



Machine Learning

机器学习经典之作

作者：Mingkui Tan

组织：SAIL

更新：June 3, 2019

版本：0.1



Victory won't come to us unless we go to it. — M Moore

目录

1	线性回归	1
1.1	线性模型	1
1.2	损失函数	2
1.3	线性回归闭式解	5
1.4	岭回归	8
1.5	梯度下降法	12
1.6	梯度下降法有效性解释	13
1.7	拓展阅读	14
1.7.1	L1 正则化回归	14
1.7.2	多项式回归	15
1.7.3	损失函数的概率解释	17
1.7.4	广义线性模型	18
1.8	本章总结	20
2	RNN & LSTM	24
2.1	RNN	24
2.1.1	为什么需要循环神经网络	24
2.1.2	不同类型的循环神经网络	25
2.1.3	参数更新	29
2.1.4	长期记忆中的问题	34
2.2	LSTM	36
2.2.1	L 还是 S?	36
2.2.2	三门分立	37
2.2.3	内外有别	39
2.2.4	LSTM 循环单元中的数据流	40
2.2.5	LSTM-变体	41
2.3	章末总结	42
2.4	Advanced Topic	43
3	Attention Mechanism	45
3.1	为什么需要注意力机制?	45
3.2	从仿生角度认识注意力	45

3.3	注意力评分函数	46
3.4	硬性注意力机制 (Hard Attention Mechanism)	47
3.5	软性注意力机制 (Soft Attention Mechanism)	48
3.6	软硬结合: 局部注意力机制 (Local Attention Mechanism)	49
3.7	自注意力机制 (Self-attention Mechanism)	51
3.7.1	键值对注意力 (Key-Value Attention)	51
3.7.2	多头注意力 (Multi-head Attention)	52
3.7.3	Attention Is All You Need	53
3.8	章末总结	55
3.9	Advanced Topic	56
4	Elegant\LaTeX 系列模板介绍	58
4.1	ElegantBook 更新说明	58
5	ElegantBook 设置说明	59
5.1	编译方式	59
5.1.1	选择 PDF \LaTeX 编译	59
5.1.2	选择 Xe \LaTeX 编译	59
5.2	语言模式	60
5.3	颜色主题	60
5.4	章标题显示风格	61
5.5	数学环境简介	61
5.5.1	定理类环境的使用	61
5.5.2	其他数学环境的使用	62
5.6	封面和徽标	62
5.7	列表环境	63
5.8	参考文献	63
5.9	添加序章	63
6	ElegantBook 写作示例	64
6.1	Lebesgue 积分	64
6.1.1	积分的定义	64
A	基本数学工具	67
A.1	求和算子与描述统计量	67
B	最小示例	68



第 1 章 线性回归

1.1 线性模型

坐在电脑前的小明，眼神凝重，握着鼠标的手来回移动，似乎在犹豫着什么。原来小明在考虑究竟要不要选机器学习这门课，毕竟近些年机器学习、人工智能的热度挺火的。富有好奇心的小明当然也有着一窥机器学习的想法。但是小明本身也成绩平平，要是最后没有通过机器学习的考核，或者这门课分数很低，这些都不是小明想要的。这该怎么办呢？要是能预测一下自己的机器学习成绩就好了。突然小明想到中学时期学过这样的一次函数

$$f(x) = kx + b \quad (1.1)$$

其中 k 是这个线性函数的斜率， b 为截距，也就是偏置项

也就是说，某数学函数或数量关系的函数图形呈现一条直线或线段，这样的关系就是一个线性关系。同时，这条公式也能代表一个模型，只有一个自变量的线性模型。但是，这个只有一个自变量的线性模型过于简单了，小明需要通过几门课程的成绩预测自己的机器学习成绩，这将如何改进呢？当我们需要表示多个自变量对一个因变量的关系时，我们需要更一般的线性模型。即

$$\mathbf{x} = (x_1, x_2, \dots, x_d)^T \quad (1.2)$$

$$\mathbf{w} = (w_1, w_2, \dots, w_d)^T \quad (1.3)$$

$$f(\mathbf{x}) = w_1x_1 + w_2x_2 + \dots + w_dx_d + b \quad (1.4)$$

$$\begin{aligned} &= \sum_{i=1}^d w_i x_i + b \\ &= \mathbf{w}^T \mathbf{x} + b \end{aligned}$$

其中 \mathbf{x} 是指包含 d 个自变量的列向量，我们称之为特征向量。关于 \mathbf{w} ，我们不用斜率来命名，用更一般的权重来命名，可以表示每个自变量的重要程度，则 \mathbf{w} 是包含 d 个权重的列向量，我们称之为权重向量。 b 为偏置项。

为了使本章公式简洁以及易于推导，我们将线性模型简写成

$$\hat{\mathbf{x}} = (1, x_1, x_2, \dots, x_d)^T \quad (1.5)$$

$$\hat{\mathbf{w}} = (b, w_1, w_2, \dots, w_d)^T \quad (1.6)$$

$$f(\hat{\mathbf{x}}) = \hat{\mathbf{w}}^T \hat{\mathbf{x}} \quad (1.7)$$

其中 $\hat{\mathbf{w}}$ 是增广权重向量， $\hat{\mathbf{x}}$ 是增广特征向量。

在保证本章后续符号表示没有歧义的情况下，我们使用 \mathbf{w} 代表增广权重向量， \mathbf{x} 代表增广特征向量。那么线性模型最后写为

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \quad (1.8)$$

线性模型是机器学习中最简单的模型，但是其中也包含了机器学习的一些重要思想。而线性回归要做的是通过一个用于训练的包含 n 个样本的数据集 $\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^{n \times 1}$, $y_i \in \mathbb{R}$ ，试图学得一个线性模型

$$f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i \quad (1.9)$$

可以使得 $f(\mathbf{x}_i) \approx y_i$ 。那如果我们需要用这个线性模型帮助小明预测他的机器学习成绩，首先就需要收集以前师兄师姐的各门课程成绩作为 \mathbf{x}_i ，对应的机器学习成绩作为 y_i 构造训练集，通过这些数据学得一个可以预测机器学习成绩的线性模型。那么首先要做的就是找到合适的 \mathbf{w} 使得线性模型尽可能预测准训练集中各个师兄师姐的机器学习成绩，换句话说就是让预测值与真实值的误差尽可能小。只有这样，预测小明的机器学习成绩才可能比较准确。如何寻找合适的 \mathbf{w} ，使得的训练集中每一个样本 \mathbf{x}_i ，经过线性模型的映射之后得到的预测值 $f(\mathbf{x}_i)$ ，与真实值 y_i 的差异都尽可能小呢？这是一个最优化问题，那么首先就需要一个函数可以衡量所有预测值与真实值的差异的总和，即损失函数。

1.2 损失函数

关于损失函数，我们可以很天然地想到使用差值函数作为描述两个值之间的差异，即该损失函数如下

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i) \quad (1.10)$$

但是在线性回归中，大量的样本通过线性模型的映射后，得到的预测值可能大于对应的真实值，也可能小于对应的真实值。如果使用差值函数作为损失函数，

对每个预测值与真实值的差值求和后，正负值抵消，无法衡量整体预测值与真实值的差异。

于是，我们很自然地进一步改进为对每个预测值与真实值的差值取绝对值后求和，那么这个绝对值损失函数表示为

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n |y_i - \mathbf{w}^T \mathbf{x}_i| \quad (1.11)$$

但是，在线性回归中，绝对值损失函数并不算是一个上佳的损失函数，因为寻找合适的 \mathbf{w} 需要损失函数最优化，即最小化。而绝对值损失函数存在不可导点，必然会对优化过程存在影响。那么我们也容易想到另一个损失函数——平方损失函数，即

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \quad (1.12)$$

平方损失函数有着很好的几何意义，对应了我们常用的距离，即欧几里得距离。所以在线性回归中，使用平方损失函数的意义就是希望找到一条直线，使得所有样本到直线上的欧几里得距离之和最小。况且，平方损失函数处处可导，且导函数的形式也较为简单。相比于四次方、六次方等高次方损失函数，平方损失函数的函数结构简单，便于计算。所以对于线性回归来说，这是一个上佳的损失函数。

在不影响平方损失函数的使用效果的情况下，为了方便平方损失函数后续的求导化简，我们把平方损失函数写成如下形式

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \quad (1.13)$$

为了方便本章后续的公式推导，我们需要把训练集中的每个样本中的 \mathbf{x}_i 综合起来作为整体设计成一个矩阵，每一个对应的 y_i 作为整体也要设计成一个向

量。

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \quad (1.14)$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad (1.15)$$

其中 $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$, $\mathbf{y} \in \mathbb{R}^{n \times 1}$, 因为 \mathbf{X} 中有 d 个特征元素和一个增广后的元素 1, 并且有 n 个样本, 所以 \mathbf{X} 为 n 行 $d+1$ 列, \mathbf{y} 为 n 行 1 列。

所以平方损失函数可以化成如下形式

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \\ &= \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \mathbf{w})^2 \\ &= \frac{1}{2} \begin{bmatrix} y_1 - \mathbf{x}_1^\top \mathbf{w} \\ \vdots \\ y_n - \mathbf{x}_n^\top \mathbf{w} \end{bmatrix}^\top \begin{bmatrix} y_1 - \mathbf{x}_1^\top \mathbf{w} \\ \vdots \\ y_n - \mathbf{x}_n^\top \mathbf{w} \end{bmatrix} \\ &= \frac{1}{2} \left(\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \mathbf{w} \right)^\top \left(\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \mathbf{w} \right) \\ &= \frac{1}{2} (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) \end{aligned} \quad (1.16)$$

如果读者读到这里, 对于损失函数的选取还存在疑问, 不妨阅读本章拓展阅读部分, 该部分会从概率的角度阐述为何选取平方损失函数作为线性回归的损失函数。而下一节我们会介绍在使用平方损失函数的情况下, 如何寻找 \mathbf{w} , 使得损失函数尽可能小, 从而求解线性回归。

1.3 线性回归闭式解

如何寻找合适的 \mathbf{w} ，使得损失函数 $\mathcal{L}(\mathbf{w})$ 最小化？根据最优化理论，可列如下等式

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad (1.17)$$

上式中 \mathbf{w}^* 表示最优的 \mathbf{w} 。

一般来说，优化平方损失函数 $\mathcal{L}(\mathbf{w})$ 需要知道 $\mathcal{L}(\mathbf{w})$ 对于 \mathbf{w} 的导数，这个对向量求导的导数是一个包含所有向量元素偏导数的向量，即梯度。

为了简化梯度计算，令

$$\mathbf{a} = \mathbf{y} - \mathbf{X}\mathbf{w} \quad (1.18)$$

那么 $\mathcal{L}(\mathbf{w})$ 对 \mathbf{w} 的梯度为

$$\begin{aligned} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \frac{\partial (\frac{1}{2} \mathbf{a}^T \mathbf{a})}{\partial \mathbf{a}} \\ &= \frac{1}{2} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \frac{\partial \mathbf{a}^T \mathbf{a}}{\partial \mathbf{a}} \\ &= \frac{1}{2} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} (2\mathbf{a}) \\ &= \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \mathbf{a} \\ &= \frac{\partial (\mathbf{y} - \mathbf{X}\mathbf{w})}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= -\frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= -\mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) \end{aligned} \quad (1.19)$$

上式中的 $\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$ 与 \mathbf{w} 的维度相同。

易知平方损失函数 $\mathcal{L}(\mathbf{w})$ 是凸二次函数，当其梯度为 $\mathbf{0}$ 时， $\mathcal{L}(\mathbf{w})$ 取得最小值。

$$\begin{aligned} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = \mathbf{0} \\ \Rightarrow \mathbf{X}^T \mathbf{X}\mathbf{w} &= \mathbf{X}^T \mathbf{y} \end{aligned} \quad (1.20)$$

上式中的 $\mathbf{0}$ 指的是与 $\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$ 维度相同的全零向量。

若 $\mathbf{X}^T\mathbf{X}$ 为可逆矩阵或满秩矩阵时，可以解得 \mathbf{w}^*

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \quad (1.21)$$

这就是线性回归的闭式解，利用这个 \mathbf{w}^* 就可以顺利地预测小明的机器学习成绩了，而这种方法也叫最小二乘估计（Least-squares Estimation），二乘即平方。

下面我们通过一个小算例来说明线性回归的闭式解，小明通过问询得到十位师兄师姐的 C++(上) 成绩、线性代数成绩和机器学习成绩，如表 1.1 所示。

表 1.1: 十位师兄师姐的成绩

C++(上)	线性代数	机器学习
78	66	77
70	93	86
61	71	60
73	66	69
79	81	70
93	95	88
74	77	72
90	85	88
66	64	70
81	90	91

如果利用前六位师兄师姐的成绩构造训练集去进行线性回归，则可令

$$\mathbf{X} = \begin{bmatrix} 1 & 78 & 66 \\ 1 & 70 & 93 \\ 1 & 61 & 71 \\ 1 & 73 & 66 \\ 1 & 79 & 81 \\ 1 & 93 & 95 \end{bmatrix} \quad (1.22)$$

$$\mathbf{y} = \begin{bmatrix} 77 \\ 86 \\ 60 \\ 69 \\ 70 \\ 88 \end{bmatrix} \quad (1.23)$$

根据公式1.21，此时可以得到 \mathbf{w}^* 为

$$\mathbf{w}^* = \begin{bmatrix} 7.74 \\ 0.42 \\ 0.45 \end{bmatrix} \quad (1.24)$$

在这个数据集中，我们可以看到线性代数成绩在模型中的权重是要比 C++(上) 成绩大的，即线性模型在这个数据集中认为线性代数成绩比 C++(上) 成绩重要。

于是， \mathbf{X} 经过这个线性模型的映射后，为

$$\mathbf{X}\mathbf{w}^* = \begin{bmatrix} 70.26 \\ 79.09 \\ 65.39 \\ 68.16 \\ 77.45 \\ 89.65 \end{bmatrix} \quad (1.25)$$

根据公式1.16，可得训练集误差为 $\mathcal{L}_{train} = 90.53$ 。尽管训练集误差 \mathcal{L}_{train} 已经降到最小了，由于样本还是比较少，以及线性模型本身的局限性， $\mathbf{X}\mathbf{w}^*$ 对 \mathbf{y} 的预测只能说差强人意吧。

剩下四位师兄师姐的成绩作为训练集，相关计算如下

$$\mathbf{X}_{test} = \begin{bmatrix} 1 & 74 & 77 \\ 1 & 90 & 85 \\ 1 & 66 & 64 \\ 1 & 81 & 90 \end{bmatrix} \quad (1.26)$$

$$\mathbf{y}_{test} = \begin{bmatrix} 72 \\ 88 \\ 70 \\ 91 \end{bmatrix} \quad (1.27)$$

$$\mathbf{X}_{test}\mathbf{w}^* = \begin{bmatrix} 73.55 \\ 83.87 \\ 64.32 \\ 82.35 \end{bmatrix} \quad (1.28)$$

$$\mathcal{L}_{test} = 63.21 \quad (1.29)$$

其中 \mathcal{L}_{test} 表示代表训练集误差。

而小明 C++(上) 成绩和线性代数成绩如表1.2所示，则通过该线性模型预测

他的机器学习成绩为 77.74。

表 1.2: 小明两门课程的成绩

C++ (上)	线性代数
84	77

但我们知道，在不少情况下 $\mathbf{X}^T\mathbf{X}$ 是不可逆的，比如 \mathbf{X} 的行数少于列数，即训练集的样本数过少。比如小明想通过六门课程的成绩预测机器学习的成績，但是只收集到三个师兄师姐的六门课程及机器学习成绩。这时，可能会存在多个 \mathbf{w}^* ，那么下面我们介绍岭回归来解决这个问题。

1.4 岭回归

从上一节我们知道，存在 $\mathbf{X}^T\mathbf{X}$ 不可逆的情况。或者就算 $\mathbf{X}^T\mathbf{X}$ 是可逆的，但存在极小的特征值， $\mathbf{X}^T\mathbf{X}$ 接近于奇异矩阵。这时，特征变量间存在较大的线性相关关系，使用传统的线性回归模型并不能很好地估计 \mathbf{w} ，对于不同的训练集估计方差会变得很大，这表现为存在很大的正权重系数项，然后被另一个同样大的与之相关的负权重系数项抵消，那么整个线性模型对输入变量中的噪音非常敏感。如果小明的各门成绩输入到如此敏感的线性模型去预测机器学习成绩时，可能得到的成绩会低于 0 分，或者高于 100 分，不符合实际。而如果我们去限制参数的大小，理所当然地会使线性模型对噪音的敏感度降低。这也就是岭回归（Ridge Regression）的基本思想。

岭回归实际上是一种改良的最小二乘估计法，通过损失模型部分的解释性、降低回归精度使得权重系数更为可靠及更符合实际。而岭回归的具体操作是让平方损失函数 $\mathcal{L}(\mathbf{w})$ 增加一个惩罚项，通过限制 \mathbf{w} 的增长，从而使线性模型对输入的噪音的敏感程度降低。公式如下

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{1}{2}\lambda\mathbf{w}^T\mathbf{w} \quad (1.30)$$

式中的 λ 是惩罚系数， $\lambda > 0$ ，其值越大，对 \mathbf{w} 的限制能力越强，而 $\frac{1}{2}\lambda$ 中的 $\frac{1}{2}$ 则是为了后续的求导化简。

同样地，为了简化梯度计算，令 $\mathbf{a} = \mathbf{y} - \mathbf{X}\mathbf{w}$ ， $\mathcal{L}(\mathbf{w})$ 对 \mathbf{w} 的梯度为

$$\begin{aligned}
 \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{2} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \frac{\partial \mathbf{a}^\top \mathbf{a}}{\partial \mathbf{a}} + \frac{1}{2} \lambda \frac{\partial \mathbf{w}^\top \mathbf{w}}{\partial \mathbf{w}} \\
 &= \frac{1}{2} \left(2 \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \mathbf{a} + 2\lambda \mathbf{w} \right) \\
 &= \frac{\partial (\mathbf{y} - \mathbf{X}\mathbf{w})}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w} \\
 &= -\frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w} \\
 &= -\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w} \\
 &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}
 \end{aligned} \tag{1.31}$$

让 $\mathcal{L}(\mathbf{w})$ 的梯度为 $\mathbf{0}$ ，则

$$\begin{aligned}
 \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w} = \mathbf{0} \\
 \Rightarrow (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} &= \mathbf{X}^\top \mathbf{y}
 \end{aligned} \tag{1.32}$$

式中的 \mathbf{I} 是单位矩阵，除了对角线的元素为 1，其余部分皆为 0，该式中的 \mathbf{I} 与 $\mathbf{X}^\top \mathbf{X}$ 维度相同。

岭回归可以解决 $\mathbf{X}^\top \mathbf{X}$ 不可逆的问题，因为 $\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}$ 必然是可逆矩阵，证明如下。

首先，根据线性代数的奇异值分解（Singular Value Decomposition），可以把 \mathbf{X} 分解成如下形式

$$\mathbf{X} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^\top \tag{1.33}$$

其中 $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$ ， $\mathbf{U} \in \mathbb{R}^{n \times n}$ 为正交矩阵， $\mathbf{V} \in \mathbb{R}^{(d+1) \times (d+1)}$ 为正交矩阵，而 $\mathbf{\Sigma} \in \mathbb{R}^{n \times (d+1)}$ 则是非负实数对角矩阵。

根据正交矩阵的性质，可化简 $\mathbf{X}^\top \mathbf{X}$ 为

$$\begin{aligned}
 \mathbf{X}^\top \mathbf{X} &= \mathbf{V} \mathbf{\Sigma}^\top \mathbf{U}^\top \mathbf{U} \mathbf{\Sigma} \mathbf{V}^\top \\
 &= \mathbf{V} \mathbf{\Sigma}^\top \mathbf{\Sigma} \mathbf{V}^\top
 \end{aligned} \tag{1.34}$$

那么

$$\begin{aligned}
 \mathbf{X}^T \mathbf{X} + \lambda \mathbf{I} &= \mathbf{V} \Sigma^T \Sigma \mathbf{V}^T + \lambda \mathbf{I} \\
 &= \mathbf{V} \Sigma^T \Sigma \mathbf{V}^T + \lambda \mathbf{V} \mathbf{V}^T \\
 &= (\mathbf{V} \Sigma^T \Sigma + \lambda \mathbf{V}) \mathbf{V}^T \\
 &= (\mathbf{V} \Sigma^T \Sigma + \lambda \mathbf{V} \mathbf{I}) \mathbf{V}^T \\
 &= \mathbf{V} (\Sigma^T \Sigma + \lambda \mathbf{I}) \mathbf{V}^T
 \end{aligned} \tag{1.35}$$

易知, 存在 $\lambda > 0$, 使得 $\Sigma^T \Sigma + \lambda \mathbf{I}$ 是对角线上皆为非 0 元素的对角矩阵, $\Sigma^T \Sigma + \lambda \mathbf{I}$ 可逆, 且其逆矩阵是对角元素皆为原对角元素倒数的对角矩阵。故 $\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}$ 为可逆矩阵。

所以, \mathbf{w} 在岭回归下的最优估计 \mathbf{w}^* 是

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \tag{1.36}$$

最后我们通过一个小算例, 结束本小节的内容。小明得到了六位师兄师姐的六门成绩及机器学习成绩, 如表 1.3 所示。

表 1.3: 六位师兄师姐的成绩

C++ (上)	C++(下)	数学分析 (一)	数学分析 (二)	线性代数	概率论	机器学习
78	89	68	62	66	73	77
70	77	87	95	93	77	86
61	64	60	62	71	71	60
73	56	49	66	66	68	69
79	81	73	74	81	51	70
93	85	100	100	95	97	88

那么利用前三位师兄师姐的成绩构造训练集去进行岭回归, 则可令

$$\mathbf{X} = \begin{bmatrix} 1 & 78 & 89 & 68 & 62 & 66 & 73 \\ 1 & 70 & 77 & 87 & 95 & 93 & 77 \\ 1 & 61 & 64 & 60 & 62 & 71 & 71 \end{bmatrix} \tag{1.37}$$

$$\mathbf{y} = \begin{bmatrix} 77 \\ 86 \\ 60 \end{bmatrix} \tag{1.38}$$

此时根据公式1.36，设 $\lambda = 0.1$ ，得到 \mathbf{w}^* 为

$$\mathbf{w}^* = \begin{bmatrix} -0.006 \\ 0.19 \\ 0.43 \\ 0.39 \\ 0.33 \\ -0.07 \\ -0.27 \end{bmatrix} \quad (1.39)$$

于是， \mathbf{X} 经过这个线性模型的映射后，为

$$\mathbf{X}\mathbf{w}^* = \begin{bmatrix} 76.998 \\ 85.998 \\ 60.005 \end{bmatrix} \quad (1.40)$$

根据公式1.30，可得训练集误差 $\mathcal{L}_{train} = 0.028$ 。利用后三个师兄师姐的成绩作为测试集，相关计算如下

$$\mathbf{X}_{test} = \begin{bmatrix} 1 & 73 & 56 & 49 & 66 & 66 & 68 \\ 1 & 79 & 81 & 73 & 74 & 81 & 51 \\ 1 & 93 & 85 & 100 & 100 & 95 & 97 \end{bmatrix} \quad (1.41)$$

$$\mathbf{y}_{test} = \begin{bmatrix} 69 \\ 70 \\ 88 \end{bmatrix} \quad (1.42)$$

$$\mathbf{X}_{test}\mathbf{w}^* = \begin{bmatrix} 56.98 \\ 84.58 \\ 95.15 \end{bmatrix} \quad (1.43)$$

$$\mathcal{L}_{test} = 204.11 \quad (1.44)$$

其中 \mathcal{L}_{test} 为测试集误差。

小明的六门成绩如表1.4所示，则通过该线性模型预测他的机器学习成绩为75.17。

表 1.4: 小明的六门成绩

C++（上）	C++(下)	数学分析（一）	数学分析（二）	线性代数	概率论
84	74	67	79	77	77

由于训练集的样本过少，样本没有进行归一化， λ 的惩罚力度不够，训练集

误差和测试集误差相差很大，这是很明显的过拟合现象，至于过拟合的详细解释，读者可以自行查阅后续章节进行学习。

1.5 梯度下降法

我们根据多元微积分的知识可以知道，某一标量场某点的方向导数的最大值是其梯度的模，当且仅当沿着梯度的方向取到，也就是说梯度的方向是该函数该点瞬时变化率最大的方向，即增长最快的方向。反过来，梯度的负方向也就是函数该点下降最快的方向。线性回归需要最小化 $\mathcal{L}(\mathbf{w})$ ，那我们需要得到 $\mathcal{L}(\mathbf{w})$ 每一点的梯度，然后用 \mathbf{w} 沿着梯度的负方向进行更新，从而实现了对 $\mathcal{L}(\mathbf{w})$ 的优化。而这就是梯度下降法的精髓所在。

为了直观理解梯度下降法，我们不妨把梯度下降过程理解成从山上某一点（初始点）下到山谷（局部最低点）的过程。一开始我们并不知道什么路径可以下山，只能观望到所站位置的周围，对周围环境进行足够地观察后，我们决定从最陡峭向下的方向（梯度的负方向）下山，步长（学习率）决定了我们下一刻的位置在哪。就这样走一步看一步，直到我们走到一个四处平坦（梯度为 0）的位置，我们就完成下到山谷的过程了。如果我们的步长过大的话，很可能直接从山坡的一边迈步到山坡的另一边，错过了山谷，然后又从山坡新的一边迈步回原来的一边，呈现“之”字形下山，也可能来回迈步无法到达山谷。但步长过小的话，下山又太慢了。

通过上面的理解我们不难看出，寻找一个合适的学习率很重要，学习率太小，整个梯度下降过程所需要的时间太长了，学习率太大可能梯度下降过程无法收敛。而梯度下降不一定能找到函数的最小值，而是找到一个局部最小值。若所需优化的函数是凸函数的话，恰好局部最小值就是全局最小值。下面，我们给出梯度下降法的具体运行过程

Algorithm 1 梯度下降法

Require:训练集 \mathcal{T} 学习率 η 迭代次数 t **Ensure:****for** $i = 1 \dots t$ **do** $\mathbf{w} := \mathbf{w} - \eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$ **end for****return** \mathbf{w}

回到线性回归，所幸平方损失函数是凸函数，这意味着通过合适的学习率 η ，



我们可以找到 \mathbf{w}^* ，使得 $\mathcal{L}(\mathbf{w})$ 取得最小值. 更新规则如下

$$\mathbf{w} := \mathbf{w} - \eta \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \quad (1.45)$$

上式的 $:=$ 代表赋值。

至此，梯度下降法介绍完毕。如果读者对梯度下降法的有效性还存疑的话，不妨继续阅读下一小节，这将会看到较为严谨的代数解释。对此没有兴趣的读者可以跳过下一小节，这并不影响后续章节的阅读。

1.6 梯度下降法有效性解释

因为 $\mathcal{L}(\mathbf{w})$ 是连续可微的，根据泰勒展开公式，当 $\Delta\mathbf{w} \rightarrow \mathbf{0}$ 时， $\mathcal{L}(\mathbf{w})$ 的一阶泰勒展开为

$$\mathcal{L}(\mathbf{w} + \Delta\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \Delta\mathbf{w}^\top \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \quad (1.46)$$

设学习率 $\eta > 0$ ，不妨令

$$\Delta\mathbf{w} = -\eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \quad (1.47)$$

易知，当 $\eta \rightarrow 0$ 时， $\Delta\mathbf{w} \rightarrow \mathbf{0}$ ，那么 $\mathcal{L}(\mathbf{w})$ 的一阶泰勒展开改写为

$$\mathcal{L}(\mathbf{w} + \Delta\mathbf{w}) = \mathcal{L}(\mathbf{w}) - \eta \left(\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \right)^\top \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \quad (1.48)$$

又知

$$\left(\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \right)^\top \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \geq 0 \quad (1.49)$$

故可以找到一个很小的正数 η ，使得

$$\mathcal{L}(\mathbf{w} + \Delta\mathbf{w}) \leq \mathcal{L}(\mathbf{w}) \quad (1.50)$$

$$\Rightarrow \mathcal{L}\left(\mathbf{w} - \eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}\right) \leq \mathcal{L}(\mathbf{w}) \quad (1.51)$$

这就比较严谨地解释了为什么梯度下降法是有效的，同时也指出了为什么学习率 η 过大会使梯度下降无法收敛到局部最优解。

1.7 拓展阅读

1.7.1 L1 正则化回归

L1 正则化回归 (L1 Regularization regression), 也称 Lasso 回归, 是最小绝对值收敛和选择算子 (Least absolute shrinkage and selection operator) 回归的简称。它在普通的线性回归模型上与岭回归类似地构造一个惩罚函数使得能够压缩权重系数。岭回归按不同比例压缩每个权重系数但始终保留每个权重系数, 而 L1 正则化回归在惩罚系数比较大的时候能够使部分权重系数精确地缩减为 0, 即不保留部分特征。而 L1 正则化回归的这种思想在数据时代的今天是很有意义的, 很多时候我们并不缺少数据, 反而是数据过多, 真正有用的数据比较少。在成百上千个变量中, 起决定作用的数据只有有限的几个。而 L1 正则化回归的具体操作类似岭回归, 对损失函数的增加了一个惩罚项, 公式如下。

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\|\mathbf{w}\|_1 \quad (1.52)$$

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{d+1} |w_i| \quad (1.53)$$

其中 λ 是惩罚系数, $\|\mathbf{w}\|_1$ 是 \mathbf{w} 的 L1 范数, 等于列向量 \mathbf{w} 的每个分量的绝对值之和。而这也揭示了何为 L1 正则化。同理, 岭回归也叫 L2 正则化回归。

下面, 我们略微介绍一下该损失函数推导。我们知道普通线性回归需要优化的问题如下

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) \quad (1.54)$$

而 L1 正则化回归对此优化问题改进成如下形式

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) \quad \text{s. t. } \|\mathbf{w}\|_1 \leq t \quad (1.55)$$

其中 t 是一个不依赖于 \mathbf{w} 的常数, $t > 0$ 。这是一个非线性的规划问题, 带有不等式约束。利用拉格朗日乘数法 (Lagrange multiplier) 及 KKT 条件 (Karush-Kuhn-Tucker Conditions), 我们可以把该优化问题等价成无约束形式, 如下

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\|\mathbf{w}\|_1 - \lambda t \quad (1.56)$$

其中 $\lambda > 0$, 和 t 一样不依赖于 \mathbf{w} 。末尾的常数项不影响优化过程, 可以去掉。故

优化问题最后化为

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \|\mathbf{w}\|_1 \quad (1.57)$$

这就略微解释了公式1.52的由来，而在岭回归部分为了讲述的连贯性，没有涉及到这部分内容。至于拉格朗日乘数法和 KKT 条件的数学过程略为高深，篇幅有限，就不予展示了，有兴趣的读者可以自行查阅。

L1 正则化回归的求解方法有多种，该部分不是本文重点，有兴趣的读者可以查阅参考文献。

1.7.2 多项式回归

多项式回归（Polynomial Regression），回归函数是特征变量多项式的回归。在现实生活中，很多变量与变量的关系不是线性的，例如在匀加速运动中，位移与时间的关系。而普通线性回归的局限性在于假设变量间是线性关系，即使尽可能去拟合，但对于本身是非线性关系的变量间，回归的效果并不好。根据微积分的知识，我们知道连续函数可以用多项式去逼近，故多项式回归还是有着较为广泛的应用。下面我们通过一元函数的拟合例子来解释多项式回归，假设因变量和自变量的数据如图1.1所示。

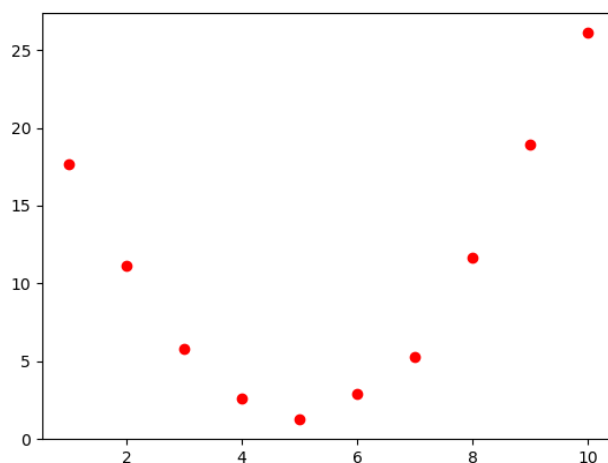


图 1.1: 因变量与自变量数据

如果我们使用，普通的线性回归对这些数据进行拟合，得到的拟合图像如图1.2所示。

我们直接的观察都可以很容易看出，自变量和因变量之间没什么线性关系，不妨先假设他们之间存在二次关系。即假设一个线性模型如下

$$f(x) = w_1 x + w_2 x^2 + b \quad (1.58)$$

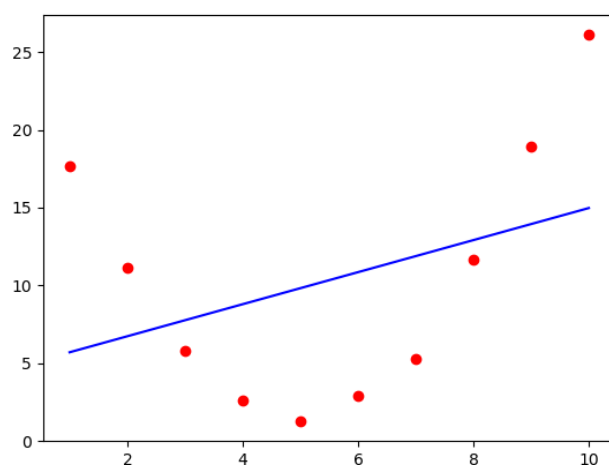


图 1.2: 普通线性回归拟合图像

而当我们把 x^2 当成另一个新的特征变量时，这本质上就可以看成是线性回归了。

$$\mathbf{x} = (1, x, x^2)^\top \quad (1.59)$$

$$\mathbf{w} = (b, w_1, w_2)^\top \quad (1.60)$$

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \quad (1.61)$$

如此这般，该多项式回归的拟合图像如图1.3所示。

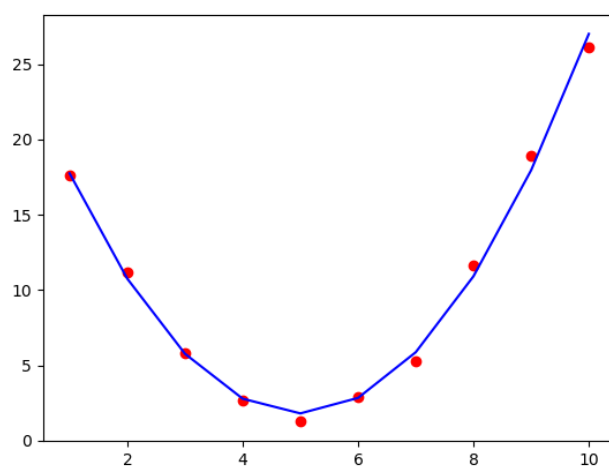


图 1.3: 多项式回归拟合图像

我们可以看到拟合的效果比起普通的线性回归好很多。那么对于多个特征变

量如何进行多项式回归呢，我们用二元二次多项式回归作为例子。

$$\mathbf{x} = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)^\top \quad (1.62)$$

$$\mathbf{w} = (b, w_1, w_2, w_3, w_4, w_5)^\top \quad (1.63)$$

$$\begin{aligned} f(\mathbf{x}) &= w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2 \\ &= \mathbf{w}^\top \mathbf{x} \end{aligned} \quad (1.64)$$

更高次项的多项式回归如此类推，但可以发现随着特征变量的个数、最高次项增加，“真实”的特征向量的数量急剧增加，计算量剧增。而且太高次的多项式回归会导致过拟合问题，详细的相关内容读者可自行查阅后续章节。

1.7.3 损失函数的概率解释

在本小节中，我们首先会给出一些概率的基本假设，然后基于这些假设推出选择平方损失函数作为线性回归的损失函数是一个很自然的事情。首先我们假设训练集 \mathcal{T} 每个样本和线性模型间存在这样的关系。

$$y_i = f(\mathbf{x}_i) + \epsilon_i = \mathbf{w}^\top \mathbf{x}_i + \epsilon_i \quad (1.65)$$

这里的 ϵ_i 是误差项，误差项存在是因为模型本身的缺陷（例如 \mathbf{x} 与 y 存在很明显的非线性关系），或者是噪音。

这个误差项应该是对结果有微小影响的众多独立随机变量叠加产生的，中心极限定理指出这种误差项近似服从正态分布。故假设该误差项是独立同分布的，且服从均值为 0，方差为 σ^2 的正态分布，即

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2) \quad (1.66)$$

那么这样就可以写出 ϵ_i 的概率密度函数为

$$p(\epsilon_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right) \quad (1.67)$$

因 $\epsilon_i = y_i - \mathbf{w}^\top \mathbf{x}_i$ ，假设 y_i 为一个随机变量，故

$$p(y_i|\mathbf{x}_i; \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2}\right) \quad (1.68)$$

这里的 $p(y_i|\mathbf{x}_i; \mathbf{w})$ 表示为对于给定输入为 \mathbf{x}_i 时输出正好等于 y_i 的条件概率密度函数，其中 \mathbf{w} 为该分布的参数。

我们知道线性回归的目标是让全部样本的预测值与真实值之间的差异尽可

能小。从概率的角度来说，就是在参数 \mathbf{w} 未知的情况下，给定所有样本的 \mathbf{x}_i ，让对应的 y_i 同时发生的概率最大，即概率积最大。而这个概率积就是 \mathbf{w} 在训练集 \mathcal{T} 下的似然函数

$$\mathbb{L}(\mathbf{w}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2}\right) \quad (1.69)$$

为了区分损失函数的符号 \mathcal{L} ，上式用了 \mathbb{L} 表示似然函数。

在得出似然函数的形式后，怎么样得到 \mathbf{w} 的最佳估计呢？这里就要用到最大似然估计，它告诉我们， \mathbf{w} 的最佳估计能让似然函数 $\mathbb{L}(\mathbf{w})$ 取得最大值。我们可以看到 $\mathbb{L}(\mathbf{w})$ 的函数结构比较复杂，且容易发生数值下溢。而我们对其取对数后，函数结构较为简单且单调性不变。所以我们可以用对数似然函数 $\ln \mathbb{L}(\mathbf{w})$ 来代替似然函数 $\mathbb{L}(\mathbf{w})$ 。

$$\begin{aligned} \ln \mathbb{L}(\mathbf{w}) &= \ln \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^n \ln \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2}\right) \right) \\ &= n \ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \end{aligned} \quad (1.70)$$

而我们对 \mathbf{w} 的估计不依赖于 σ ，故为了让 $\ln \mathbb{L}(\mathbf{w})$ 取得最大值，也就意味着要让 $\frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$ 取得最小值，而这也就是平方损失函数 $\mathcal{L}(\mathbf{w})$ 本身。所以，线性回归选取平方损失函数作为损失函数是一种很自然的做法。

1.7.4 广义线性模型

通过1.7.3小节的介绍我们可以知道，线性回归的线性模型本质等同于假设训练集 \mathcal{T} 中的每个样本都符合正态分布，即

$$y_i | \mathbf{x}_i; \mathbf{w} \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i, \sigma^2) \quad (1.71)$$

在不引起歧义的情况下，下面我们用 \mathbf{x}, y 泛指样本的特征向量和标签值。我们知道，正态分布只是一个假设，现实中很多事物的分布并不符合正态分布。并且标签值 $y_i \in (-\infty, \infty)$ ，难以扩展成其他形式，通过这样假设得到的线性模型，有较大的局限性。下面我们介绍广义线性模型，来降低局限性。在广义线性模型中，定义 y 符合指数族分布（Exponential family distributions），即

$$y | \mathbf{x}; \mathbf{w} \sim \text{ExponentialFamily}(\eta) \quad (1.72)$$

其中 ExponentialFamily 即是指指数族分布，泛指正态分布、泊松分布、指数分布、伯努利分布等分布。 η 是该分布的自然参数 (Natural parameter)，且广义线性模型假设 $\eta = \mathbf{w}^T \mathbf{x}$ ，说是假设，倒不如用设计二字描述更为准确。指数族的通用概率分布表示如下

$$p(y|\eta) = b(y) \exp(\eta T(y) - a(\eta)) \quad (1.73)$$

其中 $T(y)$ 叫做充分统计量 (Sufficient statistic)，通常情况下 $T(y) = y$ 。而 $a(\eta)$ 是对数分割函数 (Log partition function)。 $\exp(-a(\eta))$ 扮演了归一化常数 (Normalization constant) 的作用，保证 $p(y|\eta)$ 求和或者积分为 1。 $b(\cdot)$ 只是代表指数外与 y 相关的函数。

那么在我们知道指数族分布之后，如何构造具体的广义线性模型呢？我们知道构造线性模型是想构造 $f(\mathbf{x})$ 尽可能地去预测 y ，从概率的角度来说就是， $f(\mathbf{x})$ 即为 y 分布的期望。下面我们用正态分布作为例子，去构造广义线性模型，而这就是普通的线性模型。

$$y|\eta \sim \mathcal{N}(\mu, \sigma^2) \quad (1.74)$$

其中 μ 为均值，即期望。 σ^2 为方差。那么可以把该概率分布变换为通用的指数族概率分布形式，即

$$\begin{aligned} p(y|\eta) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{y^2}{2\sigma^2}\right) \cdot \exp\left(\frac{\mu}{\sigma^2}y - \frac{\mu^2}{2\sigma^2}\right) \end{aligned} \quad (1.75)$$

那么我们很容易看出 $\eta = \frac{\mu}{\sigma^2}$ ，那么我们定义一个 μ 到 η 的链接函数 (Link function) $g(\mu)$ 如下

$$\begin{aligned} g(\mu) &= \frac{\mu}{\sigma^2} \\ &= \eta \end{aligned} \quad (1.76)$$

那么这个线性模型 $f(\mathbf{x})$ 的构造就呼之欲出了。

$$\begin{aligned}
 f(\mathbf{x}) &= E(y|\eta) \\
 &= \mu \\
 &= g^{-1}(\eta) \\
 &= \sigma^2 \eta \\
 &= \sigma^2 \mathbf{w}^\top \mathbf{x}
 \end{aligned} \tag{1.77}$$

其中 E 表示这个分布的期望函数， $g^{-1}(\cdot)$ 为响应函数（Response function），是 $g(\cdot)$ 的反函数。

对于给定 \mathbf{x} ，认为 σ^2 是常量，完全可以由 \mathbf{w} 体现。故该线性模型最后为

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \tag{1.78}$$

同理，其他指数族分布同样如此构造广义线性模型，如通过伯努利分布（Bernoulli distribution）构造的广义线性模型，就是 logistic 回归的模型，对具体构造推导感兴趣的读者可以自行查阅相关文献。而损失函数的构造主要通过极大似然法，在 1.7.3 小节中，我们有详细介绍，不再赘述。

1.8 本章总结

在本章中，我们介绍了线性回归及相关内容。首先我们提出了基本的线性模型 $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ ，以及线性回归的目的。为了达到线性回归的目的，我们提出了损失函数 $\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$ 。为了最小化损失函数，可以得到线性回归的闭式解 $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$ 。为了解决 $\mathbf{X}^\top \mathbf{X}$ 可能不可逆等问题，我们介绍了岭回归。在岭回归中，主要是把损失函数变换成 $\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{1}{2} \lambda \mathbf{w}^\top \mathbf{w}$ ，相应地，闭式解为 $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$ 。最后，我们在拓展阅读中介绍了 L1 正则化回归、多项式回归、损失函数的概率解释以及广义线性模型。对线性回归尚存疑问的读者，可以查阅本章末尾的参考文献。



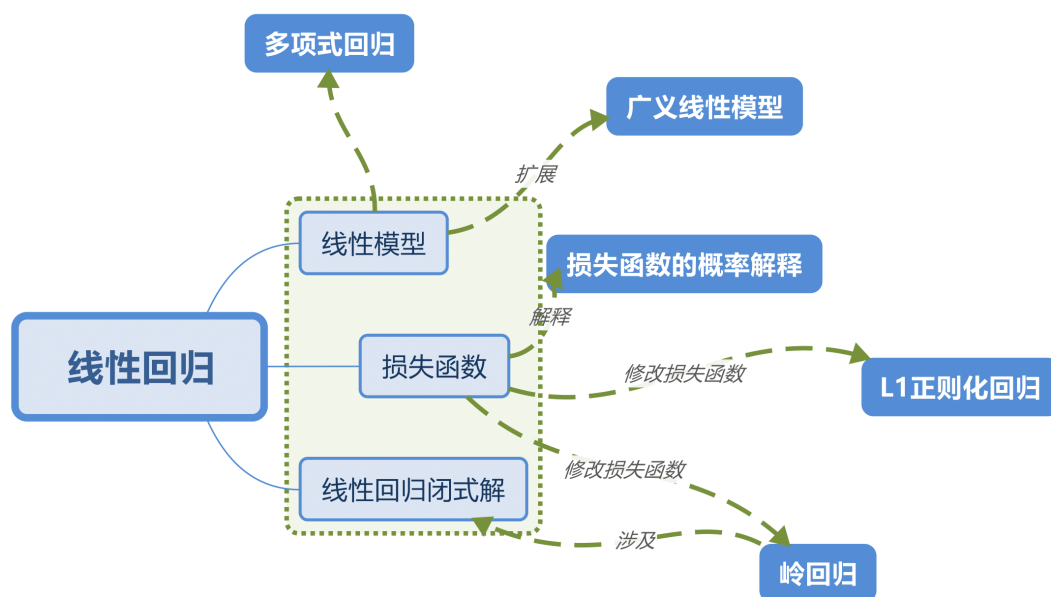


图 1.4: 线性回归知识框架

高斯与最小二乘法

18 世纪末，天文学的发展以测量和观测为基础。1794 年，高斯利用了最小二乘法解决了多余观测问题，当时他只有 17 岁。他通过实验得到两组数据，目的是用一个简单的式子来表示这些数据之间的关系。在数据分析的过程中，他首先把这些点绘制在二维平面中，发现这些点差不多在一条直线上，因此自然想到用一条直线，也就是一个线性函数来表达这些数之间的关系，求解的过程相当于求方程组中的待定系数。高斯想到，这个拟合过程毕竟是估计，不是真实的。要是用来做预测的话，必然存在一定的误差，这个误差如果能控制在相对小范围内的话，那么结果相对就好一些。所以，他找到了我们现在广为应用的残差平方和取最小值的方法，得到了待定系数的最小二乘估计。

在 1801 年，意大利著名天文学家朱赛普皮亚齐发现了第一颗小行星——谷神星。经过 40 多天的持续观察后，发现谷神星运转到太阳的背后，使得皮亚齐无法观测到谷神星的位置。在此之后，全世界的科学家利用皮亚齐的观测数据开始寻找谷神星，但是利用大多数人计算的结果来寻找谷神星，都没有找到。时年 24 岁的高斯也计算了谷神星的位置和运行轨道函数。奥地利天文学家海因里希奥尔伯斯根据高斯计算出来的轨道结果重新发现了谷神星。这一神奇的计算结果，得到了全世

界的瞩目。

高斯使用的最小二乘法的方法，最早发表于1809年他的著作《天体运动论》中。法国科学家勒让德于1806年独立发现“最小二乘法”。但因名气太小而不为人所知。勒让德曾与高斯为“谁最早创立最小二乘法原理”发生争执。根据普拉克特记载，普通最小二乘是勒让德“确定彗星轨道的新方法”。阿比意力挺高斯，声称高斯是1795年发明的，认为勒让德是在1806年首次发表的。最终在1829年，高斯提供了最小二乘法的优化效果强于其他方法的证明，因此被称为“高斯—莫卡夫”定理。最小二乘法的基本概念区别于以往任何的求得极值的方法，它的运用是相对灵活的。所用的具体的极小化方法可以是十分复杂的，要利用非线性规划的算法；也可以是相对地简单，只要用中学生的计算水平就可以解决了。所以最小二乘法在实践中得以广泛使用。



参考文献

- [1] Gene H Golub, Per Christian Hansen, and Dianne P O'Leary. Tikhonov regularization and total least squares. *SIAM Journal on Matrix Analysis and Applications*, 21(1):185–194, 1999.
- [2] Peter McCullagh. *Generalized linear models*. Routledge, 2018.
- [3] John Ashworth Nelder and Robert WM Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384, 1972.
- [4] George AF Seber and Alan J Lee. *Linear regression analysis*, volume 329. John Wiley & Sons, 2012.
- [5] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- [6] 周志华. 机器学习. Qing hua da xue chu ban she, 2016.
- [7] 邱锡鹏. 神经网络与深度学习.
- [8] 霍建新. 高斯的“大手笔”. *中国统计*, (5):57–58, 2010.

第 2 章 RNN & LSTM

2.0.0.0.1 本章基本概念 (专有名词)

超参数 在开始学习过程之前设置值的参数, 定义关于模型的更高层次的概念, 如复杂性或学习能力。在神经网络中又叫网络参数, 网络超参数。

隐藏层 网络中除输入层和输出层以外的其他各层, 不直接接受外界的信号, 也不直接向外界发送信号。

隐状态 隐藏层节点的状态, 随时间变化而变化, 又称隐藏层神经元的活性值, 可以理解为一种神经元中存储的动态信息。

非线性激活函数 logistic 函数, tanh 函数。

2.1 RNN

2.1.1 为什么需要循环神经网络

首先我们从一个简单问题讲起, 你能看懂下面这句话吗? “神学络喜我。欢经网”, 显然, 即使你看得懂每一个字, 但是你无法理解整句话的意思。但是如果写成下面这样呢, “我喜欢学神经网络。”, 这下整个句子都通顺了, 意思也很好理解。是的, 在现实生活中, 把某个整体割裂成众多部分, 每一小部分单独理解, 意义是浅薄的。而当这些小部分乱序地拼凑, 然后当作整体来理解时, 意义很可能是混乱的。而这种序列又广泛存在于现实生活中, 比如音频、视频、文字信息。这些都说明了序列信息的重要性与广泛性。而读者通过前面章节的阅读, 可能会有这样的疑问, 能够几乎模拟“任意”连续函数的神经网络, 又是否有处理序列信息的能力呢?

传统的神经网络的确有处理序列的能力, 但很有限, 比如只能处理特定长度的输入输出, 而且无法记忆上一时刻的输入输出, 更别说处理上一时刻的输入输出了。即使人为地用上一时刻的输入输出对这一时间的输入简单地进行处理, 也无法达到很好的效果。而卷积神经网络的确有着对区域处理的能力, 但是卷积神经网络认为各部分的区域之间是独立的, 而序列信息则认为序列中各部分区域是有顺序联系的, 显然卷积神经网络也不太适合处理序列信息。

那么现在就需要一个天然能处理序列的模型了。你以前可能听过自回归模型、延时神经网络等能够天然处理序列信息的模型, 但这些都不是本章的重点, 本章重点是循环神经网络 (Recurrent Neural Network, RNN), 一个天然能处理序列的神经网络。

为什么循环神经网络能够可以很好地处理序列信息呢？类比我们人脑，当我们用大脑进行学习的时候，除非是婴儿，不然都不可能是大脑一片空白地进行学习。我们很多时候会忘记一些发生在我们身上不太重要的事情，但更多时候我们会带着过往的经历去学习新的内容。循环神经网络正是借鉴了这种思想，让神经网络带有可以自反馈的神经元，这种神经元可以利用上一时刻的自带信息与这一时刻的输入信息进行更新，这理论上可以处理任意长度的序列信息。但是该神经元在不断更新的过程中，必然会逐渐丢失掉过去太久的信息。故循环神经网络是一种具有短期记忆的神经网络。在下一节中，我们会简单介绍循环神经网络的结构和不同类型的循环神经网络。

2.1.2 不同类型的循环神经网络

在了解不同类型的循环神经网络之前，我们首先需要知道循环神经网络的一般结构是什么。假设存在 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ 长度为 T 的输入序列，一般来说， $\mathbf{x}_t \in \mathbb{R}^{d \times 1}$, $t \in \{1, 2, \dots, T\}$ 。那么在 t 时刻隐藏层状态为 \mathbf{h}_t ，而 t 时刻的隐藏层状态不止和这时刻的输入 \mathbf{x}_t 有关，还和上时刻的隐藏层状态 \mathbf{h}_{t-1} 有关，所以他们之间的关系如下。

$$\mathbf{z}_t = \mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b} \quad (2.1)$$

$$\mathbf{h}_t = f(\mathbf{z}_t) \quad (2.2)$$

上式中的 \mathbf{z}_t 指的是未经激活的净输入， \mathbf{W} 是输入-状态权重矩阵， \mathbf{U} 是状态-状态权重矩阵， \mathbf{b} 是偏置项， f 是激活函数，如 \tanh 和 ReLU 函数等。其中 $\mathbf{z}_t \in \mathbb{R}^{m \times 1}$, $\mathbf{W} \in \mathbb{R}^{m \times d}$, $\mathbf{U} \in \mathbb{R}^{m \times m}$, $\mathbf{b} \in \mathbb{R}^{m \times 1}$, $\mathbf{h}_t \in \mathbb{R}^{m \times 1}$, m 由隐藏层的神经元个数决定。示意图如图2.1所示。

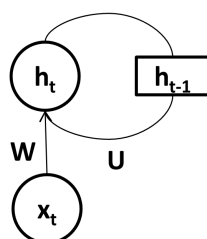


图 2.1: 循环神经网络的输入层和隐藏层

特别地，当隐藏状态 \mathbf{h} 还没有被初始化的时候，即 \mathbf{h}_1 的赋值存在如下关系

$$\mathbf{h}_0 = \mathbf{0} \quad (2.3)$$

$$\mathbf{h}_1 = f(\mathbf{W}\mathbf{x}_1 + \mathbf{U}\mathbf{h}_0 + \mathbf{b}) \quad (2.4)$$

$$= f(\mathbf{W}\mathbf{x}_1 + \mathbf{b}) \quad (2.5)$$

这里的 $\mathbf{0}$ 是维度与 $\mathbf{W}\mathbf{x}_1$ 相同的零向量。

而当只有一层隐藏层的时候，在 t 时刻循环神经网络的输出为 \mathbf{o}_t ，它与隐藏层的关系如下

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_h \quad (2.6)$$

上式中 \mathbf{V} 为状态-输出矩阵， \mathbf{b}_h 为偏置项。其中 $\mathbf{o}_t \in \mathbb{R}^{q \times 1}$ ， $\mathbf{V} \in \mathbb{R}^{q \times m}$ ， $\mathbf{b}_h \in \mathbb{R}^{q \times 1}$ ， q 的存在是因为假定训练集 \mathcal{T} 中在 t 时刻的标签值 $\mathbf{y}_t \in \mathbb{R}^{q \times 1}$ 。示意图如图2.2所示。为了示意图的简洁，该小节的示意图均省略偏置项。

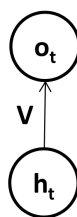


图 2.2: 循环神经网络的隐藏层和输出层

如果我们让循环神经网络凸显出循环之意，我们可以把循环神经网络按时间维度展开。我们可以清晰地看到循环神经网络中隐藏层参数是共享的，示意图如图2.3所示。

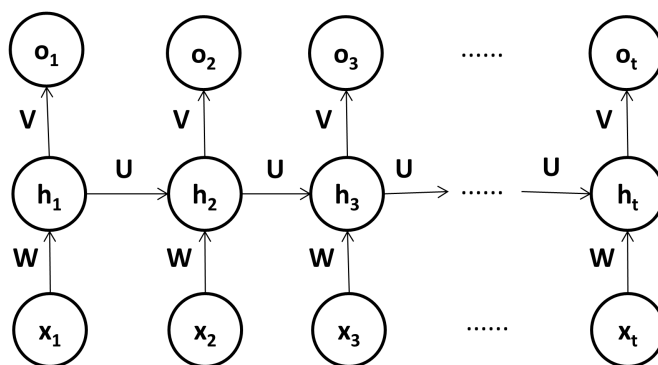


图 2.3: 按时间维度展开的循环神经网络

对于机器学习的不同任务，输入输出很可能结构不一样。那么根据这些任务特点不同，我们应用不同类型的循环神经网络。下面我们来看下这几种不同类型的循环神经网络。

2.1.2.1 多对一循环神经网络

多对一的循环神经网络模型主要用于解决机器学习中的分类问题，如文本的分类、视频的分类。输入是序列信息，输出是类别。

例如，训练集 \mathcal{T} 的一个样本为 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ ，标签值为 $y \in \{1, 2, \dots, C\}$ ，那么该样本 \mathbf{x} 按序列顺序输入到多对一的循环神经网络之后，可以得到 $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T$ 一系列的隐藏状态。我们可以将 \mathbf{h}_T 最后用于输出，即

$$\mathbf{o} = \mathbf{V}\mathbf{h}_T + \mathbf{b}_h \quad (2.7)$$

上式的 \mathbf{o} 是整个多对一循环神经网络的输出。具体示意图如图2.4所示。

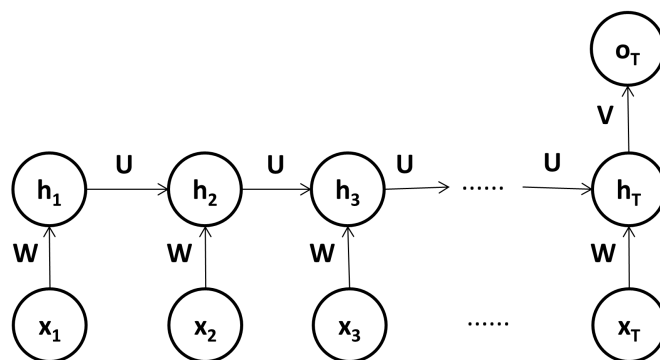


图 2.4: 多对一循环神经网络

当然，在一般情况下，会对 y 进行 one-hot 向量化， \mathbf{h}_T 最后连一层前馈神经网络，即利用 softmax 等函数对 \mathbf{o} 进行激活，最后分类。在本小节中就不详细介绍了。

2.1.2.2 一对多循环神经网络

一对多的循环神经网络的典型应用是图片的描述，输入一张图片，输出文本序列。又或者是给定一个情感类别，生成一个音符序列。

训练集 \mathcal{T} 的一个样本为 \mathbf{x} ，对应标签值则是一个序列 $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T)$ ，那么样本 \mathbf{x} 输入到一对多循环神经网络之后，得到 $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T$ 一系列的隐藏状态，但是每个隐藏状态都会进行输出，即

$$\mathbf{h}_1 = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad (2.8)$$

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}), t \in \{2, \dots, T\} \quad (2.9)$$

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_h, t \in \{1, \dots, T\} \quad (2.10)$$

具体示意图如图2.5所示。

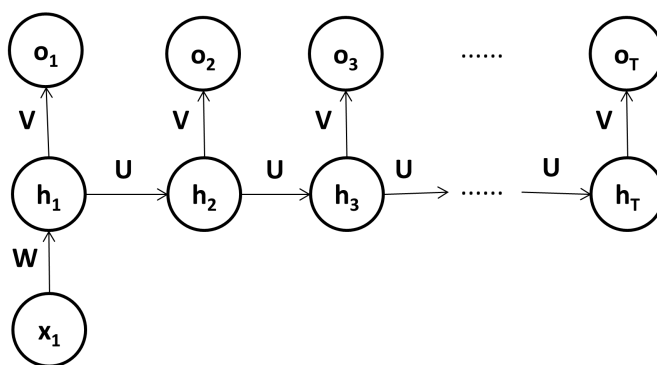


图 2.5: 一对多循环神经网络

2.1.2.3 同步的多对多循环神经网络

同步的多对多循环神经网络主要是用在序列标注，比如一个英语单词可能有多种词性，但是在句子的具体位置就只有一种词性了，对每一个单词进行词汇标注。或者是对一个视频的每帧进行分类，又或者用于一个字对下一个字的预测。

训练集的一个样本 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ ，对应标签序列为 $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T)$ ，样本按序列顺序输入到多对一的循环神经网络之后，可以得到 $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T$ 一系列的隐藏状态。每个隐藏状态都会进行输出，即

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_h \quad (2.11)$$

上式中的 $t \in \{1, \dots, T\}$ ，具体示意图如图2.6所示。

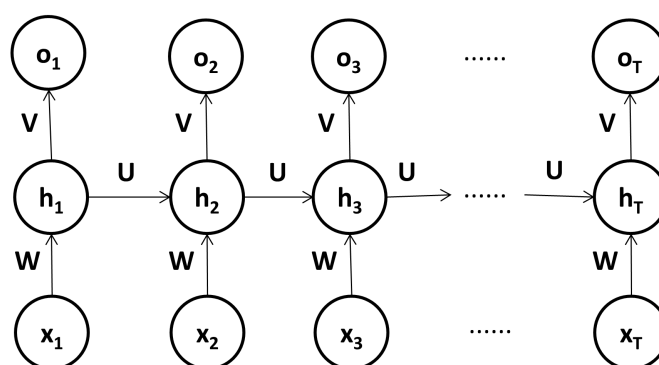


图 2.6: 同步的多对多循环神经网络

2.1.2.4 异步的多对多循环神经网络

异步的多对多循环神经网络又叫编码-解码模型（Encoder-Decoder model），主要用于机器翻译。因为在翻译的过程中，一种语言翻译成另一种语言的时候，字符序列的数量大多数情况下都是不一致的。异步的多对多循环神经网络并不要求输入序列和输出序列有一一对应关系，也不需要保持相同的长度。

训练集的一个样本 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{T_1})$ ，标签序列为 $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{T_2})$ 。那么样本 \mathbf{x} 按序列顺序输入到一个循环神经网络中，最后得到隐藏状态 \mathbf{h}_{T_1} ，传进另一个循环神经网络中得到输出序列 $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_{T_2}$ 。具体关系如下

$$\mathbf{h}_t = f_1(\mathbf{W}\mathbf{x}_t + \mathbf{U}_1\mathbf{h}_{t-1} + \mathbf{b}_1), t \in \{1, 2, \dots, T_1\} \quad (2.12)$$

$$\mathbf{h}_{T_1+t} = f_2(\mathbf{U}_2\mathbf{h}_{T_1+t-1} + \mathbf{b}_2), t \in \{1, 2, \dots, T_2\} \quad (2.13)$$

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_{T_1+t} + \mathbf{b}_h, t \in \{1, 2, \dots, T_2\} \quad (2.14)$$

上式中的 f_1 和 f_2 分别代表两个循环神经网络的激活函数。 $\mathbf{U}_1, \mathbf{U}_2, \mathbf{b}_1, \mathbf{b}_2$ 分别为两个循环神经网络的对应参数。具体示意图如图2.7所示。

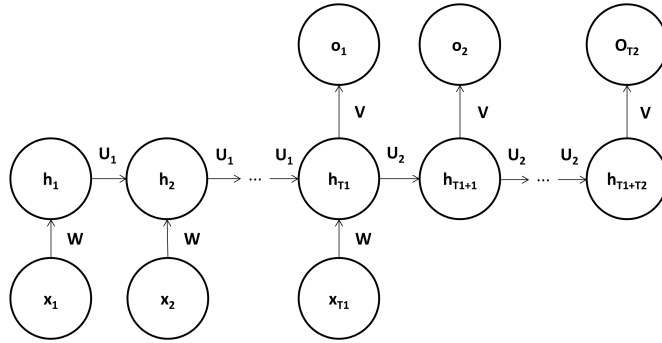


图 2.7: 异步的多对多循环神经网络

2.1.3 参数更新

循环神经网络的参数的更新，一般需要梯度的计算，可以通过随机梯度下降算法进行更新。

为了不失一般性，我们利用同步的多对多循环神经网络辅随机梯度下降算法，进行参数更新的计算。那么给定训练集 \mathcal{L} 的一个样本 (\mathbf{x}, \mathbf{y}) ， $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ ，输入序列长度为 T ， $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T)$ ，输出序列的长度也为 T 。若没有特殊说明，本节出现的变量维度表示如下， $\mathbf{x}_t \in \mathbb{R}^{d \times 1}$ ， $\mathbf{y}_t \in \mathbb{R}^{q \times 1}$ ， $\mathbf{z}_t \in \mathbb{R}^{m \times 1}$ ， $\mathbf{W} \in \mathbb{R}^{m \times d}$ ， $\mathbf{U} \in \mathbb{R}^{m \times m}$ ， $\mathbf{b} \in \mathbb{R}^{m \times 1}$ ， $\mathbf{h}_t \in \mathbb{R}^{m \times 1}$ ， $\mathbf{o}_t \in \mathbb{R}^{q \times 1}$ ， $\mathbf{V} \in \mathbb{R}^{q \times m}$ ， $\mathbf{b}_h \in \mathbb{R}^{q \times 1}$ 。

我们定义 t 时刻的损失函数为

$$\mathcal{L}_t = \mathcal{L}(\mathbf{y}_t, g(\mathbf{o}_t)) \quad (2.15)$$

上式中的 g 一般为分类函数， \mathcal{L} 为损失函数，如交叉熵损失函数。整个序列的损失函数为

$$\mathcal{L} = \sum_{t=1}^T \mathcal{L}_t \quad (2.16)$$

因 $\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_h$ ，所以 \mathcal{L}_t 对参数 \mathbf{V} 第 i 行第 j 列元素 V_{ij} 的梯度为

$$\begin{aligned} \frac{\partial \mathcal{L}_t}{\partial V_{ij}} &= \left(\frac{\partial (\mathbf{V}\mathbf{h}_t + \mathbf{b}_h)}{\partial V_{ij}} \right)^\top \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \\ &= \mathbb{I}_i(\mathbf{h}_t^j)^\top \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \\ &= \left(\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \right)^i \mathbf{h}_t^j \end{aligned} \quad (2.17)$$

上式中的 $\mathbb{I}_i(\mathbf{h}_t^j)$ 是一个列向量，其第 i 行是列向量 \mathbf{h}_t 的第 j 个分量，其余行皆为 0。而 $(\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t})^i$ 是列向量 $\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$ 的第 i 个分量，我们在前馈神经网络用过类似的处理方式。

那么 \mathcal{L}_t 对参数 \mathbf{V} 梯度为

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{V}} = \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \mathbf{h}_t^\top \quad (2.18)$$

$\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$ 由具体的损失函数和分类函数 g 所决定，这里暂不讨论。

所以整个序列的损失函数 \mathcal{L} 对参数 \mathbf{V} 梯度为

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{V}} &= \sum_{t=1}^T \frac{\partial \mathcal{L}}{\partial \mathcal{L}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \mathbf{h}_t^\top \\ &= \sum_{t=1}^T \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \mathbf{h}_t^\top \end{aligned} \quad (2.19)$$

而整个序列的损失函数 \mathcal{L} 对参数 \mathbf{b}_h 梯度为

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}_h} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \quad (2.20)$$

整个序列的损失函数 \mathcal{L} 对参数 \mathbf{U} 、 \mathbf{W} 、 \mathbf{b} 的梯度却并不是那么好求的，下面我们介绍随时间反向传播算法来解决这个问题。

2.1.3.1 随时间反向传播算法

随时间反向传播（Backpropagation Through Time, BPTT）算法类似反向传播算法，但是，BPTT 算法把循环神经网络看成一个展开的多层神经网络，循环神经网络每个时刻的隐藏层相当于展开的多层神经网络的中的一层，但是层与层之间的参数是共享的。所以当进行反向传播计算时，共享参数的梯度是所有展开层对应参数梯度之和。

首先我们先计算 \mathcal{L} 对 \mathbf{U} 的梯度 $\frac{\partial \mathcal{L}}{\partial \mathbf{U}}$ 。

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{U}} &= \sum_{t=1}^T \frac{\partial \mathcal{L}}{\partial \mathcal{L}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{U}} \\ &= \sum_{t=1}^T \frac{\partial \mathcal{L}_t}{\partial \mathbf{U}}\end{aligned}\quad (2.21)$$

而直接求 $\frac{\partial \mathcal{L}}{\partial \mathbf{U}}$ 的话，难以计算，我们先求 $\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}_{ij}}$ 。在时刻 $k (1 \leq k \leq t)$ 中，存在下列关系

$$\mathbf{z}_k = \mathbf{W}\mathbf{x}_k + \mathbf{U}\mathbf{h}_{k-1} + b \quad (2.22)$$

$$\mathbf{h}_k = f(\mathbf{z}_k) \quad (2.23)$$

故 \mathcal{L}_t 对 \mathbf{U}_{ij} 的梯度是

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}_{ij}} &= \sum_{k=1}^t \left(\frac{\partial^* \mathbf{z}_k}{\partial \mathbf{U}_{ij}} \right)^\top \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k} \\ &= \sum_{k=1}^t \mathbb{I}_i(\mathbf{h}_{k-1}^j)^\top \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k}\end{aligned}\quad (2.24)$$

上式中的 $\frac{\partial^* \mathbf{z}_k}{\partial \mathbf{U}_{ij}}$ 是较为特殊的偏导数，即 $\mathbf{z}_k = \mathbf{W}\mathbf{x}_k + \mathbf{U}\mathbf{h}_{k-1} + b$ 中 \mathbf{h}_{k-1} 暂视为与 \mathbf{U}_{ij} 无关的变量，对 \mathbf{U}_{ij} 进行求偏导数。而 $\mathbb{I}_i(\mathbf{h}_{k-1}^j)$ 是一个列向量，其第 i 行是列向量 \mathbf{h}_{k-1} 的第 j 个分量，其余行皆为 0，我们在上面的章节有过相似的操作。

设 $\delta_{t,k} = \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k}$ ，借鉴反向传播算法的思路，可列

$$\begin{aligned}\delta_{t,k} &= \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k} \\ &= \frac{\partial \mathbf{h}_k}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{h}_k} \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_{k+1}} \\ &= \text{diag}(f'(\mathbf{z}_k)) \mathbf{U}^\top \delta_{t,k+1}\end{aligned}\quad (2.25)$$

上式 $\delta_{t,k} \in \mathbb{R}^{m \times 1}$, $f'(\mathbf{z}_k) \in \mathbb{R}^{m \times 1}$, $\text{diag}(f'(\mathbf{z}_k)) \in \mathbb{R}^{m \times m}$, 其中 $\text{diag}(f'(\mathbf{z}_k))$ 表示由向量 $f'(\mathbf{z}_k)$ 中的元素组成对角矩阵的矩阵, 非对角位置皆为 0。

特别地, 因为 $\delta_{t,t+1}$ 不存在, $\delta_{t,t}$ 的计算公式不能沿用上式, 故 $\delta_{t,t}$ 为

$$\begin{aligned}\delta_{t,t} &= \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_t} \\ &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \\ &= \text{diag}(f'(\mathbf{z}_t)) \mathbf{V}^\top \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}\end{aligned}\tag{2.26}$$

那么 $\frac{\partial \mathcal{L}_t}{\partial U_{ij}}$ 则改写为

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial U_{ij}} &= \sum_{k=1}^t \mathbb{I}_i(\mathbf{h}_{k-1}^j)^\top \delta_{t,k} \\ &= \sum_{k=1}^t (\delta_{t,k})^i \mathbf{h}_{k-1}^j\end{aligned}\tag{2.27}$$

上式的 $(\delta_{t,k})^i$ 是列向量 $\delta_{t,k}$ 的第 i 个分量, \mathbf{h}_{k-1}^j 是列向量 \mathbf{h}_{k-1} 的第 j 个分量。故 $\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}}$ 为

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}} = \sum_{k=1}^t \delta_{t,k} \mathbf{h}_{k-1}^\top\tag{2.28}$$

所以

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} \mathbf{h}_{k-1}^\top\tag{2.29}$$

同理, 可以得到 \mathcal{L} 对 \mathbf{W} 和 \mathbf{b} 的梯度为

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} \mathbf{x}_k^\top\tag{2.30}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k}\tag{2.31}$$

2.1.3.2 损失函数与激活函数

在上面内容的讨论中, 我们发现因为损失函数和激活函数没有定义, 无法写出 $\frac{\partial \mathcal{L}}{\partial \mathbf{o}_t}$ 和 $f'(\mathbf{z}_t)$ 的具体形式。为了说明 $\frac{\partial \mathcal{L}}{\partial \mathbf{o}_t}$ 和 $f'(\mathbf{z}_t)$ 的常见形式, 设利用该循环神经网络解决的问题是分类问题。其中 \mathbf{y}_t 是 one-hot 列向量, 即当 \mathbf{y}_t 属于第 c 类

时, \mathbf{y}_t 的第 c 个分量为 1, 剩余分量皆为 0。分类函数用 softmax 函数, 损失函数设为交叉熵损失函数, 即

$$\begin{aligned}
 \mathcal{L}_t &= -\mathbf{y}_t^\top \ln(\text{softmax}(\mathbf{o}_t)) \\
 &= -\mathbf{y}_t^\top \ln\left(\frac{\exp(\mathbf{o}_t)}{\mathbf{1}^\top \exp(\mathbf{o}_t)}\right) \\
 &= -\mathbf{y}_t^\top (\ln(\exp(\mathbf{o}_t)) - \mathbf{1} \ln(\mathbf{1}^\top \exp(\mathbf{o}_t))) \\
 &= -\mathbf{y}_t^\top \mathbf{o}_t + \mathbf{y}_t^\top \mathbf{1} \ln(\mathbf{1}^\top \exp(\mathbf{o}_t)) \\
 &= -\mathbf{y}_t^\top \mathbf{o}_t + \ln(\mathbf{1}^\top \exp(\mathbf{o}_t))
 \end{aligned} \tag{2.32}$$

其中 $\mathbf{1}$ 是维度与 \mathbf{o}_t 相同的全 1 列向量, 因为 \mathbf{y}_t 是 one-hot 列向量, 故 $\mathbf{y}_t^\top \mathbf{1}$ 为标量 1。

此时, \mathcal{L}_t 对 \mathbf{o}_t 的梯度为

$$\begin{aligned}
 \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} &= \frac{\partial(-\mathbf{y}_t^\top \mathbf{o}_t + \ln(\mathbf{1}^\top \exp(\mathbf{o}_t)))}{\partial \mathbf{o}_t} \\
 &= -\mathbf{y}_t + \frac{1}{\mathbf{1}^\top \exp(\mathbf{o}_t)} \frac{\partial(\mathbf{1}^\top \exp(\mathbf{o}_t))}{\partial \mathbf{o}_t} \\
 &= -\mathbf{y}_t + \frac{1}{\mathbf{1}^\top \exp(\mathbf{o}_t)} \left(\frac{\partial \mathbf{1}}{\partial \mathbf{o}_t} \exp(\mathbf{o}_t) + \frac{\partial \exp(\mathbf{o}_t)}{\partial \mathbf{o}_t} \mathbf{1} \right) \\
 &= -\mathbf{y}_t + \frac{1}{\mathbf{1}^\top \exp(\mathbf{o}_t)} \left(\frac{\partial \exp(\mathbf{o}_t)}{\partial \mathbf{o}_t} \mathbf{1} \right) \\
 &= -\mathbf{y}_t + \frac{1}{\mathbf{1}^\top \exp(\mathbf{o}_t)} (\text{diag}(\exp(\mathbf{o}_t)) \mathbf{1}) \\
 &= -\mathbf{y}_t + \frac{\exp(\mathbf{o}_t)}{\mathbf{1}^\top \exp(\mathbf{o}_t)} \\
 &= \text{softmax}(\mathbf{o}_t) - \mathbf{y}_t
 \end{aligned} \tag{2.33}$$

然后我们设该循环神经网络的激活函数为常见的 tanh 函数, tanh 的形式以及对标量的导数如下

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{2.34}$$

$$\begin{aligned}
 \tanh'(x) &= 1 - \left(\frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \right)^2 \\
 &= 1 - \tanh^2(x)
 \end{aligned} \tag{2.35}$$

其中 $x \in \mathbb{R}$ 。

所以 $f'(\mathbf{z}_t)$ 为

$$\begin{aligned} f'(\mathbf{z}_t) &= \tanh'(\mathbf{z}_t) \\ &= \mathbf{1} - \tanh^2(\mathbf{z}_t) \end{aligned} \quad (2.36)$$

其中 $\mathbf{1}$ 是与 \mathbf{z}_t 维度相同的全 1 列向量。所以，该循环神经网络的 \mathbf{h}_t 对 \mathbf{z}_t 的梯度为

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} = \text{diag}(\mathbf{1} - \tanh^2(\mathbf{z}_t)) \quad (2.37)$$

2.1.4 长期记忆中的问题

相信读者已经了解了循环神经网络的基本原理，它在理论上可以建立长时间间隔的隐状态之间的依赖关系，类似于我们在很久以前学到的知识到现在都还记得。然而，实际上我们并不能完全记得十年前这个时候是在上哪一节课，同理，简单的循环神经网络实际上也只能学到短期的依赖关系，即其记忆是短期的。为什么呢？让我们来弄清在实现长期记忆功能时具体会出现什么问题。

2.1.4.1 梯度爆炸问题

深度神经网络中基于梯度学习的根本问题是梯度不稳定的问题，当不同隐藏层进行更新的速度差异很大时，就容易产生问题。每一次状态更新，都要用到误差梯度，即确定正确的更新方向和更新的量，从而正确的更新网络权重。然而，训练过程中大的误差梯度可能不断累积，每个节点和层的误差梯度可能总是大于 1.0，呈现指数式增长，导致神经网络模型的权重出现大幅更新的问题，即梯度爆炸问题。这会使网络模型不稳定，某些权重的值可能会大到溢出系统最大数值，导致出现 NaN 值，从而无法继续更新。这就像一个人每天都活在昨天的世界里，放不下过去，看不到未来，最终被往事所吞灭。

让我们从公式的角度更准确地理解梯度爆炸。在 BPTT 算法中，我们最终得到了损失函数 \mathcal{L} 对参数 U 的梯度：

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} \mathbf{h}_{k-1}^\top \quad (2.38)$$

其中 \mathbf{h}_{k-1} 是 $k-1$ 时刻隐藏层状态，可以理解为隐藏层神经元在这一时刻的记忆信息，神经元利用它的记忆可以产生输出，即 $\mathbf{o}_t = V\mathbf{h}_t + \mathbf{b}_h$ 。为了保证隐藏状态 \mathbf{h}_{t-1} 稳定、合理的传递给 \mathbf{h}_t （避免突然失忆，进入“失忆态”，或者与之相反，避免沉溺于过去的记忆，新输入的信息基本被忽略，进入“抑郁态”）就要保证上式中的 $\delta_{t,k}$ 不能出现异常，比如不能在较长一段时间内一直严格单调递增，或者

不能长期严格单调递减。在上一节我们得到的 $\delta_{t,k}$ 计算公式为：

$$\delta_{t,k} = \text{diag}(f'(\mathbf{z}_k))U^T \delta_{t,k+1} \quad (2.39)$$

注意到 $\delta_{t,k}$ 依赖于 $\delta_{t,k+1}$ ， $\delta_{t,k+1}$ 又依赖于 $\delta_{t,k+2}$ ，最终会依赖于 $\delta_{t,t}$ ，从而有：

$$\delta_{t,k} = \prod_{i=k}^{t-1} \left(\text{diag}(f'(\mathbf{z}_i))U^T \right) \delta_{t,t} \quad (2.40)$$

可见，在反向传播时为了避免 $\delta_{t,k}$ 出现异常， $\text{diag}(f'(\mathbf{z}_i))U^T$ （一个 m 行 m 列的矩阵）中的元素不能长期都大于 1.0。如果长期大于 1.0， $\delta_{t,k}$ 随着 k 的减小会一项比一项大，从而 \mathbf{h}_k 在损失函数 \mathcal{L} 对参数 U 的梯度的贡献中的比重也会随着 k 的减小而增大，即过去的记忆越来越重要，活在过去，逃避未来，这种“抑郁态”就是长期记忆中经典的梯度爆炸问题。

解决梯度爆炸问题有以下几种方法：

1. 修改网络模型与架构，比如减少隐藏层层数、使用更小的批尺寸 (batch_size)、在 t 与 k 相差较大时停止继续反向传播（沿时间截断的反向传播）；
2. 使用长短期记忆网络，下一节将会对这种方法详细讲解；
3. 进行梯度截断，将梯度的模或者将每个梯度值限制在一个范围内；
4. 在计算损失时加入权重正则化项，检查网络权重的大小，惩罚产生较大权重值的损失项。

2.1.4.2 梯度消失问题

梯度消失问题是指走向了与梯度爆炸问题相反的另一个极端。因为展开的循环神经网络中靠前的隐藏层梯度接近于 0，所以参数更新时，利用不到过去太久的信息进行学习。好比一个人每天都只保留前一天的记忆的百分之九十，100 天后，他可能连自己的姓名都不记得了。

从公式的角度来理解梯度消失，即式 2.40 中矩阵 $\text{diag}(f'(\mathbf{z}_i))U^T$ 中的元素长期小于 1.0， $\delta_{t,k}$ 随着 k 的减小会一项比一项小， \mathbf{h}_k 在损失函数 \mathcal{L} 对参数 U 的梯度的贡献中所占的比重（式 2.38）也会随着 k 的减小而减小，即当 k 时刻与 t 时刻相距很远时 $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$ 这个梯度消失了，参数 U 的更新主要依赖于最近几个时刻的记忆 \mathbf{h} 。直观上看，就好似当要用到某张银行卡的密码时，由于这个密码上次用到还是一年前，之后也不知道将密码保存在哪里了，这种对长期信息进行记忆的功能未能实现，在极端情况下，就相当于“失忆”，这种“失忆态”就是长期记忆中经典的梯度消失问题。

解决梯度爆炸的方法中修改模型（调参等）、使用长期记忆网络、梯度截断（给梯度设置一个下限）也适用于解决梯度消失问题，此外，还可以从激活函数

的选择、简化模型的角度来考虑：

1. 注意到式2.40中有用到激活函数的导数 $f'(\mathbf{z}_i)$ ，尝试对 logistic 函数和 tanh 函数求导，可以发现，logistic 函数的导数值小于 0.25，而 tanh 函数的导数值小于 1.0（除了在零点为 1.0），这使 $\text{diag}(f'(\mathbf{z}_i))U^\top$ 有小于 1.0 的趋势，从而容易导致梯度消失问题。可以使用 ReLU、Leaky ReLU、ELU 等在正区间内导数正好为 1.0 的激活函数，但各有所长也各有所缺，这里不再细谈；
2. 回顾本章第一个公式 $\mathbf{h}_t = f(W\mathbf{x}_t + U\mathbf{h}_{t-1} + \mathbf{b})$ ，依旧是为了使式2.40中 $\text{diag}(f'(\mathbf{z}_i))U^\top$ 的元素都趋于 1.0，可以考虑对模型做些简化。比如将激活函数的导数设置为 1.0，即 $f'(\mathbf{z}_i) = \mathbf{1}$ ，然后将 U 简化为 \mathbf{I} ，则 $\mathbf{h}_t = f(\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b})$ ，这就使得 $\text{diag}(f'(\mathbf{z}_i))U^\top$ 直接简化为一个单位矩阵，从理论上消除了梯度消失和梯度爆炸问题，但同时也相当于将所有时刻的 \mathbf{h} 平等对待，降低了模型的表示能力。

2.1.4.3 记忆容量问题

训练一个神经网络需要耗费的 CPU 资源与内存资源往往很大，需要的时间也很长，同时也不利于调参。如果把一个 RNN 比作一个大脑，RNN 的训练过程就是大脑思考一个现实问题的过程。由于问题的复杂性，大脑可能不足以存储所需的记忆容量，不能快速回忆到所记忆的信息，甚至总是遗忘一些重要记忆，这时就称遇到了“记忆容量问题”。为了在一定程度上缓解对记忆容量的高要求，快速找到对解决问题有帮助的重要记忆，记忆力机制（Attention Mechanism）的引入取得了很好的效果，详情请见第3章。

2.2 LSTM

为了在有限的空间实现长期记忆功能，选择性的遗忘以前的重要性低的记忆，保留重要程度高的记忆从而“不断提高自己的智商”，这就需要一种综合可行的解决机制——Long Short-term Memory(LSTM)。

2.2.1 L 还是 S?

我们知道，Long 是“长”的意思，Short 是“短”的意思，term 取“学期、期限”之意，Memory 则指我们所求的“记忆”。那么，LSTM 所指的记忆功能到底是长的还是短的呢？

我们记得我们十年前的事情，记得我们上小学、初中、高中的事情，这是一种长期的记忆，但同时我们记得的事情都只是些片段，并不确切的记得十年前的这个时候我们到底在做一件什么样的具体的事，此外，我们更多能回忆起来的是最近发生的事情，比如上个月、上周我们学到了什么，这些短片段和近期的记忆

即 short-term memory，其中那些发生在很久之前的记忆片段又称“长的短期记忆”。

让我们对 LSTM 的理解上升到机器学习的层面。在神经网络中，长期记忆隐含了通过长期的迭代传播、更新训练得到的经验、重要信息，而记忆单元在最近一段时期捕捉、更新得到的某些关键信息称为短期记忆。长期记忆的更新是缓慢的、比较稳定的，而短期记忆的更新是频繁的比较剧烈的。换言之，那年，那个夏天，或许有我们终身难忘的事，而昨天、今天我们的所见所闻可能即见即忘。

2.2.2 三门分立

2.2.2.1 门

神奇的 LSTM 到底是怎么实现更长期的记忆功能的呢？其中一个显著的特征就是基于门的机制。LSTM 中的门是一扇魔法门，能以一定的百分比让想通过这道门的信息通过，也即过滤掉一定百分比的信息，就好似我们在学习的时候会吸取百分之多少的信息，抛弃百分之多少的信息。

那么，该设置几道门呢？又怎么确定该让信息通过一扇门的比率呢？人们常说“过去的就让它过去吧”，这就是种遗忘门，简称忘门；我们刷淘宝时无视那些不感兴趣、不需要的商品，虽然看到了，但也只是一扫而过，这是种输入门，简称入门；我们做演讲、和别人谈话时要表达我们的思想、意思，决定什么话该说什么话不该说的，就是输出门，简称出门。三门分立，是怎么各司其职，怎么确定过滤信息的比率，又是怎么共组一个高效而“高情商”的循环单元的呢？下面三小部分将介绍三门的过滤信息比率，下一小节 (2.2.3) 将介绍什么样的信息会进入这三种门。

2.2.2.2 忘门

忘门，即“往事知多少”，控制多少回忆值得珍存。用 \mathbf{f}_t 表示保留上一层信息的比例， $\mathbf{f}_t \in [0, 1]$ ，其中 f 表示“unforget”， t 表示时刻 t 。 t 时刻会有新的信息输入网络，即 \mathbf{x}_t ，上一时刻的隐藏层的外部状态（循环单元的输出信息）也会影响到信息过滤比，即 \mathbf{h}_{t-1} ，再结合偏置项 \mathbf{b} 和我们的老朋友 logistic，自然的得到了如下计算“回忆珍存率”的式子：

$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f) \quad (2.41)$$

其中 $\mathbf{f}_t \in \mathbb{R}^{h \times n}$ (h 是隐藏单元的个数)， $\mathbf{x}_t \in \mathbb{R}^{d \times n}$ (d 是单个输入样本的维度)， $\mathbf{h}_{t-1} \in \mathbb{R}^{h \times n}$ ， W_f 和 U_f 是忘门的权重参数， $W_f \in \mathbb{R}^{h \times d}$ ， $U_f \in \mathbb{R}^{h \times h}$ ， \mathbf{b}_f 是偏差参数， $\mathbf{b}_f \in \mathbb{R}^{h \times 1}$ 。上式可用图2.8形象的表示。

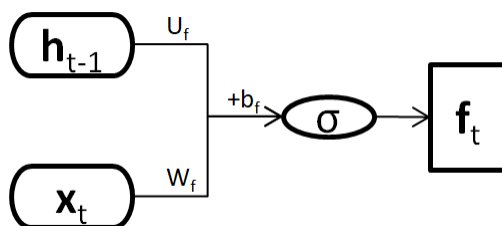


图 2.8: 忘门结构

当 $f_t = 0$ 时，忘门紧锁，记忆单元拒绝接受它的历史，获得“重生”；当 $f_t = 1$ 时，忘门完全敞开，记忆单元继承上一时刻的所有记忆。

2.2.2.3 入门

入门，即“今日知多少”，控制在 t 时刻外部输入信息可以成功流入记忆单元的比例，这个比例用 i_t 表示， $i_t \in [0, 1]$ ， i 指“input”。影响“input”的因素和遗忘门一样，都是 h_{t-1} 和 x_t ，不同的是其中的权重参数和偏差参数，故求 i_t 的公式为：

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (2.42)$$

相关矩阵的维度与忘门对应的矩阵一致，上式也可以用图2.9形象的表示。

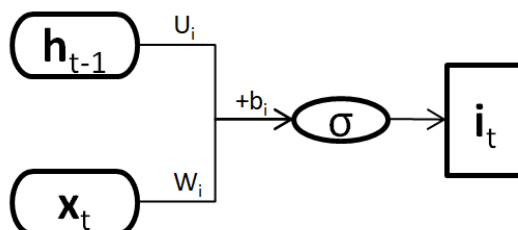


图 2.9: 入门结构

和忘门相似，当 $i_t = 0$ 时，“闭关锁国”，拒绝了一切的输入；当 $i_t = 1$ 时，“海纳百川”，接受所有的输入。

2.2.2.4 出门

出门决定哪些信息应该输出给外部状态 h_t ，用 o_t 表示输出信息的比例， $o_t \in [0, 1]$ ， o 指“output”。想通过出门出去的信息自然是成功进入忘门和入门的信息，但决定多少信息应该从出门出去的比例依旧受 h_{t-1} 和 x_t 影响，试想，我们的一言一语、一举一动，何曾不与当时当下的外界刺激和我们之前的行为所产生的影响有关呢？这样一来，求 o_t 的公式与数据流图呼之欲出：

$$\mathbf{o}_t = \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o) \quad (2.43)$$

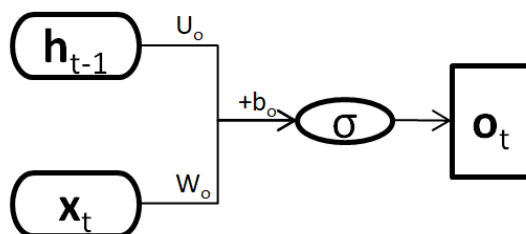


图 2.10: 出门结构

结合三门对信息的过滤比的公式2.41-2.43，可以发现所有门的比率都与当前直接的输入（ \mathbf{x}_t ）和上一时刻的输出（ \mathbf{h}_{t-1} ，这可以理解为 t 时刻一种间接的输入）有关，即我们决定一条信息是否值得记忆或发出去时应该根据这一时刻接受的信息的质量和过去我们所做的事情的效果来判断。举个出门的例子：当我们确定是否去操场上跑步时，应该根据此时的天气情况、时间（即 \mathbf{x}_t ）和我们上一次跑步的效果比如腿疼、喉咙疼、是否摔伤、是否突破个人纪录（即 \mathbf{h}_{t-1} ）来判断，至于是怎么利用这三道魔法门去确定我们是否去跑步或者我们真的去跑步了的概率的，请看下个回合——内外有别。

2.2.3 内外有别

现在，我们已经确定了三个门的“信息过滤比”，那么，哪些信息想进入这三门呢？即这三门应该安装在哪儿呢？这就要谈及 *LSTM* 与 *RNN* 另一个重要的不同——内部状态，用 \mathbf{c}_t 表示，其中 t 表示 t 时刻。

\mathbf{c}_t 作为一个存储单元内部的状态，或者说一个存储着信息的矩阵，它记录了到当前时刻为止的所有记忆，并专门负责进行线性的循环的信息传递，同时对本层神经元的外部状态 \mathbf{h}_t （即对外部的输出）产生一种非线性的影响。

\mathbf{c}_t 就像我们现在大脑里知道的所有信息，有些是昨天大脑里所有的记忆（即 \mathbf{c}_{t-1} ）经过睡一晚（忘门）后所保留下来的，所以忘门应该作用在 \mathbf{c}_{t-1} 上；有些是今天我们上课、看书、刷手机所获得的（即 \mathbf{x}_t ），或是昨天我们在淘宝上下的订单（即 \mathbf{h}_{t-1} ，注意，下单是昨天发生的事件，是我们昨天的输出，相当于隐藏层在上一时刻的外部状态），所以入门应该作用在整个存储单元所能获得的除了上一时刻记忆信息（ \mathbf{c}_{t-1} ）之外的所有信息上。为了简便，我们引入一个叫候选状态的变量来表示前面所述的“整个信息”，用 $\tilde{\mathbf{c}}_t$ 表示这个候选状态，其计算公式为：

$$\tilde{\mathbf{c}}_t = \tanh(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c) \quad (2.44)$$

容易发现 $\tilde{\mathbf{c}}_t$ 与前面三门的过滤比的计算公式很相似，都依赖于当前的输入 \mathbf{x}_t 和上一时刻的输出 \mathbf{h}_{t-1} ，矩阵的维度也一一对应。

然而，门的过滤信息的比率是通过 logistic 直接算出来的一个 0–1 之间的百分比组成的矩阵，而候选状态是通过 tanh 激活函数将输入转化为-1 到 1 之间的数值组成的矩阵，其意义是可以用于计算这一时刻到底该存储哪些信息作为一个存储单元的记忆，或者说，用于计算内部状态（ \mathbf{c}_t ），而这，也是候选状态之所以是“候选”的原因，就好像我们在真正把知识学到手之前要对知识进行不断的分析与筛选，得到我们的“干货”（ $\tilde{\mathbf{c}}_t$ ），再经过不可避免的遗忘（ \mathbf{f}_t ）与吸收（ \mathbf{i}_t ）过程，最终将知识转化为我们“大脑硬盘”里扎实的可以灵活运用的记忆。

说到这里，希望读者能理解或者写出自己“大脑硬盘”里存储着的记忆的计算公式，即我们隐藏层的内部状态：

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (2.45)$$

这里引入的一种新的运算， \odot ，表示逐元素相乘，即对应位上的值两两相乘，维度不变，都是 $(h \times n)$ ，其意义就是在每一位数据上获取相应比率的信息，体现了门的作用。如果读者仍有疑惑，可以再仔细理解一下前面的推理过程。

至此，我们知道了忘门和入门的“安装位置”，知道了候选状态和内部状态的求解公式，回想上一小节和本小节的主题，容易发现还有出门和外部状态没有解决。如果把内部状态比作我们的本领，外部状态比作我们的行为，那么出门则是连接二者之间的桥，本节最后一个公式由此而生：

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (2.46)$$

内有 $\tilde{\mathbf{c}}_t$ 、 \mathbf{c}_t ，外有 \mathbf{h}_t ，把握了 LSTM “内外有别”的特征，就把握了 LSTM 不同于 RNN 的精髓之所在。

2.2.4 LSTM 循环单元中的数据流

2.2.2 节有简单地孤立地描述三门的数据流图，让我们结合 2.2.3 节的思想从整体上理解一个小小的 LSTM 循环单元内部是怎样有序的运行的，如图 2.11 所示。

上图就像是由忘门、入门、出门和内外状态组成的一个小迷宫，我们可以通过以下三步的数据流动路径走出这个迷宫：

1. 由上一时刻的输出 \mathbf{h}_{t-1} （间接输入）和当前时刻的直接输入 \mathbf{x}_t ，得到当前时刻循环单元的总输入，即内部候选状态 $\tilde{\mathbf{c}}_t$ ，同时计算出三个门的过滤比矩阵 \mathbf{f}_t 、 \mathbf{i}_t 、 \mathbf{o}_t ；
2. 通过忘门对上一时刻的内部状态（ \mathbf{c}_{t-1} ）的筛选、入门对当前时刻候选状态信息（ $\tilde{\mathbf{c}}_t$ ）的过滤，得到当前时刻循环单元所应存储的内部状态信息，这一

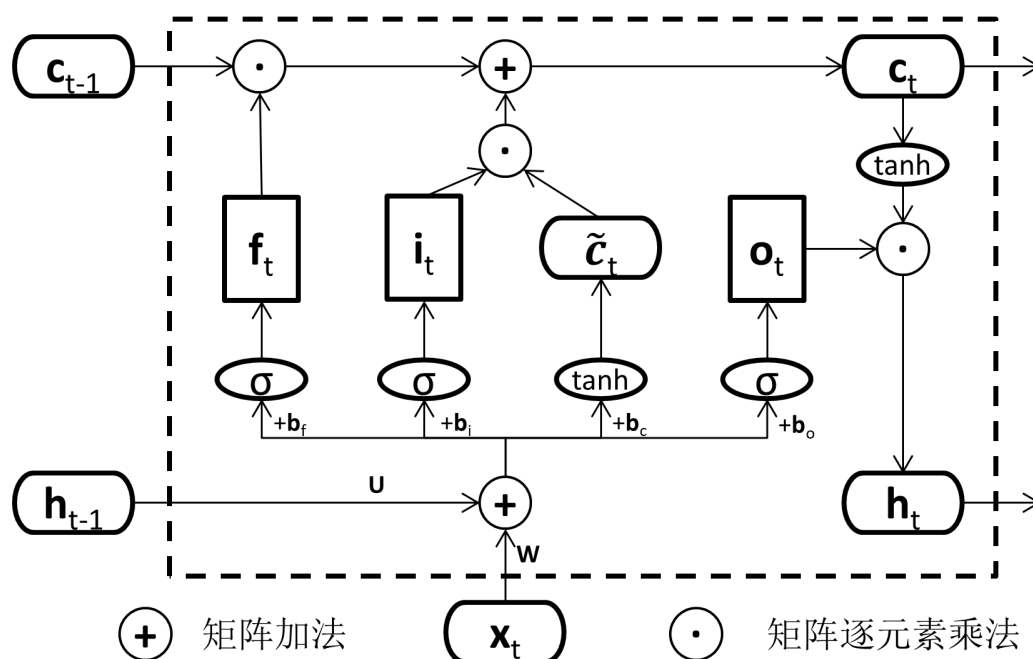


图 2.11: LSTM 隐藏层结点数据流图

信息将被线性的传递给下一时刻的循环单元；

3. 通过出门对当前时刻的内部状态信息的选择作用，得到当前时刻循环单元的外部状态。

2.2.5 LSTM-变体

LSTM 自 1997 年提出以来，有了很多的变体，主要是对各种门的增减和计算方式的改变，主导思想都是模拟人的大脑来让机器进行学习，都可以理解为对人类神经元的简化。下面从对门的增、删、改三个角度介绍三种变体，点到即止，更多内容请移步参考文献。

2.2.5.1 增

ICLR 2019 高分论文前 Top10 《Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks》中提出了一种名为 Ordered Neurons LSTM 的变体，通过对神经元进行排序，引入更高级的输入门 $\tilde{\mathbf{i}}$ 和遗忘门 $\tilde{\mathbf{f}}$ 向量，保证了当更新一个单元时其之后的所有有序单元都将被更新。新引入的 $\tilde{\mathbf{i}}$ 向量中的值从 0 ~ 1 单调递增， $\tilde{\mathbf{f}}$ 向量中的值则从 1 ~ 0 单调递减，它们是通过 `cumsum` 函数和 `softmax` 函数复合组成的一种新的激活函数计算得到，最终计算隐藏层内部状态 \mathbf{c}_t 时结合了新旧忘门和入门共四个门的影响。详细解释请读 [6]。

2.2.5.2 删

回顾计算内部状态的公式2.45:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (2.47)$$

可知把 \mathbf{c}_t 由忘门删选出的信息和入门筛选出的信息组成。如果把 \mathbf{i}_t 看成一个百分比，比如 $i\%$ ，决定 \mathbf{c}_t 中 $i\%$ 的信息，则 \mathbf{c}_t 中还剩下 $(1-i)\%$ 的“存储空间”，为了简化运算，直接用 $\mathbf{1} - \mathbf{i}_t$ 代替 \mathbf{f}_t ，则有：

$$\mathbf{c}_t = (\mathbf{1} - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (2.48)$$

这一简化相当于将忘门与入门耦合了，将两个门合并为一个门，即在原 LSTM 基础上删除了一个门。

2.2.5.3 改

peephole 连接，考虑到 \mathbf{c}_{t-1} 的影响。一种经典的对 LSTM 中门的修改的方式是使用“猫眼儿窥视”（peephole connection，见 [3]），即在计算门的过滤比时考虑上一时刻隐藏层神经元内部状态（ \mathbf{c}_{t-1} ）的影响。比如，求忘门的过滤比时：

$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1} + \mathbf{b}_f) \quad (2.49)$$

可以这样理解上式：信息 \mathbf{c}_{t-1} 想通过忘门 \mathbf{f}_t 进入 \mathbf{c}_t ， \mathbf{f}_t 为了筛选出有用的信息，直接通过猫眼儿窥视 \mathbf{c}_{t-1} 内部到底有多少是水分，有多少是干货，从而决定最终允许 \mathbf{c}_{t-1} 通过的比例，于是在计算 \mathbf{f}_t 时考虑 \mathbf{c}_{t-1} 的影响。

值得一提的是，许多论文在计算门的过滤比时会只在某些门上应用 peephole connection。这种方法可以使得 LSTM 在没有任何短期训练示例的帮助下，学习到长期的由离散时间步长分开的峰值序列之间的细微区别，生成稳定的高度非线性、具有精确时间峰值的序列，适合于时序预测问题（[3]），但它增加了模型参数，使运算复杂度变大。

2.3 章末总结

本章主要介绍了循环神经网络及其变体。从 RNN 的基本作用与结构出发，在此基础上从网络的输入输出角度介绍了 RNN 的四种应用类型；然后，同其他章节类似，详细讲解了 RNN 中参数更新的运算过程，为了优化求导的计算时间复杂度，引入了随时间方向传播的算法，详细讲解了损失函数对隐藏状态 \mathbf{h} 的参数 U 的求导过程，并在 2.1.3.2 章节对参数更新过程中遗留的 $\delta_{t,k}$ 中的两项（ $\frac{\partial \mathcal{L}}{\partial \mathbf{o}_t}$

和 $f'(z_t)$) 的计算做了具体阐述。最后, 对神经网络中常见的梯度爆炸问题与梯度消失问题从 RNN 实现长期记忆功能的角度进行了分析并给出了一些常见的解决方法。

RNN 的变体主要围绕 LSTM 展开描述, LSTM 一节主要介绍了 LSTM 的含义、LSTM 在 RNN 基础上的两个主要改进、LSTM 的三种变体。其中, 两个主要改进是指 LSTM 中引入了门的机制和内部状态的概念, 通过对遗忘、信息输入、信息输出三种活动的模拟, 神经元更具“思维能力”了, 而内部状态以及候选状态的概念是对记忆体或者是对某时刻的总体记忆的模拟, 这让一个神经元更具独立性、完整性了。在此基础上, 从增、删、改三个角度介绍了对循环单元结构的三种改进, 一方面, 是从理论上对人脑的一种模拟, 另一方面, 是联系实际情况所作的合适的调整。

本章总体知识框架图如下所示:

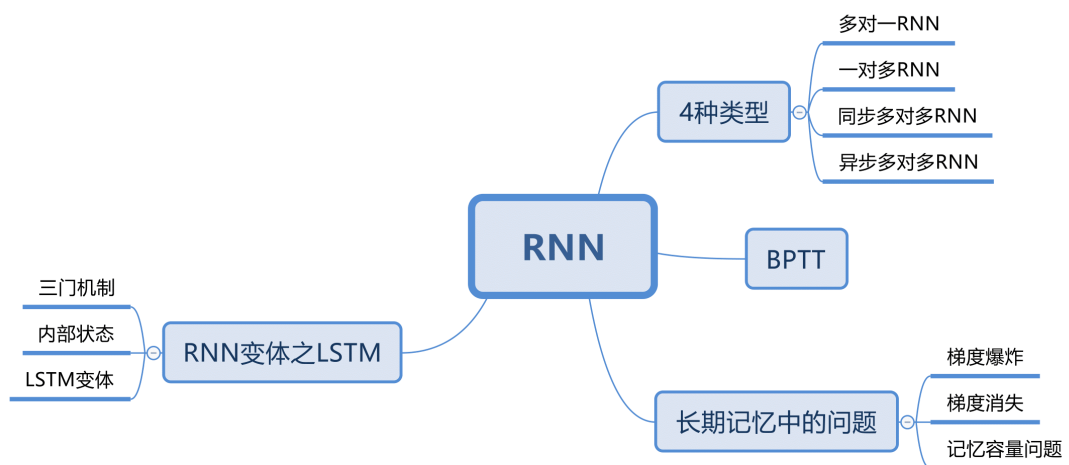


图 2.12: RNN 知识框架

2.4 Advanced Topic

GRUN 门控循环单元网络 (Gated Recurrent Unit Network) 同 LSTM 一样也是一种基于门的 RNN 变体, 它没有引入内部状态的概念, 而是在 RNN 的隐藏状态 (\mathbf{h}_t) 的基础上引入其候选状态 ($\tilde{\mathbf{h}}_t$), 即对当前时刻净输入的一种模拟, 但计算 $\tilde{\mathbf{h}}_t$ 时考虑的因素与 LSTM 中的记忆态的候选状态 ($\tilde{\mathbf{c}}_t$) 不同, 引入了一种重置门 (\mathbf{r}_t) 来决定是否需要考虑接受上一时刻的隐藏状态 (\mathbf{h}_{t-1}) 作为候选状态的组成部分之一, 因为在 GRU 中更新内部状态 (\mathbf{h}_t) 时已经通过一种更新门 (即耦合的入门与忘门) 将 \mathbf{h}_{t-1} 考虑进去了, 再计算候选状态时再考虑一次显得有些多余, 而完全不考虑又显得有些不足, 因为候选状态 ($\tilde{\mathbf{h}}_t$) 毕竟是一种净输入经过简单处理后的候选状态。

DRNN 前面所介绍的都是只有一层隐藏层的 RNN，只有一个神经元沿时间展开形成的层，如果在网络的输入和输出之间考虑到多个神经单元的信息传递作用，就形成了深层循环神经网络（Deep RNN）。如果层与层之间是从上一层到下一层依次地传递信息，则称为堆叠的循环神经网络（Stacked RNN）；如果层与层之间没有直接干扰，一层是顺着时间的流逝而传递信息，另一层是逆着时间的流逝而传递信息，即将当前时刻获取的信息传递给之前时刻的状态，再将两层的记忆综合起来考虑得到输出，则称这种网络为双向循环神经网络（Bidirectional RNN）。看到这里，读者是否还可以设计一些其他类型的深层神经网络？

RecNN 递归神经网络（Recursive Neural Network）是种基于有向无循环图的 RNN。比如，为了获取相邻时刻的输入之间紧密的关系，可以在计算某时刻的隐藏状态（ \mathbf{h}_t ）时考虑到多个时刻的输入信息。此外，LSTM 也可以基于图的机制来实现（[8]）。

参考文献

- [1] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [2] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.
- [3] Felix Gers and Jurgen Schmidhuber. Recurrent nets that time and count. volume 3, pages 189 – 194 vol.3, 02 2000.
- [4] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.
- [5] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [6] Yikang Shen*, Shawn Tan*, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. 2018.
- [7] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.
- [8] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. 2015.
- [9] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- [10] 邱锡鹏. 神经网络与深度学习, 2019.

第 3 章 Attention Mechanism

3.1 为什么需要注意力机制？

大学两年多来，你是否会因老师常说“这门课其实可以分成很多门课来讲，一个学期的课时是远远不够的”而背上一层沉重的包袱？你是否会因自己的毕业去向的选择过多而迷茫？你是否会因各种因素而无法入睡？其实，基于注意力机制的方法，你可以重点学习那门课所要求掌握的知识点而轻松地拿到一个期末好成绩，而不必从这门课所关联的多个领域的历史起源一直学到各个领域的世界前沿课题研究；你可以一头扎进研究生生涯，在某一个很小很小的研究方向上凿出一个个有影响力的成就，而不必把大把大把的时间花在思考自己是否适合读研上；你可以轻闭眼睛集中注意力告诉自己现在要入眠、入眠，而不是胡思乱想。当信息量很大时，通过注意力机制可以快速获取到对解决问题最有帮助的信息，把有限的精力集中在有意义的事情上，从而让生活更加美好。

将注意力机制应用到神经网络中也能达到类似的效果。之前我们所学的卷积神经网络、循环神经网络在理论上具有很强的表达能力，可以捕获长距离依赖关系，但实际运用起来因模型复杂度太大、参数太多，常常要等上好几天才能得到一个结果，不可避免地会遇到对计算的需求突破现有计算能力上限的问题，这使得计算能力多年来一直是限制神经网络发展的瓶颈。为了解决网络容量问题或者说信息超载问题、将有限的计算资源使用在最重要的计算任务上，同时使得长期记忆问题得到更好的实现，注意力机制应运而生。

3.2 从仿生角度认识注意力

如果说神经网络是对人脑的一种简化和模拟，那么注意力机制则是对人的记忆（或者说，人获取信息的方式）的一种模拟。让我们想象一下如场景：小明在考研自习室做英语一的阅读理解题目，因为有计时，所以他注意力高度集中在文章和五个选择题上，以至于旁边有人走进自习室、发出开门和关门声，他都毫无察觉，这时，小明的妈妈打电话过来，手机震动，小明从题目中惊醒，立刻走出自习室接电话。仔细观察这个几秒钟的过程，相信读者很容易发现其中所体现的注意力，但读者能否试着把具体的注意力示例抽象出来，并给其分类呢？

小明在仔细读文章、做题，不受他人干扰，即有意地从周围大量的视觉、听觉、触觉、嗅觉等感官上带来的信息中选择一小部分信息（文章和题目）来重点

处理，同时忽略其他不重要信息，这显然可以是一种注意力，并且是有目标的集中注意力；小明被手机震动惊醒，即无意地中断了做题的过程，去处理接电话的事情，这也是一种注意力，一种受外界刺激所驱动、和当前任务没有直接联系的注意力。基于以上观察，我们可以把注意力分成两大类——前者称为聚焦式注意力（Focused Attention），是自上而下的、有意识的，后者称为显著性注意力（Saliency-Based Attention），是自下而上的、无意识的。

实际上，卷积神经网络中的最大汇聚（max pooling）、循环神经网络中的门控机制就可以近似为是一种自下而上的显著性注意力，当在当前环境下突然接受到一个非常重要的刺激信息时，就把网络的注意力转向这个刺激。本章接下来主要介绍聚焦式注意力，以人做阅读理解题目和机器完成阅读理解任务（主要是翻译）为例，“仿人之注意力，造机器之智慧”，讲解当前主流的几种注意力机制。

3.3 注意力评分函数

以阅读理解任务作为例子，给定一篇长文，对文章的内容进行提问，问题的答案可以由文章的关键词或者关键句子给出。假设提出的问题只和段落中的有限几个句子相关，与段落中的其他句子没有太大的关系。那么我们很自然地可以想到，要是只挑出相关片段让后续的神经网络进行处理，那么后续的神经网络的参数势必会大大减少。

我们把一篇文章进行编码预处理后，用 $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$, $\mathbf{X} \in \mathbb{R}^{d \times n}$ 表示 n 个输入信息的矩阵，其中 $\mathbf{x}_i \in \mathbb{R}^{d \times 1}$ 。如果给定一个查询向量 \mathbf{q} ，那么应该如何挑选合适的输入信息片段呢？为此我们定义一个注意力分数以及注意力评分函数，来表示评定每段输入信息得到注意力程度的高低。

$$s_i = \text{score}(\mathbf{q}, \mathbf{x}_i) \quad (3.1)$$

其中 s_i 是注意力分数， $\text{score}(\mathbf{x}_i, \mathbf{q})$ 是注意力评分函数，注意力评分函数有几种形式。

（1）加性模型

$$\text{score}(\mathbf{q}, \mathbf{x}_i) = \mathbf{v}^T \tanh(\mathbf{W}\mathbf{x}_i + \mathbf{U}\mathbf{q}) \quad (3.2)$$

加性模型实际上是只含有一层隐藏层的前馈神经网络，使用 \tanh 函数作为激活函数。 \mathbf{v} 、 \mathbf{W} 和 \mathbf{U} 是可以通过学习训练的参数。

（2）点积模型

$$\text{score}(\mathbf{q}, \mathbf{x}_i) = \mathbf{x}_i^T \mathbf{q} \quad (3.3)$$

点积模型的复杂度与加性模型相差并不大，由于矩阵乘法的并行化，点积模型计算更快且更节省空间。

(3) 缩放点积模型

$$\text{score}(\mathbf{q}, \mathbf{x}_i) = \frac{\mathbf{x}_i^\top \mathbf{q}}{\sqrt{d}} \quad (3.4)$$

与点积模型相比，缩放点积模型多了一个缩放因子 $\frac{1}{\sqrt{d}}$ ，而 d 正是输入信息的维度。在后面的内容中，注意力评分函数之后会用到 softmax 函数进行计算。当 d 过大的时候，点积模型得到的值比较大，导致 softmax 函数的梯度过小，不利于训练。而缩放点积便可以解决这个问题。

(4) 双线性模型

$$\text{score}(\mathbf{q}, \mathbf{x}_i) = \mathbf{x}_i^\top \mathbf{W} \mathbf{q} \quad (3.5)$$

其中 \mathbf{W} 是可以通过学习训练的参数。其实上面的三种模型中，我们都默认 \mathbf{x} 和 \mathbf{q} 是同样的维度，但双线性模型中的 \mathbf{W} 可以不是方阵，这样的好处是便于不同维度的 \mathbf{x} 与 \mathbf{q} 对齐。

3.4 硬性注意力机制 (Hard Attention Mechanism)

还是类比做阅读理解，给定一个题目，硬性注意力就是选择文中最可能相关的一段进行解题，而不去参考别的段落。在上面，我们提到了注意力分数，那么怎么根据注意力分数来进行选择呢？为了更加客观地进行选择，我们引入概率。使用注意力变量 $z \in \{1, 2, \dots, n\}$ 来表示选择的输入信息的索引位置，也就是说 $z = i$ 表示选择了第 i 个输入信息。那么给定 \mathbf{X} 和 \mathbf{q} ，那么选择第 i 个输入信息的概率为 α_i ，即

$$\begin{aligned} \alpha_i &= p(z = i | \mathbf{X}, \mathbf{q}) \\ &= \frac{\exp(s_i)}{\sum_{j=1}^n \exp(s_j)} \end{aligned} \quad (3.6)$$

我们很容易发现，这是一个 softmax 的过程，可以很容易地表示概率。

那么设最后传入后续神经网络的注意力信息为 \mathbf{a} ，那么硬性注意力机制的实现有如下两种方式：

(1) 选取概率最高的输入信息，即

$$\mathbf{a} = \mathbf{x}_j \quad (3.7)$$

$$j = \arg \max_{i=1}^n \alpha_i \quad (3.8)$$

其中 j 是概率最大的输入信息的下标。

(2) 得到选取每个输入信息对应的概率后，通过随机取样进行输入信息的选取。

通过上面内容，可以发现硬性注意力机制，从直观上很符合人类对于注意力的理解。但硬性注意力机制通过最大采样或者随机采样来选择输入信息，这会导致最后的损失函数和注意力分布之间的函数关系不可导，那么这就无法利用通用的反向传播方法进行神经网络的训练了。硬性注意力机制需要通过强化学习方法对其训练。为了易于训练，可以使用软性注意力机制来代替硬性注意力机制。

3.5 软性注意力机制 (Soft Attention Mechanism)

其实在阅读理解中，有些题目的解答并非只看某一段就可以了，甚至需要纵观全局才能进行解题。软性注意力机制与这种解题方式类似。同样设传入后续神经网络的注意力信息为 \mathbf{a} ，选择输入信息的概率为 α_i ，基于公式3.6，我们可以给出软性注意力机制的实现方式：

$$\mathbf{a} = \sum_{i=1}^n \alpha_i \mathbf{x}_i \quad (3.9)$$

在这里，注意力信息 \mathbf{a} 其实就是输入信息的加权平均，用概率大小来表示该段输入的重要程度。

软性注意力机制的示意图如图3.1所示。

我们额外介绍另一种与软性注意力机制很类似的注意力机制——全局注意力机制 (Global Attention Mechanism)，其实现方式如下：

$$\mathbf{c} = \sum_{i=1}^n \alpha_i \mathbf{x}_i \quad (3.10)$$

$$\mathbf{a} = \tanh(\mathbf{W}_c(\mathbf{c} \oplus \mathbf{q})) \quad (3.11)$$

其中 \mathbf{c} 是上下文信息变量， \mathbf{W}_c 为权重矩阵， \oplus 意为向量拼接。简单来说，全局注意力机制就是在软性注意力机制的基础上对查询向量 \mathbf{q} 进行拼接后加了一层神经网络，作为后续神经网络的输入。

与硬性注意力机制相比，软性注意力机制或者全局注意力机制最大的优点便

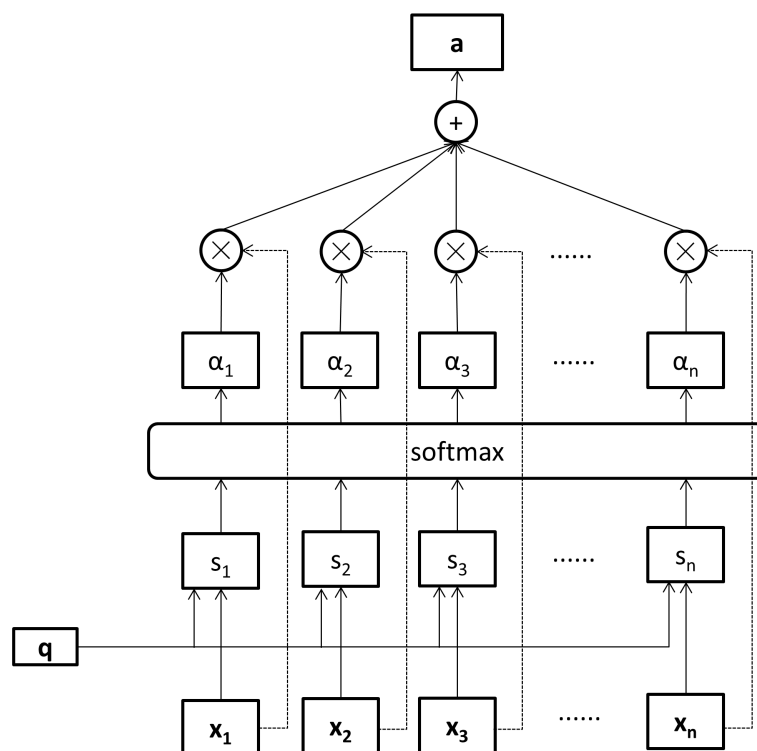


图 3.1: 软性注意力机制示意图

是函数关系可导，可以通过反向传播算法进行训练。但是存在一个不足之处便是，当输入信息的规模越加庞大的时候，对所有输入信息进行加权平均所需的计算力是昂贵的，且加权平均后得到的注意力信息，也很难说真正有效。在下一节，我们介绍一种软硬结合的注意力机制——局部注意力机制，在软硬之间折中，兼具两者优点。

3.6 软硬结合：局部注意力机制（Local Attention Mechanism）

继续类比做阅读理解，有些题目的解答的不需要通读全文才能做出解答，但只局限于某一段文字也无法进行解答。那么如果进行折中，不进行全文的阅读，只阅读其中几段文字来进行解题，或许准确性和速度都会有不错的提升。局部注意力机制就是与这种方式类似，软硬结合。局部注意力机制，既然是关注于局部的输入信息，那么这个局部的窗口应该怎么确定呢？那我们设这个局部窗口的中心位置位置为 p ，整个窗口的范围是 $[p - D, p + D]$ ，其中 D 是根据实际情况通过经验挑选的长度，是一个正整数。关于局部注意力模型的实现，我们介绍两种模型：

(1) 单调对齐模型（Monotonic alignment model），我们可以简称为 local-m 模

型。此模型假设有序且不同的查询变量 \mathbf{q} 按顺序大致与输入信息 \mathbf{x}_i 相关，那么 p 则为：

$$p = i \quad (3.12)$$

那么此时选择第 i 个输入信息的概率与软硬注意力机制中的 α_i 类似，即

$$\alpha_i = \frac{\exp(s_i)}{\sum_{j=\max(1, p-D)}^{\min(n, p+D)} \exp(s_j)} \quad (3.13)$$

其中 $\max(1, p-D)$ 意为取两者中的最大值， $\min(n, p+D)$ 意为取两者中的最小值。 $i \in [\max(1, p-D), \min(n, p+D)]$ ，这意味着并不用计算所有输入信息被挑选的概率。

(2) 预测对齐模型 (Predictive alignment model)，我们可以简称为 local-p 模型。此模型通过神经网络预测位置 p ，具体如下：

$$p = n \cdot \text{sigmoid}(\mathbf{v}_p^\top \tanh(\mathbf{W}_p \mathbf{q})) \quad (3.14)$$

这里的 n 是输入信息的数量， sigmoid 这函数保证了取值在 $(0, 1)$ 之间，这确保了 p 在 $(0, n)$ 之间。然后 \mathbf{v}_p 是一个参数列向量， \mathbf{W}_p 是一个参数矩阵，他们都是通过学习训练的。

而在 local-p 模型中，选择第 i 个输入信息的概率与之前的有所不同，为了尽可能使位置 p 附近的输入信息被挑选的概率尽可能大些，以位置 p 为中心放置高斯分布，具体如下：

$$\alpha_i = \frac{\exp(s_i)}{\sum_{j=\max(1, \lceil p-D \rceil)}^{\min(n, \lfloor p+D \rfloor)} \exp(s_j)} \exp\left(-\frac{(i-p)^2}{2\sigma^2}\right) \quad (3.15)$$

其中 $\lceil p-D \rceil$ 意为 $p-D$ 的向上取整， $\lfloor p+D \rfloor$ 意为 $p+D$ 的向下取整， $i \in [\max(1, \lceil p-D \rceil), \min(n, \lfloor p+D \rfloor)]$ ，一般来说，设置 $\sigma = \frac{D}{2}$ 。

但不管是 local-m 模型还是 local-p 模型，上下文信息变量的计算公式如下：

$$\mathbf{c} = \sum_{i=\max(1, \lceil p-D \rceil)}^{\min(n, \lfloor p+D \rfloor)} \alpha_i \mathbf{x}_i \quad (3.16)$$

简单来说，上下文信息变量就是对窗口 $[p-D, p+D]$ 内的输入信息进行加权平均。

与全局注意力机制相似，最后我们可以得到注意力信息为

$$\mathbf{a} = \tanh(\mathbf{W}_c(\mathbf{c} \oplus \mathbf{q})) \quad (3.17)$$

其中 \oplus 为向量拼接。

由于 D 可以自行调节，即使输入信息的规模越加扩大，基本不会增加额外的计算，局部注意力机制充分保留了硬性注意力机制和软性注意力机制的优点。

3.7 自注意力机制 (Self-attention Mechanism)

自注意力机制由《Attention Is All You Need》[5] 提出，这篇论文为了降低总的计算复杂度、提高算法的可并行成分、实现序列处理问题中更长期的记忆功能（或者说学习到更长期的依赖关系，见2.1.4），完全抛弃了我们前面所学的 CNN 和 RNN，取而代之的是，在基于“堆叠的编码器-解码器”（Encoder and Decoder Stacks）这一经典模型上，采用了自注意力机制和用于获取位置信息的全连接前馈神经网络。本节不对论文中的各种方法、模型和证明做详细介绍，而是重点解释论文中所用到的关于注意力的方法，同时也忽略了论文中“注意力机制是怎么应用到模型中”的相关内容。笔者强烈推荐这篇论文给初学者，希望本节的内容能激发读者对注意力的兴趣，为读者能读懂这篇论文铺路。

3.7.1 键值对注意力 (Key-Value Attention)

前面三小节主要介绍了三种注意力机制，从我们做阅读理解的角度用三句话来简略的概括就是：依据评分最高（最重要）的一段或者几段来答题（硬性注意力）、依据每一段的评分（重要程度）从每一段抽取不同量的信息来答题（软性注意力）、结合全文总体内容的评分找到切入点再联系切入点的上下文去答题（局部注意力）。

然而，我们做阅读理解时真的会去先花大量时间给每一段算出一个评分以表示其重要程度、然后再去做题吗？非也！我们是给每一段总结出关键词句并以此来代替上文所介绍的数值型评分的，大致步骤分以下四步：

1. 粗读每一段 v_i 的内容（一共 n 段），给其总结出几个关键词或者句子 k_i ，得到一对对键值对向量，进而构成“键矩阵” K 和“值矩阵” V ，即 $(K, V) = [(k_1, v_1), \dots, (k_n, v_n)]$ ；
2. 读第一题 q_1 ，将 q_1 与每一段的关键词句 v_i 一一做比较，得到一系列模糊的相关程度，用 $score(q_i, k_1)$ 表示，最终可以得到 $score(q_1, K) = [score(q_1, k_1), score(q_1, k_2), \dots, score(q_1, k_n)]$ ；
3. 当某一段的相关程度 $score(q_1, k_i)$ 很高时，就从这一段 v_i 里多读几句、多获取一些信息，当很低时就只看首尾句、少获取一些信息，最终综合从每一段获取的信息得到对问题 q_1 的答案；
4. 对第二题、第三题... 一直到最后一题的做法和第一题一样，最终完成这篇阅读理解。

将以上这种做题方法应用到机器学习中，可得到如下注意力函数：

$$\text{Attention}(\mathbf{q}_i, (\mathbf{K}, \mathbf{V})) = \text{softmax}(\text{score}(\mathbf{q}_i, \mathbf{K}))\mathbf{V} \quad (3.18)$$

其中， $\text{score}(\mathbf{q}_i, \mathbf{K})$ 是一种可选的打分函数，比较的是问题向量和键矩阵，而不是之前的问题向量和输入向量。上式可理解为将问题向量与所有的键向量相比较后通过 softmax 函数将所有得出的评分归一化，并将评分作用到对应的值向量上，基于此，可以将上式的键矩阵 \mathbf{K} 按行展开：

$$\begin{aligned} \text{Attention}(\mathbf{q}_i, (\mathbf{K}, \mathbf{V})) &= \sum_{j=1}^n \alpha_j \mathbf{v}_j \\ &= \sum_{j=1}^n \frac{\exp(\text{score}(\mathbf{q}_i, \mathbf{k}_j))}{\sum_{p=1}^n \exp(\text{score}(\mathbf{q}_i, \mathbf{k}_p))} \mathbf{v}_j \end{aligned} \quad (3.19)$$

接下来让我们从矩阵维度的角度来验证上式的正确性并加深理解。设输入信息条数（文章的段落数）为 N ，每条输入信息的维度为 d_v ，对应的关键词句的维度为 d_k ，则值矩阵 $\mathbf{V} \in \mathbb{R}^{N \times d_v}$ ，键矩阵 $\mathbf{K} \in \mathbb{R}^{N \times d_k}$ ，每个问题向量应该与键矩阵能够相比较、匹配，故维度也是 d_k ，即 $\mathbf{q}_i \in \mathbb{R}^{1 \times d_k}$ 。

那么，式3.19中 $\exp(\text{score}(\mathbf{q}_i, \mathbf{k}_j))$ 是一个表示相关性程度或者评分的数值，每一个段落信息 \mathbf{k}_j 都与问题向量 \mathbf{q}_i 进行比较、匹配，得到 n 个评分，将这一系列评分通过 softmax 进行归一化后得到的值与维度为 $\mathbb{R}^{1 \times d_v}$ 的 \mathbf{v}_j 上每个数值相乘，得到的是从第 j 段信息或者从第 j 个输入中提取的有用信息。最后，将从每一段信息中提取的有用信息汇总，得到 N 段文字对问题 \mathbf{q}_i 的回答向量 $\text{Attention}(\mathbf{q}_i, (\mathbf{K}, \mathbf{V}))$ ，维度仍为 $\mathbb{R}^{1 \times d_v}$ 。显然，这获取了文章中每一段文字以及每一段的每一个维度上的信息。值得注意的是，这里的回答向量并不是最终模型给出的答案，它可以被看做是一个比较原始的答案，还需要思考、加工后才能成为最终的输出答案。

以上即是对键值对注意力机制的一种基于阅读理解的解释，既然有了公式，不如可视化一下，把公式背后的数据流显示出来，一目了然，见图3.2。

可见，键值对的注意力机制和软性注意力机制很相似——当不引入键矩阵 \mathbf{K} 或者说当 $\mathbf{K} = \mathbf{V}$ 时，键值对注意力就退化为软性的注意力机制。

3.7.2 多头注意力 (Multi-head Attention)

不知读者是否注意到，前面的问题向量 \mathbf{q}_i 一直带着个似乎可有可无的下标 i ，表示第 i 个题目。读者是不是可以大胆想象一下，结合做阅读理解题的经验，还可以有怎样的注意力呢？

试想，我们做题时有时候是不是为了加快解题速度，一次读多个题目，带着问题去读文章，而不是像上文键值对注意力一样机器式的做一题答一题再做一题

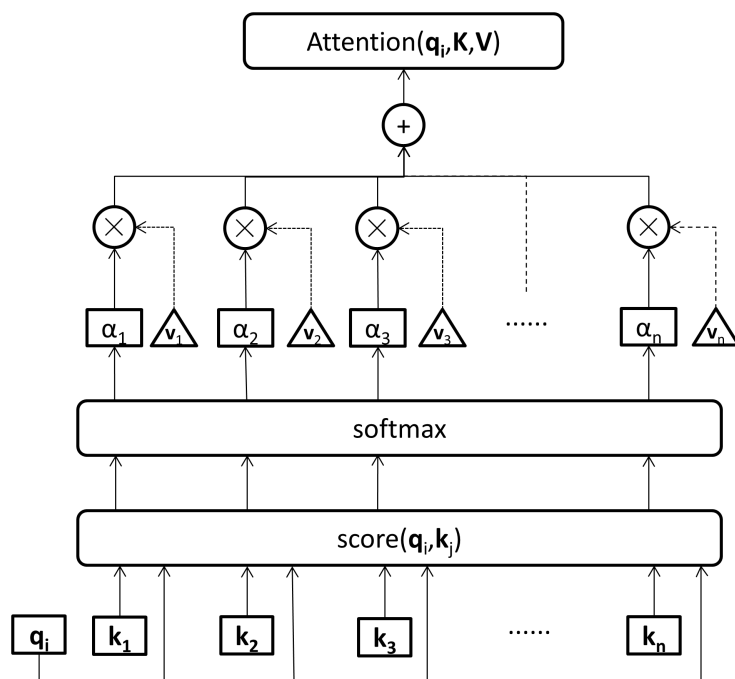


图 3.2: 键值对注意力机制流程图

再答一题？由此，便有了注意力机制中的多头注意力，每一头注意力关注不同的题目，从而能够并行化的答题，仿佛每一时刻有多个大脑在做题。于是， Attention 函数可以同时处理多个问题组成的矩阵 \mathbf{Q} 了：

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Attention}(\mathbf{q}_1, \mathbf{K}, \mathbf{V}) \oplus \cdots \oplus \text{Attention}(\mathbf{q}_n, \mathbf{K}, \mathbf{V}) \quad (3.20)$$

\oplus 是将向量拼接成矩阵的符号，一般是按列拼接，但这里是按行拼接，每一行对应一个问题的阶段性解。

继续激发我们的想象力，多头难道就只能是对多个问题的注意吗？其实，多头注意力并不完全就是指关注多个问题， \mathbf{Q} 、 \mathbf{K} 、 \mathbf{V} 都有各自的维度 (d_k 、 d_k 、 d_v)，“多头”也可以指每一头关注一部分维度呀！这，就是下一节将介绍的自注意力机制的核心！聪明的你，是否可以将以上两种机制结合起来，设计出一种属于自己的注意力了呢？

3.7.3 Attention Is All You Need

前面两小节介绍了键值对注意力和多头注意力，似乎与本节的主题自注意力机制没什么关系。其实，自注意力机制本质上就是键值对注意力和多头注意力机制结合在一起以高效实现长期记忆功能的一种综合应用，下面就让我们正式学习论文 [5] 所提出的自注意力机制。

首先, 从3.3节中选择一种合适的基于键值对的缩放点积评分函数:

$$\text{score}(\mathbf{Q}, \mathbf{K}) = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \quad (3.21)$$

其中, $\mathbf{Q} \in \mathbb{R}^{m \times d_k}$, 表示 m 个查询, 每个查询有 d_k 个维度, $\mathbf{K} \in \mathbb{R}^{n \times d_k}$, 表示一共有 N 个段落或者说输入信息, 每个段落总结出了 d_k 个维度的关键信息。二者相乘后 $\mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{m \times n}$, 实现了将每个问题从 d_k 个角度与每个关键信息 K 的比较, $\mathbf{Q}\mathbf{K}_{ij}^T$ 表示从关键信息来看第 j 段的内容与第 i 个问题的匹配程度。

之所以除以 $\sqrt{d_k}$, 论文中的解释是当 d_k 的值很大时, 将 $\mathbf{Q}\mathbf{K}^T$ 矩阵的值看成一个随机变量后, 每一个随机变量就是 d_k 个随机变量的乘积和, 于是其方差会变大, 从而其增长幅度很大, 这会将 softmax 函数推向梯度很小的区域, 不利于梯度下降, 而除以 $\sqrt{d_k}$ 后可以在一定程度上抵消这种影响。

选定了评分函数后的注意力函数为:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \quad (3.22)$$

其中, $\text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})$ 按行对评分进行归一化, 每一行的和是 1, 其维度仍为 $\mathbb{R}^{m \times n}$, 值矩阵 $\mathbf{V} \in \mathbb{R}^{n \times d_v}$, 二者相乘即表示依据每个问题与每个段落的匹配程度, 从每一段输入的 d_v 个维度中获取一定比例的信息, 得到的每一行 $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_i$ 代表第 i 个问题在 d_v 个维度上获得的初始答案。

接下来可以运用针对维度的多头注意力机制了, 每一头关注问题与段落的不同维度, 从而实现将问题“分治”, 化为一个个简单些的小问题, 同时也将段落的维度分层, 针对同一个问题下不同的小问题选择对应层上的维度的段落信息来答题。假设一共有 h 个“头”, 模型的输入向量(经过预处理后)和输出向量的维度为 d_{model} , 每一头平均分配到 $\frac{d_{model}}{h}$ 个维度上的输入并负责 $\frac{d_{model}}{h}$ 个维度上的输出。论文[5]中做了 $d_k = d_v = d_{model}$ 的简化, 从而每一头注意力 h_i 的维度统一都是 $\frac{d_{model}}{h}$ 。在此, 不失一般性, 每一头注意力 h_i 上与 $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ 对应的 $\mathbf{Q}'_i, \mathbf{K}'_i, \mathbf{V}'_i$ 矩阵的维度为: $\mathbf{Q}'_i \in \mathbb{R}^{m \times \frac{d_k}{h}}, \mathbf{K}'_i \in \mathbb{R}^{n \times \frac{d_k}{h}}, \mathbf{V}'_i \in \mathbb{R}^{n \times \frac{d_v}{h}}$ (d_k, d_v 仍是不使用多头注意力机制时的维度, 与前两小节符号保持一致)。

那么, 如何得到将 $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ 降维后每一头 h_i 上的 $\mathbf{Q}'_i, \mathbf{K}'_i, \mathbf{V}'_i$? 很简单, 给每一头 h_i 赋予三个参数 W_i^Q, W_i^K, W_i^V , 用于对 $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ 作线性变换即可:

$$\begin{aligned} \mathbf{Q}'_i &= \mathbf{Q}W_i^Q \\ \mathbf{K}'_i &= \mathbf{K}W_i^K \\ \mathbf{V}'_i &= \mathbf{V}W_i^V \end{aligned} \quad (3.23)$$

易知: $W_i^Q \in \mathbb{R}^{d_k \times \frac{d_k}{h}}, W_i^K \in \mathbb{R}^{d_k \times \frac{d_k}{h}}, W_i^V \in \mathbb{R}^{d_v \times \frac{d_v}{h}}$ 。将 $\mathbf{Q}'_i, \mathbf{K}'_i, \mathbf{V}'_i$ 的表达式传入每一头

上的注意力函数，有：

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad (3.24)$$

$head_i$ 只关注 $\frac{1}{h}$ 的维度，即 $head_i \in \mathbb{R}^{m \times \frac{d_v}{h}}$ ，故最后直接将每个注意力头按列拼接：

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O \quad (3.25)$$

$W^O \in \mathbb{R}^{d_v \times d_{model}}$ ，是将初始答案的维度转化为欲输出维度的网络参数。容易发现上式的每个 head 的计算互不干扰，可以并行计算。

以上就是自注意力机制的核心内容：利用键值对使得模型更加科学合理，使用缩放点积评分函数在计算复杂度较低、可以应用高度优化并行实现的情况下避免局部梯度过小问题，采用多头注意力实现并行化计算每一层降低了维度后的注意力函数。

值得一提的是，自注意力机制没有使用 CNN 和 RNN，基于前面的公式得到的评分没有考虑到 (K, V) 中行与行之间前后位置排列的顺序，也就是将 (K, V) 中对应的两行互换后得到的结果是一样的，因此论文 [5] 中使用自注意力机制前加入了位置编码信息来捕捉重要的序列信息。当然，自注意力机制也并不是一定要与 CNN 或 RNN 分道扬镳，论文 [4] 就将二者结合起来实现了很好的效果，本章的 Advanced Topic 中将其进行简单介绍。

3.8 章末总结

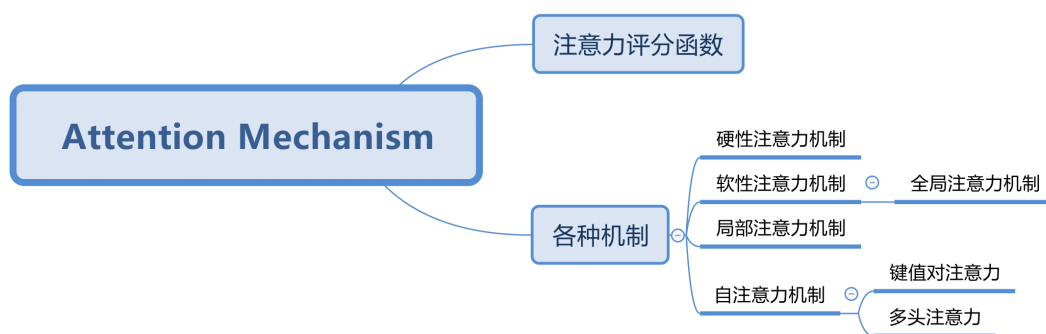


图 3.3: Attention Mechanism 知识框架

本章首先结合现实生活介绍了注意力机制的普遍存在，进而引入注意力机制在机器学习中的作用，随后从仿生的角度去认识注意力，将注意力分成了两大类——自上而下的聚焦式注意力和自下而上的显著性注意力。

显著性注意力类似门控机制的方法前面章节已有介绍，本章着重介绍比较容

易理解的四种聚焦式注意力机制，通过围绕现实生活中人做阅读理解题的例子，从本质上解释了四种聚焦式注意力机制所关注的不同点。

为了模拟人脑的决策判断过程、确定注意点或者注意方向，于是引入注意力评分函数来确定两组信息之间的相关程度，本章的四种注意力机制都是基于评分函数实现的。常见的评分模型有加性模型、点积模型、缩放点积模型、双线性模型，针对具体的问题选择不同的评分模型对注意力机制的效果有很大影响。

做阅读理解时快速锁定最有可能找到答案的一段或几段信息，比如题目中明确说了是从第3段至第7段得出答案，于是就只关注第3至7段，这就是种硬性注意力机制。

当问及通观全文可以看出作者对某某事是什么态度时，我们要综合每一段信息及其重要程度去理解，这时就用到了软性注意力机制。

有时我们做题时先从前几段比如1-3段得出第一题的答案，而后从5-7段、8-11段等等分别得出第二题、第三题以及后续题的答案，这就体现了一种结合软性硬性注意力、假设每题的答案在文章中是按阅读顺序分布的模型，即局部注意力机制中的单调对齐模型；而当根据题意而去判断答案最有可能出现的段落，然后结合从那段的上下文来得出答案时，就体现了局部注意力机制中的预测对齐模型。

本章介绍的第四种聚焦式注意力——自注意力机制，是对键值对注意力与多头注意力的综合运用。键值对注意力就类似给每个段落总结出关键信息，基于关键信息和问题得到评分，再运用软性注意力机制解题；多头注意力机制就类似先把所有题目理解透再去阅读，边读边解题，让解题并行化，同时做所有的题目。自注意力机制深化了多头注意力机制，将一个问题分解成多个子问题，即将问题向量的维度分层，每一头注意力关注对应的层上的维度，从而实现并行化运行。

至此，读者是否发现结合做阅读理解题的方法来理解机器学习中的注意力机制是一个很有效的方法、甚至觉得注意力机制的所有机理就是以做题的方法为依据而产生的？其实，只要我们留心观察，很多学术上的方法模型都可以在现实生活中找到它们的原型或者说“出处”！

3.9 Advanced Topic

Pointer Networks Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015. 指针网络，一种软性的自上而下注意力机制，也是序列到序列模型，输出序列是输入序列的下标（索引）。

Structured Attention Networks Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured attention networks. arXiv preprint arXiv:1702.00887, 2017 结构化注意力，类比小学中学语文课上给好几页长的散文、故事分段

落层次，“形散神不散”

DiSAN [4]

参考文献

- [1] Karim Ahmed, Nitish Shirish Keskar, and Richard Socher. Weighted transformer network for machine translation. *arXiv preprint arXiv:1711.02132*, 2017.
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [3] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
- [4] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. DiSAN: Directional self-attention network for RNN/CNN-free language understanding. page 10, 2018.
- [5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. pages 5998–6008, 2017.
- [6] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.
- [7] 邱锡鹏. 神经网络与深度学习, 2019.

第 4 章 Elegant \LaTeX 系列模板介绍

值此版本发行之际，我们 Elegant \LaTeX 项目组向大家重新介绍一下我们的工作，我们致力于打造一系列美观优雅简便的模板方便用户使用 Elegant \LaTeX 系列模板目前由 **ElegantNote**, **ElegantBook**, **ElegantPaper** 组成，分别用于排版笔记，书籍和工作论文这些子项目的名词是一体的，请在这些名词的时候不要将其断开（如 **Elegant Note** 是不正确的写法）并且，Elegant \LaTeX Book 指的即是 **ElegantBook**

最新版本下载地址：[Github:ElegantBook/releases](https://github.com/ElegantBook/releases) 本文将介绍本模板的一些设置内容以及基本使用方法如果您在使用此模板，欢迎把您使用此模板制作的成品发一份给我们，谢谢！如果您有其他问题，建议或者意见，欢迎联系我们

4.1 ElegantBook 更新说明

在这几年间，我们收到了很多用户的反馈，主要的问题涉及到字体安装，编码支持，定理浮动，定理跨页，交叉引用等等我们思前想后，原先让用户安装字体以追求视觉上的美观并不完美，用户陷入了巨大的麻烦，这违背了我们的模板初衷因此我们在新版中删除了这部分，用户无需安装任何字体让我们来看下此次 **ElegantBook** 模板 3x 更新的主要内容有：

1. 删除了自定义字体设置，改用 **ctex** 宏包或者系统默认字体；
2. 模板拆分为中英文模式 (**lang**=cn/en)；
3. **PDF \LaTeX** 与 **Xe \LaTeX** 支持；
4. 使用 **tcolorbox** 宏包改写定理类环境，可跨页；
5. 定理类环境名字更新，修复定理环境交叉引用；
6. 颜色名字更新，统一链接颜色；
7. 重新绘制 Elegant \LaTeX 的 Logo；
8. 更新封面与装饰物，删除水印；
9. 修正附录相关内容；
10. 增加灰色主题 **color**=plain；
11. 增加代码高亮；
12. 美化列表环境.

第 5 章 ElegantBook 设置说明

5.1 编译方式

本模板基于基础的 book 文类，所以 book 的选项对于本模板也是有效的默认编码为 UTF-8，推荐使用 T_EX Live 编译本文编写环境为 Win10 (64bit) + T_EX Live 2018，支持 PDFLaTeX 以及 XeLaTeX 编译

5.1.1 选择 PDFLaTeX 编译

如果你使用 PDFLaTeX 编译，默认的 Computer Modern 字体被换成了 newtx 系列字体，默认的字体字号是 12 pt 关于字体设置的宏包主要用到了：

- newtxtext 用于文档正文字体，类似于 Times New Roman 字体
- newtxmath 用于数学字体，搭配 newtxtext 非常合适
- FiraMono 用于打字机字体，并使用了 scale=07 选项
- ctex 用于中文字体设置，并使用了 scheme=plain 选项

一次完整的编译：PDFLaTeX -> BibTeX -> PDFLaTeX*2

5.1.2 选择 XeLaTeX 编译

如果你选择 XeLaTeX 编译的话，那么设置字体的宏包为 fontspec 和 xeCJK 由于模板中使用的字体是 Windows 中的字体，所以如果你使用其他操作系统，比如 Linux 或者 Mac OS，那么你需要把所用字体替换为你系统中的字体设置字体的命令：

```
\RequirePackage{fontenc}
\RequirePackage[no-math]{fontspec}
\setmainfont{Times New Roman}[NFSSFamily=ntx1f]
\setsansfont{Arial}
%\setmonofont[Scale=07]{Courier New}
\RequirePackage{xeCJK}
\RequirePackage{xunicode}
\setCJKmainfont[BoldFont={SimHei},ItalicFont={KaiTi}]{
  SimSun}
```



```

\setCJKsansfont[BoldFont={SimHei},ItalicFont={KaiTi}]{
  KaiTi}
\setCJKmonofont[BoldFont={SimHei},ItalicFont={KaiTi},Scale
  =07]{Microsoft YaHei}
\XeTeXlinebreaklocale "zh"
\XeTeXlinebreakskip = 0pt plus 1pt minus 0.1pt
\RequirePackage{newtxmath}

```

一次完整的编译: XeLaTeX -> BibTeX -> XeLaTeX*2

5.2 语言模式

本模板内含两套语言环境, 改变语言环境会改变图表标题的引导词(图, 表), 文章结构词(比如目录, 参考文献等), 以及定理环境中的引导词(比如定理, 引理等) 不同语言模式的启用如下:

```

\documentclass[cn]{elegantbook}
\documentclass[lang=cn]{elegantbook}
\documentclass[en]{elegantbook}
\documentclass[lang=en]{elegantbook}

```

注: 不管选用中文环境(`lang=cn`) 还是英文环境(`lang=en`) 均可输入中文 另外如果在笔记中使用了抄录环境(`lstlisting`), 并且其中包括了中文, 请务必使用 XeLaTeX 编译

5.3 颜色主题

本模板内置 4 组颜色主题, 分别为 `green` (默认) `cyanblueplain`, 另外还有一个自定义的选项 `nocolor` 调用颜色主题 `green` 的方法为

```

\documentclass[green]{elegantbook} %or
\documentclass[color=green]{elegantbook}

```

其中 `plain` 主题为全灰色如果需要自定义颜色的话请选择 `nocolor` 选项或者使用 `color=none`, 然后在导言区定义 `mainsecondthird` 颜色, 具体方法如下:

```

\definecolor{main}{RGB}{70,70,70}
\definecolor{second}{RGB}{115,45,2}
\definecolor{third}{RGB}{0,80,80}

```

表 5.1: ElegantBook 模板中的颜色主题

	green	cyan	blue	主要使用的环境
main	<div></div>	<div></div>	<div></div>	definition
second	<div></div>	<div></div>	<div></div>	theorem lemma corollary
third	<div></div>	<div></div>	<div></div>	proposition

5.4 章标题显示风格

本模板内置 2 套章标题显示风格，包含 hang（默认）与 display 两种风格，区别在于章标题单行显示（hang）与双行显示（display），本说明使用了 hang 调用方式为

```
\documentclass[hang]{elegantbook} %or
\documentclass[titlestyle=hang]{elegantbook}
```

5.5 数学环境简介

- 在我们这个模板中，定义了四大类环境
- 定理类环境，包含标题和内容两部分，全部定理类环境的编号均以章节编号根据格式的不同分为 3 种
 - definition 环境，颜色为 main;
 - theoremlemmacorollary 环境，颜色为 second;
 - proposition 环境，颜色为 third
 - 示例类环境，有 exampleexerciseproblem 环境（对应于例，练习，例题），自动编号，编号以章节为单位
 - 证明类环境，有 proofnote 环境，特点是，有引导符或者结尾符，note 环境有引导符号，proof 环境有证明完毕符号
 - 结论类环境，有 conclusionassumptionproperty, remarksolution 环境，三者均以粗体的引导词为开头，和普通段落格式一致

5.5.1 定理类环境的使用

由于本模板使用了 tcolorbox 宏包来定制定理类环境，所以和普通的定理环境的使用有些许区别，定理的使用方法如下：

```
\begin{theorem}{<theorem name>}{<label>}
```




```
The content of theorem
\end{theorem}
```

第一个必选项 <theorem name> 是定理的名字，第二个必选项 <label> 是交叉引用时所用到的标签，交叉引用的方法为 \ref{thm:label} 请注意，交叉引用时必须加上前缀 thm:效果如下：

定理 5.1: <theorem name>

The content of theorem

♡

其他相同用法的定理类环境有：

表 5.2: 定理类环境

环境名	标签名	前缀	交叉引用
definition	label	def	\ref{def:label}
theorem	label	thm	\ref{thm:label}
lemma	label	lem	\ref{lem:label}
corrlary	label	cor	\ref{cor:label}
proposition	label	pro	\ref{pro:label}

5.5.2 其他数学环境的使用

其他三种数学环境因为没有选项，可以直接使用，比如 example 环境的使用：

```
\begin{example}
  This is the content of example environment
\end{example}
```

效果如下：

例 5.1: This is the content of example environment

这几个都是同一类环境，区别在于

- 示例环境（example）练习（exercise）与例题（problem）章节自动编号；
- 注意（note）环境有提醒引导符，证明（proof）环境有证明结束符；
- 结论（conclusion）等环境都是普通段落环境，引导词加粗

5.6 封面和徽标

本模板使用的封面图片来源于 [pixabaycom](#)¹，图片完全免费，可用于任何场景封面图片的尺寸为 1280 × 1024，更换图片的时候请严格按照封面图片尺寸进行

¹感谢 ChinaTeX 提供免费图源网站，另外还推荐 [pexelscom](#)



裁剪推荐一个免费的在线图片裁剪网站 befunky.com

本文用到的 Logo 比例为 1:1，也即正方形图片，在更换图片的时候请选择合适的图片进行替换

5.7 列表环境

本模板借助于 `tikz` 定制了 `itemize` 和 `enumerate` 环境，其中 `itemize` 环境修改了 3 层嵌套，而 `enumerate` 环境修改了 4 层嵌套（仅改变颜色）示例如下

- | | |
|--------------------------|-----------------------------|
| • first item of nesti; | 1. first item of nesti; |
| • second item of nesti; | 2. second item of nesti; |
| • first item of nestii; | (a). first item of nestii; |
| • second item of nestii; | (b). second item of nestii; |
| • first item of nestiii; | I. first item of nestiii; |
| • second item of nestiii | II. second item of nestiii |

5.8 参考文献

此模板使用了 `BibTeX` 来生成参考文献，默认使用的文献样式（`bib style`）是 `aer` 参考文献示例：[?] 使用了中国一个大型的 P2P 平台（人人贷）的数据来检验男性投资者和女性投资者在投资表现上是否有显著差异

你可以在谷歌学术，Mendeley，Endnote 中获得文献条目（`bib item`），然后把它们添加到 `referencebib` 中在文中引用的时候，引用它们的键值（`bib key`）即可注意需要在编译的过程中添加 `BibTeX` 编译如果你想在参考文献中添加未引用的文献，可以使用

```
\nocite{EINAV2010,Havrylchuk2018}
```

5.9 添加序章

如果你想在第一章前面添加序章，不改变原本章节序号，你可以在第一章内容前面使用

```
\chapter*{序章}
\addcontentsline{toc}{chapter}{序章}
\markboth{序章}{}
序章的内容
```

第 6 章 ElegantBook 写作示例

6.1 Lebesgue 积分

在前面各章做了必要的准备后，本章开始介绍新的积分在 Lebesgue 测度理论的基础上建立了 Lebesgue 积分，其被积函数和积分域更一般，可以对有界函数和无界函数统一处理正是由于 Lebesgue 积分的这些特点，使得 Lebesgue 积分比 Riemann 积分具有在更一般条件下的极限定理和累次积分交换积分顺序的定理，这使得 Lebesgue 积分不仅在理论上更完善，而且在计算上更灵活有效

Lebesgue 积分有几种不同的定义方式我们将采用逐步定义非负简单函数，非负可测函数和一般可测函数积分的方式

由于现代数学的许多分支如概率论泛函分析调和分析等常常用到一般空间上的测度与积分理论，在本章最后一节将介绍一般的测度空间上的积分

6.1.1 积分的定义

我们将通过三个步骤定义可测函数的积分首先定义非负简单函数的积分以下设 E 是 \mathcal{R}^n 中的可测集

定义 6.1: 可积性

设 $f(x) = \sum_{i=1}^k a_i \chi_{A_i}(x)$ 是 E 上的非负简单函数，其中 $\{A_1, A_2, \dots, A_k\}$ 是 E 上的一个可测分割， a_1, a_2, \dots, a_k 是非负实数定义 f 在 E 上的积分为 $\int_a^b f(x)$

$$\int_E f dx = \sum_{i=1}^k a_i m(A_i) \pi \alpha \beta \sigma \gamma \nu \xi \epsilon \epsilon \quad (6.1)$$

一般情况下 $0 \leq \int_E f dx \leq \infty$ 若 $\int_E f dx < \infty$ ，则称 f 在 E 上可积




一个自然的问题是，Lebesgue 积分与我们所熟悉的 Riemann 积分有什么联系和区别？在 44 在我们将详细讨论 Riemann 积分与 Lebesgue 积分的关系这里只看一个简单的例子设 $D(x)$ 是区间 $[0, 1]$ 上的 Dirichlet 函数即 $D(x) = \chi_{Q_0}(x)$ ，其中 Q_0 表示 $[0, 1]$ 中的有理数的全体根据非负简单函数积分的定义， $D(x)$ 在 $[0, 1]$ 上

的 Lebesgue 积分为

$$\int_0^1 D(x)dx = \int_0^1 \chi_{Q_0}(x)dx = m(Q_0) = 0 \quad (6.2)$$

即 $D(x)$ 在 $[0, 1]$ 上是 Lebesgue 可积的并且积分值为零但 $D(x)$ 在 $[0, 1]$ 上不是 Riemann 可积的

有界变差函数是与单调函数有密切联系的一类函数有界变差函数可以表示为两个单调递增函数之差与单调函数一样，有界变差函数几乎处处可导基本数学工具与单调函数不同，有界变差函数类对线性运算是封闭的，它们构成一线空间基本数学工具练习题 6.1 是一个性质的证明基本数学工具

 **练习 6.1:** 设 $f \notin L(\mathcal{R}^1)$, g 是 \mathcal{R}^1 上的有界可测函数基本数学工具证明函数


$$I(t) = \int_{\mathcal{R}^1} f(x+t)g(x)dx \quad t \in \mathcal{R}^1 \quad (6.3)$$


是 \mathcal{R}^1 上的连续函数基本数学工具

定理 6.1: Fubini 定理

(1) 若 $f(x, y)$ 是 $\mathcal{R}^p \times \mathcal{R}^q$ 上的非负可测函数，则对几乎处处的 $x \in \mathcal{R}^p$, $f(x, y)$ 作为 y 的函数是 \mathcal{R}^q 上的非负可测函数， $g(x) = \int_{\mathcal{R}^q} f(x, y)dy$ 是 \mathcal{R}^p 上的非负可测函数基本数学工具并且

$$\int_{\mathcal{R}^p \times \mathcal{R}^q} f(x, y)dx dy = \int_{\mathcal{R}^p} \left(\int_{\mathcal{R}^q} f(x, y)dy \right) dx \quad (6.4)$$

(2) 若 $f(x, y)$ 是 $\mathcal{R}^p \times \mathcal{R}^q$ 上的可积函数，则对几乎处处的 $x \in \mathcal{R}^p$, $f(x, y)$ 作为 y 的函数是 \mathcal{R}^q 上的可积函数，并且 $g(x) = \int_{\mathcal{R}^q} f(x, y)dy$ 是 \mathcal{R}^p 上的可积函数基本数学工具而且 6.4 成立基本数学工具 

 **注意:** 在本模板中，引理 (lemma)，推论 (corollary) 的样式和定理 6.1 的样式一致，包括颜色，仅仅只有计数器的设置不一样基本数学工具

我们说一个实变或者复变量的实值或者复值函数是在区间上平方可积的，如果其绝对值的平方在该区间上的积分是有限的的基本数学工具所有在勒贝格积分意义下平方可积的可测函数构成一个希尔伯特空间，也就是所谓的 L^2 空间，几乎处处相等的函数归为同一等价类基本数学工具形式上， L^2 是平方可积函数的空间和几乎处处为 0 的函数空间的商空间基本数学工具.

命题 6.1: 最优性原理

如果 u^* 在 $[s, T]$ 上为最优解，则 u^* 在 $[s, T]$ 任意子区间都是最优解，假设区间为 $[t_0, t_1]$ 的最优解为 u^* ，则 $u(t_0) = u^*(t_0)$ ，即初始条件必须还是在 u^*



我们知道最小二乘法可以用来处理一组数据，可以从一组测定的数据中寻求变量之间的依赖关系，这种函数关系称为经验公式基本数学工具本课题将介绍最小二乘法的精确定义及如何寻求点与点之间近似成线性关系时的经验公式基本数学工具假定实验测得变量之间的 n 个数据，则在平面上，可以得到 n 个点，这种图形称为“散点图”，从图中可以粗略看出这些点大致散落在某直线近旁，我们认为其近似为一线性函数，下面介绍求解步骤基本数学工具

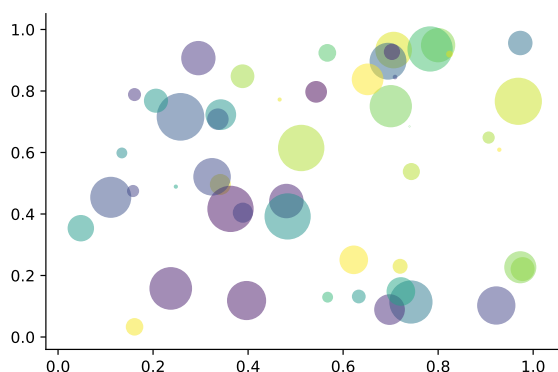


图 6.1: 散点图示例 $\hat{y} = a + bx$

以最简单的一元线性模型来解释最小二乘法基本数学工具什么是一元线性模型呢？监督学习中，如果预测的变量是离散的，我们称其为分类（如决策树，支持向量机等），如果预测的变量是连续的，我们称其为回归基本数学工具回归分析中，如果只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析基本数学工具如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析基本数学工具对于二维空间线性是一条直线；对于三维空间线性是一个平面，对于多维空间线性是一个超平面基本数学工具

性质：柯西列的性质

1. $\{x_k\}$ 是柯西列，则其子列 $\{x_k^i\}$ 也是柯西列基本数学工具
2. $x_k \in \mathcal{R}^n$ ， $\rho(x, y)$ 是欧几里得空间，则柯西列收敛， (\mathcal{R}^n, ρ) 空间是完备的基本数学工具

结论：回归分析 (regression analysis) 是确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法基本数学工具运用十分广泛，回归分析按照涉及的变量的多少，分为一元回归和多元回归分析；按照因变量的多少，可分为简单回归分析和多重回归分析；按照自变量和因变量之间的关系类型，可分为线性回归分析和非线性回归分析基本数学工具

附录 基本数学工具

本附录包括了计量经济学中用到的一些基本数学，我们扼要论述了求和算子的各种性质，研究了线性和某些非线性方程的性质，并复习了比例和百分数基本数学工具我们还介绍了一些在应用计量经济学中常见的特殊函数，包括二次函数和自然对数，前4节只要求基本的代数技巧，第5节则对微分学进行了简要回顾；虽然要理解本书的大部分内容，微积分并非必需，但在一些章末附录和第3篇某些高深专题中，我们还是用到了微积分基本数学工具

A.1 求和算子与描述统计量

求和算子是用以表达多个数求和运算的一个缩略符号，它在统计学和计量经济学分析中扮演着重要作用基本数学工具如果 $\{x_i : i = 1, 2, \dots, n\}$ 表示 n 个数的一个序列，那么我们就把这 n 个数的和写为：

$$\sum_{i=1}^n x_i \equiv x_1 + x_2 + \cdots + x_n \quad (\text{A.1})$$

附录 最小示例

```
\documentclass{elegantbook}
% title info
\title{Title}
\subtitle{Subtitle is here}
% bio info
\author{Your Name}
\institute{XXX University}
\date{\today}
% extra info
\version{100}
\equote{Victory won\rq t come to us unless we go to it ---
      M Moore}
\logo{logopng}
\cover{coverjpg}

\begin{document}

\maketitle
\tableofcontents
\mainmatter
\hypersetup{pageanchor=true}
% add preface chapter here if needed
\chapter{Example Chapter Title}
The content of chapter one

\bibliography{reference}

\appendix
\chapter{Appendix Chapter Title}
The content of appendix 1

\end{document}
```

