电子信息学中的机器学习

期末总结

宇航员Dale

2023 年 7 月 15 日

目录

§1 AI 及机器学习发展简史

1.1 AI 发展简史

1.1.1 AI 的诞生

AI 的起源可以追溯到 1956 年的达特茅斯会议，John McCarthy、Marvin Minsky、Oliver Selfridge、Allen Newell、Claude Shannon 和 Hebert Simon 等科学家开始讨论关于人工智能的理念和方法。其中，"Artificial Intelligence" 这个词也在这个会议中被提出。

1.1.2 第一次浪潮 1956-1974

1958 年，H. A. Simon 和 Allen Newell 预测：" 十年之内，数字计算机将成为国际象棋世界冠军" 和" 十年之内，数字计算机将发现并证明一个重要的数学定理"。

1965 年，H. A. Simon 声明：" 二十年内，机器将能完成人能做到的一切工作"。

1967 年，Marvin Minsky 预言：" 一代之内……创造人工智能的问题将获得实质上的解决"。

1970 年，Marvin Minsky 预计：" 在三到八年的时间里我们将得到一台具有人类平均智能的机器"。

然而，这一阶段的 AI 研究并未如预期那样取得突破。

1.1.3 第一次低谷 1974-1980

这一阶段，AI 研究者尝试使用传统的方法进行研究，即首先了解人类是如何产生智能的，然后让计算机按照人的思路去做。然而，这种方法在语音识别、机器翻译等领域却迟迟不能突破。

1.1.4 第二次浪潮 1980-1987

在这个阶段，一些关键的技术和观念开始成形，例如专家系统和决策树模型等。同时，研究者重新发现了神经网络的价值，重新发掘了反向传播算法。

1.1.5 第二次“低谷”1987-1993

由于各种原因，包括资本对技术期望过高，以及全球经济环境等因素，人工智能再次进入了“低谷”阶段。

但是与此同时，研究在学术界并没有停滞，卷积神经网络、循环神经网络，以及核函数与支持向量机诞生。

1.1.6 第三次浪潮

尽管经历了两次低谷，但是人工智能的发展并未停止。随着计算机性能的提升，AI 开始从小数据集的研究和实验中慢慢崭露头角。

在这个阶段，神经网络的发展越来越快，特别是在深度神经网络和卷积神经网络的应用上取得了显著的进步。同时，大数据和计算能力的提升也为 AI 的发展提供了更大的空间。

现在，人工智能已经渗透到了各个领域和行业，包括自然语言处理、推荐系统、通信网络、智能控制、微波天线设计和脑科学等。然而，当人们发现人工智能并没预想的那么赚钱时，是否会暂时冷静下来，甚至进入一个低谷？这个问题我们无法预知，但是我们相信，无论如何，人工智能的发展都将继续。

1.2 从 AI 到机器学习

人工智能（AI）有三个主要流派，分别是符号主义，行为主义和连接主义。

1. 符号主义以数学和逻辑为基础，其主要表现形式是计算机模拟数学证明和推导，例如 Mathematica。

2. 行为主义认为只有可以进行实际动作的（例如机器人）才能称之为人工智能。

3. 连接主义则主要指以神经网络为代表的基于大量数据生成数学模型并进行预测的人工智能。这种基于数据的建模和预测过程通常被称为机器学习，这也是当前主流的 AI 实现方式。

1.2.1 机器学习简要介绍

机器学习的基本流程可以概括为：基于大量数据，建立和优化数学模型，预测新的个体。这个流程和人的学习过程相似，人也是不断地学习知识，综合所学知识训练解决各类

问题的基本技能，然后使用训练好的技能解决类似的问题。

此外，机器学习可以进一步分为监督学习，无监督学习，和强化学习，分别应用在不同的场景中。

1. 监督学习，即根据自变量（特征）和因变量（标签）的关系建立数学模型，例如手写数字识别。

2. 无监督学习，则是根据自变量自身的特征建立数学模型，例如聚类。

3. 强化学习则是基于每一个样本包含的行动随着时间产生的状态变化来建立数学模型，例如学下棋。

在实际应用中，机器学习已经在许多领域发挥了重要作用，例如信号处理，通信网络，移动社交网络，电磁场与微波等。在这些领域中，机器学习方法可以被用来解决无线感知，资源分配，用户关联分析，天线设计，电磁成像等问题。

然而，尽管机器学习在许多领域具有显著的应用价值，但并不是所有问题引入机器学习都会有更好的效果。在某些情况下，传统的基于统计学和优化论的方法可能仍然能够提供有效的解决方案。同时，机器学习也不能解决所有传统方法难以解决的问题，它的应用效果取决于问题的特性和可用数据的质量和数量。

下面按照老师讲解的顺序以及上面的分类进行总结。

§2 监督学习

监督学习主要解决的是回归问题和分类问题。

2.1 回归问题

回归问题多用来预测一个具体的数值，如预测房价、未来的天气情况等等。例如我们根据一个地区的若干年的 PM2.5 数值变化来估计某一天该地区的 PM2.5 值大小，预测值与

当天实际数值大小越接近，回归分析算法的可信度越高。

回归问题的一般求解思路如下：

1. 选定训练模型，即我们为程序选定一个求解框架，如线性回归模型等。
2. 导入训练集 train_set，即给模型提供大量可供学习参考的正确数据。
3. 选择合适的学习算法，通过训练集中大量输入输出结果让程序不断优化输入数据与输出数据间的关联性，从而提升模型的预测准确度。
4. 在训练结束后即可让模型预测结果，我们为程序提供一组新的输入数据，模型根据训练集的学习成果来预测这组输入对应的输出值。

2.1.1 线性回归模型

用线性函数逼近训练数据。即

$$f_\theta(x) = \theta_0 + \theta_1 x$$

因此损失函数为

$$L(F_\theta(x), y) = (f_\theta(x) - y)^2$$

最小化损失函数，即求

$$\min_\theta \sum_{i=1}^{N} L(F_\theta(x_i), y_i) = \min_\theta \sum_{i=1}^{N} (\theta_0 + \theta_1 x_i - y_i)^2$$

求偏导数，有

$$\theta_1 = \frac{\sum_{i=1}^{N} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} (x_i - \overline{x})^2}$$

$$\theta_0 = \overline{y} - \theta_1 \overline{x}$$

2.1.2 非线性回归模型

非线性回归是在模型的趋势可以观察得出时，尝试通过更复杂的函数进行拟合，以达到更高的精确度。例如，使用二次函数进行逼近，整个函数图像不仅可以显示出“上升趋势”，而且还能显示出“越增越快”的特性。随着函数的次数增加，拟合的精度可能会提高，但也可能出现过拟合和欠拟合的问题。

欠拟合：模型过于简单，不能很好地拟合数据，表现出不够精确的问题。

过拟合：模型过于复杂，过度拟合了训练数据，缺乏泛化能力，无法很好地适应新的、未见过的数据。

为了解决这个问题，可以采用正则化方法。正则化方法通过对模型参数加入一定的约束或者惩罚，使模型保持简单（尽可能多的参数为 0），从而防止过拟合，同时也可以让模型选择一个较为复杂的拟合函数，以保证损失函数尽可能的小，防止欠拟合。其中，Lasso（套索回归）和 Ridge（岭回归）是两种常用的正则化方法。Lasso 使用 1 范数（绝对值）作为正则项，可以实现参数的稀疏，即让部分参数为 0；Ridge 使用 2 范数的平方作为正则项，可以使得参数尽量小，从而实现模型的平滑。

使用 0 范数时，带有正则项的目标函数为

$$\min_{\theta} \sum_{i=1}^{N} L\left(f_{\theta}\left(x_i\right), y_i\right) + \lambda \sum_{k}^{\infty} \mathbb{I}(\theta_k)$$

对于一般情况，带有正则项的目标函数为

$$\min_{\theta} \sum_{i=1}^{N} L\left(f_{\theta}\left(x_i\right), y_i\right) + \lambda g(\theta)$$

其中 $g(\theta)$ 为正则项。

2.1.3 多变量回归

解决实际问题时，往往会遇到多变量的回归问题，按照与单变量类似的思路，可以采用多变量线性回归和非线性回归进行处理。

多变量线性回归是针对有多个预测变量的线性回归问题。例如，考虑一个预测 CPU 性能的问题，可能会用到 CPU 的多个相关属性，如主存的最小/最大值，高速缓存的大小，通道的最小/最大值，指令周期等。将第 i 个样本的属性向量表示为 $x_i = x_{i_0}, x_{i_1}, \cdots, x_{i_D}$ ，优化目标是最小化预测值与真实值之间的平方差。

多变量多项式回归则是对于非线性关系的情况，使用多项式来进行拟合。例如，可以将一个二元二次多项式作为决策函数。但需要注意的是，当特征数量（变量数）增加时，多项式回归可能会变得过于复杂，导致计算量过大，且容易出现过拟合问题，因此并不总是适用。

在实际问题中，通常需要根据数据的具体情况和任务的需求来选择合适的模型。例如，如果任务是预测任务，并且预测变量与目标变量之间的关系较为复杂，可能需要选择多项式回归；而如果预测变量与目标变量之间的关系较为简单，或者任务是分类任务，可能选择线性回归或者逻辑回归会更加合适。

2.2 分类问题

2.2.1 逻辑斯蒂回归

统计学上，以最大似然估计为目标，使用 Sigmoid function 作为决策函数形式的模型建立过程，叫作逻辑斯蒂回归（Logistic Regression）

逻辑斯蒂回归主要用于解决二分类问题，损失函数的目的是为了定义拟合的精确性，与回归问题有所区别，分类问题中每个样本的标签是其类别，取值只有两种。

逻辑斯蒂回归的基本思想是将线性回归的输出通过 Sigmoid 函数转化为概率值，用于二分类问题。具体来说，逻辑斯蒂回归模型首先计算线性函数，然后将其结果传递给逻辑函数，最后输出的是一个介于 0 和 1 之间的概率值。如果输出概率大于或等于 0.5，模型就将这个样本分类为正类；如果输出概率小于 0.5，模型就将这个样本分类为负类。

逻辑斯蒂回归的决策函数为

$$f_\theta(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$$

常用损失函数为

$$L\big(f_\theta(x), y\big) = -\log_2\big(f_\theta(x)\big)^y\big(1-f_\theta(x)\big)^{1-y}$$

目标函数为

$$\min_\theta \sum_{i=1}^{N} L\big(f_\theta(x_i), y_i\big)$$

因此需要求解

$$\frac{\partial \sum\limits_{i=1}^{N} L\big(f_\theta(x_i), y_i\big)}{\partial\theta} = 0$$

2.2.2 多分类问题 (Softmax 回归)

多分类问题可以强行看成二分类问题进行求解，但是开销很大，因此可类比二分类，从概率分布的角度扩展。

输出为“i”的概率为

$$P\big(y=i|x\big) = \frac{e^{-\left(\theta_{i,0}+\theta_{i,0}x\right)}}{\sum_{k=0}^{k} e^{-\left(\theta_{k,0}+\theta_{k,1}x\right)}}$$

因此，第 i 个样本 x_i 对应于各个类别的概率（θ 是一个 D 维向量）为

$$\mathbf{f}_\Theta(\mathbf{x}_i) = \begin{bmatrix} p_\Theta(y=1|\mathbf{x}_i) \\ p_\Theta(y=2|\mathbf{x}_i) \\ \cdots \\ p_\Theta(y=K|\mathbf{x}_i) \end{bmatrix} = \frac{1}{\sum_{k=0}^{K} e^{\theta_k^T\mathbf{x}_i}} \begin{bmatrix} e^{\theta_1^T\mathbf{x}_i} \\ e^{\theta_2^T\mathbf{x}_i} \\ \cdots \\ e^{\theta_k^T\mathbf{x}_i} \end{bmatrix}$$

其中，$\mathbf{\Theta} = [\theta_1, \cdots, \theta_K]$

因此目标函数为

$$\min_{\Theta} -\frac{1}{N}\left(\sum_{i=1}^{N}\sum_{k=1}^{K}\mathbb{I}(y_i = k)\log\frac{e^{\theta_k^T \mathbf{x}_i}}{\sum_{j=1}^{K} e^{\theta_j^T \mathbf{x}_i}}\right)$$

可以引入正则项以解决“冗余”问题，即

$$\min_{\Theta} -\frac{1}{N}\left(\sum_{i=1}^{N}\sum_{k=1}^{K}\mathbb{I}(y_i = k)\log\frac{e^{\theta_k^T \mathbf{x}_i}}{\sum_{j=1}^{K} e^{\theta_j^T \mathbf{x}_i}}\right) + \frac{\lambda}{2}\sum_{k=1}^{K}\sum_{d=1}^{D}\theta_{kd}^2$$

2.2.3 附：数学基础与问题求解算法

2.2.3.1 优化相关数学基础

凸集

集合 $C \in \mathbb{R}^n$ 是凸集当且仅当

$$\forall \vec{x}, \vec{y} \in C, \forall \alpha \in [0,1], \alpha\vec{x} + (1-\alpha)\vec{y} \in C$$

凸集的几何解释是凸集内任意两点的连线 (线段) 上的点，仍属于这个集合。也就说，这个集合是“连通”的。

凸函数

凸函数是定义域 $dom(f)$ 为凸集，且满足如下性质的函数：

$$\forall x, y \in dom(f), \forall \alpha \in [0,1], f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$$

凸函数判定条件

一阶：$f(\vec{y}) \geq f(\vec{x}) + \nabla f^T(\vec{x})(\vec{y} - \vec{x}), \forall \vec{x}, \vec{y} \in dom(f)$

二阶：Hessian 矩阵半正定

最优化问题形式

求

$$\min_x f_0(x) \quad s.t.\ f_i(x) \leq 0, i = 1, \cdots, m \quad h_i(x) = 0, i = 1, \cdots, P$$

可行域

$$C = \{x | x \in \{\bigcap_{i=1}^{m} dom f_i\} \bigcap \{\bigcap_{i=1}^{p} dom h_i\}, f_i(x) \leq 0, h_i(x) = 0, \forall i\}$$

凸优化问题形式

求 $\min f_0(x),\ s.t. Ax = b, f_i(x) \leq 0, i = 1, \cdots, m,$ 其中 $f_0, \cdots, f_m$ 是凸函数，$A \in \mathbb{R}^{p \times m}, b \in \mathbb{R}^p$

对于上述问题，任何一个局部最优点都是全局最优点。

对偶理论求解

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{i=1}^{p} \mu_i h_i(x)$$

KKT 条件

$$\begin{cases} g_i(x) \leq 0, h_i(x) = 0 \\ \lambda_i \geq 0, \lambda_i g_i(x) = 0 \\ \nabla_x L(x, \lambda, \mu) = 0 \end{cases}$$

2.2.3.2 梯度下降算法

标准梯度下降

目标：想要找到最优的 $\boldsymbol{\epsilon}$，即原来的点为 $\boldsymbol{\Theta}$，新的点为 $\boldsymbol{\Theta}+\boldsymbol{\epsilon}$, 希望在 $\boldsymbol{\Theta}+\boldsymbol{\epsilon}$ 处梯度为 0，即

$$\nabla f(\boldsymbol{\theta}+\boldsymbol{\epsilon})=0 \Rightarrow \nabla f(\boldsymbol{\theta})+\nabla^2 f(\boldsymbol{\theta})\cdot\boldsymbol{\epsilon}+O(\cdot)=0$$
$$\Rightarrow \nabla f(\boldsymbol{\theta})+\boldsymbol{H}\boldsymbol{\epsilon}=0 \Rightarrow \boldsymbol{\epsilon}=-\boldsymbol{H}^{-1}\nabla f(\boldsymbol{\theta})$$

可表示为：$\theta \rightarrow \theta - H^{-1}\nabla f(\theta)$

优点：

1）每次迭代是对所有样本进行计算，利用矩阵运算，可实现并行；

2）由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。当目标函数为凸函数时，一定能够得到全局最优。

缺点：当样本数目很大时，训练过程很慢。

随机梯度下降

每次迭代中，对数据样本随机均匀采样一个样本 $i, i \in \{1,2,...,n\}$ ，即

$$\theta \leftarrow \theta - \eta \nabla_\theta f(\boldsymbol{x_i}, \boldsymbol{\theta})$$

优点：在每轮迭代中，随机优化某一条训练数据上的损失函数，因此每一轮参数的更新速度快。

缺点：

1）准确度下降。由于即使在目标函数为强凸函数的情况下，SGD 仍旧无法做到线性收敛；

2）可能会收敛到局部最优，由于单个样本并不能代表全体样本的趋势；

3）不易于并行实现。

小批量随机梯度下降

设 S 是样本集 N 的子集（一个 batch），则小批量 SGD 更新为

$$\theta \leftarrow \theta - \frac{\eta}{|S|}\sum_{i\in S}\nabla_\theta f(\boldsymbol{x}_i,\boldsymbol{\theta}), S \subset N$$

优点：

1）减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。

2）可实现并行化。

缺点：batch_size 的不当选择可能会带来一些问题。

其余问题求解算法及适用范围如下表所示。

	迭代公式	收敛性	擅长场景	应用代表
梯度下降法	$\theta_{k+1} \leftarrow \theta_k - \eta\nabla\mathcal{L}_k$	一阶收敛	神经网络	SGD Adam 动量法
牛顿法	$\theta_{k+1} \leftarrow \theta_k - H_k^{-1}\nabla\mathcal{L}_k$	二阶收敛	非线性最小二乘法	LM法
拟牛顿法	$\theta_{k+1} \leftarrow \theta_k - C_k\nabla\mathcal{L}_k$ 其中 $C_k y_{k-1} = S_{k-1}$	二阶收敛	逻辑回归	BFGS算法

2.2.4 感知机

2.2.4.1 简介

假设输入空间 (特征空间) 是 $\mathcal{X} \subseteq \mathbb{R}^n$，输出空间是 $\mathcal{Y} = \{+1, -1\}$ 输入 $x \in \mathcal{X}$ 表示实例的特征向量，对应于输入空间 (特征空间) 的点输出 $x \in \mathcal{Y}$ 表示实例的类别，由输入空间到输出空间的函数

$$f(x) = sign(\mathbf{w} \cdot \mathbf{x} + b)$$

被称为感知机，其中 **w** 和 **x** 是感知机模型参数，$\mathbf{w} \in \mathbb{R}^n$ 叫做权值 (weight) 或者权值向量，

$\mathbf{b} \in \mathbb{R}$ 叫做偏置，$\mathbf{w} \cdot \mathbf{x}$ 是内积，$sign()$ 是符号函数，即

$$\text{sign}(x) = \begin{cases} +1, & x \geqslant 0 \\ -1, & x < 0 \end{cases}$$

2.2.4.2 几何含义

其几何含义为用一个分类超平面，将特征空间划分为两个部分，分离正、负类。

2.2.4.3 损失函数

损失函数为：

$$L(\boldsymbol{w}, b) = -\sum_{\boldsymbol{x_i} \in M} y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b)$$

2.2.4.4 求解算法

标准梯度下降法是行不通的

1. 因为目标函数限定：只有误分类的点才能参与损失函数的优化
2. 而标准 GD 是基于所有样本的梯度和来计算的
3. 可以采用随机梯度下降算法或者小批量梯度下降算法

若随机梯度下降算法：每次仅需要使用一个误分类的点来更新梯度。

先求梯度

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, b) = -\sum_{x_i \in M} y_i \boldsymbol{x}_i \quad \nabla_b L(\boldsymbol{w}, b) = -\sum_{x_i \in M} y_i$$

随机选取一个误分类点 $\boldsymbol{x_i}$, y_i ，其对 $\boldsymbol{w}, b$ 的贡献：

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_i x_i$$

$$b \leftarrow b + \eta y_i$$

对偶形式：

将 $\boldsymbol{w}, b$ 均表示为实例 x_i 和标签 y_i 的线性组合的形式，通过求解该线性组合的系数，求得 $\boldsymbol{w}, b$ 根据原始算法，已知对误分类点 $\boldsymbol{x}_i$, y_i 的更新为

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_i x_i$$

$$b \leftarrow b + \eta y_i$$

假设初始值 $\boldsymbol{w}_0, b_0$ 均为 0，按照以上规则逐步修改 $\boldsymbol{w}_0, b_0$，设修改 n_i 次，则 $\boldsymbol{w}_0, b_0$ 关于 $(\boldsymbol{x}_i, y_i)$ 的增量分别为 $\alpha_i y_i x_i$ 和 $\alpha_i y_i$，其中 $\alpha_i = n_i \eta$

最后学习到的 $\boldsymbol{w}_0, b_0$ 可以分别表示为

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \quad b = \sum_{i=1}^{N} \alpha_i y_i$$

2.2.4.5 总结

不足:

1. 要求线性模型，且数据集线性可分

2. 比如，解决不了异或问题

改进方向:

1. 使用更多的感知机：神经网络

2. 引入非线性模型，如 SVM（核函数）进行处理

2.2.5 支持向量机

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 还包括核技巧，这使它成为实质上的非线性分类器。SVM 的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM 的的学习算法就是求解凸二次规划的最优化算法。

支持向量指线性可分情况下，训练数据集的样本点中与分离超平面距离最近的样本点的实例，即满足 $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$.

2.2.5.1 问题描述与形式

问题描述：求解能够正确划分训练数据集并且几何间隔最大的分离超平面。

其问题形式为：

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$s.t. y_i(\boldsymbol{w}^T x_i + b) \geqslant 1, i = 1, 2, \cdots, N$$

2.2.5.2 对偶算法

构建拉格朗日函数，引入拉格朗日乘子 $\{\alpha_i\}$

$$L(w, b, \alpha) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N} \alpha_i y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) + \sum_{i=1}^{N} \alpha_i$$

再根据 KKT 条件，代入化简后得到对偶问题的目标函数

$$\max_{\alpha} -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) + \sum_{i=1}^{N} \alpha_i$$

因此对偶问题写作：

$$\min_{\alpha} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) - \sum_{i=1}^{N} \alpha_i$$

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

$$\alpha_i \geq 0, i = 1, 2, \cdots, N$$

再通过对偶算法的解便可以解出原问题的解。

2.2.5.3 推广到线性不可分数据集

以上线性 SVM 适用于线性可分数据集，当应用于线性不可分数据集时，不等式约束无法都成立。

解决办法：

1. 从硬间隔最大化变为软间隔最大化；

2. 对每个松弛变量支付一个代价 ，即目标函数变为

$$\frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{j=1}^{N}\xi_i$$

建模得到拉格朗日函数

$$L(\boldsymbol{w}, b, \xi, \alpha, \mu) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{N}\mu_i\xi_i$$

与上面类似通过对偶算法求解。

2.2.5.4 非线性分类问题

无法通过线性模式去分类，需要引入非线性模型。

对于分类问题，如果能用一个超曲面将正负实例分开，则称该问题是非线性可分的。

解决思路：采用非线性变换，将非线性问题转化为线性问题，然后利用线性 SVM 进行求解。即需要找到合适的 $\phi(\boldsymbol{x})$ 进行映射，然后用 $\phi(\boldsymbol{x}_i)$ 去替代每个实例点 $\boldsymbol{x}_i$.

核函数：

设输入空间为 $\mathcal{X}$，映射后的特征空间为 $\mathcal{H}$，该映射记作 $\phi(\boldsymbol{x}) : \mathcal{X} \to \mathcal{H}$，若对所有的 $x, z \in \mathcal{X}$, 有 $K(x,z) = \phi(x) \cdot \phi(z)$ 则称 $K(x,z)$ 为核函数。

利用核函数进行映射后，对偶问题的目标函数变为

$$\min_{\alpha}\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\big(K(\boldsymbol{x}_i, \boldsymbol{x}_j)\big) - \sum_{i=1}^{N}\alpha_i$$

得到的 SVM 决策函数为

$$f(x)=\operatorname{sign}\left(\sum_{i=1}^{N} a_i^* y_i \phi(\boldsymbol{x}_i)\cdot\phi(\boldsymbol{x})+b^*\right)=\operatorname{sign}\left(\sum_{i=1}^{N} a_i^* y_i K(\boldsymbol{x}_i,\boldsymbol{x})+b^*\right)$$

核函数的选取：

$K(\boldsymbol{x},\boldsymbol{z})$ 可写作两个映射函数的内积，当且仅当 $K(\boldsymbol{x},\boldsymbol{z})$ 满足以下两个条件。此时，正定核函数 $K(x,z)=\phi(x)\cdot\phi(z)$

1）$K(x,z)=K(z,x)$, 即核函数为对称函数

2）对于任意输入数据集 $\mathcal{X}=\{\boldsymbol{x}_1,...,\boldsymbol{x}_N\}$, 核函数对应的矩阵 $\left[K(\boldsymbol{x}_i,\boldsymbol{x}_j)\right]_{N\times N}$ 是半正定矩阵，即对 $\forall C_i(i=1\sim N)$, 有 $\sum_{i=1}^{N}\sum_{j=1}^{N} C_i C_j K\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)\geq 0$

常用核函数：

名称	表达式	参数
线性核	$\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)=\boldsymbol{x}_i^{\top}\boldsymbol{x}_j$	
多项式核	$\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)=(\boldsymbol{x}_i^{\top}\boldsymbol{x}_j)^d$	$d\geq 1$为多项式的次数
高斯核	$\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)=\exp\left(-\frac{\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2}{2\delta^2}\right)$	$\delta>0$为高斯核的带宽(width)
拉普拉斯核	$\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)=\exp\left(-\frac{\|\boldsymbol{x}_i-\boldsymbol{x}_j\|}{\delta}\right)$	$\delta>0$
Sigmoid核	$\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)=\tanh(\beta\boldsymbol{x}_i^{\top}\boldsymbol{x}_j+\theta)$	tanh为双曲正切函数, $\beta>0,\theta<0$

2.2.5.5 支持向量回归

目标：学习一个形式为 $f(\boldsymbol{x})=\boldsymbol{w}^{\top}\boldsymbol{x}+b$ 的决策函数。

传统回归：希望所有的点都落在决策函数对应的曲线上，距离越小越好。

SVR 回归：允许有 的偏差，即落入红色区域的样本点不计入损失。

SVR 问题可以形式化为

$$\min_{\boldsymbol{w},b}\quad \frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{N}\ell_{\epsilon}(f(\boldsymbol{x}_i)-y_i)$$

可以利用对偶问题求解，也可以利用核函数推广到非线性。

2.2.5.6 半监督 SVM

针对二分类问题，在部分数据实例无标签的情况下，学习分类模型对无标签数据进行分类。

算法思路：

先利用有标签样本进行训练，得到分类器，用该分类器去标注无标签样本，得到相应的临时标签 $\hat{\boldsymbol{y}}$；

将有标签样本和临时标签的样本放在一起，再一次训练，得到分类器 SVM1；

从那些临时标签的样本中，找出两个标签相反且极可能发生标记错误的样本，交换它们的标签，再重新进行训练，得到更新后的分类器 SVM2；

以上步骤不断重复，每一次都增大 C_u 的值来提高无标签样本对优化目标的影响，直至 $C_u = C_l$ 终止；

最终得到的 SVM 为所有的无标签样本提供了标签，并且可以用于对测试数据进行分类。

2.2.6 决策树

2.2.6.1 简介

决策树（Decision Tree），它是一种以树形数据结构来展示决策规则和分类结果的模型，作为一种归纳学习算法，其重点是将看似无序、杂乱的已知数据，通过某种技术手段将它们转化成可以预测未知数据的树状模型，每一条从根结点（对最终分类结果贡献最大的属性）到叶子结点（最终分类结果）的路径都代表一条决策的规则。

分类决策树由节点和有向边组成。节点分为内部节点（表示一个特征或者属性）和叶节点（表示一个类别），最上面的节点为根节点，每个分支是一个新的内部节点。可以看做是 If-then 规则的集合，其每一条路径对应一条规则，规则集合互斥且完备。每个实例都被有且仅有一条规则描述

2.2.6.2 决策树学习

任务：给定训练数据集

$$D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$$

其中 $x_i = \left(x_i^{(1)}, x_i^{(2)}, \cdots, x_i^{(n)}\right)^{\mathrm{T}}$ 为输入实例（特征向量），n 为特征个数，$y_i \in \{1, 2, \cdots, K\}$ 为类别标记（标签），N 为数据集的实例个数。建立一个决策树模型，使之能够对实例进行正确的分类。

其本质上是从训练数据集中归纳出一组分类规则，与训练数据集不相矛盾的决策树。

能对训练数据进行正确分类的决策树可能有多个，也可能一个也没有。我们需要的是一个与训练数据矛盾较小的决策树，同时具有很好的泛化能力。

决策树学习是由训练数据集估计条件概率模型。基于特征空间划分的条件概率模型可能有无穷多个，我们需要的条件概率模型应该不仅对训练数据有很好的拟合，而且对未知数据有很好的预测。

学习策略：以损耗函数为目标函数，为了最小化损失函数来选择最优决策树。

手段：特征选择（递归法）-> 决策树生成（基于特征构造树）-> 决策树剪枝

2.2.6.3 信息增益与特征选择

信息增益定义：特征 A 对训练数据集 D 的信息增益，$g(D, A)$, 定义为集合 D 的经验熵 $H(D)$ 与特征 A 给定条件下 D 的经验条件熵 $H(D|A)$ 之差，即

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

其表示得知特征 X 的信息而使得类 Y 的信息的不确定性减少的程度

其中，熵的本质是一个系统 “内在的混乱程度”，表征随机变量不确定性的度量。即

$$H(\boldsymbol{p}) = -\sum_{k=1}^{K} p_k \log_2 p_k$$

条件熵表示在已知随机变量 X 的条件下随机变量 Y 的不确定性，即

$$H(Y \mid X) = \sum_{i=1}^{n} p_i H(Y \mid X = x_i)$$

特征选择准则：对于训练数据集 D，计算每个特征的信息增益，并比较大小，选择信息增益最大的特征。

2.2.6.4 信息增益比

信息增益的问题：可能会偏向于选取取值较多的特征。引入信息增益比：消除特征本身取值个数所带来的影响。即

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

$$H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$$

2.2.6.5 ID3 算法

核心：

在决策树的各个节点上应用**信息增益准则**选择特征，递归的构建决策树步骤：

- 从根结点 (root node) 开始，对结点计算所有可能的特征的信息增益；
- 选择信息增益最大的特征作为结点的特征，由该特征的不同取值建立子结点；
- 再对子结点递归地调用以上方法，构建决策树；
- 直到所有特征的信息增益均很小或没有特征可以选择为止。

2.2.6.6 C4.5 算法

核心：在决策树的各个节点上应用**信息增益比准则**选择特征，递归的构建决策树其步骤与 ID3 一样。

缺陷：

C4.5 生成的是多叉树，即一个父节点可以有多个节点。很多时候，在计算机中二叉树模型更方便；

C4.5 只能用于分类，是否能将决策树用于回归；

使用了熵模型，包含有大量耗时的对数运算。

2.2.6.7 剪枝

目的：减弱过拟合现象（生成树时，过多的考虑如何提高正确分类率，生成了规则复杂的树）

手段：从已生成的树上裁掉一些子树或叶结点，并将其根结点或父结点作为新的叶结点，从而简化分类树模型

设树 T 的叶结点个数为 $|T|$，t 是树 T 的一个叶结点，该叶结点有 N_t 个样本点，其中 k 类的样本点有 N_{tk} 个，$k = 1, 2, \cdots, K$,K 为样本空间中的所属分类数量。叶结点 t 上的经验熵 $H_t(T)$ 为

$$H_t(T) = -\sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t}$$

损失函数：

$$C_\alpha(T) = C(T) + \alpha|T|$$

$$C(T) = \sum_{t=1}^{|T|} N_t H_t(T) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t}$$

2.2.6.8 CART

CART 决策树

Scikit-learn 官方指定用树，其为二叉树，可以用于回归和分类，将熵模型变成了简单的线性运算。

CART 用于回归

设 与 是输入和输出，其中 是连续变量，给定训练数据集

$$D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$$

则 CART 回归树对应着输入空间（即特征空间）的一个划分以及在划分单元上的输出值。

使用基尼指数来代替信息增益比

在分类问题中，假设有 K 个类别，第 k 个类别的概率为 p_k, 则基尼指数的表达式为

$$Gini(p) = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

对于二类分类问题，若样本点属于分类 1 的概率是 p，则基尼指数的表达式为：

$$Gini(p) = 2p(1-p)$$

对于个给定的样本 D, 假设有 K 个类别, 第 k 个类别的数量为 C_k，则样本 D 的基尼指数表达式为

$$Gini(D) = 1 - \sum_{k=1}^{K} \left(\frac{|c_k|}{|D|} \right)^2$$

如果根据特征 A 的某个值 a, 把 D 分成 D_1 和 D_2 两部分，则在特征 A 的条件下，D 的基尼指数表达式为：

$$Gini(D, A) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

基尼指数更加简便，省去了对数运算，提供了对熵模型的近似。

损失函数：

$$C(T) = \sum_{t=1}^{|T|} N_t H_t(T) = \sum_{t=1}^{|T|} \left(1 - \sum_{k=1}^{K} \frac{N_{tk}}{N_t} \right)$$

剪枝原则

设生成的整体树为 T_0，对 T_0 的任意内部节点 t，以 t 为单节点树的损失函数为

$$C_\alpha(t) = C(t) + \alpha$$

以 t 为根节点的子树 T_t 的损失函数为

$$C_\alpha(T_t) = C(T_t) + \alpha |T_t|$$

当 $\alpha = \frac{C(t) - C(T_t)}{|T_t| - 1}$ 时，$C_\alpha(t) = C_\alpha(T_t)$，若 α 大于此值，则应该剪枝。

逻辑回归与决策树的对比

	逻辑回归	决策树
数学形式	$h_\theta(x)=g(\theta^T x)=\frac{1}{1+e^{-\theta^T x}}$ $\theta^T x=\sum_{i=1}^{n}\theta_i x_i$	$G(x)=\sum_{t=1}^{T} q_t(x)\cdot g_t(x)$ $q_t(x)$代表决策树分支t能否覆盖实例x， $g_t(x)$代表决策树分支t的结果
输出	概率	分类：众数 回归：均值
数据实例输入	连续变量	连续或离散
数据实例标签	0,1	多标签也可以
损失函数	交叉熵	分类：基尼系数 回归：平方误差
求解方法	梯度下降	离散穷举
学习结果	权重θ	切分特征、切分点
优点	分类比决策树准确 没有决策树容易过拟合	计算量小，属于非线性、非参数方法； 不需要做特征筛选，适合高维数据

2.2.6.9 提升方法

AdaBoost：利用前一轮迭代弱学习器的误差率来更新训练集的权重。

GBDT：针对 CART 回归树，利用残差拟合来训练弱学习器。

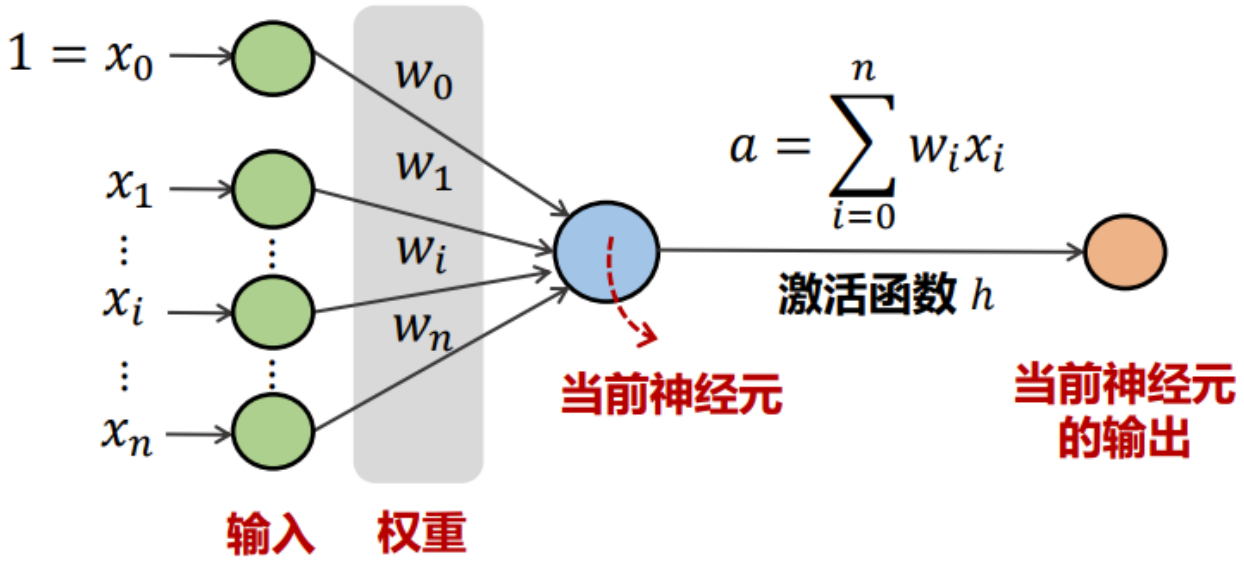
随机森林：随机选择一部分特征，在这部分特征里面选最优的，增强泛化能力。

§3 神经网络

3.1 全连接神经网络

3.1.1 基本概念

神经元模型：把逻辑斯蒂回归的决策函数形象化，就得到了一个神经元。决策函数中也可以引入更多的非线性因素。神经元结构不断重复，就构成了一个神经网络。如下图所示。



3.1.2 激活函数

激活函数是向神经网络中引入非线性因素，通过激活函数神经网络就可以拟合各种曲线。

下面是一些常见的激活函数：

1. 线性激活函数（Identity）：

$$f(x) = x$$

这是最简单的激活函数，它返回输入值本身。线性函数的问题在于，无论多少层的神经网络，如果所有的激活函数都是线性的，那么整个神经网络仍然是线性的，这意味着它的表达能力不足。

2.Sigmoid 函数：

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

Sigmoid 函数可以将任何值都映射到一个位于 0 到 1 之间的值。因此，它经常被用于二分类问题的最后输出层，输出的是概率值。但是，Sigmoid 函数在值的两端有所谓的梯度消失问题，也就是在这些区域内，梯度几乎为 0，这对于神经网络的学习是不利的。

3.Tanh 函数：

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

Tanh 函数可以将任何值都映射到-1 和 1 之间。与 Sigmoid 函数相比，Tanh 函数的输出结果是以 0 为中心的，但是，和 Sigmoid 函数一样，Tanh 函数也存在梯度消失的问题。

4.ReLU（Rectified Linear Units）函数：

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

ReLU 函数在输入值小于 0 时输出 0，在输入值大于 0 时输出输入值本身。相比于 Sigmoid 和 Tanh 函数，ReLU 函数在输入值大于 0 的部分没有梯度消失的问题，计算速度也更快，因此，ReLU 函数目前在深度学习中使用最为广泛。

5.Softmax 函数：

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

Softmax 函数可以将一组数值转换为概率分布，即所有的输出值都在 0 到 1 之间，且所有的输出值之和为 1。因此，Softmax 函数常用于多分类问题的输出层。

3.1.3 神经网络的结构

神经网络的基本构成包含以下三个主要组成部分：输入层（Input Layer）、隐藏层（Hidden Layer）、输出层（Output Layer）。

1. 输入层（Input Layer）：输入层是神经网络接收信息的部分。

2. 隐藏层（Hidden Layer）：隐藏层位于输入层和输出层之间。一个神经网络可以有一层或者多层隐藏层。隐藏层中的每一层都由多个神经元（也称为节点或单位）组成，每个神经元都有一个权重和偏置值，并且通过激活函数将其输入转换为输出。隐藏层的目标是从输入层接收的数据中提取出有用的信息，并将其传递给输出层。

3. 输出层（Output Layer）：输出层是神经网络的最后一层，用于输出神经网络的预测结果。输出层的神经元数量取决于问题的类型：对于二分类问题，输出层通常只有一个神经元；对于多分类问题，输出层的神经元数量通常与类别的数量相等；对于回归问题，输出层通常只有一个神经元。

3.1.4 算法

目标函数：

$$\min_{W_1,\ldots,W_L} L = \frac{1}{2}\sum_{j=1}^{N}\left\|\boldsymbol{y}^{(j)} - \widetilde{\boldsymbol{y}}^{(j)}\right\|^2$$

其中，

$$\boldsymbol{y} = f\left(h\left(\boldsymbol{W_n}h\left(\boldsymbol{W_{n-1}}\ldots h(\boldsymbol{W_1}\boldsymbol{x})\right)\right)\right)$$

采用梯度下降法计算。针对神经网络的复合函数特性，可以利用链式法则从后往前分层算，即反向传播求梯度。可以采用小批量梯度下降的方式来更新权重和阈值参数。

3.1.5 结构优化

- 合理确定神经元层数。神经元太少则刻画非线性的能力不够，神经元太多则容易过拟合。
- 选取恰当的损失函数。如交叉熵损失函数。
- 选取恰当的激活函数。如选取线性整流函数来减小梯度消失的风险。
- 解决过拟合问题。可以引入 dropout 层，随机删除隐藏层神经元，提升模型的泛化能力。
- 设置合理的权重初值。
- 对数据进行归一化。

3.1.6 缺点

- 提取了过多无用信息。

- 数据的形状被忽视。

较好的解决方案为采用部分连接的卷积神经网络。

3.2 卷积神经网络

3.2.1 卷积

一维卷积

给定两个连续函数 $f(x)$ 和 $g(x)$，则二者的卷积定义为

$$h(x) = (f * g)(x) = \int f(\tau)g(x-\tau)d\tau$$

给定两个离散序列 $f(x)$ 和 $g(x)$，则二者的卷积定义为

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(n-\tau)$$

多项式形式：

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$
$$g(x) = b_m x^m + b_{m-1}x^{m-1} + \cdots + b_1 x + b_0$$
$$\Rightarrow (f * g)(i) = \sum_i c_i x^i$$

其中，$c_i = \sum_k a_k b_{i-k}$。

二维卷积

卷积核：权重全层共用一个卷积核减少了参数量，防止过拟合卷积核又称为滤波器。

步幅：卷积核向前移动的窗口的间隔。

假设输入大小为 (H, W)，卷积核大小为 (FH, FW)，输出大小为 (OH, OW)，填充为 P，步幅为 S。此时，输出大小

$$OH = \frac{H + 2P - FH}{S} + 1$$
$$OW = \frac{W + 2P - FW}{S} + 1$$

三维卷积对于图像，除了长和高，还有通道（R、G、B）。把每个通道的卷积结果加起来即可。

3.2.2 卷积层

卷积层即对输入图像进行卷积操作的一层。一个精心设计的卷积层可以有效抽取图像的初级特征，如转角、边等。

卷积层看上去运算非常复杂，嵌套在一起会带来极高的复杂度，因此引入池化层。

3.2.3 池化层

池化层（Pooling Layer）的主要目标是通过降采样（downsampling）来减少数据的维度，降低计算复杂度，同时也能在一定程度上提高模型的鲁棒性，避免过拟合。

池化操作通常有两种形式：最大池化（Max Pooling）和平均池化（Average Pooling）。在最大池化中，我们选择一块区域（例如，2x2 像素）内的最大值作为该区域的代表值；在平均池化中，我们取这块区域内的平均值作为代表值。就像卷积操作一样，池化操作也是滑动的，通常池化窗口的大小和步长会设置为相同的值，这样可以避免窗口之间的重叠。

池化层的引入可以帮助提取更高层次的特征，比如不仅可以检测到边缘和纹理，还可以检测到物体的部分和整体等。它可以在保持图像特征的同时，大大减少后续层需要处理的数据量。

3.2.4 算法

计算时仍采用与全连接网络类似的反向传播，得到 CNN 的反向传播公式：

$$\frac{\partial L}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{z}^{(l-1)} * \frac{\partial L}{\partial \boldsymbol{a}^{(l)}}$$

$$\frac{\partial L}{\partial \boldsymbol{a}^{(l)}} = upsample\left(\frac{\partial L}{\partial \boldsymbol{a}^{(l+1)}} * \text{rot180}(\boldsymbol{W}^{(l+1)})\right) \odot h'\left(\boldsymbol{a}^{(l)}\right)$$

3.2.5 适用范围

不止图像，任何具有空间相关性的 2D 和 3D 数据如图片、文字、声音等。

如果实例的每部分没有相关性，则效果下降如一个人的姓名、地址、邮箱、爱好等。

3.3 循环神经网络

3.3.1 简介与基本概念

循环神经网络（Recurrent Neural Network，RNN）是一类用于处理序列数据的神经网络，例如时间序列数据、文本、语音等。RNN 的主要特点是它具有一种“记忆”机制，为每个神经元配一个存储模块，记录从其他部分学习到的信息。因此能够将前面的信息传递到后面，这使得它在处理序列数据时能够考虑到数据之间的时序关系。

RNN 具有权值共享的特性，即每个单元的权值 w 采用同一套，能够降低复杂度，同时可以应对不同实例中各种长度的输入和输出数据。

3.3.2 损失函数

对于信息提取类的任务：交叉熵，把每一个输出的损失加起来。

对于感情识别任务：整个 RNN 输出一个结果，单向损失。

3.3.3 缺陷与解决方法

当序列过长时，容易出现梯度消失现象。因为参数更新只能捕捉到局部的依赖关系，无法捕捉到序列之间的长期关联。

解决方法：

LSTM：设计一个记忆细胞，具备选择性记忆的功能，只记重要的，减轻记忆负担。LSTM 通过引入各种门，确保至少有一条通路不为 0。

§4 无监督学习

无监督学习：训练样本的标签信息未知，目标是通过对无标签样本的学习来揭示数据的内在性质及规律。

4.1 聚类

4.1.1 基本概念与基本思想

聚类：将数据集中的样本划分为若干个不相交子集，每个子集为一个簇。其指标有簇内相似度和簇间相似度等。

基本思想：对于给定的类别数量 K，首先给出初始划分，通过迭代改变样本和簇的隶属关系，使得每一次改进之后的划分方案都比前一次好。

4.1.2 性能度量

聚类算法有如下的性能度量：

簇中心

$$\operatorname{avg}(C)=\frac{2}{|C|(|C|-1)}\sum_{1\le i\le j\le |C|}\operatorname{dist}(\boldsymbol{x_i},\boldsymbol{x_j})$$

簇内最远距离

$$\operatorname{diam}(C)=\max_{1\le i\le j\le |C|}\operatorname{dist}(\boldsymbol{x_i},\boldsymbol{x_j})$$

簇间最近距离

$$d_{min}(C_i,C_j)=\min_{\boldsymbol{x_i}\in C_i,\boldsymbol{x_j}\in C_j}\operatorname{dist}(\boldsymbol{x_i},\boldsymbol{x_j})$$

簇中心之间的距离

$$d_{cen}(C_i,C_j)=\operatorname{dist}(\boldsymbol{\mu_i},\boldsymbol{\mu_j})$$

4.1.3 算法实现

4.1.3.1 K-means

K-means 是一种常见的无监督学习算法，用于数据的聚类分析。其基本思想是：通过迭代的方式，将样本划分到 K 个集合中，使得每个样本都属于距离其最近的均值（centroid）对应的集合，以此来最小化每个集合内部样本与均值之间的距离。

K-means 聚类算法的**主要步骤**如下：

1. 初始化：首先，选择 K 个样本点作为初始的聚类中心。
2. 分配样本到最近的聚类中心：对每个样本，计算它与 K 个聚类中心的距离，然后将其分配到距离最近的聚类中心对应的集合。
3. 更新聚类中心：然后，根据新的集合，重新计算每个集合的聚类中心（每个集合中所有样本的平均值）。
4. 迭代：重复上述的“分配-更新”过程，直到聚类中心不再显著改变，或者达到预设的最大迭代次数。

存在问题：

若簇中含有异常点，则将导致均值偏离严重；当数据量非常大时，运行速度较慢。

K 值选取：

求轮廓系数

$$s = \frac{b-a}{max(a,b)}$$

如果每个簇中的样本轮廓系数都很大，则说明聚类效果好，K 取的好。

初始簇心选取：

原始 K-means 问题：簇中心是随机选的。

改进：只有第一个簇中心是随机选的；假设已经选择了 n 个簇中心，则距离当前 n 个簇中心越远的点会有更高的概率被选为第 n+1 个聚类中心。

应用：降维：通过 K-means 聚类得到那些相似的点，然后在不减少样本点个数和特征数量的情况下，减少每个样本携带的信息。

4.1.3.2 高斯混合聚类 (含 EM 算法)

高斯混合模型（Gaussian Mixture Models，GMM）是一种广泛使用的概率模型，主要用于聚类任务，也可用于密度估计。与 K-means 聚类相比，GMM 可以看作是 K-means 的扩展，能够处理更复杂的情况，如非球形的集合，并且每个数据点的类别不再是硬分配，而是用概率（软分配）表示。

GMM 的主要思想是假设数据是由多个高斯分布的混合生成的。每个高斯分布对应一个类别，每个类别有自己的均值和协方差矩阵。

GMM 的实现主要采用期望最大化（Expectation-Maximization，EM）算法，以下是具体步骤：

1. **初始化**：选择 K 个高斯分布（即 K 个类别）。这些高斯分布的均值和协方差矩阵可以随机初始化，或者使用一些更复杂的初始化方法，如 K-means 聚类的结果。每个高斯分布的权重（即每个类别的先验概率）可以设为均等，或者按照数据点的初始类别分布来设定。

2. **E 步骤**：计算每个数据点属于每个高斯分布的后验概率。对于第 i 个数据点和第 k 个高斯分布，其后验概率计算公式为：

$$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i; \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i; \mu_j, \Sigma_j)}$$

其中，$\mathcal{N}(x_i; \mu_k, \Sigma_k)$ 是高斯分布的密度函数，π_k 是第 k 个高斯分布的权重。

3. **M 步骤**：根据 E 步骤计算的后验概率来更新高斯分布的参数。具体来说，对于第 k 个高斯分布，其参数的更新公式为：

- 权重更新公式：

$$\pi_k = \frac{1}{N} \sum_{i=1}^{N} \gamma_{ik}$$

- 均值更新公式：

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i$$

- 协方差矩阵更新公式：

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T$$

其中，N 是数据点的总数，N_k 是第 k 个高斯分布的有效数据点数，即所有数据点属于第 k 个高斯分布的后验概率的和。

4. **迭代**：重复 E 步骤和 M 步骤，直到高斯分布的参数不再显著变化，或者达到预设的最大迭代次数。

5. **聚类**：根据每个数据点属于每个高斯分布的后验概率进行聚类。具体来说，每个数据点被分配到后验概率最大的高斯分布对应的类别。

4.1.3.3 K 近邻聚类算法

面对一个新的数据点 $\left(x_{new}^{(1)}, x_{new}^{(2)}\right)$，判断其应该归在第 0 类还是第 1 类。然后选择离新数据点最近的 K 个点，这个“K”可以自己定。看这 K 个点中哪一类的点多，说明新的点离哪一类近。这种计算方法称为 K-近邻（KNN）算法。

K 越大，分界线倾向于越平滑，但是 太大会增加不必要的计算量。

KNN 误差来源：当不同簇的点非常接近（甚至交叠）时，聚类可能不准。

4.1.3.4 密度聚类算法

密度聚类是一种基于数据密度的聚类方法，其中类别的形成主要依赖于数据的分布密度。这类方法的一个显著特征是可以找出任意形状的聚类，而不仅仅是球形或凸形的聚类。DBSCAN 是一种常用的密度聚类算法。

动机：对密度进行量化，样本点密度较高（即样本点大量聚集）的地方设为簇中心。

4.1.3.5 DBSCAN 与 K-Means 的对比

K-means：倾向于把数据点聚成团状/球状，适用于凸集；对于非凸的数据集，效果不佳。

DBSCAN：基于密度进行聚类，对数据集的凸性没有要求，而且不需要输入 K；但是受参数影响大，需要进行参数选取；在聚类的同时，发现异常点。但是密度差别大时，性能不好。

4.1.3.6 谱聚类

主要思路：把所有的数据看做空间中的点，点之间可以用边连接起来。距离较远的点之间的边权重值低，而距离较近的点之间的边权重值高。通过对所有数据点组成的图进行切图，子图间边权重和尽可能的低，而子图内的边权重和尽可能的高，从而达到聚类的目的。

4.1.4 算法总结

算法	优点	缺点
K-Means	速度快，简单，性能稳定	受参数影响大，只适用于球状簇结构，对异常点敏感
DBSCAN	不需要设定簇的数量，可以划分非凸簇，能够找到异常点	当不同簇的密度差异较大时，聚类效果不佳
高斯混合	适用于类之间存在相关性、类大小不一的情况	迭代计算复杂
谱聚类	适用于类比较少的情况，对点的空间特征没有要求	需要提前指定K，类比较多的时候效果不好

4.2 降维

4.2.1 矩阵的特征分解

设 $\boldsymbol{A}$ 为 n 阶矩阵，若存在常数 λ 及 n 维非零向量 $\boldsymbol{v}$，使得 $\boldsymbol{A}\boldsymbol{v}=\lambda\boldsymbol{v}$，则称 λ 是 $\boldsymbol{A}$ 的特征值，$\boldsymbol{v}$ 是 $\boldsymbol{A}$ 属于特征值 λ 的特征向量。假设 $\boldsymbol{A}$ 有 n 个不同的特征值，即

$$\boldsymbol{A}\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1$$
$$\boldsymbol{A}\boldsymbol{v}_2 = \lambda_2 \boldsymbol{v}_2$$
$$\cdots$$
$$\boldsymbol{A}\boldsymbol{v}_n = \lambda_n \boldsymbol{v}_n$$

令 $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n], \boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_n]^\top$，则有

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{V} &= [\boldsymbol{A}\boldsymbol{v}_1, \boldsymbol{A}\boldsymbol{v}_2, ...\boldsymbol{A}\boldsymbol{v}_n] \\
&= [\lambda_1 \boldsymbol{v}_1, \lambda_2 \boldsymbol{v}_2, \cdots \lambda_n \boldsymbol{v}_n] \\
&= [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots\cdots, \boldsymbol{v}_n] \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix}
\end{aligned}$$

因此 $\boldsymbol{A} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots\cdots, \boldsymbol{v}_n]\text{diag}(\boldsymbol{\lambda}) \cdot \boldsymbol{V}^{-1} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})V^{-1}$

由于 $\boldsymbol{A}$ 的 n 个特征值不同，因此对应的特征向量是线性不相关的，即 $\boldsymbol{V} = [\boldsymbol{v_1}, \boldsymbol{v_2}, \cdots, \boldsymbol{v_n}]$ 是一个正交矩阵。因此 $\boldsymbol{V}\boldsymbol{V}^\top = \boldsymbol{I} \Rightarrow \boldsymbol{V}^{-1} = \boldsymbol{V}^\top$。因此方阵 $\boldsymbol{A}$ 的特征分解可以表示为

$$\boldsymbol{A} = \boldsymbol{V}\text{diag}(\lambda)\boldsymbol{V}^\top$$

特征值分解就是要得到这个矩阵最主要的变化方向，特征值表示的是这个特征到底有多重要，而特征向量表示这个特征是什么。

4.2.2 MDS

多维缩放（Multi-Dimensional Scaling，MDS）是一种用于降维的技术，常被用于探索高维数据的结构。MDS 旨在将高维度的数据降低到二维或三维以进行可视化。

MDS 通过尝试保留原始数据集中对象之间的距离来降低数据的维度。它通过计算数据点之间的距离来生成一个距离矩阵，然后使用这个距离矩阵来在低维空间中定位每个数据点。其中，常见的距离度量包括欧氏距离、曼哈顿距离、马氏距离等。

具体步骤如下：

1. 计算高维数据中每对数据点之间的距离，生成距离矩阵。

2. 使用 MDS 算法将距离矩阵映射到低维空间。MDS 的目标是找到一组低维坐标，使得这些坐标在低维空间中的距离尽可能接近原始的高维距离。

最常见的 MDS 算法是经典的 MDS，其主要步骤是：首先，通过计算每个数据点到所有其他数据点的平均距离来中心化距离矩阵；然后，对中心化后的距离矩阵进行特征分解；最后，选择前 d 个最大的特征值对应的特征向量来构造低维坐标，其中 d 是目标维度。

3. 然后，你就可以将低维坐标用于可视化或者其他的机器学习任务。

MDS 的一个主要优点是可以处理非欧氏距离，例如曼哈顿距离或马氏距离。然而，它也有一些缺点，例如计算量大，因为需要计算所有数据点之间的距离；并且对噪声和异常值敏感。

4.2.3 PCA

主成分分析（Principal Component Analysis，PCA）是一种常用的数据降维技术，旨在通过线性变换将原始数据转化为一组各维度线性无关的表示，称为主成分，主成分按照其解释原始数据的方差大小排序。

给定一个含有 n 个 d 维数值向量的数据集，我们希望找到一个新的 k 维空间（k<d），将每个 d 维向量转化为 k 维向量，而且希望在某种意义上，这个 k 维向量“尽可能好”的表示原始数据。

PCA 算法步骤如下：

1. 中心化：首先对原始数据进行中心化处理，即每一维数据减去该维的均值：

$$x^{(i)} = x^{(i)} - \frac{1}{n}\sum_{j=1}^{n} x^{(j)}$$

2. 计算协方差矩阵：协方差矩阵 C 的元素是原始数据在各个维度上的协方差，用于表示数据在不同维度间的相关性。C 的元素 c_{ij} 计算公式为：

$$c_{ij} = \frac{1}{n}\sum_{k=1}^{n}(x_k^{(i)} - \mu_i)(x_k^{(j)} - \mu_j)$$

其中 μ_i 和 μ_j 为数据在第 i 和 j 维度的均值。

3. 计算协方差矩阵的特征值和特征向量：利用特征分解或奇异值分解等方法，求解协方差矩阵 C 的特征值和特征向量。

4. 选择主成分：根据需要保留的主成分个数或者需要保留的方差比例，选择最大的 k 个特征值对应的特征向量构成投影矩阵 W。

5. 生成降维后的数据：将原始数据通过投影矩阵 W 转换到新的 k 维空间，得到降维后的数据。

在 PCA 中，第一主成分就是数据方差最大的方向，第二主成分是与第一主成分正交且具有次大方差的方向，以此类推。PCA 的一个主要优点是降低数据的维度，使问题更易处理，同时也有助于去除噪声，并可能发现数据内在的结构。然而，PCA 假定数据的主要结构是线性的，对于存在非线性结构的数据，PCA 可能不是最好的降维方法。

对于存在非线性结构的数据，可以采用核主成分分析的方法来实现降维。引入核函数，再进行特征分解。

4.2.4 SVD

奇异值分解（Singular Value Decomposition，SVD）是一种在线性代数中常用的分解方法，可以将任意的 m×n 矩阵分解为三个简单的矩阵的乘积。对于任意实数矩阵 A，它的奇异值分解可以表示为：

$$A = U\Sigma V^T$$

其中：

- A 是需要分解的 m×n 矩阵。

- U 是一个 m×m 的正交矩阵，其列向量被称为左奇异向量，它们构成了 A 的列空间。

- Σ 是一个 m×n 的对角矩阵，其对角线上的元素称为 A 的奇异值，它们是 A^TA 或者 AA^T 的非零平方根。奇异值一般按照从大到小的顺序排列。

- V 是一个 n×n 的正交矩阵，其列向量被称为右奇异向量，它们构成了 A 的行空间。

奇异值分解的一个重要性质是：对于任意实数矩阵 A，其总是存在奇异值分解。这使得 SVD 成为一种强大的工具，可以被用于许多应用，包括最小二乘线性回归，信号处理和统计等。因此，在机器学习中，SVD 常被用于降维和特征抽取。

§5 强化学习

强化学习是机器学习的一种类型，它强调如何基于与环境的互动，通过试错的方式，学习和优化决策策略。它的目标是学习一个策略，使得通过这个策略做出的行为序列能够最大化预期的累积奖励。

5.1 基本概念

两大交互对象：智能体（Agent）与环境（Environment）。

智能体可以感知外界环境的状态 (State) 和反馈的奖励 (Reward) 并进行学习和决策。智能体的决策功能是指根据外界环境的状态来做出不同的动作 (Action), 而学习功能是指根据外界环境的奖励来调整策略。

环境 (Environment) 是智能体外部的所有事物，对智能体动作和状态做出响应，并反馈给智能体相应的奖励。

五大基本要素：

1. 状态 $s \in \mathcal{S}$: 对环境的描述，可以是离散的或连续的。如：当前的棋局；马里奥当前所处环境（一张图片）。

2. 动作 $a \in \mathcal{A}$: 对智能体行为的描述，可以是离散的或连续的。

3. 策略 $\pi(a|s)$：智能体根据环境状态 来决定下一步动作 a 的函数。

确定性策略:π: $\mathcal{S} \to \mathcal{A}$；随机性策略：$\pi(a|s) = p(a|s)$, 且 $\sum_{a \in S} p(a|s) = 1$

4. 状态转移概率 $p(s'|s,a)$：在智能体根据当前状态 s 做出动作 a 之后，环境在下一个时刻转变为状态 s' 的概率。

5. 即时奖励 $r(s,a,s')$：智能体根据当前状态 s 做出动作 a 之后转移到状态 s' ，环境反馈给智能体的奖励。

智能体与环境的交互可以看作离散的时间序列，是一个马尔科夫决策过程。

强化学习的目标是学习到一个策略 $\pi_\theta(a \mid s)$ 来最大化期望回报（expected return），即

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[G(\tau)] = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} \gamma^t r_{t+1} \right]$$

5.2 值函数

在强化学习中，值函数（Value Function）是一个非常重要的概念，它用来评估在某个状态下执行某个策略的长期效益。值函数分为两类：状态值函数（State Value Function）和动作值函数（Action Value Function）。

1. 状态值函数 $V(s)$: 状态值函数描述了在给定状态下，遵循某个策略 π 所能获得的期望总回报。如果我们定义奖励函数 R 为某个状态 s 和动作 a 在时间 t 带来的即时奖励，γ 为折扣因子，那么状态值函数可以表示为：

$$V^\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$$

其中 E 表示期望，即在所有可能的行为序列上取平均。

2. 动作值函数 $Q(s,a)$: 动作值函数描述了在给定状态下，选择某个动作并遵循某个策略 π 所能获得的期望总回报。如果我们同样定义奖励函数 R 和折扣因子 γ，那么动作值函数

可以表示为：

$$Q^{\pi}(s,a) = E_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s, A_t = a]$$

值函数的计算是强化学习中的关键部分，它可以通过迭代算法如贝尔曼方程进行更新。

值函数可看作对策略 π 的评估，因此可以根据值函数来优化策略，而寻找最优策略通过寻找最优价值函数完成。

最优策略（Optimal Policy）是指在任何给定状态下都能最大化期望回报的策略。更正式地，策略 π' 被认为优于或等于策略 π，如果对所有的状态 s，$V^{\pi'}(s) \geq V^{\pi}(s)$。如果策略 π^* 对所有状态和所有策略 π 都满足 $V^{\pi^*}(s) \geq V^{\pi}(s)$，则策略 π^* 被称为最优策略。值得注意的是，可能存在多个最优策略，但它们都有同样的状态值函数和动作值函数。

最优状态值函数 $V^*(s)$ 是在所有策略中，对于给定状态 s，可以获得的最大期望回报：

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

同样地，**最优动作值函数** $Q^*(s,a)$ 是在所有策略中，对于给定状态 s 和动作 a，可以获得的最大期望回报：

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

最优值函数满足贝尔曼最优性方程，这个方程描述了一个事实：如果我们知道了后续状态的最优值，那么当前状态的最优策略就应该选择能够最大化当前即时奖励加上折扣后的后续状态最优值的动作。

对于**最优状态值函数**，贝尔曼最优性方程如下：

$$V^*(s) = \max_a E[R_{t+1} + \gamma V^*(S_{t+1})|S_t = s, A_t = a]$$

对于**最优动作值函数**，贝尔曼最优性方程如下：

$$Q^*(s,a) = E[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a')|S_t = s, A_t = a]$$

在实际应用中，强化学习的目标通常是找到最优策略和最优值函数。

5.3 蒙特卡洛法

蒙特卡洛（Monte Carlo）方法是一种用于估计值函数的统计方法，它通过采样和平均回报来计算值函数的估计值。

蒙特卡洛方法有以下几个步骤：

1. 策略初始化：首先，我们需要一个策略 π，这个策略可以是任意的，比如随机策略。
2. 执行试验：然后，我们在环境中执行多次试验。在每次试验中，我们根据策略 π 选择动作，直到试验结束（例如，到达终止状态）。在每次试验中，我们记录下每一步的状态、动作和奖励。
3. 计算回报：对于每一次试验，我们都可以计算每一步的回报，即从那一步开始，未来所有步的奖励的折扣和。
4. 估计值函数：最后，我们对每一步的状态（对于状态值函数）或状态-动作对（对于动作值函数），计算所有试验中那一步的回报的平均值，作为值函数的估计值。

潜在问题：对于确定性策略，每次得到的轨迹都是一样的，没有采样平均的意义。

解决办法：采用 ϵ 贪心法，即从某状态出发，执行策略 π ，以 ϵ 的概率从所有动作中随机选取一个，以 $1-\epsilon$ 的概率选择最优动作。

蒙特卡洛算法的局限：需要所有的采样序列都是完整的状态序列；执行完整个采样轨迹后才进行值函数更新，没有利用 MDP 的特性。

5.4 时序差分学习

时序差分学习（Temporal Difference Learning，简称 TD 学习）是一种强化学习方法，它结合了蒙特卡洛（Monte Carlo）方法和动态规划（Dynamic Programming）的优点。它可以从不完整的序列中学习，同时也无需知道环境的完全信息。其主要思想是利用当前估计的值函数来更新自身。

5.4.1 SARSA 算法

SARSA 是一种时序差分（TD）学习算法，用于解决强化学习问题。SARSA 是"State-Action-Reward-State-Action" 的缩写，反映了在算法中用于更新动作值函数（Q 函数）的五个关键元素。

在 SARSA 中，智能体在每个时刻 t 根据当前策略（通常是基于 ϵ-贪婪的策略）选择动作 a_t，然后观察奖励 r_{t+1} 和下一个状态 s_{t+1}，在 s_{t+1} 中再次根据当前策略选择动作 a_{t+1}。这样，就有了一组五元组（$s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$），用于更新动作值函数 $Q(s, a)$。

SARSA 的更新规则如下：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

其中，α 是学习率，控制着学习的速度；γ 是折扣因子，决定了未来奖励的重要性。

SARSA 是一种在线型（on-policy）学习算法，即在学习和更新过程中，智能体始终遵循同一策略。

5.4.2 Q-Learning 算法

Q-learning 的核心思想是利用" 贝尔曼最优等式" 对 Q 函数进行迭代更新。该等式基于一个观察，即最优策略下的 Q 函数值等于执行某个动作并从那以后遵循最优策略的预期回报。

更新公式如下：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

其中，α 是学习率，决定了学习的速度；γ 是折扣因子，确定了未来奖励的重要性。公式中的 r_{t+1} 是从状态 s_t 采取动作 a_t 后得到的即时奖励，$\max_a Q(s_{t+1}, a)$ 是对下一状态 s_{t+1} 所有可能动作的 Q 值取最大，代表在下一状态中选择最优动作的 Q 值。

不同于 SARSA 的是，Q-learning 是一种离线策略（off-policy）算法，也就是说在学习过程中，智能体并不总是遵循当前最优的策略行动。在更新 Q 值时，它会假设在下一个状态中总能选择能使 Q 值最大的动作，而不是根据当前策略选择动作。这使得 Q-learning 能够学习到最优策略，即使智能体并没有始终执行最优策略。

5.4.3 深度 Q 网络

深度 Q 网络（Deep Q-Network，简称 DQN）是一种结合了深度学习和强化学习的方法，特别是结合了深度神经网络和 Q-Learning 算法。

在传统的 Q-Learning 中，我们维护一个表格来记录每一个状态-动作对的 Q 值。然而，这种方法在状态空间或动作空间非常大甚至连续的情况下，将会面临" 维度诅咒" 的问题。此时，表格方法将会不切实际，因为无法为每一个状态-动作对都存储一个值。

深度 Q 网络就是为了解决这个问题，它使用深度神经网络作为函数逼近器，用来逼近真实的 Q 函数。网络的输入是状态（或观察），输出是每个动作的 Q 值。即 $Q(s, a; \theta) \approx Q^*(s, a)$，其中 θ 是神经网络的参数，$Q^*(s, a)$ 是真实的 Q 值。

DQN 的训练过程中采用了两个关键的技巧：

1. **经验回放（Experience Replay）**：为了打破数据之间的相关性并提高数据利用率，DQN 会将每个时间步的经验（状态，动作，奖励，下一个状态）存储在一个数据缓存（称为经验回放池）中。在训练过程中，DQN 会随机从经验回放池中抽取一批经验进行学习。

2. **目标网络（Target Network）**：为了保证训练的稳定性，DQN 使用了两个完全一样，但参数不同步的网络：一个是行动网络，用于选择动作；另一个是目标网络，用于计算目标 Q 值。在训练过程中，目标网络的参数会定期被行动网络的参数更新。

通过上述技巧，DQN 显著提升了深度强化学习的稳定性和效率，从而在一些复杂的任务中取得了显著的性能。然而，DQN 仍然存在一些局限性，例如可能过度估计 Q 值，对于有大量动作的环境可能效果不佳。

5.5 基于策略函数的强化学习

值函数方法的缺陷:

1. 以 DQN 为例，核心是更新 Q 函数，比较适合处理状态空间连续取值的情况，对于动作空间连续的情况不适用。

2. 值函数方法输出的是确定性策略，但有些问题的最优策略是随机策略（博弈）。

策略网络:

在某些强化学习算法中，特别是策略梯度方法中，策略是由一个参数化的函数，通常是一个神经网络来表示的，我们称之为策略网络。

策略网络的输入是环境状态，输出是每个可能动作的概率分布。网络的参数通过优化一个目标函数（如期望回报）来进行学习。优化通常采用梯度上升方法，因为我们希望最大化目标函数。策略梯度算法正是基于这样的思想，通过计算目标函数关于策略参数的梯度，然后沿梯度方向更新策略参数，以达到提高期望回报的目的。

与值函数方法（如 Q-learning）相比，策略网络方法直接对策略进行优化，无需维护一个值函数。因此，策略网络方法能更自然地处理连续动作空间和随机策略，这在许多任务中是非常重要的。

5.5.1 REINFORCE 算法

REINFORCE 算法是一种基于策略梯度的强化学习算法。它直接优化智能体的策略，而不是通过估计值函数（如在 Q-learning 中）来寻找最优策略。

REINFORCE 算法的基本思想是：如果在某一序列中，智能体采取了某一动作并得到了好的结果（也就是高的回报），那么我们应该鼓励智能体在未来的序列中更可能地采取这个动作。这就是" 策略梯度定理" 的基本思想。

REINFORCE 算法的具体步骤如下：

1. 初始化策略参数 θ，通常策略是用神经网络表示的，那么 θ 就是神经网络的权重。

2. 通过执行当前的策略，采集一系列的经验（状态，动作，奖励）。

3. 对每个时间步 t，计算回报 G_t，它等于从时间步 t 开始到序列结束的所有未来奖励的折扣和。

4. 计算策略梯度并更新策略参数：$\theta = \theta + \alpha * G_t * \nabla_\theta \log \pi(A_t|S_t, \theta)$，这里的 α 是学习

率，$\nabla_\theta \log \pi(A_t|S_t,\theta)$ 是策略的梯度，$\pi(A_t|S_t,\theta)$ 表示在状态 S_t 下，按照当前策略采取动作 A_t 的概率。

5. 重复步骤 2-4 直到收敛。

REINFORCE 算法的一个主要缺点是方差较大，这会导致学习的不稳定性和低效率。在实践中，通常会引入一些技巧来降低方差，如引入 baseline（通常为值函数的估计），使用优势函数等。

5.5.2 Actor-Critic 算法与 A2C 算法

Actor-Critic 算法是一种强化学习算法，它结合了策略梯度方法和值函数方法的优点。具体来说，Actor-Critic 方法中有两个主要的组件：Actor 和 Critic。

1.**Actor**：Actor 是一个策略函数，负责根据当前环境状态选择一个动作。通常，Actor 会以一定的概率选择动作，这个概率由策略函数 $\pi_\theta(a|s)$ 决定，这里的 θ 是策略的参数。

2. **Critic**：Critic 是一个值函数，负责评估 Actor 选取的动作的好坏。它通常用于计算 TD 误差或者优势函数，这个误差或者优势函数可以用来更新 Actor 的策略。

Actor-Critic 方法的基本步骤如下：

1. 根据 Actor 的策略 $\pi_\theta(a|s)$ 选择一个动作 a。

2. 执行这个动作并观察环境的反馈，得到下一个状态 s' 和奖励 r。

3. 根据反馈和 Critic 的评估来计算 TD 误差或者优势函数。对于值函数 $V(s)$，TD 误差可以定义为：$\delta = r + \gamma V(s') - V(s)$，其中 γ 是折扣因子。

4. 使用 TD 误差或者优势函数来更新 Actor 的策略。策略的更新规则可以表示为：$\theta = \theta + \alpha\delta\nabla_\theta \log \pi_\theta(a|s)$，其中 α 是学习率。

5. 使用 TD 误差来更新 Critic 的值函数。值函数的更新规则可以表示为：$V(s) = V(s) + \alpha\delta$。

6. 重复上述步骤直到收敛。

Actor-Critic 方法的优点在于，它结合了策略梯度方法直接优化策略的优点，以及值函数方法估计状态或状态-动作值的优点。它也可以有效地处理连续状态和动作空间。

但是，Actor-Critic 方法也存在一些问题。例如，由于 Actor 和 Critic 的更新是相互影

响的，所以这种方法的稳定性和收敛性是一个需要注意的问题。此外，如何选择合适的 Critic 也是一个重要的问题，因为 Critic 的质量直接影响到 Actor 的性能。

Advantage Actor-Critic (A2C) 是一种常见的 Actor-Critic 算法的变种。A2C 的主要想法是使用" 优势函数"（Advantage Function）来代替原来的 TD 误差或奖励信号，作为 Actor 的策略更新依据。

优势函数是在 Q 值（动作-状态值函数）与 V 值（状态值函数）之间的差值，其定义为：

$$A(s,a) = Q(s,a) - V(s)$$

这个优势函数的意义在于衡量采取某个动作 a 比按照当前策略的平均性能（即 V 值）好多少，如果优势函数为正，那么就说明这个动作比平均水平更好，如果为负，那么就说明这个动作比平均水平差。

然后，A2C 的更新规则就可以改写为：

1. 策略更新：$\theta = \theta + \alpha A(s,a) \nabla_\theta \log \pi_\theta(a|s)$

2. 值函数更新：$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$

其中，α 是学习率，γ 是折扣因子。

A2C 算法中的优势函数可以有多种估计方法，一种常见的方法是用 TD 误差的样本估计来近似，也就是说，我们将 $r + \gamma V(s') - V(s)$ 作为优势函数的估计。

5.5.3 确定策略梯度（DPG）

确定性策略梯度 (Deterministic Policy Gradient，DPG) 是一种在连续动作空间中进行策略优化的强化学习算法。

DPG 算法的学习过程可以分为以下几个步骤：

1. 使用 Actor 的策略从环境中采样经验。

2. 使用采样的经验来更新 Critic 的值函数估计。

3. 通过最大化 Critic 的值函数估计来更新 Actor 的策略。

5.5.4 多智能体强化学习（MARL）

多智能体强化学习 (Multi-Agent Reinforcement Learning，MARL) 是强化学习的一种扩展，它考虑了一个环境中有多个智能体（agent）同时学习和行动的情况。在 MARL 中，每个智能体都要通过自己的策略来最大化自己的累积奖励。这些智能体可能是合作的，也可能是竞争的，或者是部分合作部分竞争的。因此多智能体 A2C 算法根据智能体之间是否合作也有所不同。

§6 总结

在《电子信息学中的机器学习》这门课程中，我们对机器学习的多个方面进行了全面而深入的学习。首先，我们学习了监督学习的多种模型和方法，如线性回归、逻辑回归、支持向量机、决策树、随机森林等。然后，我们学习了神经网络的结构与应用，如全连接神经网络、卷积神经网络和循环神经网络。随后，我们深入研究了无监督学习的主要算法，包括聚类和降维，如 K-means、高斯混合模型、主成分分析和奇异值分解等。

我们还介绍了强化学习的基础理论，包括马尔科夫决策过程（MDP）、值函数、策略、蒙特卡罗方法、时序差分学习以及 Q 学习等。我们还探讨了基于神经网络的函数逼近在强化学习中的应用，如深度 Q 网络（DQN）、策略梯度方法如 REINFORCE 以及 Actor-Critic 架构。在多智能体强化学习部分，我们介绍了一些主要的算法和挑战。

在整个学期的学习内容中，数学推导占了相当大的比例。但念及像我这样的同学数学基础较差，许多线代和高数的知识掌握的不好，老师花了一定的篇幅讲了数学基础，比如凸优化分析、梯度下降、卷积、马尔可夫过程等等，同时也尽可能的减少了数学推导，尽量让我们多应用。因此在这里十分感谢邸老师的精心准备与耐心指导，同时也希望这门课越来越好！