD` 和 `ANIMAL` 的属性，所以金丝雀有翅膀、有羽毛、能呼吸、有皮肤。
2. 覆盖或例外：`OSTRICH` 虽然是鸟，但显式写了 `cannot fly`，它覆盖了鸟类“一般能飞”的默认属性。

2.3 心理学证据：为什么层级越远反应越慢

语义网络也被用来解释人类回答判断题的反应时间。核心想法是：属性不是全部重复存储在每个个体上，而是存储在合适的上位类别上；判断时需要沿层级向上查找。

以金丝雀为例：

- 判断 “A canary is a canary” 最快，因为它直接匹配对象本身。
- 判断 “A canary is a bird” 需要从 `CANARY` 查到上位 `BIRD`。
- 判断 “A canary is an animal” 需要从 `CANARY` 经 `BIRD` 再到 `ANIMAL`。
- 判断 “A canary can sing” 比较快，因为 `can sing` 存在 `CANARY` 本层。
- 判断 “A canary can fly” 需要查 `CANARY` 或 `BIRD` 层的飞行属性。
- 判断 “A canary has skin” 更慢，因为 `has skin` 存在更高的 `ANIMAL` 层。

因此，如果题目问 `psychological evidence for semantic networks`，可以答：反应时间随语义距离和继承路径长度增加而增加，这支持知识按层级组织、属性沿层级查找的观点。

2.4 语义网络上的继承推理

给定事实：

- `BIRD can-fly`：鸟能飞。
- `CANARY is-a BIRD`：金丝雀是鸟。
- `CANARY color YELLOW`：金丝雀是黄色的。
- `TWEETY instance-of CANARY`：Tweety 是一只金丝雀。
- `SYLVESTER owns TWEETY`：Sylvester 拥有 Tweety。

可以推出：

- `CANARY can-fly`：因为金丝雀是鸟，继承鸟能飞。
- `TWEETY can-fly`：因为 Tweety 是金丝雀，金丝雀继承鸟能飞。
- `TWEETY color YELLOW`：因为 Tweety 是金丝雀，继承金丝雀黄色。
- `TWEETY is-a BIRD`：因为 Tweety 是金丝雀，金丝雀是鸟。
- `SYLVESTER owns a CANARY`：因为 Sylvester 拥有 Tweety，而 Tweety 是金丝雀。
- `SYLVESTER owns a BIRD`：因为金丝雀是鸟。
- `SYLVESTER owns something that can fly`：因为 Tweety 能飞。

作答时不要只写结论，最好写推理链：

$$\textbf{TWEETY} \rightarrow \textbf{CANARY} \rightarrow \textbf{BIRD} \rightarrow \textbf{can-fly}$$

中文解释：先确认 Tweety 是金丝雀，再利用 `is-a` 继承到鸟类，最后继承鸟类的 `can-fly` 属性。

2.5 语义网络优缺点

优点：

- 直观，适合表达对象、类别、属性、层级关系。
- 继承机制减少重复存储。
- 易于解释推理路径。

局限：

- 关系语义可能不够严格，同一条边在不同系统中含义可能不同。
- 处理例外、否定、时间变化、多重继承冲突时需要额外机制。
- 对复杂量词、嵌套命题和不确定性表达能力有限。

3. Frame 框架表示

3.1 基本定义

框架是一种结构化知识表示方法，用来描述一个典型对象、场景或概念。一个框架由若干 `slot` 槽组成，每个槽有 `filler` 槽值。

可以把框架理解为“带继承和默认值的记录结构”：

```
Frame: HOTEL-ROOM
  is-a: ROOM
  location: HOTEL
  contains: HOTEL-CHAIR, HOTEL-PHONE, HOTEL-BED
```

关键术语：

- `frame`：框架，表示一个类、对象或情境。
- `slot`：槽，表示某一方面的属性或关系。
- `filler`：槽值，可以是具体值、另一个框架、默认值、规则或过程。
- `default value`：默认值，在没有特殊信息时采用。
- `inheritance`：继承，下位框架继承上位框架的槽。
- `procedural attachment`：过程附着，槽中可以放入需要时执行的过程。

3.2 酒店房间框架：把对象结构写完整

酒店房间可以表示为一个框架系统，而不是一张图。

`HOTEL-ROOM`：

槽	槽值	含义
`specialization of`	`ROOM`	酒店房间是房间的一种
`location`	`HOTEL`	位于酒店
`contains`	`HOTEL-CHAIR`，`HOTEL-PHONE`，`HOTEL-BED`	包含酒店椅、酒店电话、酒店床

`HOTEL-CHAIR`：

槽	槽值	含义
`specialization of`	`CHAIR`	酒店椅是椅子的一种
`height`	`20-40 cm`	高度范围
`legs`	`4`	四条腿
`use`	`sitting`	用途是坐

`HOTEL-PHONE`：

槽	槽值	含义
`specialization of`	`PHONE`	酒店电话是电话的一种
`use`	`calling`，`room service`	可打电话、叫客房服务
`billing`	`through room`	费用计入房间

`HOTEL-BED`：

槽	槽值	含义
`superclass`	`BED`	属于床这一类
`use`	`sleeping`	用途是睡觉
`size`	`king`	大号床
`part`	`MATTRESS`，`FRAME`	包含床垫和床架

`MATTRESS`：

槽	槽值	含义
`superclass`	`CUSHION`	床垫可看作软垫类
`firmness`	`firm`	硬度偏硬

这个例子说明框架特别适合表达“一个场景中有哪些典型组成部分、每个部分有哪些默认属性”。如果题目问 `frame representation for a hotel room`，不要只写概念名，要写出框架名、槽名、槽值和继承关系。

3.3 Types of Slot 槽的类型

框架中的槽不只保存普通属性，还可以承担不同功能：

slot type	中文解释	作用
`Frame identification information`	框架识别信息	标识框架名称、类型、它描述的对象
`Relationship of this frame to other frames`	与其他框架的关系	表示 `is-a`、`part-of`、`instance-of` 等联系
`Descriptors of requirements for a frame`	对框架适用条件的描述	说明什么时候可使用该框架，或槽值需满足什么限制
`Procedural information`	过程性信息	槽值可触发计算、检查或动作
`Frame default information`	默认信息	未知时采用默认槽值
`New instance information`	新实例信息	当生成实例时补充或覆盖槽值

3.4 框架推理：继承、默认值、槽填充

框架推理最常见的是三件事：

1. 从上位框架继承槽。例如 `HOTEL-CHAIR` 是 `CHAIR` 的特化，所以可以继承椅子的一般性质。
2. 使用默认值。例如如果没有特别说明，酒店椅默认有四条腿。
3. 根据关系填充或传播信息。例如一个对象与另一个对象有 `likes`、`contains`、`part` 关系时，可以沿关系查询相关槽。

例子：`John likes a fire engine`。

相关框架可以写成：

`JOHN`：

槽	槽值
`isa`	`HUMAN`
`gender`	`male`
`enterprise`	`average`
`activity`	`high`
`likes`	`FIRE-ENGINE`

`FIRE-ENGINE`：

槽	槽值
`isa`	`MOTOR-VEHICLE`
`color`	`red`
`activity`	`high`
`volume`	`very high`
`fuel efficiency`	`average`
`ladder`	`5 m`

可以回答的问题：

- 问 John 喜欢什么：沿 `likes` 槽得到 `FIRE-ENGINE`。
- 问 John 喜欢的东西是什么颜色：先由 `JOHN likes FIRE-ENGINE` 找到对象，再查 `FIRE-ENGINE color red`。
- 问 John 喜欢的东西是否很响：查 `FIRE-ENGINE volume very high`。
- 问 fire engine 是什么：由 `FIRE-ENGINE isa MOTOR-VEHICLE` 得到它是机动车的一种。

注意：框架推理不是单纯逻辑证明，而是围绕槽的查询、继承、默认填充和关系导航。

3.5 框架表示优缺点

优点：

- 结构清楚，适合描述典型对象和典型场景。
- 支持继承和默认值，能减少重复。
- 便于把声明性知识和过程性知识结合。

局限：

- 框架之间继承冲突时需要额外规则。

- 默认值可能被例外推翻，需要非单调处理。
- 表达严格逻辑量词、复杂约束时不如逻辑语言精确。

4. Conceptual Graph 概念图

4.1 基本符号

概念图用图结构表示语义：

- `concept node` 概念结点：通常画成矩形，表示实体、概念、事件、命题等。
- `relation node` 关系结点：通常画成椭圆，连接一个或多个概念。
- `arity` 元数：关系连接的概念个数。

也可以把概念图写成链式表达：

```
[DOG] -> (COLOR) -> [BROWN]
```

含义：一只狗的颜色是棕色。

4.2 一元、二元、三元关系

`one-ary relation` 一元关系：

```
[BIRD] -> (FLIES)
```

含义：鸟会飞。关系 `FLIES` 只连接一个概念。

`two-ary relation` 二元关系：

```
[DOG] -> (COLOR) -> [BROWN]
```

含义：狗的颜色是棕色。关系 `COLOR` 连接对象和颜色两个概念。

`three-ary relation` 三元关系：

```
[CHILD] -> (PARENTS) -> [FATHER], [MOTHER]
```

含义：一个孩子的父母包括父亲和母亲。关系 `PARENTS` 同时关联三个角色。

4.3 Unique Marker 唯一标记

概念图可以用唯一标记表示某个具体个体：

```
[DOG:#007] -> (COLOR) -> [BROWN]
[DOG:#007] -> (NAME) -> ["Emma"]
```

中文含义：编号为 `#007` 的那只狗叫 Emma，并且是棕色的。

可写成自然语言：一只名叫 Emma 的狗是棕色的。

这里 `#007` 的作用是避免把“任意狗”误认为“这只具体的狗”。

4.4 Generic Marker 通用标记

通用标记表示某个未具体命名的个体，常写成 `*X`。

```
[DOG:*X] -> (AGENT) -> [SCRATCH]
[SCRATCH] -> (OBJECT) -> [EAR]
[SCRATCH] -> (INSTRUMENT) -> [PAW]
[EAR] -> (PART-OF) -> [DOG:*X]
[PAW] -> (PART-OF) -> [DOG:*X]
```

中文含义：某只狗用自己的爪子抓自己的耳朵。

关键点是：`EAR` 和 `PAW` 都通过 `PART-OF` 连接到同一个 `DOG:*X`，所以表达的是“它自己的耳朵”和“它自己的爪子”，不是别的狗的耳朵或爪子。

4.5 Propositional Concept 命题概念

概念图可以把一个完整命题当作另一个命题的对象，用来表示相信、知道、说、希望等心理或语言行为。

例子：

```
[PERSON:"Tom"] -> (EXPERIENCER) -> [BELIEVE]
[BELIEVE] -> (OBJECT) -> [PROPOSITION:
    [PERSON:"Jane"] -> (SUBJECT) -> [LIKE] -> (OBJECT) -> [PIZZA]
]
```

中文含义：Tom 相信 Jane 喜欢披萨。

重点：Tom 相信的是一个命题“Jane likes pizza”，而不是直接等同于系统断言 Jane 一定喜欢披萨。`believe` 的对象是嵌套命题。

4.6 概念图推理操作：restriction, join, simplify

概念图的推理可以通过图操作完成。

Restriction 限制

限制操作把较一般的概念变得更具体，或给通用个体加上更具体的类型。

初始事实：

```
[ANIMAL:"Emma"] -> (LOCATION) -> [PORCH]
[ANIMAL:"Emma"] -> (COLOR) -> [BROWN]
```

如果知道 Emma 是狗，可以限制为：

```
[DOG:"Emma"] -> (LOCATION) -> [PORCH]
[DOG:"Emma"] -> (COLOR) -> [BROWN]
```

含义：`DOG` 比 `ANIMAL` 更具体，所以可以把 Emma 从动物限制成狗，同时保留位置和颜色事实。

Join 合并

合并操作把两个共享同一概念的图连接起来。

图 A：

```
[DOG:"Emma"] -> (AGENT) -> [EAT]
[EAT] -> (OBJECT) -> [BONE]
```

图 B：

```
[DOG:"Emma"] -> (LOCATION) -> [PORCH]
[DOG:"Emma"] -> (COLOR) -> [BROWN]
```

合并后：

```
[DOG:"Emma"] -> (AGENT) -> [EAT]
[EAT] -> (OBJECT) -> [BONE]
[DOG:"Emma"] -> (LOCATION) -> [PORCH]
[DOG:"Emma"] -> (COLOR) -> [BROWN]
```

中文含义：Emma 是一只棕色的狗，在门廊上吃骨头。

Simplify 化简

化简操作删除合并后重复或冗余的结点与关系，但不改变含义。例如合并后出现两个完全相同的 `[DOG:"Emma"]` 结点，应保留一个，并把所有关系接到同一个 Emma 上。

答题时可以写：`restriction` 增加具体性，`join` 汇合共享实体的事实，`simplify` 去除重复结构以保持图简洁。

4.7 概念图优缺点

优点：

- 比普通语义网络更适合表达事件、角色和嵌套命题。
- 图结构直观，可和自然语言语义结构对应。
- `restriction`、`join`、`simplify` 提供了形式化操作。

局限：

- 大规模知识库中图匹配成本较高。
- 复杂否定、量词、概率不确定性仍需额外机制。

5. Knowledge Graph 与 RDF Triple

知识图谱常把事实表示成三元组：

$$(\text{subject},\ \text{predicate},\ \text{object})$$

含义：

- `subject` 主体：被描述的实体。
- `predicate` 谓词：关系或属性名。
- `object` 客体：另一个实体或属性值。

例子：描述 Sean Penn。

subject	predicate	object
`Sean_Penn`	`rdf:type`	`Person`
`Sean_Penn`	`fullName`	`"Sean Penn"`
`Sean_Penn`	`mailbox`	`"sean_penn@xxx.com"`
`Sean_Penn`	`occupation`	`"Actor"`

这些三元组可以被查询。例如题目给出作者、书名、主页、价格等关系时，查询的思路是：用变量占住要找的实体或属性值，再用三元组模式匹配知识库中的事实。

典型查询语义：

```
SELECT ?author ?title ?homepage
```

中文解释：查询满足条件的作者、书名和主页。`?author`、`?title`、`?homepage` 是变量，系统会从三元组集合里找能同时满足关系约束的绑定。

6. 从自然语言到内部知识表示

自然语言理解可以分成几个层次。

原句：

```
Tarzan kissed Jane.
```

6.1 Parsing 句法分析

句法结构：

- 名词短语：`Tarzan`
- 动词：`kissed`
- 名词短语：`Jane`

这个阶段只说明句子的语法组成，还没有充分表示动作角色和常识。

6.2 Semantic Interpretation 语义解释

内部表示可以写成：

```
[PERSON:"Tarzan"] -> (AGENT) -> [KISS]
[KISS] -> (OBJECT) -> [PERSON:"Jane"]
[KISS] -> (INSTRUMENT) -> [LIPS]
```

中文含义：Tarzan 是亲吻动作的施事，Jane 是受事，亲吻通常用嘴唇完成。

6.3 Contextual / World Knowledge Interpretation 上下文和世界知识解释

加入背景知识后，可以扩展为：

- `Tarzan` 是一个人。
- `Jane` 是一个人。
- `Tarzan` 亲吻 `Jane`。
- 亲吻的工具通常是 `lips`。
- `Tarzan` 可能生活在 `jungle` 场景中。
- `Tarzan` 可能拥有宠物 `Cheetah`。
- `Tarzan` 对 `Jane` 有 `love` 关系。

重点：系统从句子本身得到核心事件，再用世界知识补充隐含信息。自然语言理解不是简单词典替换，而是句法、语义和背景知识共同作用。

6.4 这些表示可用于什么

内部知识表示可以服务于：

- `question answerer`：问答系统。
- `database query handler`：把自然语言问题转成数据库查询。
- `translator`：机器翻译。
- `language generator`：自然语言生成。
- `speech synthesis`：语音合成。

7. 其他知识表示方法

本章还可能出现以下方法，考场至少要能识别它们属于知识表示。

方法	适合表达什么
`propositional logic`	命题真值及命题连接
`first-order logic`	对象、谓词、量词和关系
`description logic`	概念层级、属性限制、本体
`modal logic`	必然、可能、知识、信念等模态
`production rule`	`IF condition THEN action/conclusion` 规则
`artificial neural network`	分布式、数值化、从数据学习的知识
`decision table`	条件组合与决策结果
`knowledge Petri net`	状态、事件、并发和流转
`script`	典型事件序列，如餐馆脚本
`Bayesian network`	不确定因果关系和概率推理
`object-attribute-value triples`	对象、属性、属性值事实
`neurules`	神经网络和规则结合
`language field theory`	语言场相关的知识组织方式

8. 常见答题模板

8.1 问：Compare semantic networks and frames

中文作答：

语义网络和框架都能表示概念、对象和属性，也都可以支持继承。语义网络强调图结构，结点表示概念或对象，边表示关系，适合表达 `is-a` 层级和属性继承。框架强调结构化槽，使用 `slot-filler` 描述对象或场景，适合表达典型对象的组成部分、默认值、过程信息和实例化信息。语义网络更像关系图，框架更像带继承的对象模板。

8.2 问：Explain inheritance in a semantic network

中文作答：

继承是指下位概念或实例可以获得上位概念的属性。例如 `TWEETY instance-of CANARY`，`CANARY is-a BIRD`，`BIRD can-fly`，则 Tweety 通过 `TWEETY -> CANARY -> BIRD` 的路径继承 `can-fly` 属性，所以可推出 Tweety can fly。如果下位概念有更特殊的相反属性，例如 `OSTRICH cannot fly`，则特殊属性覆盖一般属性。

8.3 问：What is a frame? Explain slots

中文作答：

框架是一种结构化知识表示方法，用来描述典型对象、概念或情境。框架由多个槽组成，槽表示属性、关系、默认值、约束或过程，槽值称为 filler。框架之间可以通过 `is-a`、`part-of` 等关系连接，下位框架可以继承上位框架的槽。槽还可以保存默认值或过程性信息，使系统在信息不完整时仍能推理。

8.4 问：Explain conceptual graph operations

中文作答：

概念图由概念结点和关系结点组成。`restriction` 是把一般概念变得更具体，例如把 `ANIMAL:"Emma"` 限制为 `DOG:"Emma"`。`join` 是把两个共享同一实体的概念图合并，例如把 “Emma eats a bone” 和 “Emma is on the porch” 合并成 “Emma eats a bone on the porch”。`simplify` 是删除合并后重复的结点和关系，在不改变含义的前提下使图更简洁。

8.5 问：What is a knowledge graph triple?

中文作答：

知识图谱三元组用 `(subject, predicate, object)` 表示一个事实。`subject` 是被描述实体，`predicate` 是关系或属性，`object` 是另一个实体或属性值。例如 `(Sean_Penn, occupation, Actor)` 表示 Sean Penn 的职业是演员。大量三元组可以组成知识图谱，并通过变量匹配进行查询。

9. 易错点

- 不要把 `is-a` 和 `instance-of` 混在一起：`CANARY is-a BIRD` 是类和类的关系；`TWEETY instance-of CANARY` 是个体和类的关系。
- 继承不是无条件永远成立：显式例外可以覆盖默认属性，如鸟一般能飞，但鸵鸟不能飞。
- 框架不只是属性表，还包括继承、默认值、过程附着和实例化信息。
- 概念图中的唯一标记表示具体个体，通用标记表示未命名但同一的个体。
- 命题概念中，被相信或被说出的内容是嵌套命题，不一定等于系统直接断言的事实。
- 三元组的 `object` 可以是实体，也可以是字符串、数字等属性值。

05 Uncertainty 考场资料

本章的核心问题是：智能系统经常无法获得完整、精确、可靠的信息，但仍然要判断、诊断和决策。确定性逻辑适合“前提都确定为真”的场景；一旦信息缺失、冲突、含糊或带噪声，就需要不确定性推理。

可以按四条线理解本章：

1. 不确定性从哪里来：信息不足、错误、模糊、随机、推理缺陷。
2. 确定性推理为什么不够：演绎推理保证真，但现实中常只能做归纳、默认推理和可撤销推理。
3. 如何量化不确定性：概率、贝叶斯推理、决策风险、Bayesian network。
4. 如何处理专家系统和模糊语言：certainty factor、fuzzy set、fuzzy inference，以及时间变化中的 Markov chain。

0. 英文题目触发词速查

English trigger	中文定位	作答时要写出的核心
`uncertainty`	不确定性	缺少足够信息时仍需判断或决策
`ambiguous`, `incomplete`, `incorrect`	含糊、不完整、不正确	信息本身可能有多种解释、缺项或错误
`false positive`, `false negative`	假阳性、假阴性	系统误报存在；系统漏报存在
`deduction`, `induction`	演绎、归纳	演绎保真；归纳从样本推广，结论带置信度
`affirming the consequent`	肯定后件谬误	`p -> q` 和 `q` 不能推出 `p`
`nonmonotonic reasoning`	非单调推理	新信息可能撤销旧结论
`Bayes' rule`, `posterior`, `prior`, `likelihood`	贝叶斯公式	用证据更新假设概率
`Bayesian decision theory`, `loss`, `risk`	贝叶斯决策	同时考虑概率和损失，选期望损失小的行动
`certainty factor`, `MB`, `MD`, `CF`	确信因子	专家系统中用相信程度和不相信程度表示证据支持
`Markov chain`, `transition matrix`, `steady state`	马尔可夫链	状态按转移概率随时间变化
`PageRank`	页面排名	用马尔可夫过程迭代网页重要性
`fuzzy set`, `membership function`	模糊集合、隶属函数	元素属于集合的程度是 0 到 1
`hedge`, `very`, `more or less`, `not`	模糊修饰词	用函数改变隶属度
`max-min composition`	最大-最小合成	模糊关系推理的组合方式

1. Uncertainty 是什么

不确定性可以概括为：由于没有足够信息，系统不能确定某个结论一定为真或一定为假，但仍然需要做判断或行动选择。

典型原因：

- 观察不完整：只看到部分症状、部分传感器读数、部分证据。
- 观察有噪声：测量误差、传感器故障、数据录入错误。
- 世界本身随机：天气、用户行为、市场变化都不是确定的。
- 概念边界模糊：什么叫“高个子”“年轻”“温度较高”没有清晰边界。
- 推理规则有例外：鸟一般会飞，但鸵鸟不会飞；默认结论可能被新信息推翻。

考场可写：不确定性不是系统完全无知，而是在证据不足或证据质量有限时，用概率、置信度、模糊隶属度等方式表达“支持到什么程度”。

2. 错误和不确定性的类型

2.1 Ambiguous 含糊

同一信息可以有多种解释。例如一句自然语言、一个医学症状或一个视觉对象可能对应多个含义。

答题关键词：`multiple possible interpretations`。中文解释：证据不是缺失，而是解释不唯一。

2.2 Incomplete 不完整

系统缺少某些关键事实。例如医生只知道病人发烧，但不知道血检结果；机器人只看到局部环境。

答题关键词：`missing information`。中文解释：不是已有信息错，而是信息不够。

2.3 Incorrect 不正确

信息本身是错的。例如传感器读数错误、记录错误、专家规则写错。

答题关键词：`wrong information`。中文解释：系统基于错误输入推理，即使推理形式正确，也会得到错误结论。

2.4 False positive 与 false negative

- `false positive` 假阳性：系统判断某事件存在，但实际上不存在。例如诊断系统说有病，实际没有病。
- `false negative` 假阴性：系统判断某事件不存在，但实际上存在。例如诊断系统说无病，实际有病。

两者区别：

类型	系统判断	实际情况	中文记忆
`false positive`	有	无	误报
`false negative`	无	有	漏报

2.5 其他常见误差词

English	中文	解释
`imprecise`	不精密	结果分散，重复测量不稳定
`inaccurate`	不准确	结果偏离真实值
`unreliable`	不可靠	来源或过程不能稳定信任
`random error`	随机误差	方向不固定，随机波动
`systematic error`	系统误差	总是朝某个方向偏
`reasoning error`	推理错误	证据使用方式或推理形式有问题

精密度和准确度的区别：精密强调重复结果彼此接近，准确强调结果接近真实值。一个系统可以很精密但不准确，例如每次都稳定地偏高。

3. Deduction, Induction 与推理谬误

3.1 Deduction 演绎推理

演绎推理的特点：如果前提为真，推理规则有效，则结论必真。

典型形式：

$$p \to q,\quad p \quad \therefore q$$

中文解释：如果 `p` 蕴含 `q`，并且 `p` 已经成立，那么可以推出 `q`。这叫 `modus ponens`。

例子：

- 如果一个物体是鸟，则它有羽毛。
- Tweety 是鸟。
- 所以 Tweety 有羽毛。

在现实中，问题是很多前提并不完全确定，规则也可能有例外。

3.2 Induction 归纳推理

归纳推理从有限观察推广到一般结论，结论不能保证必真，只能给出某种置信度。

例子：观察到许多乌鸦都是黑色，于是归纳出“乌鸦通常是黑色的”。这个结论可能很可靠，但不是逻辑必然，因为未来可能观察到例外。

作答时可写：演绎推理关注有效性和保真性；归纳推理关注证据支持程度，因此天然带有不确定性。

3.3 Affirming the Consequent 肯定后件谬误

错误形式：

$$p \to q,\quad q \quad \therefore p$$

为什么错：`q` 可能由许多原因造成，并不一定由 `p` 造成。

医学例子：

- 如果得了某病，可能会发烧。
- 病人发烧。
- 不能直接推出病人一定得了该病，因为发烧也可能由其他疾病造成。

正确做法：把 `q` 当作证据，用贝叶斯方法更新 `p` 的概率，而不是把它当作确定证明。

4. Nonmonotonic Reasoning 非单调推理

4.1 单调与非单调

在经典逻辑中，如果一组前提能推出结论，加入更多前提后这个结论仍然成立。这叫 `monotonic` 单调性。

非单调推理中，新信息可以撤销旧结论。它适合表达默认规则、常识推理和例外。

例子：

1. 已知 `BIRD(Tweety)`，默认规则是鸟通常会飞。
2. 推出 `can-fly(Tweety)`。
3. 后来知道 `PENGUIN(Tweety)` 或 `OSTRICH(Tweety)`。
4. 因为企鹅或鸵鸟是鸟的例外，撤销 `can-fly(Tweety)`。

4.2 Support 与 Defeat

非单调推理常需要区分：

- `support` 支持：某些证据支持一个结论。
- `defeat` 击败：新证据削弱或推翻该结论。

例子：

- 支持：`Tweety is a bird` 支持 `Tweety can fly`。
- 击败：`Tweety is an ostrich` 击败 `Tweety can fly`。

4.3 Lottery 与 Preface 类问题

这些问题用于说明“每个单独判断都很合理”不代表“所有判断合起来仍然合理”。

`lottery` 彩票直觉：对于一张有很多票的彩票，每张具体票“不会中奖”都很可能是真的，但不能把所有票都判断为不会中奖，因为必然有一张中奖。

`preface` 前言直觉：作者相信书中每个具体陈述都经过检查，但也承认整本书可能有错误。单个信念有支持，整体合取却不一定可接受。

这类例子说明：常识推理中，结论强度、例外和组合方式都要谨慎处理。

5. Probability 与 Bayes Rule

概率方法用数值表示不确定性。核心思想是：先有一个先验判断，得到证据后更新为后验判断。

5.1 基本术语

English	中文	含义
`hypothesis H`	假设	要判断的命题，如“有油”“患病”
`evidence E`	证据	观察到的信息，如测试阳性、症状
`prior P(H)`	先验概率	看到证据前对假设的概率
`likelihood P(E|H)`	似然	假设为真时出现证据的概率
`evidence probability P(E)`	证据概率	证据整体出现的概率
`posterior P(H|E)`	后验概率	看到证据后假设为真的概率

5.2 Bayes Rule 贝叶斯公式

单个假设：

$$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}$$

多个互斥且完备假设 $H_1, \ldots, H_n$：

$$P(H_i \mid E) = \frac{P(E \mid H_i)P(H_i)}{\sum_j P(E \mid H_j)P(H_j)}$$

变量解释：

- H_i：第 i 个可能假设。
- E：观察到的证据。
- $P(H_i)$：证据出现前，假设 H_i 的先验概率。
- $P(E \mid H_i)$：如果 H_i 为真，看到证据 E 的概率。
- 分母 $\sum_j P(E \mid H_j)P(H_j)$：证据 E 的总概率，用来归一化。

一句话理解：后验概率与“假设本来有多可能”和“该假设能多好解释证据”同时相关。

5.3 Oil Test 例题：先验、测试结果、后验

设：

- O：有油。
- O'：无油。
- $+$：测试阳性。
- $-$：测试阴性。

先验：

$$P(O) = 0.6, \quad P(O') = 0.4$$

测试可靠性：

$$P(+ \mid O) = 0.8, \quad P(- \mid O) = 0.2$$

$$P(+ \mid O') = 0.1, \quad P(- \mid O') = 0.9$$

先算联合概率：

$$P(+ \cap O) = P(+ \mid O)P(O) = 0.8 \times 0.6 = 0.48$$

$$P(-\cap O)=P(-\mid O)P(O)=0.2\times0.6=0.12$$
$$P(+\cap O')=P(+\mid O')P(O')=0.1\times0.4=0.04$$
$$P(-\cap O')=P(-\mid O')P(O')=0.9\times0.4=0.36$$

证据总概率：

$$P(+)=0.48+0.04=0.52$$
$$P(-)=0.12+0.36=0.48$$

如果测试阳性：

$$P(O\mid+)=\frac{P(+\cap O)}{P(+)}=\frac{0.48}{0.52}=\frac{12}{13}$$
$$P(O'\mid+)=\frac{P(+\cap O')}{P(+)}=\frac{0.04}{0.52}=\frac{1}{13}$$

如果测试阴性：

$$P(O\mid-)=\frac{P(-\cap O)}{P(-)}=\frac{0.12}{0.48}=\frac{1}{4}$$
$$P(O'\mid-)=\frac{P(-\cap O')}{P(-)}=\frac{0.36}{0.48}=\frac{3}{4}$$

中文解释：阳性时，有油概率从 0.6 提高到 $12/13$；阴性时，有油概率降到 $1/4$。证据不是绝对证明，而是更新概率。

5.4 医学诊断中的贝叶斯更新

设 D_i 表示第 i 种疾病，E 表示症状或检查结果。

单个证据：

$$P(D_i\mid E)=\frac{P(E\mid D_i)P(D_i)}{\sum_j P(E\mid D_j)P(D_j)}$$

含义：把一个症状作为证据，计算每种疾病的后验概率。

多个证据依次更新：

$$P(D_i\mid E_2,E_1)=\frac{P(E_2\mid D_i,E_1)P(D_i\mid E_1)}{\sum_j P(E_2\mid D_j,E_1)P(D_j\mid E_1)}$$

含义：先用 E_1 更新得到 $P(D_i\mid E_1)$，再把它作为新的先验，结合 E_2 继续更新。

答题时要强调：贝叶斯诊断不是“症状推出疾病”，而是“症状改变疾病概率”。

6. Bayesian Decision Theory 贝叶斯决策

概率只回答“哪个状态更可能”，决策还要回答“采取哪个行动更划算”。如果错误行动损失很大，最可能的状态不一定对应最优行动。

6.1 基本组成

符号	含义
θ	世界状态，如降雨量、真实疾病、市场情况
a	可选行动，如种哪种作物、做哪种治疗
$l(\theta,a)$	损失函数，状态为 θ 时采取行动 a 的损失
$R(\theta,a)$	风险或期望损失
δ	决策规则，把观察结果映射到行动
$r(\pi,\delta)$	在先验 π 下决策规则 δ 的 Bayes risk

贝叶斯决策原则：选择期望损失最小的行动或决策规则。

6.2 作物选择例题

设：

- a_1：种抗旱作物 `drought-resistant crop`。
- a_2：种高产作物 `high-yielding crop`。
- θ：降雨量。

损失函数：

$$l(\theta,a)=\begin{cases}200-2\theta, & a=a_1\\ 3000-10\theta, & a=a_2\end{cases}$$

解释：

- 对抗旱作物，降雨量越高，损失 $200-2\theta$ 越低。
- 对高产作物，降雨量越高，损失 $3000-10\theta$ 降得更快，但降雨不足时损失可能更大。

天气预报模型给出观测 x 在真实降雨量 θ 下的密度：

$$f(x\mid\theta)=\begin{cases}\frac{1}{\pi}\frac{400}{400^2+(x-\theta)^2}, & x\geq 0\\ 0, & x<0\end{cases}$$

含义：预报值 x 围绕真实 θ 波动，密度函数描述观测误差。

如果决策规则根据预报值选择作物，例如预报低时选抗旱作物，预报高时选高产作物，则风险可以写成对可能观测结果的积分：

$$R(\theta,a_1)=\int_0^{400} l(\theta,a_1)f(x\mid\theta)\,dx$$

$$R(\theta,a_2)=\int_{400}^{\infty} l(\theta,a_2)f(x\mid\theta)\,dx$$

整体 Bayes risk：

$$r(\pi,\delta)=\sum_{\theta}R(\theta,\delta)\pi(\theta)$$

这里 $\pi(\theta)$ 是状态 θ 的先验概率，δ 是根据观测选择行动的规则。如果题目要求解释公式含义，可以说明：这套公式是在“预报不完全可靠”的条件下，计算某个决策规则的期望损失。

7. Bayesian Network 贝叶斯网络

贝叶斯网络用有向无环图表示随机变量之间的依赖关系。

核心思想：

- 结点表示随机变量。
- 有向边表示直接概率依赖或因果影响。
- 每个结点有条件概率表 `conditional probability table, CPT`。
- 整个联合分布可以分解为局部条件概率的乘积。

若变量为 $X_1,\dots,X_n$，父结点集合为 $Parents(X_i)$，则：

$$P(X_1,\dots,X_n)=\prod_i P(X_i\mid Parents(X_i))$$

中文解释：贝叶斯网络的价值在于不用直接列出完整联合概率表，而是利用条件独立性把复杂分布拆开。

常见任务：

- `prediction`：由原因推结果。
- `diagnosis`：由结果反推原因。
- `explaining away`：多个原因竞争解释同一证据。

8. MYCIN 与 Certainty Factor

8.1 为什么不直接用概率

MYCIN 是医学专家系统，使用规则进行诊断。专家常能说“某些证据强烈支持某疾病”，但不一定愿意或能够给出严格概率。

典型规则：

```
IF stain is gram positive
AND morphology is coccus
AND growth conformation is chains
THEN there is suggestive evidence 0.7
that identity is streptococcus
```

如果把它硬解释为：

$$P(H\mid E_1,E_2,E_3)=0.7$$

会有问题，因为专家说的 0.7 往往是“支持程度”，不一定等于严格概率；剩下 0.3 也不一定表示假设为假的概率，可能只是未知或证据不足。

因此 MYCIN 使用 `certainty factor, CF` 表示证据对假设的支持和反支持。

8.2 MB, MD, CF 的定义

三个量：

- $MB(H,E)$：`measure of belief`，证据 E 对假设 H 的相信程度。
- $MD(H,E)$：`measure of disbelief`，证据 E 对假设 H 的不相信程度。

- $CF(H,E)$：`certainty factor`，净确信度。

关系：

$$CF(H,E) = MB(H,E) - MD(H,E)$$

范围：

$$-1 \le CF(H,E) \le 1$$

解释：

- $CF = 1$：完全支持 H。
- $CF = -1$：完全反对 H。
- $CF = 0$：没有净支持，可能是无证据，也可能是支持和反对抵消。

基于概率的定义：

$$MB(H,E) = \begin{cases} 1, & P(H) = 1 \\ \dfrac{\max(P(H \mid E), P(H)) - P(H)}{1 - P(H)}, & P(H) \neq 1 \end{cases}$$

$$MD(H,E) = \begin{cases} 1, & P(H) = 0 \\ \dfrac{\min(P(H \mid E), P(H)) - P(H)}{0 - P(H)}, & P(H) \neq 0 \end{cases}$$

含义：

- 如果证据使 $P(H \mid E)$ 大于 $P(H)$，则增加相信程度 MB。
- 如果证据使 $P(H \mid E)$ 小于 $P(H)$，则增加不相信程度 MD。
- 分母把变化量归一化到 0 到 1。

8.3 证据组合：AND, OR, NOT

MYCIN 中常用简单规则组合证据的 CF。

合取：

$$CF(E_1 \wedge E_2, e) = \min(CF(E_1, e), CF(E_2, e))$$

中文解释：多个条件同时成立时，整体支持程度受最弱条件限制。

析取：

$$CF(E_1 \vee E_2, e) = \max(CF(E_1, e), CF(E_2, e))$$

中文解释：多个条件任一成立时，整体支持程度取最强支持。

否定：

$$CF(\neg E, e) = -CF(E, e)$$

中文解释：对证据取反时，支持方向反过来。

复杂条件：

$$E = (E_1 \wedge E_2 \wedge E_3) \vee (E_4 \wedge \neg E_5)$$

则：

$$CF(E,e) = \max\left[\min(CF(E_1,e), CF(E_2,e), CF(E_3,e)), \min(CF(E_4,e), -CF(E_5,e))\right]$$

8.4 规则强度与结论 CF

如果规则本身有强度 $CF(H,E)$，实际观察到证据的 CF 为 $CF(E,e)$，则：

$$CF(H,e) = CF(E,e) \times CF(H,E)$$

例子：

$$CF(E_1,e) = 0.5, \quad CF(E_2,e) = 0.6, \quad CF(E_3,e) = 0.3$$

条件是 $E_1 \wedge E_2 \wedge E_3$，所以：

$$CF(E,e) = \min(0.5, 0.6, 0.3) = 0.3$$

规则强度：

$$CF(H,E) = 0.7$$

结论：

$$CF(H,e) = 0.3 \times 0.7 = 0.21$$

中文解释：证据链中最弱条件只有 0.3，规则本身也只是 0.7 强度，所以最终对假设的支持是 0.21。

8.5 多条规则支持同一假设时如何合并

若两个证据来源分别给同一假设 H 产生 CF_1 和 CF_2，组合公式为：

$$CF_{comb}(CF_1, CF_2) = \begin{cases} CF_1 + CF_2(1 - CF_1), & CF_1 > 0,\ CF_2 > 0 \\ \dfrac{CF_1 + CF_2}{1 - \min(|CF_1|, |CF_2|)}, & CF_1 CF_2 < 0 \\ CF_1 + CF_2(1 + CF_1), & CF_1 < 0,\ CF_2 < 0 \end{cases}$$

解释：

- 两个都支持时，支持增强但不会超过 1。
- 一个支持一个反对时，需要抵消。
- 两个都反对时，反对增强但不会小于 -1。

8.6 CF 方法的困难

CF 方法实用，但有局限：

- 数值含义不如概率严格。
- 不同专家给出的 CF 可能不一致。
- `min`、`max` 等组合规则比较经验化。
- 证据之间独立或相关的问题没有像概率模型那样严格处理。

答题时可写：CF 是专家系统中处理不确定证据的工程化方法，强调证据对假设的支持程度，而不是完整概率分布。

9. Temporal Reasoning 与 Markov Chain

9.1 为什么需要时间推理

很多问题不是静态的：用户偏好、市场份额、网页访问、天气状态都会随时间变化。时间推理关注状态如何从一个时刻转移到下一个时刻。

9.2 Markov Chain 马尔可夫链

马尔可夫链假设：下一时刻状态只依赖当前状态，而不依赖更早历史。

$$P(X_{t+1} \mid X_t, X_{t-1}, \ldots, X_0) = P(X_{t+1} \mid X_t)$$

转移矩阵 P 的元素 P_{ij} 表示从状态 i 转移到状态 j 的概率。

例子：顾客在 `Lenovo` 和 `Apple` 两种电脑品牌之间转换。

状态：

- C_1：买 Lenovo。
- C_2：买 Apple。

转移矩阵：

$$P = \begin{bmatrix} 0.1 & 0.9 \\ 0.6 & 0.4 \end{bmatrix}$$

含义：

- 当前买 Lenovo 的顾客，下次继续买 Lenovo 的概率 0.1，转买 Apple 的概率 0.9。
- 当前买 Apple 的顾客，下次转买 Lenovo 的概率 0.6，继续买 Apple 的概率 0.4。

若初始状态分布为：

$$S_1 = [0.8,\ 0.2]$$

表示 80% 买 Lenovo，20% 买 Apple。下一期：

$$S_2 = S_1 P$$

计算：

$$S_2 = [0.8,\ 0.2] \begin{bmatrix} 0.1 & 0.9 \\ 0.6 & 0.4 \end{bmatrix} = [0.2,\ 0.8]$$

中文解释：经过一次转移后，买 Lenovo 的比例变成 0.2，买 Apple 的比例变成 0.8。

9.3 Steady State 稳态

稳态分布满足：

$$S = SP$$

并且：

$$P_1 + P_2 = 1$$

对上面的矩阵，解得：

$$P_1 = 0.4, \quad P_2 = 0.6$$

中文解释：长期来看，如果转移规律不变，系统会稳定在 40% Lenovo、60% Apple。

9.4 PageRank

PageRank 可以看作网页之间随机跳转的马尔可夫过程。网页的重要性由其他网页指向它的结构决定。

迭代形式：

$$\pi^{(k)} = \pi^{(k-1)} G$$

变量解释：

- $\pi^{(k)}$：第 k 次迭代后的页面排名向量。
- G：Google matrix，表示网页之间的跳转概率。
- 迭代直到排名向量接近稳定。

中文解释：如果很多重要网页指向某网页，该网页的 PageRank 会更高。公式本质上是在转移矩阵上不断更新状态分布。

10. Fuzzy Set 模糊集合

10.1 为什么需要模糊集合

概率处理“事件是否发生的不确定性”，模糊集合处理“概念边界不清晰”。例如“高个子”“温暖”“速度快”不是非黑即白，而是有程度。

经典集合：

$$x \in A \quad 或 \quad x \notin A$$

模糊集合：

$$\mu_A(x) : X \to [0,1]$$

其中 $\mu_A(x)$ 是 x 属于集合 A 的隶属度。

解释：

- $\mu_A(x) = 1$：完全属于。
- $\mu_A(x) = 0$：完全不属于。
- $0 < \mu_A(x) < 1$：部分属于。

10.2 Linguistic Variable 语言变量

语言变量用自然语言词表示取值，例如：

- 变量：`height` 身高。
- 语言值：`short`、`medium`、`tall`、`very tall`。

这些词通过隶属函数转成数值程度。

例子：

$$TALL = \{0.125/5.5,\ 0.5/6,\ 0.875/6.5,\ 1/7\}$$

含义：

- 身高 5.5 对 `TALL` 的隶属度是 0.125。
- 身高 6 对 `TALL` 的隶属度是 0.5。
- 身高 6.5 对 `TALL` 的隶属度是 0.875。
- 身高 7 对 `TALL` 的隶属度是 1。

写法 `0.5/6` 不是除法，而是“元素 6 的隶属度为 0.5”。

10.3 Hedges 模糊修饰词

`hedge` 是对模糊集合的修饰，如 very、more or less、not。

设原隶属度为 $\mu_F(x)$。

`Very F`：

$$\mu_{\text{Very } F}(x) = \mu_F(x)^2$$

作用：降低中间隶属度，让概念更严格。

`More or Less F`：

$$\mu_{\text{More or Less } F}(x) = \mu_F(x)^{0.5}$$

作用：提高中间隶属度，让概念更宽松。

`Plus F`：

$$\mu_{\text{Plus } F}(x) = \mu_F(x)^{1.25}$$

作用：比原概念稍严格，但没有 `Very` 那么强。

`Not F`：

$$\mu_{\text{Not } F}(x) = 1 - \mu_F(x)$$

`Not Very F`：

$$\mu_{\text{Not Very } F}(x) = 1 - \mu_F(x)^2$$

例子：由 `TALL` 得到 `VERY TALL`：

$$VERY\ TALL = \{0.016/5.5,\ 0.25/6,\ 0.766/6.5,\ 1/7\}$$

由 `TALL` 得到 `NOT TALL`：

$$NOT\ TALL = \{0.875/5.5,\ 0.5/6,\ 0.125/6.5,\ 0/7\}$$

10.4 模糊逻辑中的 AND, OR, NOT

常用定义：

$$\mu_{A \wedge B}(x) = \min(\mu_A(x), \mu_B(x))$$
$$\mu_{A \vee B}(x) = \max(\mu_A(x), \mu_B(x))$$
$$\mu_{\neg A}(x) = 1 - \mu_A(x)$$

中文解释：

- AND 取较小值，因为同时满足两个条件受较弱条件限制。
- OR 取较大值，因为满足任一条件即可由较强条件支持。
- NOT 用 1 减去原隶属度。

10.5 Fuzzy Relation 与 Max-Min Composition

模糊关系表示两个集合元素之间关系成立的程度。

若 $R_1(x)$ 是关于 x 的模糊集合，$R_2(x, y)$ 是 x 与 y 的模糊关系，则合成结果 $R_3(y)$ 为：

$$R_3(y) = R_1(x) \circ R_2(x, y)$$

隶属度：

$$\mu_{R_3}(y) = \max_x \min(\mu_{R_1}(x), \mu_{R_2}(x, y))$$

解释步骤：

1. 对每个可能的中间变量 x，先取 $\min$，表示路径强度受较弱环节限制。
2. 在所有 x 的路径中取 $\max$，表示选择最强的一条支持路径。

10.6 Fuzzy Inference 模糊推理

模糊规则形式：

```
IF E_i THEN H_i
```

每条规则得到一个结论隶属度：

$$\mu_{H_i} = \min(\mu_{E_i}, \mu_{\text{rule}_i})$$

多条规则合成：

$$\mu_H = \max(\mu_{H_1}, \ldots, \mu_{H_n})$$

如果条件是：

$$E_1 = E_A \wedge (E_B \vee \neg E_C)$$

则条件隶属度为：

$$\mu_{E_1} = \min(\mu_{E_A}, \max(\mu_{E_B}, 1 - \mu_{E_C}))$$

中文解释：先按括号算 `OR` 和 `NOT`，再与 E_A 做 `AND`。规则结论强度由条件满足程度和规则本身强度共同决定。

11. 本章方法之间的区别

方法	处理的问题	数值含义	典型公式/机制
Probability	事件真假不确定	事件发生概率	Bayes rule
Bayesian decision	不确定状态下选行动	概率 + 损失	expected loss / risk
Bayesian network	多变量概率依赖	条件概率	$\prod_i P(X_i \mid Parents(X_i))$
Certainty factor	专家规则支持程度	支持减反支持	$CF = MB - MD$
Markov chain	状态随时间变化	转移概率	$S_{t+1} = S_t P$
Fuzzy set	概念边界模糊	隶属度	$\mu_A(x) \in [0,1]$

区分概率和模糊：

- 概率问：这个事件发生的可能性多大？例如“明天下雨的概率是 0.7”。
- 模糊问：这个对象属于某个模糊概念的程度多大？例如“这个人属于高个子的程度是 0.7”。

概率中的 0.7 表示不确定事件；模糊中的 0.7 表示概念匹配程度。

12. 常见答题模板

12.1 问：What causes uncertainty in AI?

中文作答：

AI 中的不确定性来自信息不足和信息质量问题。系统可能面对不完整、含糊、错误或带噪声的数据，也可能面对随机变化的环境和有例外的常识规则。因此系统不能总是用确定性逻辑推出绝对结论，而需要用概率、置信度或模糊隶属度表示证据支持程度。

12.2 问：Explain Bayes rule and its use

中文作答：

贝叶斯公式用于根据证据更新假设概率。$P(H)$ 是看到证据前的先验概率，$P(E \mid H)$ 是假设成立时观察到证据的概率，$P(H \mid E)$ 是看到证据后的后验概率。公式为 $P(H \mid E) = P(E \mid H)P(H)/P(E)$。它适用于诊断和分类，因为症状或测试结果不能确定推出原因，只能改变各假设的概率。

12.3 问：Why is affirming the consequent invalid?

中文作答：

`p -> q` 和 `q` 不能推出 `p`，因为 `q` 可能由其他原因导致。例如某病会导致发烧，但发烧不一定说明一定得了该病。正确处理方式是把发烧作为证据，用贝叶斯公式提高或降低该疾病的概率，而不是作确定结论。

12.4 问：Explain certainty factor in MYCIN

中文作答：

MYCIN 用 certainty factor 表示证据对假设的净支持程度。$MB(H,E)$ 表示证据对假设的相信程度，$MD(H,E)$ 表示不相信程度，$CF(H,E) = MB(H,E) - MD(H,E)$。$CF = 1$ 表示完全支持，$CF = -1$ 表示完全反对，$CF = 0$ 表示没有净支持。它适合专家只能给出支持强度、不能给出严格概率的规则系统。

12.5 问：Explain fuzzy set and membership function

中文作答：

模糊集合用于表示边界不清晰的概念。经典集合中元素要么属于集合，要么不属于集合；模糊集合中元素属于集合的程度由隶属函数 $\mu_A(x)$ 给出，取值在 0 到 1 之间。例如一个人的身高对 `TALL` 的隶属度可以是 0.5，表示他在一定程度上属于高个子。模糊集合适合处理“高”“热”“快”等自然语言概念。

12.6 问：Compare probability and fuzzy logic

中文作答：

概率处理事件是否发生的不确定性，数值表示事件为真的可能性；模糊逻辑处理概念边界的不清晰，数值表示对象属于某概念的程度。例如“明天下雨概率为 0.7”是概率问题，因为下雨事件未来要么发生要么不发生；“某人是高个子的程度为 0.7”是模糊问题，因为高个子没有清晰边界。

13. 易错点

- `P(E|H)` 和 `P(H|E)` 不能混淆：前者是假设下证据的概率，后者是证据下假设的概率。
- 看到结果不能直接确定原因；这正是肯定后件谬误。
- CF 不是严格概率，0.7 的支持程度不等于“剩下 0.3 就是假”。
- 模糊隶属度不是概率；它表示概念匹配程度。
- Markov chain 的下一状态只依赖当前状态，这是马尔可夫性质。
- `steady state` 要同时满足 $S = SP$ 和各状态概率和为 1。
- 模糊 `AND` 常用 `min`，`OR` 常用 `max`，`NOT` 常用 $1 - \mu$。

06 Genetic Algorithm 考场资料

本章主线：遗传算法 `genetic algorithm, GA` 把一个问题的候选解编码成染色体，在一组候选解组成的种群中反复执行选择、交叉、变异，让高适应度的结构更容易保留下来并被重组。考题常给出小种群，要求计算 fitness、selection probability、crossover/mutation 后的个体，或解释 schema theorem。

0. 英文题目触发词速查

English trigger	中文定位	作答时要写出的核心
`genetic algorithm`, `GA`	遗传算法	population, fitness, selection, crossover, mutation, replacement
`chromosome`, `individual`, `gene`	染色体/个体/基因	染色体是一个候选解，基因是编码位置
`encoding`, `representation`	编码	把问题解映射成 binary/permutation/Gray code 等
`fitness function`	适应度函数	衡量候选解好坏，选择概率通常与 fitness 成正比
`roulette wheel selection`	轮盘赌选择	$p_i = f(i)/\sum_j f(j)$
`crossover`	交叉	父代片段重组，产生子代
`mutation`	变异	随机改变基因，增加多样性
`knapsack problem`	背包问题	0/1 编码，超重可加 penalty
`traveling salesman problem`, `TSP`	旅行商问题	permutation 编码，交叉/变异必须保持城市不重复不遗漏
`CNF-satisfaction problem`	合取范式可满足性	染色体可表示变量赋值，fitness 可表示满足子句数
`semantic tree`	语义树	用分支枚举变量真值赋值
`schema`, `order`, `defining length`	模式、阶、定义长度	schema 是带通配符的模板；order 看固定位点数；defining length 看首末固定位距离
`schema theorem`	模式定理	短、低阶、高于平均适应度的模式会在后代中指数增长

1. GA 的基本思想

遗传算法是受自然进化启发的随机搜索方法。它不是从一个解出发一步步爬坡，而是同时维护一个 `population` 种群。每个 `individual/chromosome` 表示一个候选解，`fitness function` 衡量候选解质量。算法倾向于选择高适应度个体，让它们通过 `crossover` 重组，通过 `mutation` 产生小扰动，逐代搜索更优区域。

标准流程：

```
set time t = 0
initialize the population P(t)
while termination condition is not met:
    evaluate the fitness of each member of P(t)
    select members from P(t) based on fitness
    produce offspring using genetic operators
    replace candidates in P(t) with these offspring
    set time t = t + 1
```

各部件作用：

Component	中文	作用
`population`	种群	多个候选解的集合
`chromosome`	染色体	一个被编码的候选解
`gene`	基因	染色体中的一个位置
`fitness`	适应度	候选解质量；越大通常越好
`selection`	选择	让高适应度个体更可能产生后代
`crossover`	交叉	把两个父代的部分结构组合起来
`mutation`	变异	随机改变基因，避免种群过早集中
`replacement`	替换	用子代更新下一代种群

终止条件可以是达到最大代数、fitness 达到目标、种群变化很小，或计算预算耗尽。

2. 例题：最大化 $f(x) = x^2$

题目：

$$\max f(x) = x^2, \quad x \in [1, 31]$$

2.1 Representation 编码

因为 $\mathbf{1}$ 到 $\mathbf{31}$ 可由 5 位二进制表示，所以用 5-bit binary string 表示 $\boldsymbol{x}$。

初始种群：

Individual	二进制解释 x	Fitness $f(x)=x^2$
`01101`	13	169
`11000`	24	576
`01000`	8	64
`10011`	19	361

总适应度：

$$\sum_i f(i) = 169 + 576 + 64 + 361 = 1170$$

2.2 Roulette Wheel Selection 轮盘赌选择

选择概率：

$$p_i = \frac{f(i)}{\sum_{j=1}^{n} f(j)}$$

其中 $f(i)$ 是第 i 个个体的适应度，分母是种群总适应度。

Individual	Fitness	Probability
`01101`	169	$\mathbf{169/1170 \approx 0.14}$
`11000`	576	$\mathbf{576/1170 \approx 0.49}$
`01000`	64	$\mathbf{64/1170 \approx 0.06}$
`10011`	361	$\mathbf{361/1170 \approx 0.31}$

含义：把轮盘按 14%、49%、6%、31% 分成四块，随机指针落在哪块就选哪个个体。高 fitness 个体占的扇区更大，因此被复制到下一代的概率更高。

课件示例中一次选择结果为：

```
01101, 11000, 11000, 10011
```

注意：`11000` 被选中两次，因为它 fitness 最高；`01000` 没被选中，因为它 fitness 最低但概率不是 0。

2.3 Crossover 交叉

对选择后的个体做单点交叉。示例：

```
0110 | 1        0110 | 0
1100 | 0   ->   1100 | 1
```

中文解释：竖线是切点。切点前的片段保留，切点后的片段在两个父代之间交换。

课件最终得到的下一代示例：

```
01101, 11000, 11000, 10011
```

经过交叉：

```
01101 + 11000 -> 01100 + 11001
11000 + 10011 -> 11011 + 10000
```

2.4 Mutation 变异

变异随机改变某些基因。二进制编码中通常是 bit flip：

```
01100 -> 11100
```

中文解释：第一个 bit 从 0 变成 1。变异概率通常较小，它的作用不是主要复制优秀结构，而是维持多样性、引入种群中没有的基因，帮助跳出局部最优。

3. GA 可用于哪些问题

3.1 Knapsack Problem 背包问题

问题：有 n 个物品，第 i 个物品价值 v_i、重量 w_i。背包最多承重 W，要选择一组物品，使总价值尽量大且总重量不超过 W。

编码：

```
chromosome bit i = 1: 选择第 i 个物品
chromosome bit i = 0: 不选择第 i 个物品
```

总价值：

$$V(x) = \sum_i x_i v_i$$

总重量：

$$Weight(x) = \sum_i x_i w_i$$

一种 fitness 设计：

$$fitness(x) = \sum_i x_i v_i - \lambda \max(0, \sum_i x_i w_i - W)$$

含义：不超重时 fitness 等于总价值；超重时扣除惩罚项。λ 控制超重惩罚强度。

3.2 CNF-Satisfaction Problem 合取范式可满足性

CNF `conjunctive normal form` 是若干子句用 AND 连接，每个子句是若干文字 `literal` 用 OR 连接。

例子：

$$(\neg a \vee c) \wedge (\neg a \vee c \vee \neg e) \wedge (\neg b \vee c \vee d \vee \neg e) \wedge (a \vee \neg b \vee c) \wedge (\neg e \vee v)$$

GA 解法思路：

- 染色体表示变量赋值，例如每个 bit 对应一个命题变量是真还是假。
- fitness 可以设为“被满足的子句数量”。
- 如果一个赋值满足所有子句，则找到可满足解。

语义树 `semantic tree` 用树分支枚举变量取值。以集合

$$\{p(x), \neg p(x) \vee q(x), \neg q(f(a))\}$$

为例，树的分支可以先对 $p(a)$ 取真/假，再对 $q(a)$、$p(f(a))$ 等实例取真/假。每条从根到叶的路径表示一种可能解释；如果某路径同时包含互相矛盾的文字，该分支关闭。语义树的意义是把公式可满足性转成对赋值分支的检查。

3.3 Traveling Salesman Problem, TSP 旅行商问题

TSP 问题：给定若干城市和城市间距离，找一条路径，从某城市出发，访问每个城市恰好一次并回到起点，使总距离最短。

复杂性示例：

- 19 个城市可能路线数约为 $18! = 6.40237 \times 10^{15}$。
- 如果每条路线检查时间为 $80 \times 365 \times 24 \times 60 \times 60$ 的量级，穷举不可行。
- 课件估算即使计算速度为 10000 routes/second，也需要约 253.77 generations/year 量级的时间。

编码：permutation。

```
(1 9 2 4 6 5 7 8 3)
```

表示按该顺序访问城市。TSP 的交叉和变异必须保持排列合法：不能重复城市，也不能漏城市。

TSP Crossover 顺序交叉

父代：

```
p1 = (1 9 2 | 4 6 5 7 | 8 3)
p2 = (4 5 9 | 1 8 7 6 | 2 3)
```

切点之间片段来自一个父代。以生成 c_1 为例，保留 p_1 中间片段：

```
_ _ _ | 4 6 5 7 | _ _
```

再从另一个父代 p_2 的第二个切点之后开始读，读到末尾后从开头继续，得到顺序：

```
2 3 4 5 9 1 8 7 6
```

跳过已经在子代中出现的 `4,6,5,7`，按顺序填空：

```
c1 = (2 3 9 | 4 6 5 7 | 1 8)
```

同理，保留 p_2 中间片段并按 p_1 的顺序填空：

```
c2 = (3 9 2 | 1 8 7 6 | 4 5)
```

这个过程的核心是：重组路径片段，同时保持每个城市只出现一次。

TSP Mutation: Inversion 反转变异

对子代：

```
c1 = (2 3 9 | 4 6 5 7 | 1 8)
```

把两个切点之间的片段 `4 6 5 7` 反转，得到：

```
c1 = (2 3 9 | 7 5 6 4 | 1 8)
```

中文解释：inversion mutation 不引入重复城市，只改变局部访问顺序，适合排列编码。

4. Encoding 编码方式

4.1 Binary Coding

二进制编码适合整数优化、0/1 选择、布尔变量赋值等问题。例子：5 位字符串 `11000` 可解释为整数 24。

4.2 Gray Coding

Gray code 的特点：相邻数值的编码只相差 1 个 bit。

Decimal	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

中文解释：普通二进制中相邻数可能差很多位，例如 7 是 `0111`，8 是 `1000`，四位全变；Gray code 让相邻数只差一位，使小的基因变化更可能对应小的数值变化。

4.3 Permutation Coding

排列编码适合 TSP、排序、调度、路径规划。操作必须保持排列合法，因此常使用顺序交叉、部分映射交叉、swap mutation、inversion mutation 等。

5. GA 为什么有搜索能力

课件中的搜索空间图可以转写成下面这条逻辑：

1. `initialization`：初始种群把点随机撒在搜索空间各处。
2. `selection`：高 fitness 区域的个体更容易被保留，搜索开始集中。
3. `crossover`：在已有较好个体附近重组，把不同个体的好片段组合。
4. `mutation`：在局部区域周围产生扰动，也可能跳到新区域。
5. 多代之后，种群分布从随机分散逐渐偏向高质量解区域。

优势：

- 同时搜索多个候选解，而不是单点搜索。
- 不要求目标函数连续或可导。
- 适合离散、组合、非凸、多峰问题。
- 交叉可以组合已有优良结构。
- 变异可以维持多样性，减少早熟收敛。

局限：

- 不能保证每次都找到全局最优。
- fitness、编码、交叉和变异设计会显著影响效果。

- 参数如种群大小、交叉率、变异率需要调节。

6. Evolutionary Hardware 演化硬件

课件给出 `PLA, Programmable Logic Array` 作为演化硬件例子。思想是：硬件配置也可以编码成染色体，遗传算法搜索不同电路配置，使输出行为逐渐接近期望逻辑功能。

作答时可写：演化硬件把电路结构或连接方式作为可进化对象，fitness 衡量电路输出与目标输出的匹配程度，GA 通过选择、交叉、变异寻找较优硬件配置。

7. Schema 与 Schema Theorem

7.1 Schema 是什么

`schema` 是带通配符 `*` 的字符串模板，表示一组具有共同位置模式的染色体。

例子：

```
(1 * * 0 * 1)
```

表示所有长度为 6，且第 1 位为 1、第 4 位为 0、第 6 位为 1 的字符串；第 2、3、5 位可以是 0 或 1。

7.2 Order 阶

`order` 记为 $O(s)$，是 schema 中固定位置的数量。

对：

```
s = (1 * * 0 * 1)
```

固定位置为第 1、4、6 位，所以：

$$O(s) = 3$$

阶越低，固定位置越少，越不容易被变异破坏。

7.3 Defining Length 定义长度

`defining length` 记为 $\delta(s)$，是最左固定位置和最右固定位置之间的距离。

对：

```
s = (1 * * 0 * 1)
```

最左固定位置是 1，最右固定位置是 6，所以：

$$\delta(s) = 6 - 1 = 5$$

定义长度越短，单点交叉越不容易切断 schema 的固定结构。

7.4 Schema Fitness

schema 的 fitness 是当前种群中所有匹配该 schema 的个体的平均适应度：

$$\bar{f}(s) = \text{average fitness of strings matching schema } s$$

种群平均适应度：

$$\bar{f} = \frac{\sum_{j=1}^{n} f(j)}{n}$$

其中 n 是种群大小。

7.5 Selection 对 schema 数量的影响

第 i 个字符串被选择的概率：

$$p_i = \frac{f(i)}{\sum_{j=1}^{n} f(j)}$$

令 $m(s,t)$ 表示第 t 代中属于 schema s 的字符串数量。仅考虑选择时，下一代中 schema s 的期望数量：

$$E[m(s,t+1)] = m(s,t) \cdot n \cdot \frac{\bar{f}(s)}{\sum_{j=1}^{n} f(j)}$$

因为：

$$\bar{f} = \frac{\sum_{j=1}^{n} f(j)}{n}$$

所以：

$$E[m(s,t+1)] = m(s,t)\cdot\frac{\bar{f}(s)}{\bar{f}}$$

中文解释：如果某 schema 的平均适应度高于种群平均适应度，即 $\bar{f}(s) > \bar{f}$，它在选择后期望数量会增加。

若：

$$\bar{f}(s) = (1+c)\bar{f},\quad c>0$$

则：

$$E[m(s,t+1)] = m(s,t)(1+c)$$

连续多代近似为：

$$m(s,t) \approx m(s,0)(1+c)^t$$

这解释了“高于平均适应度的 schema 会增长”。

7.6 Crossover 对 schema 的破坏概率

设染色体长度为 l，单点交叉有 $l-1$ 个可能切点。若切点落在 schema 的最左固定位置和最右固定位置之间，就可能破坏该 schema。

不被交叉破坏的概率下界：

$$p_s \geq 1-\frac{\delta(s)}{l-1}$$

若交叉概率为 p_c：

$$p_s \geq 1-p_c\frac{\delta(s)}{l-1}$$

中文解释：$\delta(s)$ 越短，切点落入关键区间的机会越小，schema 越容易被保留。

7.7 Mutation 对 schema 的破坏概率

设每个位的变异概率为 p_m。schema 的固定位置有 $O(s)$ 个，这些位置都不变异时 schema 才保持。

不被变异破坏的概率：

$$p_s = (1-p_m)^{O(s)}$$

当 $p_m \ll 1$ 时近似为：

$$p_s \approx 1-p_mO(s)$$

中文解释：$O(s)$ 越低，需要保护的固定位越少，越不容易被 mutation 破坏。

7.8 Schema Theorem 完整形式

把 selection、crossover、mutation 合在一起，得到：

$$E[m(s,t+1)] \geq m(s,t)\frac{\bar{f}(s)}{\bar{f}}\left[1-p_c\frac{\delta(s)}{l-1}-O(s)p_m\right]$$

变量含义：

Symbol	中文含义
$m(s,t)$	第 t 代中属于 schema s 的个体数
$\bar{f}(s)$	schema s 中个体的平均 fitness
$\bar{f}$	整个种群的平均 fitness
$\delta(s)$	schema 的 defining length
$O(s)$	schema 的 order
l	染色体长度
p_c	crossover probability
p_m	mutation probability

定理含义：短、低阶、平均适应度高于种群平均的 schema，在连续世代中会以指数方式增加。英文常写作：`short, low-order schemata with above-average fitness increase exponentially in successive generations`。

中文作答时解释成：高适应度让它更容易被选择；短定义长度让它不易被交叉切断；低阶让它不易被变异破坏。

8. 常见答题模板

8.1 问：Describe the genetic algorithm

中文作答：

遗传算法是一种基于自然进化思想的随机搜索方法。它先把候选解编码为染色体，并初始化一个种群；然后用 fitness function 评价每个个体；再根据适应度进行选择，使优良个体更可能繁殖；通过 crossover 重组父代结构，通过 mutation 引入随机变化；最后用子代替换部分或全部旧种群，迭代直到满足终止条件。

8.2 问：Explain roulette wheel selection

中文作答：

轮盘赌选择把每个个体的适应度转化为选择概率，公式为 $p_i = f(i) / \sum_j f(j)$。适应度越高，个体在轮盘中占的扇区越大，被随机选中的概率越高。这种方法不是确定地只选最优个体，因此仍保留一定多样性。

8.3 问：Why are crossover and mutation both needed?

中文作答：

crossover 负责重组两个父代中已有的结构，把不同个体中的有用片段组合成新个体；mutation 负责随机改变基因，引入新变化并维持种群多样性。只有选择和交叉时，种群可能过早集中到局部区域；变异可以帮助探索未出现过的区域。

8.4 问：Explain schema theorem

中文作答：

schema 是带通配符的字符串模板，表示一类具有共同固定位点的染色体。schema theorem 说明，短、低阶、平均适应度高于种群平均的 schema 会在后代中快速增加。原因是：高适应度使它更容易被选择，短 defining length 使它不容易被 crossover 破坏，低 order 使它不容易被 mutation 破坏。

9. 易错点

- roulette wheel selection 中低 fitness 个体仍可能被选中，只是概率较小。
- TSP 不能直接用普通单点交叉，否则可能产生重复城市或漏城市。
- Gray code 的优势是相邻值编码只差一位，不是 fitness 更高。
- schema 的 `order` 是固定位置数量，`defining length` 是首末固定位置距离。
- schema theorem 是期望意义上的增长规律，不表示每次运行都严格增长。
- mutation rate 通常较小；太大时搜索会接近随机扰动，难以保留优秀结构。

07 Logics and Prolog 考场资料

本章主线：逻辑提供知识表示和推理的形式基础；Prolog 则把 Horn clauses 变成可执行的逻辑程序。考试常见任务不是背定义，而是判断 WFF、做 unification、转 clause form、用 resolution refutation 证明目标、解释 Prolog 的匹配/回溯/递归程序。

0. 英文题目触发词速查

English trigger	中文定位	作答时要写出的核心
`propositional logic`	命题逻辑	命题符号、真值、连接词、真值表
`WFF`, `well-formed formula`	合式公式	按语法构造规则判断是否合法
`interpretation`, `model`, `valid`, `satisfiable`	解释、模型、有效、可满足	解释赋真值/对象含义；模型使公式真；valid 是所有解释真
`first-order logic`, `predicate calculus`	一阶逻辑	常量、变量、谓词、函数、量词
`universal quantifier`, `existential quantifier`	全称/存在量词	$\forall x$ 对所有对象；$\exists x$ 至少存在一个对象
`inference rule`	推理规则	modus ponens, modus tollens, elimination, introduction, universal instantiation
`unification`, `mgu`	合一、最一般合一	找 substitution 使表达式相同
`clause form`, `CNF`	子句形式/合取范式	消蕴含、内移否定、标准化变量、Skolemization、分配律
`resolution refutation`	归结反证	加入目标否定，归结推出 empty clause
`answer extraction`	答案抽取	从产生空子句的 substitutions 还原变量答案
`Horn clause`	Horn 子句	至多一个正文字；Prolog 规则基础
`Prolog`	逻辑程序设计	facts, rules, goals, unification, backtracking
`closed world assumption`, `negation as failure`	闭世界假设/失败即否定	查不到就按 false 处理
`list`, `match`, `member`	列表/匹配/member	`[H
`cut`	剪枝	`!` 固定当前选择，阻止部分回溯

1. 为什么逻辑是 Knowledge Representation 的基础

知识表示的基础包括：

- `Logics`：给出符号、语义和推理规则。
- `Ontology`：定义领域中的对象、类别和关系。
- `Theory of Computation`：说明哪些推理过程可计算、复杂度如何。

逻辑的价值：它让知识不仅能被存储，还能被检查真值、判断是否满足、进行证明。

2. Propositional Logic 命题逻辑

2.1 符号和连接词

Symbol	English	中文
P, Q, R, S	propositional symbols	命题符号
`true`, `false`	truth symbols	真值符号
$\wedge$	conjunction	与
$\vee$	disjunction	或
$\neg$	not	非
$\rightarrow$	implication	蕴含
$\equiv$	equivalence	等价

命题逻辑不分析命题内部结构。例如 `It has rained` 和 `Ground is wet` 可以分别记作 P、Q。

2.2 WFF 合式公式

构造规则：

1. 每个命题符号和真值符号都是 sentence。
2. 若 P 是 sentence，则 $\neg P$ 是 sentence。
3. 若 P, Q 是 sentence，则 $P \wedge Q$、$P \vee Q$、$P \rightarrow Q$、$P \equiv Q$ 都是 sentence。

合法句子也称为 `well-formed formula, WFF`。

2.3 语义和真值条件

`interpretation` 在命题逻辑中就是给每个命题符号赋 `true` 或 `false`。

Formula	为真的条件
$\neg P$	P 为假
$P \wedge Q$	P 和 Q 都真
$P \vee Q$	至少一个为真
$P \rightarrow Q$	只有 P 真且 Q 假时为假；其他情况为真
$P \equiv Q$	两者真值相同

例子：`It has rained -> Ground is wet`，只有“下雨为真但地面湿为假”时蕴含式为假。

2.4 常用等价式

双重否定：

$$\neg(\neg P) \equiv P$$

蕴含消去：

$$P \rightarrow Q \equiv \neg P \vee Q$$

逆否律：

$$P \rightarrow Q \equiv \neg Q \rightarrow \neg P$$

De Morgan 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：

$$P \vee Q \equiv Q \vee P, \quad P \wedge Q \equiv Q \wedge P$$

分配律：

$$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$

这些等价式在 clause form 转换中反复使用。

3. First-Order Logic 一阶逻辑

命题逻辑只能把整句话当作原子命题；一阶逻辑能表达对象、属性和关系。

3.1 基本符号

Symbol type	例子	中文
truth symbol	`true`，`false`	真值
constant symbol	`socrates`，`john`，`0`	常量，表示具体对象
variable symbol	`x`，`y`，`z`	变量
predicate symbol	`man(x)`，`parent(x,y)`	谓词，表示属性或关系
function symbol	`father(x)`，`minsavings(y)`	函数，返回对象
quantifier	∀,∃	全称/存在量词

一阶逻辑的量词只作用于对象变量，不能量化谓词或函数。这是 `first-order` 的含义。

3.2 Atomic Sentence 原子句

谓词加参数形成原子句：

```
man(socrates)
parent(x,y)
greater(x, minsavings(y))
```

谓词的参数个数叫 `arity`。例如 `parent(x,y)` 是二元谓词。

3.3 Quantifiers 量词

全称：

$$\forall x(man(x) \rightarrow mortal(x))$$

中文：对所有 x，如果 x 是人，则 x 会死。

存在：

$$\exists x(likes(x, taxes))$$

中文：存在某个 x，x 喜欢税。

3.4 Family relationship 例子

事实：

```
mother(eve, abel)
mother(eve, cain)
father(adam, abel)
father(adam, cain)
```

规则：

$$\forall x \forall y((father(x, y) \vee mother(x, y)) \rightarrow parent(x, y))$$

中文：如果 x 是 y 的父亲或母亲，则 x 是 y 的 parent。

$$\forall x \forall y \forall z((parent(x, y) \wedge parent(x, z)) \rightarrow sibling(y, z))$$

中文：如果 y 和 z 有共同 parent x，则 y 和 z 是 sibling。实际应用中通常还要加 $y \neq z$，否则会推出自己和自己是 sibling。

4. Interpretation, Model, Valid

一阶逻辑解释 `interpretation` 包括：

- 给每个常量分配论域 D 中的一个对象。
- 给每个函数分配从对象到对象的映射。
- 给每个谓词分配在论域上的关系。
- 给变量分配允许的对象。

重要概念：

Term	中文解释
`satisfies`	某解释使公式为真
`model`	使某公式或公式集合为真的解释
`satisfiable`	至少存在一个模型
`unsatisfiable` / `inconsistent`	没有任何模型
`valid`	所有解释下都为真
`logically follows`	若所有满足前提集 S 的解释也满足 x，则 x 从 S 逻辑推出
`sound`	推理系统推出的结论都是真的逻辑后承
`complete`	所有真的逻辑后承都能被推理系统推出

5. Inference Rules 推理规则

5.1 Modus Ponens

$$P, \quad P \rightarrow Q \quad \therefore Q$$

5.2 Modus Tollens

$$P \rightarrow Q, \quad \neg Q \quad \therefore \neg P$$

5.3 Elimination

$$P \wedge Q \quad \therefore P$$

$$P \wedge Q \quad \therefore Q$$

5.4 Introduction

$$P, Q \quad \therefore P \wedge Q$$

5.5 Universal Instantiation

$$\forall x \, P(x) \quad \therefore P(a)$$

Socrates 例子：

前提：

$$\forall x(man(x) \rightarrow mortal(x))$$

$$man(socrates)$$

用 $socrates/x$ 实例化：

$$man(socrates) \rightarrow mortal(socrates)$$

再用 modus ponens：

$$mortal(socrates)$$

6. Unification 合一

`unification` 是找一个 substitution，使两个表达式变成相同。`mgu, most general unifier` 是最一般合一，不做多余限制。

规则：

- 常量只能和相同常量合一。
- 变量可以和常量、变量、复合项合一。
- 同一个变量在所有位置必须替换一致。
- `occurs check`：变量不能替换成包含自身的项，例如 x 不能与 $f(x)$ 合一。

例子：

```
parents(x, father(x), mother(bill))
parents(bill, father(bill), y)
```

先让第一参数相同：

```
{bill/x}
parents(bill, father(bill), mother(bill))
parents(bill, father(bill), y)
```

再让第三参数相同：

```
{mother(bill)/y}
```

mgu：

```
{bill/x, mother(bill)/y}
```

7. Logic-Based Financial Advisor

这个例子说明一阶逻辑如何表达专家规则。

规则核心：

1. 如果 savings account 不足，则应增加 savings：

$$savings\_account(inadequate) \rightarrow investment(savings)$$

2. 如果 savings account 充足且 income 充足，则投资 stocks：

$$savings\_account(adequate) \land income(adequate) \rightarrow investment(stocks)$$

3. 如果 savings account 充足但 income 不足，则选择 combination：

$$savings\_account(adequate) \land income(inadequate) \rightarrow investment(combination)$$

事实：

```
amount_saved(22000)
earnings(25000, steady)
dependents(3)
```

函数：

```
minsavings(x) = 5000 * x
minincome(x) = 15000 + 4000 * x
```

计算：

```
minsavings(3) = 15000
minincome(3) = 27000
```

因为 `22000 > 15000`，所以 savings account adequate。因为 `25000 < 27000`，所以 income inadequate。由规则 3 推出：

```
investment(combination)
```

中文结论：该用户储蓄足够，但收入低于有 3 个 dependent 时的最低收入要求，因此建议组合投资。

8. Clause Form 与 Resolution Refutation

8.1 Resolution Refutation 步骤

归结反证证明 $\boldsymbol{Goal}$：

1. 把所有 premises/axioms 转成 clause form。
2. 把目标的否定 $\neg\boldsymbol{Goal}$ 也转成 clause form，并加入子句集。
3. 反复选两个含互补文字的子句做 resolution。
4. 若推出 empty clause □，说明子句集不可满足。
5. 因为 premises 加上 $\neg\boldsymbol{Goal}$ 矛盾，所以 premises 蕴含 Goal。

重要：如果目标带变量，产生空子句过程中使用的 substitutions 可用于 answer extraction。

8.2 Resolution 规则

命题形式：

$$(A \vee B),\quad (\neg B \vee C)\quad \therefore A \vee C$$

一阶形式需要先合一互补文字。例如 $\boldsymbol{animal(x)}$ 和 $\neg\boldsymbol{animal(fido)}$ 可用 $\boldsymbol{fido/x}$ 合一后归结。

8.3 Fido 例题

前提：

$$\forall x(dog(x) \to animal(x))$$

$$\forall y(animal(y) \to die(y))$$

$$dog(fido)$$

目标：

$$die(fido)$$

转 clause form：

```
¬dog(x) ∨ animal(x)
¬animal(y) ∨ die(y)
dog(fido)
¬die(fido)          negated goal
```

归结：

1. 用 $\boldsymbol{y/x}$ 归结前两个子句：

```
¬dog(y) ∨ die(y)
```

2. 与 `dog(fido)` 用 $\boldsymbol{fido/y}$ 归结：

```
die(fido)
```

3. 与 `¬die(fido)` 归结：

```
□
```

结论：前提推出 Fido will die。

8.4 Producing Clause Form 标准流程

把公式转子句形式的步骤：

1. 消去蕴含：

$$a \to b \equiv \neg a \vee b$$

2. 把否定向内推：

$$\neg(\neg a) \equiv a$$

$$\neg(a \wedge b) \equiv \neg a \vee \neg b$$

$$\neg(a \vee b) \equiv \neg a \wedge \neg b$$

$$\neg\exists x\, a(x) \equiv \forall x\, \neg a(x)$$

$$\neg\forall x\, a(x) \equiv \exists x\, \neg a(x)$$

3. 标准化变量 apart：不同量词绑定的变量改成不同名字。
4. 把量词移到最左边，保持相对顺序，形成 prenex form。
5. Skolemization：消去存在量词。

6. 删除全称量词。
7. 用分配律变成 AND of ORs。
8. 每个合取项单独作为一个 clause。
9. 再次标准化每个 clause 的变量，避免不同 clause 变量意外同名。

Skolemization 规则：

- $\exists y$ 不在任何全称变量作用域内：用新的 Skolem constant 替换。
- $\exists y$ 在 $\forall x$ 作用域内：用 Skolem function $f(x)$ 替换。
- $\exists z$ 在 $\forall x \forall y$ 作用域内：用 $g(x, y)$ 替换。

8.5 Happy Student 例题

故事：

- 通过历史考试并中彩票的人是 happy。
- 学习或幸运的人能通过所有考试。
- John 没有学习，但 John 幸运。
- 幸运的人会中彩票。
- 问：John 是否 happy?

子句形式可写成：

```
¬pass(x, history) ∨ ¬win(x, lottery) ∨ happy(x)
¬study(y) ∨ pass(y, z)
¬lucky(u) ∨ pass(u, v)
¬lucky(w) ∨ win(w, lottery)
¬study(john)
lucky(john)
¬happy(john)       negated goal
```

归结路线：

1. 用幸运推出 John 通过 history：

```
¬lucky(w) ∨ pass(w, v)
lucky(john)
=> pass(john, v)
```

令 $v = history$ 得到 `pass(john, history)`。

2. 用幸运推出 John 中彩票：

```
¬lucky(u) ∨ win(u, lottery)
lucky(john)
=> win(john, lottery)
```

3. 与 happy 规则和 negated goal 归结：

```
¬pass(john, history) ∨ ¬win(john, lottery) ∨ happy(john)
¬happy(john)
=> ¬pass(john, history) ∨ ¬win(john, lottery)
```

再分别用 `pass(john, history)` 和 `win(john, lottery)` 消去，得到 □。

结论：John is happy。

8.6 Exciting Life 与 Answer Extraction

故事：

- 非 poor 且 smart 的人 happy。
- 会 read 的人 smart。
- John not poor 且 read。
- happy 的人有 exciting life。
- 问：是否能找到有 exciting life 的人?

子句：

```
poor(x) ∨ ¬smart(x) ∨ happy(x)
¬read(y) ∨ smart(y)
¬poor(john)
read(john)
¬happy(z) ∨ exciting(z)
¬exciting(w)       negated goal
```

归结核心：

1. `¬happy(z) ∨ exciting(z)` 与 `¬exciting(w)`，用 `{z/w}`：

```
¬happy(z)
```

2. 与 `poor(x) ∨ ¬smart(x) ∨ happy(x)`，用 `{x/z}`：

```
poor(x) ∨ ¬smart(x)
```

3. 与 `¬read(y) ∨ smart(y)`，用 `{y/x}`：

```
poor(y) ∨ ¬read(y)
```

4. 与 `¬poor(john)`，用 `{john/y}`：

```
¬read(john)
```

5. 与 `read(john)`：

```
□
```

答案抽取：

```
exciting(w)
{z/w}, {x/z}, {y/x}, {john/y}
=> exciting(john)
```

结论：找到的答案是 John has an exciting life。

8.7 Grandparent 例题

故事：

- 每个人都有 parent。
- parent 的 parent 是 grandparent。
- 给定 John，证明 John 有 grandparent。

用 Skolem function 表示“每个人的某个 parent”：

```
parent(x, pa(x))
```

规则：

```
¬parent(w,y) ∨ ¬parent(y,z) ∨ grandparent(w,z)
```

目标否定：

```
¬grandparent(john,v)
```

利用：

```
parent(john, pa(john))
parent(pa(john), pa(pa(john)))
```

归结得到：

```
grandparent(john, pa(pa(john)))
```

答案：John 的 grandparent 可以表示为 `pa(pa(john))`。

9. Resolution Strategies 与 Herbrand/Semantic Tree

策略：

- `breadth-first strategy`：广度优先枚举可能归结。
- `set of support strategy`：优先使用与目标否定相关的子句。
- `unit preference strategy`：优先使用单文字子句。
- `linear input form strategy`：每次至少有一个父子句来自原始输入或上一步结果；不总是 complete。

Herbrand domain：

- 若子句集中有常量，则 H_0 为所有常量集合；若没有常量，引入一个常量。
- 若有函数 f，用 H_i 中项不断生成 $f(t_1, \ldots, t_n)$。
- Herbrand universe 是所有这些 ground terms 的并集。

Semantic tree：

以

$$\{p(x), \neg p(x) \vee q(x), \neg q(f(a))\}$$

为例，语义树对 ground atoms 如 $p(a)$、$q(a)$、$p(f(a))$、$q(f(a))$ 等逐个分支取真/假。若某分支和子句集矛盾则关闭；若所有分支关闭，子句集不可满足。

10. 其他逻辑系统

System	中文定位
expressive but undecidable logics	表达能力强但不可判定，如高阶逻辑或复杂一阶变体
quantifier-free formalisms	无量词形式，复杂度低一些，如部分 description logic
default reasoning	默认推理，允许通常成立但可被例外推翻
circumscription	最小化异常或某些谓词的扩张
modal logic	模态逻辑，表达 necessarily $\Box p$、possibly $\Diamond p$
truth maintenance system	为每个结论保存 justification，前提变化时更新结论

模态逻辑常见等式：

$$\Box p \leftrightarrow \neg\Diamond\neg p$$

以及：

$$\Box(p \to q) \to (\Box p \to \Box q)$$

11. Horn Clauses 与 Prolog

11.1 Horn Clause

Horn clause 至多有一个正文字：

$$a \vee \neg b_1 \vee \neg b_2 \vee \cdots \vee \neg b_n$$

通常写成 implication：

$$a \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

意思：如果 $b_1, \ldots, b_n$ 都成立，则 a 成立。

三种形式：

Form	写法	中文
fact	$a \leftarrow$	事实，无条件成立
rule	$a \leftarrow b_1 \wedge \cdots \wedge b_n$	规则
goal/headless clause	$\leftarrow a_1 \wedge \cdots \wedge a_n$	查询目标

11.2 Prolog 执行机制

给定 goal：

```
?- a1, a2, ..., an.
```

Prolog：

1. 从左到右选择最左 goal `a1`。
2. 从程序顶部开始找第一个 head 能与 `a1` 合一的 fact/rule。
3. 若匹配到 rule：

```
a1 :- b1, b2, ..., bm.
```

则把当前 goal 替换为：

```
b1, b2, ..., bm, a2, ..., an
```

并应用合一 substitution。 4. 若某路径失败，则 backtrack 到最近选择点，尝试下一条 fact/rule。 5. 若目标化为空目标，则成功。

11.3 Prolog 语法对照

Logic	Prolog
and $\wedge$	`,`
or $\vee$	`;`
implication $\leftarrow$	`:-`
not	`not` 或 `\\+`
variable	大写开头，如 `X`
atom/constant	小写开头，如 `john`

11.4 Facts, Rules, Queries

事实：

```
likes(george, kate).
likes(george, susie).
likes(george, wine).
likes(susie, wine).
likes(kate, gin).
```

查询：

```
?- likes(george, susie).
yes

?- likes(george, gin).
no

?- likes(george, X).
X = kate ;
X = susie ;
X = wine ;
no
```

规则：

```
friends(X,Y) :-
    likes(X,Z),
    likes(Y,Z).
```

解释：如果 X 和 Y 喜欢同一个 Z，则他们是 friends。

查询：

```
?- friends(george, susie).
yes
```

因为二者都喜欢 `wine`。

`closed world assumption`：Prolog 数据库中无法证明为真的事实，会被当作 false。因此 `likes(george, gin)` 返回 `no`，不是因为系统知道 George 讨厌 gin，而是因为没有可证明的事实。

12. Lists, Match, Member

12.1 List 表示

```
[1,2,3,4]
[[george,kate], [allen,amy], [don,pat]]
[tom,dick,harry,fred]
[]
```

`[H|T]` 表示列表头和尾：

- `H` 是第一个元素。
- `T` 是剩余列表。

12.2 Match 例子

```
[tom,dick,harry] = [X|Y]
```

得到：

```
X = tom
Y = [dick,harry]
```

```
[tom,dick,harry] = [X,Y|Z]
```

得到：

```
X = tom
Y = dick
Z = [harry]
```

```
[tom,dick,harry] = [X,Y,Z|W]
```

得到：

```
X = tom
Y = dick
Z = harry
W = []
```

```
[tom,dick,harry] = [tom, X, [harry]]
```

得到：

```
X = dick
```

但：

```
[tom,dick,harry] = [X,Y,Z,W|U]
```

不能匹配，因为左边只有 3 个元素，右边至少需要 4 个。

12.3 Member 程序

```
member(X, [X|T]).
member(X, [Y|T]) :-
    member(X, T).
```

第一条：X 是列表头，则 X 是列表成员。

第二条：如果 X 不是当前头，就递归检查尾列表。

查询：

```
?- member(a, [a,b,c,d,e]).
yes

?- member(a, [1,2,3,4]).
no

?- member(X, [a,b,c]).
X = a ;
X = b ;
X = c ;
no
```

执行 `member(c,[a,b,c])`：

1. 与第一条尝试：`c` 与 `a` 不同，失败。
2. 用第二条递归到 `member(c,[b,c])`。
3. 第一条：`c` 与 `b` 不同，失败。
4. 第二条递归到 `member(c,[c])`。
5. 第一条成功。

13. 递归、顺序与回溯

祖先关系：

```
predecessor(Parent, Child) :-
    parent(Parent, Child).

predecessor(Predecessor, Successor) :-
    parent(Predecessor, Child),
    predecessor(Child, Successor).
```

解释：第一条是基例，直接 parent 是 predecessor；第二条是递归，若 Predecessor 是 Child 的 parent，且 Child 是 Successor 的 predecessor，则 Predecessor 也是 Successor 的 predecessor。

Prolog 对子目标从左到右执行，对规则从上到下尝试。因此目标顺序和规则顺序可能影响终止性和搜索效率。把递归调用放在过早位置可能导致无限递归。

14. Search Problems in Prolog

14.1 Knight's Tour / Path Search

棋盘编号 1 到 9，`move(X,Y)` 表示马可以从 X 跳到 Y。

路径程序模式：

```
path(X, X, L).
path(X, Y, L) :-
    move(X, Z),
    not(member(Z, L)),
    path(Z, Y, [Z|L]).
```

含义：

- 基例：起点等于终点，路径找到。
- 递归：从 X 找一个可走的 Z，要求 Z 不在已访问列表 L 中，避免循环；再从 Z 继续找 Y。
- 若某条路失败，Prolog backtrack 尝试下一个 `move`。

14.2 Cut 控制搜索

无 cut：

```
path2(X,Y) :-
    move(X,Z),
    move(Z,Y).
```

查询：

```
?- path2(1,W).
W = 7 ;
W = 1 ;
W = 3 ;
W = 1 ;
no
```

带 cut：

```
path3(X,Y) :-
    move(X,Z),
    !,
    move(Z,Y).
```

查询结果：

```
?- path3(1,W).
W = 7 ;
W = 1 ;
no
```

解释：`!` 一旦执行，就固定此前对 `move(X,Z)` 的选择，不再回溯尝试其他 Z。cut 可提高效率，但会改变程序逻辑含义，使用要谨慎。

14.3 Farmer, Wolf, Goat, Cabbage

状态写成：

```
state(Farmer, Wolf, Goat, Cabbage)
```

每个位置是 `w` 或 `e`，表示河西岸或东岸。初始：

```
state(w,w,w,w)
```

目标：

```
state(e,e,e,e)
```

安全约束：

- farmer 不在时，wolf 和 goat 不能单独在一起。
- farmer 不在时，goat 和 cabbage 不能单独在一起。

动作包括：

- farmer alone 过河。
- farmer takes wolf。
- farmer takes goat。
- farmer takes cabbage。

课件给出的求解过程可文字化为：

```
farmer takes goat to east
farmer returns alone to west
farmer takes wolf to east
farmer brings goat back to west
farmer takes cabbage to east
farmer returns alone to west
farmer takes goat to east
```

对应最终所有对象都到 east，且中途不违反安全约束。

15. 常见答题模板

15.1 问：Explain resolution refutation

中文作答：

归结反证先把前提转为 clause form，再把待证目标的否定加入子句集。然后反复对含互补文字的子句进行 resolution，若推出 empty clause，说明前提与目标否定不可同时满足，因此前提蕴含目标。如果目标含变量，产生空子句时使用的 substitutions 可以用于抽取答案。

15.2 问：How to convert to clause form?

中文作答：

先消去蕴含，把否定移到原子公式前；再标准化变量，使不同量词变量不冲突；然后把量词移到公式左端；接着用 Skolemization 消去存在量词；删除全称量词；最后用分配律转成合取范式，并把每个合取项作为一个子句。

15.3 问：What is unification?

中文作答：

合一是寻找 substitution，使两个逻辑表达式变成相同。最一般合一 mgu 是不加入多余限制的替换。合一时常量只能与相同常量匹配，变量可以替换为项，但替换必须在所有出现位置一致，并且不能让变量替换成包含自身的项。

15.4 问：How does Prolog execute a query?

中文作答：

Prolog 把程序看作 Horn clauses。执行查询时，它从左到右选择当前 goal，从程序顶部开始找 head 能与该 goal 合一的 fact 或 rule。若匹配 rule，就用 rule body 替换该 goal；若某条路径失败，就回溯尝试下一条规则。目标全部消去时查询成功。

16. Prolog 考场语法总表

这一节用于遇到 Prolog 程序题时快速定位语法。

写法	中文含义	注意点
`fact.`	事实	以句点结束
`head :- body.`	规则	`:-` 读作 if；body 中子目标用逗号连接
`?- goal.`	查询	Prolog 尝试证明 goal
`X`，`Name`	变量	大写或下划线开头
`john`，`parent`	常量/谓词名	小写开头
`_`	匿名变量	每次出现都表示不同的无名变量
`,`	AND	从左到右执行
`;`	OR	可产生多个分支
`not(G)` / `\+ G`	失败即否定	不是经典逻辑否定，而是无法证明 G
`[H	T]`	列表头尾
`!`	cut	固定已作出的选择，阻止部分回溯

16.1 写 Prolog 规则的基本模式

一阶逻辑：

$$\forall x \forall y (parent(x, y) \rightarrow predecessor(x, y))$$

Prolog：

```
predecessor(X,Y) :-
    parent(X,Y).
```

一阶逻辑：

$$parent(x, z) \land predecessor(z, y) \rightarrow predecessor(x, y)$$

Prolog：

```
predecessor(X,Y) :-
    parent(X,Z),
    predecessor(Z,Y).
```

写递归规则时要有基例，否则容易无限递归。通常把更具体、能立即成功或失败的目标放在前面，把递归调用放在后面。

16.2 Prolog 和一阶逻辑的区别

Prolog 来源于 Horn clause 逻辑，但不等同于完整一阶逻辑定理证明器：

- Prolog 使用深度优先、从左到右、按程序顺序的搜索。
- Prolog 的否定通常是 `negation as failure`。
- Prolog 采用 closed world assumption，数据库中不能证明的事实按失败处理。
- 规则和目标顺序会影响效率，甚至影响是否终止。

因此，考试写 Prolog 程序时不仅要逻辑正确，还要考虑子目标顺序、递归基例和回溯行为。

17. 易错点

- $P \to Q$ 只有 P 真且 Q 假时为假。
- `valid` 是所有解释真；`satisfiable` 是至少一个解释真。
- 一阶逻辑中函数返回对象，谓词返回真值。
- Skolem function 的参数取决于它所在作用域内的全称变量。
- resolution refutation 要加入目标否定，不是直接加入目标。
- Horn clause 至多一个正文字。
- Prolog 变量大写开头，常量/谓词小写开头。
- Prolog 的 `no` 常表示无法证明，不等于现实中明确为假。
- cut `!` 会影响回溯，可能改变程序答案集合。

08 CLIPS 考场资料

本章主线：CLIPS 是一种用于构建规则专家系统的语言。它把当前已知信息放在 `fact list / working memory` 中，把领域知识写成 `defrule` 规则；推理机不断做 `match -> conflict resolution -> act`，即匹配规则左部、从 agenda 中选规则、执行规则右部。

这章考试重点不是背 “CLIPS 是什么”，而是能读懂和写出 CLIPS 程序：`deftemplate`、`deffacts`、`defrule`、变量、通配符、约束、条件元素、模块、agenda、salience、事实修改、Rete/效率优化。

0. 英文题目触发词速查

English trigger	中文定位	作答时要写出的核心
`CLIPS`	C Language Integrated Production System	基于规则的专家系统语言
`rule-based expert system`	规则专家系统	knowledge base + working memory + inference engine + agenda
`fact`, `fact list`, `working memory`	事实/工作记忆	当前系统已知数据
`deftemplate`	模板	定义事实结构，包含 slot/multislot
`deffacts`	初始事实集合	reset 时加入 fact list
`defrule`	规则	LHS patterns 匹配事实，RHS actions 执行动作
`assert`, `retract`, `modify`	事实操作	增加、删除、修改事实
`fact address`	事实地址	`?f <- pattern` 绑定事实以便 modify/retract
`variable`	变量	`?x` 匹配并绑定单字段
`single-field wildcard`	单字段通配符	`?` 匹配一个字段但不绑定
`multifield wildcard`	多字段通配符	`$?x` 匹配零个或多个字段
`connective constraint`	连接约束	`&`, `|`
`predicate constraint`	谓词约束	`&:(> ?age 18)` 用谓词测试字段
`test conditional element`	测试条件元素	`(test <expression>)` 在 LHS 做额外检查
`not`, `exists`, `forall`, `or`, `and`, `logical`	条件元素	控制模式是否存在、是否所有都满足、逻辑依赖
`agenda`, `activation`	议程/激活	LHS 满足的规则实例进入 agenda 等待执行
`salience`	优先级	`(declare (salience n))` 控制 firing 顺序
`defmodule`, `focus`	模块/焦点	用模块组织阶段，控制规则执行范围
`Rete`, `pattern network`	模式匹配网络	共享匹配结果，减少重复搜索
`pattern order`, `efficiency`	模式顺序/效率	具体、少匹配、稳定的 pattern 放前面；volatile facts 放后面

1. Rule-Based Expert System 结构

规则专家系统的结构可以文字化为：

```
Knowledge Base (Rules)
        |
Inference Engine -- Agenda
        |
Working Memory / Fact List (Facts)
        |
User Interface
```

辅助部分：

- `Explanation Facility`：解释为什么得到某结论。
- `Knowledge Acquisition Facility`：帮助获取和维护知识。

核心三件事：

Component	中文	作用
`fact list` / `working memory`	事实表/工作记忆	存放当前已知事实
`knowledge base`	知识库	存放规则
`inference engine`	推理机	匹配事实与规则，触发规则
`agenda`	议程	存放已激活但尚未执行的规则实例

特点：

- 模块化：规则、事实、模板可分别维护。
- 可解释：规则 firing 可以说明推理过程。

- 类似人类认知：根据当前事实应用知识，逐步推出新事实或动作。

2. CLIPS 基础语法

2.1 记号约定

Notation	含义
`( )`	CLIPS 基本分隔符，所有函数/规则/事实都用前缀括号形式
`< >`	语法说明中的占位符
`[ ]`	可选项
`+`	一个或多个
`*`	零个或多个
`	`

2.2 Primitive Data Types

Type	例子	说明
`integer`	`1`，`+3`，`-1`，`65`	整数
`float`	`1.5`，`0.7`，`9e+1`，`3.5e10`	浮点数
`symbol`	`fire`，`345B`，`activate-sprinkler-system`	符号，不加引号
`string`	`"John Smith"`，`"string"`	字符串，加双引号

分隔符包括空格、tab、换行、`"`、`(`、`)`、`;`、`&`、`|`、`~`、`<`、`>`。

注意：`?` 和 `$?` 不能放在 symbol 开头，因为它们用于变量和通配符。

2.3 Prefix Form 前缀表达式

CLIPS 函数使用前缀形式：

```
(+ 2 3)
(* 4.2 5)
(+ 3 (* 4 5))
```

对应普通数学写法：

```
 2 + 3
4.2 * 5
3 + 4 * 5
```

比较表达式也用前缀：

```
(> (/ (- ?y2 ?y1) (- ?x2 ?x1)) 0)
```

含义：

$$\frac{y_2 - y_1}{x_2 - x_1} > 0$$

3. Facts 与 Templates

3.1 Fact 是什么

fact 是工作记忆中的一条数据。

例子：

```
(person
  (name "John Smith")
  (age 21)
  (eye-color black)
  (hair-color black))
```

这是一个 `person` fact，包含四个 slot。

3.2 deftemplate

`deftemplate` 定义某类事实的结构。

语法：

```
(deftemplate <relation-name>
  [<optional-comment>]
  <slot-definition>*)
```

slot 定义：

```
(slot <slot-name> <attributes>*)
(multislot <slot-name> <attributes>*)
```

例子：

```
(deftemplate person
  "A person template"
  (slot name)
  (slot age)
  (slot eye-color)
  (slot hair-color))
```

`slot` 是单字段槽；`multislot` 可包含零个或多个字段：

```
(deftemplate number-list
  (multislot values))

(number-list (values 7 9 3 4 20))
```

3.3 Ordered Facts 与 Implied deftemplate

ordered fact：

```
(number-list 7 9 3 4 20)
```

可以理解为隐式模板：

```
(deftemplate number-list
  (multislot values))

(number-list (values 7 9 3 4 20))
```

区别：显式 `deftemplate` 适合结构清晰、slot 有名字和约束的事实；ordered fact 更短，但可读性和约束能力较弱。

4. deftemplate 属性

4.1 type

语法：

```
(type <type-specification>)
```

常见类型：

```
SYMBOL
STRING
LEXEME     ; SYMBOL or STRING
INTEGER
FLOAT
NUMBER     ; INTEGER or FLOAT
?VARIABLE  ; default
```

例子：

```
(deftemplate person
  (multislot name (type SYMBOL))
  (slot age (type INTEGER)))
```

4.2 allowed values

限制 slot 可取值。

```
(slot gender
  (allowed-values male female))
```

相关属性：

```
allowed-values
allowed-symbols
allowed-strings
allowed-lexemes
allowed-integers
allowed-floats
allowed-numbers
```

4.3 range

限制数值范围：

```
(slot age
  (type INTEGER)
  (range 0 ?VARIABLE))
```

含义：age 是整数，最小为 0，上界不固定。

4.4 cardinality

限制 `multislot` 的字段个数：

```
(multislot name
  (type SYMBOL)
  (cardinality 1 6))
```

含义：name 至少 1 个字段，最多 6 个字段。

4.5 default

默认值：

```
(slot gender
  (allowed-values male female)
  (default female))
```

默认规范：

default specification	含义
`?DERIVE`	默认，由类型和 allowed constraints 推导
`?NONE`	没有默认值，assert 时必须给出
单个表达式	单字段 slot 默认值
多个表达式	multifield slot 默认值

例子：

```
(deftemplate default-derive-example
  (slot a)
  (slot b (type INTEGER))
  (slot c (allowed-values red green blue))
  (multislot d)
  (multislot e (cardinality 2 2)
               (type FLOAT)
               (range 3.5 10.0)))

(assert (default-derive-example))
```

解释：未提供 slot 值时，CLIPS 根据默认规则填充。考试遇到默认值题，要先看 slot 是否有显式 `(default ...)`，再看类型/allowed/range/cardinality。

5. Fact Operations 事实操作

5.1 assert

添加事实：

```
(assert <fact>+)
```

例子：

```
(assert (emergency (type fire)))
```

5.2 facts

显示事实：

```
(facts [<start> [<end> [<maximum>]]])
```

5.3 retract

删除事实：

```
(retract <fact-index>+)
```

5.4 modify

修改事实：

```
(modify <fact-index-or-address>
  (<slot-name> <slot-value>)+)
```

5.5 Fact Address

为了修改或删除被匹配到的事实，需要绑定事实地址：

```
 ?f <- (person (name ?name) (address ?address))
=>
(modify ?f (address "237 Cherry Lane"))
```

`?f <- pattern` 的意思是：不仅匹配这个 fact，还把该 fact 的地址绑定到 `?f`。RHS 中可用 `modify ?f` 或 `retract ?f` 操作它。

6. deffacts

`deffacts` 定义一组初始事实。

语法：

```
(deffacts <deffacts-name>
  [<optional-comment>]
  <facts>*)
```

例子：

```
(deffacts people
  (person (name "John Smith") (age 21)
          (eye-color black) (hair-color black))
  (person (name "Susan Kelly") (age 42)
          (eye-color blue) (hair-color black)))
```

通常执行 `(reset)` 后，`deffacts` 中的 facts 被放入 fact list。

7. defrule 与 run

7.1 defrule 结构

语法：

```
(defrule <rule-name>
  [<optional-comment>]
  <patterns>*       ; LHS
  =>
  <actions>*)       ; RHS
```

LHS `left-hand side`：条件部分，写 pattern，用来匹配事实。

RHS `right-hand side`：动作部分，执行 assert/retract/modify/printout/bind 等。

例子：

```
 (deftemplate emergency (slot type))
(deftemplate response (slot action))

(defrule fire-emergency
  (emergency (type fire))
  =>
  (assert (response
    (action activate-sprinkler-system))))
```

含义：如果 fact list 中有 `(emergency (type fire))`，则断言一个 response。

7.2 run

运行规则：

```
(run [<limit>])
```

`limit` 限制最多 firing 多少条规则。

7.3 printout 与 clear

输出：

```
(printout t "Activate the sprinkler system" crlf)
```

清空环境：

```
(clear)
```

7.4 Manipulating Constructs

查看定义：

```
(list-defrules)
(list-deftemplates)
(list-deffacts)
(ppdefrule <defrule-name>)
(ppdeftemplate <deftemplate-name>)
(ppdeffacts <deffacts-name>)
```

删除定义：

```
(undefrule <defrule-name>)
(undeftemplate <deftemplate-name>)
(undeffacts <deffacts-name>)
```

8. Variables, Wildcards, Constraints

8.1 Variable 变量

变量以 `?` 开头，匹配一个字段并绑定值。

```
(deftemplate person
  (slot name)
  (slot eyes)
  (slot hair))

(defrule find-blue-eyes
  (person (name ?name) (eyes blue))
  =>
  (printout t ?name " has blue eyes." crlf))
```

事实：

```
(person (name Jane) (eyes blue) (hair red))
(person (name Joe) (eyes green) (hair brown))
(person (name Jack) (eyes blue) (hair black))
```

匹配结果：Jane 和 Jack 满足 `(eyes blue)`，`?name` 分别绑定为 `Jane`、`Jack`。

变量一致性：

```
(defrule find-eyes
  (find (eyes ?eyes))
  (person (name ?name) (eyes ?eyes))
  =>
  (printout t ?name " has " ?eyes " eyes." crlf))
```

若有：

```
(assert (find (eyes green)))
```

则第二个 pattern 只匹配 eyes 为 green 的 person。因为同一个变量 `?eyes` 在两个 pattern 中必须取同一个值。

8.2 Single-Field Wildcard

`?` 匹配一个字段，但不绑定变量。

用法：只关心某个位置存在一个字段，但不关心它的值。

若 `person` 的 name 是 multislot：

```
(person (name John Q. Smith))
```

pattern：

```
(person (name ? ? Smith))
```

含义：匹配三个字段的名字，最后一个字段是 Smith，中间两个字段任意。

8.3 Unspecified Slot

如果 template 中有 slot 但 pattern 未写该 slot，等价于该 slot 使用 wildcard。

```
(deftemplate person
  (multislot name)
  (slot identity-card-number))

(person (name John Q. Smith))
```

等价于：

```
(person (name John Q. Smith)
        (identity-card-number ?))
```

含义：没有约束 identity-card-number。

8.4 Multi-Field Wildcard

`$?` 或 `$?name` 匹配零个或多个字段。

例子：

```
(deftemplate person
  (multislot name)
  (multislot children))

(defrule find-child
  (find-child ?child)
  (person (name $?name)
          (children $?before ?child $?after))
  =>
  (printout t ?name " has child " ?child crlf)
  (printout t "Other children are " ?before " " ?after crlf))
```

事实：

```
(person (name John Smith) (children Jane Joe Paul))
(person (name Jack R. Lenny) (children Joe Joe Joe))
```

若查找 `Joe`：

- `?child` 绑定为 `Joe`。
- `$?before` 绑定 Joe 前面的所有字段。
- `$?after` 绑定 Joe 后面的所有字段。
- 如果同一列表里有多个 Joe，会产生多个匹配。

课件输出说明：

```
Jack R. Lenny has child Joe
Other children are () (Joe Joe)

Jack R. Lenny has child Joe
Other children are (Joe) (Joe)

Jack R. Lenny has child Joe
Other children are (Joe Joe) ()

John Smith has child Joe
Other children are (Jane) (Paul)
```

8.5 Connective Constraint

字段约束符：

Symbol	含义	例子
`~`	not	`~black` 表示不是 black
`|`	|	or
`&`	and / 组合约束	`?x&~black` 表示绑定 ?x 且 ?x 不是 black

例子：

```
(defrule person-without-black-hair
  (person (name ?name) (hair ~black))
  =>
  (printout t ?name " does not have black hair" crlf))
```

```
(defrule black-or-brown-hair
  (person (name ?name) (hair brown|black))
  =>
  (printout t ?name " has dark hair" crlf))
```

组合约束例子：

```
(defrule complex-eye-hair-match
  (person (name ?name1)
          (eyes ?eyes1&blue|green)
          (hair ?hair1&~black))
  (person (name ?name2&~?name1)
          (eyes ?eyes2&~?eyes1)
          (hair ?hair2&black|?hair1))
  =>
  (printout t ?name1 " has " ?eyes1 " eyes and "
              ?hair1 " hair" crlf)
  (printout t ?name2 " has " ?eyes2 " eyes and "
              ?hair2 " hair" crlf))
```

含义：

- 第一个人眼睛是 blue 或 green，头发不是 black。
- 第二个人不是同一个人，眼睛颜色不同于第一个人。
- 第二个人头发是 black 或与第一个人头发相同。

8.6 Predicate Function 与 test

谓词函数返回 TRUE/FALSE：

```
(and (> 4 3) (> 4 5))    ; FALSE
(or  (> 4 3) (> 4 5))    ; TRUE
(not (integerp 3))       ; FALSE
```

`test` 在 LHS 中执行布尔测试：

```
(test (and (integerp ?x) (>= ?x 1)))
```

字段内谓词约束：

```
?age&:(> ?age 18)
```

含义：先绑定 `?age`，再检查 `?age > 18`。

8.7 Return Value Constraint

`=` 可使用返回值作为约束。

例子：

```
?x&=(mod 13 4)
```

如果 `(mod 13 4)` 返回 1，则要求该字段等于 1。

9. Conditional Elements 条件元素

9.1 and

多个普通 pattern 默认就是 AND：

```
(defrule example
  (a)
  (b)
  =>
  ...)
```

等价于同时要求 `(a)` 和 `(b)` 存在。

9.2 or

`or` 表示多个条件分支任一满足即可。

例子：洪水或水喷淋系统开启都要关闭电力。

```
(defrule shut-off-electricity-1
  ?power <- (electrical-power (status on))
  (emergency (type flood))
  =>
  (modify ?power (status off))
  (printout t "Shut off the electricity" crlf))
```

```
(defrule shut-off-electricity-2
  ?power <- (electrical-power (status on))
  (extinguisher-system (type water-sprinkler)
                       (status on))
  =>
  (modify ?power (status off))
  (printout t "Shut off the electricity" crlf))
```

这两个规则也可用 `or` 合成一个条件结构，但考试写成两个规则更容易读。

9.3 not

`not` 表示不存在匹配事实。

常见错误：把变量只放在 `not` 内部，外部没有绑定，可能导致语义不明确。

正确模式通常先有一个外部事实绑定变量，再用 `not` 检查不存在某事实：

```
(defrule no-birthdays-on-specific-date
  (check-for-no-birthdays (date ?date))
  (not (person (birthday ?date)))
  =>
  (printout t "No birthdays on " ?date crlf))
```

含义：对某个待检查日期 `?date`，如果没有 person 的 birthday 是该日期，则输出无生日。

9.4 exists

`exists` 表示至少存在一个匹配事实。与普通 pattern 的区别是：即使有多个匹配，规则也只因“存在性”激活一次。

普通规则：

```
(defrule operator-alert-for-emergency
  (emergency)
  =>
  (printout t "Emergency: Operator Alert." crlf)
  (assert (operator-alert)))
```

如果有两个 emergency fact，会激活两次。

使用 `exists`：

```
(defrule operator-alert-for-emergency
  (exists (emergency))
  =>
  (printout t "Emergency: Operator Alert" crlf)
  (assert (operator-alert)))
```

含义：只要至少有一个 emergency，就产生一次 operator alert。

`exists` 等价于：

```
(and (not (not (and <pattern>))))
```

9.5 forall

`forall` 表示：对每一个满足 first CE 的事实，都必须满足 remaining CEs。

语法：

```
(forall <first-CE> <remaining-CEs>+)
```

例子：

```
(defrule all-fires-being-handled
  (forall
    (emergency (type fire) (location ?where))
    (fire-squad (location ?where))
    (evacuated (building ?where)))
  =>
  (printout t "All buildings that are on fire have been" crlf
              "evacuated and have firefighters on location." crlf))
```

含义：每个发生 fire emergency 的地点，都必须有 fire-squad 且对应 building 已 evacuated。

`forall` 等价于：

```
(not (and <first-CE>
          (not (and <remaining-CEs>+))))
```

中文理解：不存在这样一个 first-CE，它缺少后续条件。

9.6 logical

`logical` 表示 RHS 断言的事实依赖于 LHS 中 logical 包裹的事实。

例子：

```
(defrule use-oxygen-masks
  (logical
    (noxious-fumes-present)
    (gas-extinguishers-in-use))
  (emergency (type fire))
  =>
  (assert (use-oxygen-masks)))
```

含义：`use-oxygen-masks` 是基于 `noxious-fumes-present` 和 `gas-extinguishers-in-use` 逻辑支持产生的。如果这些支持事实消失，CLIPS 可以自动撤销依赖于它们的逻辑事实。

相关命令：

```
(dependents <fact-index-or-address>)
(dependencies <fact-index-or-address>)
```

用于查看依赖和被依赖关系。

10. Rule-Based Decision Tree Program

课件的动物决策树：

```
Is the animal warm blooded?
  yes -> Does the animal purr?
           yes -> cat
           no  -> dog
  no  -> snake
```

10.1 节点模板

```
(deftemplate node
  (slot name)
  (slot type)
  (slot question)
  (slot yes-node)
  (slot no-node)
  (slot answer))
```

slot 含义：

slot	含义
`name`	节点名，如 root、node1
`type`	`decision` 或 `answer`
`question`	决策节点要问的问题
`yes-node`	回答 yes 时去的节点
`no-node`	回答 no 时去的节点
`answer`	叶子节点的答案，如 cat/dog/snake

初始化：

```
(defrule initialize
  (not (node (name root)))
  =>
  (load-facts "animal.dat")
  (assert (current-node root)))
```

含义：如果还没有 root 节点，载入动物决策树数据，并把当前节点设为 root。

10.2 决策节点提问

```
(defrule ask-decision-node-question
  ?node <- (current-node ?name)
  (node (name ?name)
        (type decision)
        (question ?question))
  (not (answer ?))
  =>
  (printout t ?question " (yes or no) ")
  (assert (answer (read))))
```

解释：

- `current-node` 指向当前节点。
- 找到同名 `node`，且 type 是 decision。
- 如果当前还没有 answer fact，就打印问题并读取用户输入。

10.3 处理非法答案

```
(defrule bad-answer
  ?answer <- (answer ~yes&~no)
  =>
  (retract ?answer))
```

含义：如果输入既不是 yes 也不是 no，就撤销这个 answer，让系统重新提问。

10.4 沿 yes/no 分支移动

yes 分支：

```
(defrule proceed-to-yes-branch
  ?node <- (current-node ?name)
  (node (name ?name)
        (type decision)
        (yes-node ?yes-branch))
  ?answer <- (answer yes)
  =>
  (retract ?node ?answer)
  (assert (current-node ?yes-branch)))
```

no 分支：

```
(defrule proceed-to-no-branch
  ?node <- (current-node ?name)
  (node (name ?name)
        (type decision)
        (no-node ?no-branch))
  ?answer <- (answer no)
  =>
  (retract ?node ?answer)
  (assert (current-node ?no-branch)))
```

核心：移动节点时要删除旧 `current-node` 和旧 `answer`，再 assert 新的 `current-node`。

10.5 到达答案节点

```
(defrule ask-if-answer-node-is-correct
  ?node <- (current-node ?name)
  (node (name ?name) (type answer) (answer ?value))
  (not (answer ?))
  =>
  (printout t "I guess it is a " ?value crlf)
  (printout t "Am I correct? (yes or no) ")
  (assert (answer (read))))
```

如果猜对：

```
(defrule answer-node-guess-is-correct
  ?node <- (current-node ?name)
  (node (name ?name) (type answer))
  ?answer <- (answer yes)
  =>
  (assert (ask-try-again))
  (retract ?node ?answer))
```

如果猜错：

```
(defrule answer-node-guess-is-incorrect
  ?node <- (current-node ?name)
  (node (name ?name) (type answer))
  ?answer <- (answer no)
  =>
  (assert (replace-answer-node ?name))
  (retract ?answer ?node))
```

10.6 学习新节点

当猜错时，系统询问正确动物和区别问题，然后把原 answer node 替换成 decision node，并创建两个新的 answer nodes。

关键动作：

```
(bind ?newnode1 (gensym*))
(bind ?newnode2 (gensym*))

(modify ?data
  (type decision)
  (question ?question)
  (yes-node ?newnode1)
  (no-node ?newnode2))

(assert (node (name ?newnode1)
              (type answer)
              (answer ?new-animal)))

(assert (node (name ?newnode2)
              (type answer)
              (answer ?value)))

(assert (ask-try-again))
```

解释：

- `gensym*` 生成新的唯一符号作为节点名。
- 原来的错误答案节点被改成决策节点。
- 新节点一个保存用户提供的新动物，一个保存原来的旧答案。
- 这样系统完成增量学习。

11. 控制执行顺序：salience, phase, defmodule

11.1 salience

语法：

```
(declare (salience <integer>))
```

范围：

```
-10000 到 10000，默认 0
```

salience 越高，activation 优先级越高。

例子：

```
(defrule situation-emergency
  (declare (salience 10))
  (situation emergency)
  =>
  (assert (action emergency)))
```

11.2 Misuse of salience

课件强调：不要把 salience 当成主要控制逻辑的工具。

不好的写法：

```
(defrule situation-emergency
  (declare (salience 10))
  (situation emergency)
  =>
  (assert (action emergency)))

(defrule situation-important
  (declare (salience 5))
  (situation important)
  =>
  (assert (action important)))

(defrule situation-normal
  (declare (salience 0))
  (situation normal)
  =>
  (assert (action normal)))
```

问题：如果同时存在多个 situation，只靠 salience 可能让规则顺序正确，但事实逻辑没有互斥表达。

更好的写法是在条件中明确排除更高优先级 situation：

```
(defrule situation-important
  (situation important)
  (not (situation emergency))
  =>
  (assert (action important)))
```

```
(defrule situation-normal
  (situation normal)
  (not (situation emergency))
  (not (situation important))
  =>
  (assert (action normal)))
```

含义：逻辑条件本身表达优先关系，而不是只依赖 rule priority。

11.3 Control Pattern

把系统分成阶段：

```
DETECTION -> ISOLATION -> RECOVERY -> DETECTION ...
```

阶段控制事实：

```
(phase detection)
```

规则按 phase 触发：

```
(defrule detection-rule
  (phase detection)
  <patterns>*
  =>
  <actions>*)
```

阶段转换规则用低 salience，确保当前阶段普通规则先执行完：

```
(defrule detection-to-isolation
  (declare (salience -10))
  ?phase <- (phase detection)
  =>
  (retract ?phase)
  (assert (phase isolation)))
```

同理：

```
isolation -> recovery
recovery -> detection
```

中文解释：普通检测规则默认 salience 0，阶段转换规则 salience -10，因此本阶段能触发的规则先执行，最后才切换 phase。

11.4 defmodule

模块语法：

```
(defmodule <module-name> [<comment>])
```

默认模块：

```
MAIN
```

常用命令：

```
(get-current-module)
(set-current-module <module-name>)
(focus <module-name>+)
(list-focus-stack)
(clear-focus-stack)
(pop-focus)
(get-focus)
```

模块化阶段：

```
(defmodule DETECTION)
(defmodule ISOLATION)
(defmodule RECOVERY)

(deffacts MAIN::control-information
  (phase-sequence DETECTION ISOLATION RECOVERY))
```

改变焦点：

```
(defrule MAIN::change-phase
  ?list <- (phase-sequence ?next-phase $?other-phases)
  =>
  (focus ?next-phase)
  (retract ?list)
  (assert (phase-sequence ?other-phases ?next-phase)))
```

解释：每次取出 phase-sequence 的第一个模块作为 `?next-phase`，把 focus 交给该模块，再把它放到序列末尾，实现循环调度。

11.5 import/export

模块间共享 deftemplate：

```
(defmodule DETECTION
  (export deftemplate fault))

(deftemplate DETECTION::fault
  (slot component))
```

```
(defmodule RECOVERY
  (import DETECTION deftemplate fault))
```

含义：DETECTION 导出 `fault` 模板，RECOVERY 可以导入并使用它。

11.6 initial-fact

特殊规则：

```
(defrule <rule-name>
  =>
  <actions>*)
```

没有 LHS 的规则可由 `initial-fact` 触发。若跨模块使用，需要导入/导出 `initial-fact` deftemplate：

```
(defmodule MAIN
  (export deftemplate initial-fact))

(defmodule <module-name>
  (import MAIN deftemplate initial-fact))
```

12. Procedural Functions

12.1 if

语法：

```
(if <predicate-expression>
 then
   <expression>+
 [else
   <expression>+])
```

12.2 while

语法：

```
(while <predicate-expression>
  [do]
  <expression>+)
```

例子：询问是否继续。

```
(defrule continue-check
  ?phase <- (phase check-continue)
  =>
  (retract ?phase)
  (printout t "Continue? ")
  (bind ?answer (read))
  (if (eq ?answer yes)
      then (assert (phase detection))
      else (halt)))
```

`bind` 用于给变量赋值：

```
(bind ?ans (read))
```

13. Monitoring Problem

这个例子是 CLIPS 程序综合题：用模块、事实、规则、阈值、趋势记录和告警规则实现设备监控。

13.1 问题结构

系统有传感器和设备：

```
S1, S2 -> D1
S3     -> D2
S4     -> D3
S5, S6 -> D4
```

每个 sensor 有四条阈值线：

Sensor	Low Red Line	Low Guard Line	High Guard Line	High Red Line
S1	60	70	120	130
S2	20	40	160	180
S3	60	70	120	130
S4	60	70	120	130
S5	65	70	120	125
S6	110	115	125	130

规则含义：

- 若读数 $\leq$ low red line，或读数 $\geq$ high red line，则 shut down device。
- 若读数在 red line 和 guard line 之间，则 issue warning；若持续给定 cycles，则 shut down device。
- 若读数在 low guard line 与 high guard line 之间，则 normal。

13.2 MAIN 模块基础事实

```
(defmodule MAIN (export ?ALL))

(deftemplate MAIN::device
  (slot name (type SYMBOL))
  (slot status (allowed-values on off)))

(deffacts MAIN::device-information
  (device (name D1) (status on))
  (device (name D2) (status on))
  (device (name D3) (status on))
  (device (name D4) (status on)))
```

sensor 模板：

```
(deftemplate MAIN::sensor
  (slot name (type SYMBOL))
  (slot device (type SYMBOL))
  (slot raw-value
        (type SYMBOL NUMBER)
        (allowed-symbols none)
        (default none))
  (slot state
        (allowed-values low-red-line low-guard-line
                        normal high-guard-line high-red-line)
        (default normal))
  (slot low-red-line)
  (slot low-guard-line)
  (slot high-guard-line)
  (slot high-red-line))
```

解释：

- `raw-value` 可以是数字，也可以是 symbol `none`。
- `state` 表示当前读数区间。
- 每个 sensor 保存自己的四条阈值。

13.3 Cycle 控制

初始：

```
(deffacts MAIN::cycle-start
  (data-source facts)
  (cycle 0))
```

每轮开始：

```
(defrule MAIN::Begin-Next-Cycle
  ?f <- (cycle ?current-cycle)
  =>
  (retract ?f)
  (assert (cycle (+ ?current-cycle 1)))
  (focus INPUT TRENDS WARNINGS))
```

含义：周期数加一，然后依次把焦点给 INPUT、TRENDS、WARNINGS 模块。

13.4 INPUT 模块读取数据

输入数据事实：

```
(defmodule INPUT (import MAIN ?ALL))

(deftemplate INPUT::fact-data-for-sensor
  (slot name)
  (multislot data))
```

读取规则：

```
(defrule INPUT::Read-Sensor-Values-From-Facts
  (data-source facts)
  ?s <- (sensor (name ?name) (raw-value none))
  ?f <- (fact-data-for-sensor
          (name ?name)
          (data ?raw-value $?rest))
  =>
  (modify ?s (raw-value ?raw-value))
  (modify ?f (data ?rest)))
```

解释：

- 找一个 raw-value 仍为 none 的 sensor。
- 在对应 `fact-data-for-sensor` 中取出第一个 data 作为 `?raw-value`。
- 把 sensor 的 raw-value 改成该值。
- 把 data 列表剩余部分 `$?rest` 写回。

无数据时停止：

```
(defrule INPUT::No-Fact-Data-Values-Left
  (data-source facts)
  (sensor (name ?name) (raw-value none))
  (fact-data-for-sensor (name ?name) (data))
  =>
  (printout t "No fact data for sensor " ?name crlf)
  (printout t "Halting monitoring system" crlf)
  (halt))
```

13.5 TRENDS 模块判断状态

normal：

```
(defmodule TRENDS (import MAIN ?ALL))

(defrule TRENDS::Normal-State
  ?s <- (sensor (raw-value ?raw-value&~none)
                (low-guard-line ?lgl)
                (high-guard-line ?hgl))
  (test (and (> ?raw-value ?lgl)
             (< ?raw-value ?hgl)))
  =>
  (modify ?s (state normal) (raw-value none)))
```

high guard：

```
(defrule TRENDS::High-Guard-Line-State
  ?s <- (sensor (raw-value ?raw-value&~none)
                (high-guard-line ?hgl)
                (high-red-line ?hrl))
  (test (and (>= ?raw-value ?hgl)
             (< ?raw-value ?hrl)))
  =>
  (modify ?s (state high-guard-line) (raw-value none)))
```

其他状态同理：

- `High-Red-Line-State`：$raw \geq high\ red\ line$。
- `Low-Guard-Line-State`：$low\ red\ line < raw \leq low\ guard\ line$。

- `Low-Red-Line-State`：$raw \leq low\ red\ line$。

注意：每次判断完状态后把 `raw-value` 改回 `none`，为下一 cycle 读取新值做准备。

13.6 sensor-trend 趋势记录

模板：

```
(deftemplate MAIN::sensor-trend
  (slot name)
  (slot state (default normal))
  (slot start (default 0))
  (slot end (default 0))
  (slot shutdown-duration (default 3)))
```

若状态未变化：

```
(defrule TRENDS::State-Has-Not-Changed
  (cycle ?time)
  ?trend <- (sensor-trend (name ?sensor)
                          (state ?state)
                          (end ?end-cycle&~?time))
  (sensor (name ?sensor)
          (state ?state)
          (raw-value none))
  =>
  (modify ?trend (end ?time)))
```

若状态变化：

```
(defrule TRENDS::State-Has-Changed
  (cycle ?time)
  ?trend <- (sensor-trend (name ?sensor)
                          (state ?state)
                          (end ?end-cycle&~?time))
  (sensor (name ?sensor)
          (state ?new-state&~?state)
          (raw-value none))
  =>
  (modify ?trend
          (start ?time)
          (end ?time)
          (state ?new-state)))
```

解释：

- `start` 是当前状态开始的 cycle。
- `end` 是当前状态持续到的 cycle。
- 如果新 state 与旧 state 相同，只更新 end。
- 如果新 state 不同，则 start/end 都设为当前 cycle，并更新 state。

13.7 WARNINGS / shutdown 思路

根据 sensor state 和持续时间决定 warning 或 shutdown：

- red-line 状态立即关闭对应 device。
- guard-line 状态先 warning；若持续时间达到 `shutdown-duration`，关闭设备。
- normal 状态通常不告警。

核心判断：

```
duration = end - start + 1
if duration >= shutdown-duration:
    shut down device
```

关闭设备通常通过：

```
?d <- (device (name ?device) (status on))
=>
(modify ?d (status off))
```

WARNINGS 模块三条规则的完整读法

`WARNINGS` 模块先通过 `(defmodule WARNINGS (import MAIN ?ALL))` 读取 `MAIN` 模块中的 facts。三条规则都依赖 `cycle` 和 `sensor-trend`，区别在于红线立即关机，警戒线按持续时间决定 warning 或 shutdown。

1. `Shutdown-In-Red-Region`

触发条件：

```
(cycle ?time)
(sensor-trend
  (name ?sensor)
  (state ?state&high-red-line|low-red-line))
(sensor (name ?sensor) (device ?device))
?on <- (device (name ?device) (status on))
```

含义：

- `?state&high-red-line|low-red-line` 表示 state 必须绑定为 `high-red-line` 或 `low-red-line`。
- `(sensor ... (device ?device))` 把传感器映射到它监控的设备。
- `?on <- (...)` 绑定 fact address，右侧才能 `modify` 这个设备 fact。

动作：

```
(printout t "Cycle " ?time " - ")
(printout t "Sensor " ?sensor " in " ?state crlf)
(printout t "  Shutting down device " ?device crlf)
(modify ?on (status off))
```

考场解释：一旦传感器进入红线区，系统不等待持续时间，直接把对应设备从 `status on` 改为 `status off`。

2. `Shutdown-In-Guard-Region`

触发条件：

```
(cycle ?time)
(sensor-trend
  (name ?sensor)
  (state ?state&high-guard-line|low-guard-line)
  (shutdown-duration ?length)
  (start ?start)
  (end ?end))
(test (>= (+ (- ?end ?start) 1) ?length))
(sensor (name ?sensor) (device ?device))
?on <- (device (name ?device) (status on))
```

含义：

- `high-guard-line | low-guard-line` 是警戒区，不是红线区。
- `start` 和 `end` 记录该状态从哪个 cycle 持续到哪个 cycle。
- `(+ (- ?end ?start) 1)` 是持续 cycle 数，例如 start=3, end=5 时持续 3 个 cycle。
- 只有持续时间 `>= shutdown-duration` 时才关闭设备。

动作：打印 cycle、sensor、state、持续了多少 cycles，然后 `(modify ?on (status off))`。

考场解释：警戒线不是立即关机；系统先检查该异常状态是否连续存在足够久，达到阈值才 shutdown。

3. `Sensor-In-Guard-Region`

触发条件：

```
(cycle ?time)
(sensor-trend
  (name ?sensor)
  (state ?state&high-guard-line|low-guard-line)
  (shutdown-duration ?length)
  (start ?start)
  (end ?end))
(test (< (+ (- ?end ?start) 1) ?length))
```

动作：

```
(printout t "Cycle " ?time " - ")
(printout t "Sensor " ?sensor " in " ?state crlf)
```

考场解释：如果传感器在警戒区但持续时间还没达到 shutdown 阈值，只输出 warning 信息，不修改设备状态。

14. Inference Engine 与 Agenda

推理机循环是 `recognize-act cycle`。

```
while not done:
    Conflict Resolution:
        if there are activations,
        select one with highest priority;
        else done.
    Act:
        perform RHS actions of selected activation.
        remove fired activation from agenda.
    Match:
        update agenda by checking rule LHSs.
        add activations whose LHSs are satisfied.
        remove activations whose LHSs are no longer satisfied.
    Check for Halt:
        if halt action or break command, done.
end
accept new user command
```

概念关系：

- rule LHS 匹配 facts 后产生 `activation`。
- activations 存入 `agenda`。
- conflict resolution 选择一个 activation firing。
- RHS 的 assert/retract/modify 会改变 fact list，从而重新影响 agenda。

15. Pattern Network / Rete 思想

若每次都让所有 rules 搜索所有 facts，会非常低效。Rete 思想是把 pattern matching 编成网络，共享中间匹配结果。

网络可分为：

Network	作用
match/data pattern network	检查单个 pattern 对单个 fact 的约束
joint network	检查多个 pattern 之间的变量一致性

例子规则：

```
(defrule sharing-1
  (match (a ?x) (b red))
  (data (x ~green) (y ?x))
  (data (x ?x) (y ?x))
  =>
  (printout t "Matching sharing-1 rule." crlf))
```

匹配逻辑可拆为：

- 第一 pattern 要求 `b` slot 等于 red。
- 第二 pattern 要求 `x` slot 不等于 green。
- 第三 pattern 要求 `y` slot 等于 `x` slot。
- joint network 还检查不同 pattern 之间变量绑定是否一致。

中文解释：Rete 不只是逐条规则扫描事实，而是保存 pattern 的部分匹配，并在事实变化时增量更新。

16. Efficiency 与 Pattern Order

16.1 基本原则

效率原则：

1. Most specific patterns go first.
2. Patterns matching volatile facts go last.
3. Patterns matching the fewest facts go first.
4. Limit the number of multifield wildcards and variables.
5. Test conditional elements should be placed as close to the top of the rule as possible, but only after variables have been bound.
6. Use built-in pattern constraints when possible instead of separate test CEs.

16.2 Pattern order 例子

事实：

```
(deffacts information
  (find-match a c e g)
  (item a)
  (item b)
  (item c)
  (item d)
  (item e)
  (item f)
  (item g))
```

好顺序：

```
(defrule good-match
  (find-match ?x ?y ?z ?w)
  (item ?x)
  (item ?y)
  (item ?z)
  (item ?w)
  =>
  (assert (found-match ?x ?y ?z ?w)))
```

解释：`find-match` 只有一个事实，先绑定 `?x ?y ?z ?w`，后面的 item pattern 都被限制住。

坏顺序：

```
(defrule bad-match
  (item ?x)
  (item ?y)
  (item ?z)
  (item ?w)
  (find-match ?x ?y ?z ?w)
  =>
  (assert (found-match ?x ?y ?z ?w)))
```

解释：前四个 item pattern 各可匹配 7 个 item，产生大量 partial matches，最后才用 find-match 筛掉，效率很差。

16.3 Multifield wildcards 的代价

pattern：

```
(list (items $?a $?b $?c))
```

fact：

```
(list (items x 5 y 9))
```

三个 multifield wildcard 要把 4 个字段切分给 `$?a`、`$?b`、`$?c`。可能切分有 15 种。字段越多、multifield wildcard 越多，组合数增长越快。

结论：少用多个 `$?`；能用固定字段、slot 约束或更具体 pattern 时就不要写过宽的 multifield wildcard。

16.4 test 放置位置

低效写法：先匹配三个 point，再一次性 test 它们是否不同。

```
(defrule three-distinct-points
  ?point-1 <- (point (x ?x1) (y ?y1))
  ?point-2 <- (point (x ?x2) (y ?y2))
  ?point-3 <- (point (x ?x3) (y ?y3))
  (test (and (neq ?point-1 ?point-2)
             (neq ?point-2 ?point-3)
             (neq ?point-1 ?point-3)))
  =>
  ...)
```

更好：变量绑定后尽早 test，减少后续 partial matches。

```
(defrule three-distinct-points
  ?point-1 <- (point (x ?x1) (y ?y1))
  ?point-2 <- (point (x ?x2) (y ?y2))
  (test (neq ?point-1 ?point-2))
  ?point-3 <- (point (x ?x3) (y ?y3))
  (test (and (neq ?point-2 ?point-3)
             (neq ?point-1 ?point-3)))
  =>
  ...)
```

16.5 Built-in pattern constraints

低效或不够直接：

```
(defrule primary-color
  (color ?x&:(or (eq ?x red)
                 (eq ?x green)
                 (eq ?x blue)))
  =>
  (assert (primary-color ?x)))
```

更简洁：

```
(defrule primary-color
  (color ?x&red|green|blue)
  =>
  (assert (primary-color ?x)))
```

含义相同：只匹配 red、green、blue 三种颜色。内置 pattern constraint 通常更适合放在 pattern 中。

17. Loading and Saving Facts

载入 facts：

```
(load-facts <file-name>)
```

保存 facts：

```
(save-facts <file-name>
            [<save-scope> <deftemplate-names>*])
```

`save-scope`：

```
visible
local
```

facts 文件格式例子：

```
(data 34)
(data 89)
```

命令：

```
(load-facts "facts.dat")
```

18. Prolog 与 CLIPS 对比

维度	Prolog	CLIPS
核心范式	逻辑程序设计	产生式规则系统
知识形式	Horn clauses, facts, rules	facts, deftemplates, defrules
推理方式	目标驱动，深度优先，回溯	数据驱动，事实变化激活规则
查询/控制	用户提出 goal，Prolog 证明	facts 满足 LHS，rule activation 进入 agenda
变量	大写变量，如 `X`	`?x` 单字段变量，`$?x` 多字段变量
否定	negation as failure	`not` conditional element
程序关注点	goal 顺序、递归、回溯、cut	pattern matching、agenda、salience、modules、fact operations

一句话区分：Prolog 更像“给出目标，让系统证明”；CLIPS 更像“事实不断变化，满足条件的规则自动触发”。

19. 常见答题模板

19.1 问：Describe the structure of a rule-based expert system

中文作答：

规则专家系统由 working memory、knowledge base、inference engine 和 agenda 等部分组成。working memory 保存当前 facts，knowledge base 保存 rules。推理机将规则 LHS 与事实匹配，产生 activations 放入 agenda；再通过 conflict resolution 选择一个 activation 执行 RHS actions，可能 assert、retract 或 modify facts，从而继续触发新规则。

19.2 问：Explain deftemplate, deffacts, defrule

中文作答：

`deftemplate` 定义事实结构，包括 slot 和 multislot 以及类型、范围、默认值等约束。`deffacts` 定义 reset 后加入 fact list 的初始事实。`defrule` 定义规则，左部 LHS 是匹配 facts 的 patterns，右部 RHS 是匹配成功后执行的 actions，例如 assert、retract、modify 或 printout。

19.3 问：Explain variables and wildcards in CLIPS

中文作答：

CLIPS 中 `?x` 是单字段变量，匹配一个字段并绑定值；同一变量多次出现时必须取同一值。`?` 是单字段通配符，匹配一个字段但不绑定。`$?x` 是多字段变量，可匹配零个或多个字段，常用于 multislot。多字段通配符很灵活，但会产生很多匹配组合，可能降低效率。

19.4 问：Explain recognize-act cycle

中文作答：

CLIPS 推理机循环执行 recognize-act cycle。首先进行 match，检查哪些规则 LHS 被当前 facts 满足，并把对应 activations 放入 agenda；然后进行 conflict resolution，从 agenda 中选择优先级最高的 activation；接着执行 RHS actions，修改 fact list；最后更新 agenda。如果执行了 halt 或没有 activations，循环结束。

19.5 问：How to improve CLIPS rule efficiency?

中文作答：

提高效率时应把更具体、匹配事实更少的 pattern 放在前面，把 volatile facts 放在后面，减少 multifield wildcards 和不必要变量。`test` 条件应在相关变量绑定后尽早出现，以尽早过滤 partial matches。能用内置 pattern constraints 表达的约束，尽量写在 pattern 中，而不是放到后面的 test 中。

20. 易错点

- `?x` 匹配一个字段；`$?x` 匹配零个或多个字段。
- 同一个变量在同一规则中多次出现时，必须绑定为同一值。
- 未写出的 slot 通常表示不约束该 slot，不是该 slot 不存在。
- `not` 检查事实不存在，不是经典逻辑中的永久否定。
- `exists` 关注至少存在一个匹配，避免多个事实导致多次激活。
- `forall` 的意思是所有 first-CE 匹配都满足后续条件。
- `logical` 会建立依赖，支持事实消失时派生事实可被撤销。
- `salience` 可控制优先级，但不应代替清晰的逻辑条件。
- `modify` 和 `retract` 需要 fact index 或 fact address。
- 多个 multifield wildcard 会造成组合爆炸。
- CLIPS 是数据驱动规则系统；Prolog 是目标驱动逻辑程序。

09 Reinforcement Learning 考场资料

本章主线：强化学习 `reinforcement learning, RL` 研究一个有目标的 `agent` 如何在不确定环境中通过交互学习。与监督学习不同，环境通常不告诉 agent “正确动作是什么”，只在动作后返回 `reward`。agent 要在探索未知动作和利用当前较好动作之间平衡，并学习策略或价值函数。

0. 英文题目触发词速查

English trigger	中文定位	作答时要写出的核心
`reinforcement learning`	强化学习	goal-directed agent interacts with uncertain environment
`state`, `action`, `reward`	状态、动作、奖励	agent 观察状态、选择动作、环境返回奖励和新状态
`policy`	策略	$\pi(a \mid s)$ 表示状态 s 下选动作 a 的概率
`value function`	价值函数	预测从状态或状态-动作出发的长期回报
`multi-armed bandit`	多臂老虎机	只选动作并观察奖励，没有状态转移
`exploration`, `exploitation`	探索、利用	尝试未知动作 vs 选择当前最优动作
`return`	回报	折扣奖励和 G_t
`Bellman equation`	贝尔曼方程	当前价值 = 期望即时奖励 + 折扣后继价值
`policy evaluation`	策略评估	给定策略，估计 v_π
`policy improvement`	策略改进	根据价值函数让策略更贪心
`policy iteration`	策略迭代	evaluation 与 improvement 交替
`Monte Carlo`	蒙特卡洛	用完整 episode 的实际 return 平均估计价值
`TD learning`	时序差分	用一步奖励和下一状态估计 bootstrap
`Sarsa`	同策略 TD 控制	更新目标使用实际选出的下一动作
`Q-learning`	异策略 TD 控制	更新目标使用下一状态最大动作价值
`model-based methods`	基于模型方法	从真实经验学习模型，再用模拟经验 planning
`policy gradient`	策略梯度	直接优化参数化策略
`eligibility traces`, `TD(lambda)`	资格迹	在 one-step TD 和 Monte Carlo 之间折中分配 credit

1. RL 在机器学习中的位置

三类学习：

Type	中文	训练信号
`supervised learning`	监督学习	有外部监督者提供 labeled examples
`unsupervised learning`	无监督学习	从 unlabeled data 中发现结构
`reinforcement learning`	强化学习	agent 与环境交互，从 reward 学习

RL 的关键区别：没有每一步的标准答案。agent 需要通过试错发现哪些动作长期更好。

2. Agent-Environment 交互模型

迷宫例子可以文字化为：

1. agent 有内部状态或当前观察 `observation`。
2. agent 根据当前状态选择一个 `action`。
3. environment 接收动作，状态发生变化。
4. environment 返回新观察和 `reward`。
5. agent 根据 reward 调整未来行为。

标准轨迹：

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

符号：

Symbol	中文含义
S	状态集合
A	动作集合
R	可能奖励集合
$\pi(a \mid s)$	策略：状态 s 下采取动作 a 的概率
$v_\pi(s)$	状态价值：在策略 π 下从状态 s 出发的期望回报
$q_\pi(s,a)$	动作价值：在状态 s 先做动作 a，之后按 π 行动的期望回报

3. Multi-Armed Bandit 多臂老虎机

bandit 问题只有动作和奖励，没有显式状态转移。每个臂代表一个动作，选择后获得随机 reward。目标是在不断尝试中最大化累计奖励。

3.1 Action-Value Estimate

令 R_i 表示某动作第 i 次被选择后得到的奖励。令 Q_n 表示该动作被选择 $n-1$ 次后的价值估计：

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

当第 n 次奖励 R_n 到来后，新平均值：

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i$$

可改写为增量更新：

$$Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n)$$

解释：$R_n - Q_n$ 是新奖励与旧估计之间的误差；$1/n$ 是步长。样本越多，新样本对平均值影响越小。

3.2 Nonstationary 情况

如果奖励分布会随时间变化，旧样本不应一直同等重要。常用 constant step-size：

$$Q_{n+1} = Q_n + \alpha(R_n - Q_n)$$

展开后：

$$Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

解释：越新的奖励权重越大，越旧的奖励按 $(1-\alpha)$ 衰减。

3.3 Exploration vs Exploitation

- `exploitation` 利用：选择当前估计价值最高的动作。
- `exploration` 探索：尝试当前不确定或估计不高的动作，以获得信息。

ϵ-greedy 策略：

```
with probability 1-epsilon: choose argmax_a Q(a)
with probability epsilon: choose a random action
```

简单 bandit 算法：

```
Initialize, for a = 1 to k:
    Q(a) = 0
    N(a) = 0
Repeat:
    choose A by epsilon-greedy
    receive reward R
    N(A) = N(A) + 1
    Q(A) = Q(A) + 1/N(A) * [R - Q(A)]
```

4. Return 与 Bellman Equation

4.1 Return 回报

有限 horizon 的未折扣回报：

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

无限 horizon 或长期任务常用折扣回报：

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

递推形式：

$$G_t = R_{t+1} + \gamma G_{t+1}$$

γ 是折扣因子，$0 \le \gamma \le 1$。γ 越大，越重视长期奖励；γ 越小，越重视近期奖励。

4.2 State-Value Function

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

含义：在策略 π 下，从状态 s 开始的期望折扣回报。

Bellman equation：

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

变量解释：

- $\pi(a \mid s)$：策略在状态 s 选择动作 a 的概率。
- $p(s', r \mid s, a)$：环境在状态 s 做动作 a 后转到 s' 并得到奖励 r 的概率。
- $r + \gamma v_\pi(s')$：一步奖励加折扣后的后继状态价值。

一句话理解：状态价值等于所有可能动作、后继状态和奖励下的“即时奖励 + 折扣未来价值”的期望。

4.3 Action-Value Function

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$$

含义：先在 s 做 a，之后按策略 π 行动的期望回报。

5. Generalized Policy Iteration 与 Policy Iteration

`generalized policy iteration, GPI` 包含两个相互作用过程：

- `policy evaluation`：根据当前策略估计价值函数。
- `policy improvement`：根据价值函数改进策略，使策略更倾向于高价值动作。

二者循环进行：策略决定价值，价值又指导策略改进。

Policy iteration 流程：

1. 初始化 $V(s)$ 和 $\pi(s)$。
2. Policy evaluation：在当前策略下反复更新 V。
3. Policy improvement：令策略选择使期望回报最大的动作：

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$$

4. 如果策略不再变化，则停止；否则回到 evaluation。

中文解释：先弄清当前策略有多好，再根据这个评价改策略；直到改不动。

6. Monte Carlo Methods

Monte Carlo 方法用完整 episode 的实际 return 来估计价值。

First-visit MC：

```
for each episode:
    generate an episode following policy pi
    for each state first visited in the episode:
        compute return G from that time
        append G to Returns(state)
        V(state) = average(Returns(state))
```

特点：

- 不需要环境模型。
- 必须等 episode 结束才能知道完整 return。
- 不 bootstrap，因为目标是实际 G_t，不是下一状态估计。

7. Temporal-Difference Learning

TD learning 结合了 Monte Carlo 和 dynamic programming：像 MC 一样从经验学习，不需要模型；像 DP 一样 bootstrap，用已有估计更新已有估计。

TD(0)：

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

TD error：

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

解释：$R_{t+1} + \gamma V(S_{t+1})$ 是一步 TD target；如果它大于当前 $V(S_t)$，就上调价值；反之下调。

对比：

Method	Target	是否等 episode 结束	是否 bootstrap
Monte Carlo	G_t	是	否
TD(0)	$R_{t+1} + \gamma V(S_{t+1})$	否	是

8. Sarsa vs Q-Learning

8.1 Sarsa

Sarsa 是 on-policy TD control。名字来自一条更新使用的五元组：

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$$

更新公式：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

含义：下一动作 A_{t+1} 是当前行为策略实际选出来的动作，例如 ϵ-greedy 选出来的动作。因此 Sarsa 学的是包含探索行为在内的当前策略价值。

8.2 Q-Learning

Q-learning 是 off-policy TD control。

更新公式：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

含义：虽然行为策略可能为了探索而随机选动作，但更新目标假设下一状态会选择价值最大的动作。因此 Q-learning 直接逼近最优动作价值函数。

8.3 对比

维度	Sarsa	Q-learning
类型	on-policy	off-policy
下一动作	实际选出的 A_{t+1}	$\max_a Q(S_{t+1}, a)$
学习目标	当前策略的价值	最优策略的价值
探索影响	会把探索动作的后果算入价值	更新时按贪心目标估计

右侧示意图可文字化为：Expected Sarsa 对下一动作按策略求期望；Q-learning 在下一状态的多个动作中取最大值。

9. Model-Based Methods

基于模型方法不仅从真实交互中直接更新策略/价值，还学习环境模型。

基于模型方法的关系可以写成：

1. agent 与 `Environment` 交互，得到 `real experience`。
2. `real experience` 可直接更新 `Policy/value functions`，这叫 direct RL update。
3. `real experience` 还可用于 `model learning`，学习一个 `Model`。
4. `Model` 可产生 `simulated experience`。
5. simulated experience 用于 `planning update`，继续更新 policy/value functions。

中文解释：model-free 方法只用真实经验更新；model-based 方法学习环境如何转移和给奖励，再在模型中模拟经验，用 planning 提高样本效率。

10. Policy Gradient Methods

价值方法先学 V 或 Q，再由价值选动作；policy gradient 直接参数化策略并优化策略参数。

对每个 state-action pair 定义 preference：

$$h(s, a, \theta)$$

softmax 策略：

$$\pi(a \mid s, \theta) = \frac{\exp(h(s, a, \theta))}{\sum_b \exp(h(s, b, \theta))}$$

REINFORCE with baseline 的核心：

1. 生成一个 episode：

$$S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$$

2. 对每一步计算 return G_t。
3. 用 baseline value $\hat{v}(S_t, w)$ 减小方差：

$$\delta = G_t - \hat{v}(S_t, w)$$

4. 更新 value 参数：

$$w \leftarrow w + \alpha^w \gamma^t \delta \nabla_w \hat{v}(S_t, w)$$

5. 更新 policy 参数：

$$\theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla_\theta \ln \pi(A_t \mid S_t, \theta)$$

中文解释：如果某动作之后的实际回报高于 baseline，就增加该动作在该状态下的概率；如果低于 baseline，就降低它的概率。

11. n-step Bootstrapping 与 Eligibility Traces

11.1 n-step Return

n-step return：

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} \gamma^n \hat{v}(S_{t+n}, w_{t+n-1})$$

含义：先看未来 n 步真实奖励，再用第 n 步后的价值估计 bootstrap。

- $n = 1$ 接近 one-step TD。
- n 到 episode 末尾接近 Monte Carlo。

11.2 lambda-return

TD(λ) 把不同步数的 return 加权平均：

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$$

权重为：

$$(1-\lambda), \quad (1-\lambda)\lambda, \quad (1-\lambda)\lambda^2, \ldots$$

这些权重总和为 1。$\lambda = 0$ 时只看一步 TD；$\lambda = 1$ 时趋向 Monte Carlo。

11.3 Forward View 与 Backward View

`forward view`：站在当前状态 S_t，向未来看多个可能长度的 return，把它们按 λ 加权。这解释了目标是什么。

`backward view`：每次出现 TD error δ_t 时，用资格迹 z_t 给最近访问过的状态或特征分配 credit。这适合在线实现。

Semi-gradient TD(λ)：

$$z \leftarrow \gamma \lambda z + \nabla \hat{v}(S, w)$$

$$\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$$

$$w \leftarrow w + \alpha \delta z$$

中文解释：资格迹像“近期责任记录”。刚访问过的状态 trace 大，较早状态 trace 按 $\gamma\lambda$ 衰减；当前 TD error 会按 trace 分配给这些状态。

课件强调：

- eligibility traces 统一了 TD 和 Monte Carlo。
- $\lambda = 0$ 是 one-step TD。
- $\lambda = 1$ 接近 Monte Carlo。
- 中间值常在二者之间取得更好折中。
- eligibility traces 可以在线实现，适合持续任务。

12. Deep Nets + Reinforcement Learning

AlphaGo 例子说明深度网络可以与 RL 结合：

- `policy network` 用于选择动作或落子。
- `value network` 用于评估局面价值。
- 初始策略可由人类专家棋谱进行 supervised learning。
- 后续通过 self-play 产生数据，用 reinforcement learning 进一步提升。
- policy gradient 可用于改进策略网络。

中文解释：围棋搜索空间巨大，局面价值难以手工评估；深度网络提供近似的策略和价值函数，强化学习通过自我对弈不断改进这些函数。

12.5 课件应用例子补充

A Golf Example

这个例子用高尔夫球场说明 `value function` 和 `action-value function` 的区别。

图中要素：

英文标记	中文含义	图中怎么理解
`green`	果岭	目标区域，球进洞附近价值最高。
`sand`	沙坑	不利区域，图中价值标成 `-\infty` 或很低，表示应避免。
`v_putt`	使用 putter 策略时的状态价值	每个位置的数字表示从该状态开始、按 putt 策略到达目标所需代价/负回报。越接近 0 越好，越负越差。
`q_*(s, driver)`	在状态 `s` 选择 `driver` 这一动作的最优动作价值	先固定当前动作是 driver，再假设后续按最优策略行动。

关键理解：

- `v(s)` 评价“站在这个状态有多好”，不显式指定当前动作。
- `q(s,a)` 评价“在这个状态先做动作 `a` 有多好”。
- 图中等值线上的 `-1, -2, -3...` 表示到达目标的预期代价或负回报；数值越大，状态/动作越有利。
- 沙坑区域被赋予极差价值，说明 reward/value 可以把环境中的风险区域编码进策略学习。

中文答法：Golf example 展示了 RL 中价值函数如何把空间位置转化为可比较的长期回报。`v_putt` 给出某种策略下各状态的价值，而 `q_*(s, driver)` 给出在状态 `s` 先选择 driver 后的最优动作价值；agent 可据此选择长期回报更好的动作。

RL for Optimized Execution

`optimized execution` 指金融交易中的最优执行问题：要在市场中分批完成订单，同时尽量获得更好的成交价格、降低冲击成本和风险。

课件把它写成 state-based stochastic optimal control：

RL 元素	交易执行中的含义
`state`	当前时间、剩余待交易 shares，也可包含市场价格、订单簿、流动性等信息。
`action`	在 order book 中如何放置订单，例如下单价格、数量、位置。
`reward`	成交获得的价格或执行质量；也可包含成本、滑点、风险惩罚。
`stochastic`	同一状态下市场后续变化不确定，因此相同动作可能得到不同结果。

中文答法：优化执行可以看作强化学习问题，因为交易者需要在不确定市场中连续决策。状态描述时间和剩余交易量等信息，动作是订单在订单簿中的放置方式，奖励来自成交价格或执行收益；由于市场随机变化，agent 需要学习能最大化长期执行效果的策略。

13. Multi-Agent Autocurricula

多智能体自课程 `multi-agent autocurricula` 指多个 agent 在相互竞争或合作中自动生成越来越复杂的学习任务。环境难度不是人工固定设计的，而是在 agent 之间的互动中逐步升级。

作答可写：在多智能体环境中，一个 agent 的策略变化会改变其他 agent 面临的学习环境，因此群体互动可以产生 emergent tool use、复杂策略和逐步增长的课程难度。

14. RL 的挑战

课件列出的挑战可组织为：

- 状态和特征表示困难：高维连续状态和动作空间很常见。
- reward 设计困难：奖励可能稀疏、延迟、多目标或风险敏感。
- credit assignment 困难：长期之后才出现奖励，难判断哪个早期动作有贡献。
- exploration/exploitation tradeoff：既要尝试新动作，又不能损失太多奖励。
- 学习速度与样本效率：真实环境交互可能昂贵或慢。
- planning with learned models：学到的模型有误差，规划可能放大误差。
- subproblem selection：复杂任务中要决定先学哪些子问题。
- multi-agent RL：多个 agent 同时学习，环境对单个 agent 来说会变化。

15. 常见答题模板

15.1 问：What is reinforcement learning?

中文作答：

强化学习研究一个 goal-directed agent 如何在不确定环境中通过交互学习。agent 在状态中选择动作，环境返回奖励和下一状态；agent 的目标不是模仿标签，而是学习能最大化长期累计奖励的策略。

15.2 问：Explain Bellman equation

中文作答：

Bellman equation 表达价值函数的递归结构。状态 s 的价值等于在策略 π 下，对所有动作、后继状态和奖励求期望的“即时奖励加折扣后继价值”。公式为 $v_\pi(s)=\sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')]$。

15.3 问：Compare Monte Carlo and TD

中文作答：

Monte Carlo 用完整 episode 的实际 return G_t 更新价值，因此不需要模型，也不 bootstrap，但必须等 episode 结束。TD 用 $R_{t+1}+\gamma V(S_{t+1})$ 作为目标，可以一步一步在线更新，但它 bootstrap，因为目标中包含当前价值估计。

15.4 问：Compare Sarsa and Q-learning

中文作答：

Sarsa 是 on-policy 方法，更新目标使用实际按当前策略选出的下一动作 A_{t+1}，因此学习当前探索策略的价值。Q-learning 是 off-policy 方法，更新目标使用 $\max_a Q(S_{t+1},a)$，即下一状态的贪心动作价值，因此直接逼近最优动作价值。

16. 易错点

- reward 是一步反馈，return 是从某时刻开始的累计折扣奖励。
- $v_\pi(s)$ 是状态价值，$q_\pi(s,a)$ 是状态-动作价值。
- TD target 中含有估计值，所以 TD 是 bootstrap。
- Sarsa 的下一动作是实际采取的动作；Q-learning 的下一动作是取最大值的动作。
- model-based RL 的 model 是环境模型，不是神经网络模型这个泛称。
- $\lambda=0$ 接近 one-step TD，$\lambda=1$ 接近 Monte Carlo。