

一份简短的深度学习笔记

SteveZhu(朱鉴)

jzhu.nlp@gmail.com

符号定义

这里采用的符号定义引用至深度学习¹一书

数和数组

a 标量 (整数或实数)

\mathbf{a} 向量

\mathbf{A} 矩阵

\mathbf{A} 张量

\mathbf{I}_n n 行 n 列的单位阵

\mathbf{I} 维数根据上下文确定的单位阵

$\mathbf{e}^{(i)}$ 索引 i 处值 1 其他值为 0 的标准基向量
[0, ..., 0, 1, 0, ..., 0]

$\text{diag}(\mathbf{a})$ 对角方阵, 对角线值由向量 \mathbf{a} 确定

\mathbf{a} 标量随机变量

\mathbf{a} 向量随机变量

\mathbf{A} 矩阵随机变量

集合和图

\mathbb{A} 集合

\mathbb{R} 实数集

{0, 1} 包含 0 和 1 的集合

{0, 1, ..., n} 包含 0 和 n 之间所有整数的集合

[a, b] 包含 a 和 b 的实数区间

(a, b] 不包含 a 但包含 b 的实数区间

$\mathbb{A} \setminus \mathbb{B}$ 差集, 即其元素包含于 \mathbb{A} 但不包含于 \mathbb{B}

¹<https://www.deeplearningbook.org/>

索引

a_i	向量 \mathbf{a} 的第 i 个元素，索引从 1 开始
\mathbf{a}_{-i}	除第 i 个元素之外向量 \mathbf{a} 的所有元素
$A_{i,j}$	矩阵 \mathbf{A} 的 i,j 索引处的元素
$A_{i,:}$	矩阵 \mathbf{A} 的第 i 行元素
$A_{:,i}$	矩阵 \mathbf{A} 的第 i 列元素
$A_{i,j,k}$	三维张量 \mathbf{A} 的 i,j,k 索引处的元素
$\mathbf{A}_{::,i}$	三维张量的二维切片
\mathbf{a}_i	随机向量 \mathbf{a} 的第 i 个元素

线性代数运算符

\mathbf{A}^\top	矩阵 \mathbf{A} 的转置
\mathbf{A}^+	矩阵 \mathbf{A} 的 Moore-Penrose 伪逆
$\mathbf{A} \odot \mathbf{B}$	矩阵 \mathbf{A} 和 \mathbf{B} 的逐元素 (Hadamard) 乘积
$\det(\mathbf{A})$	矩阵 \mathbf{A} 的行列式

微积分

$\frac{dy}{dx}$	y 关于 x 的导数
$\frac{\partial y}{\partial x}$	y 关于 x 的偏导
$\nabla_{\mathbf{x}} y$	梯度: y 关于向量 \mathbf{x} 的导数
$\nabla_X y$	矩阵: y 关于矩阵 X 的导数
$\nabla_{\mathbf{X}} y$	张量: y 关于张量 \mathbf{X} 的导数
$\frac{\partial f}{\partial \mathbf{x}}$	函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 的 Jacobian 矩阵 $\mathbf{J} \in \mathbb{R}^{m \times n}$
$\nabla_{\mathbf{x}}^2 f(\mathbf{x})$ or $\mathbf{H}(f)(\mathbf{x})$	函数 f 在点 \mathbf{x} 处的 Hessian 矩阵
$\int f(\mathbf{x}) d\mathbf{x}$	向量 \mathbf{x} 整个定义域上的定积分
$\int_{\mathbb{S}} f(\mathbf{x}) d\mathbf{x}$	集合 \mathbb{S} 上关于向量 \mathbf{x} 的定积分

概率和信息论

$a \perp b$	随机变量 a 和 b 彼此独立
$a \perp b c$	给定随机变量 c 时彼此独立
$P(a)$	离散变量的概率分布
$p(a)$	连续变量 (或未指定类型变量) 的概率分布
$a \sim P$	随机变量 a 服从概率分布 P
$\mathbb{E}_{x \sim P}[f(x)]$ or $\mathbb{E}f(x)$	$f(x)$ 关于 $P(x)$ 的期望
$\text{Var}(f(x))$	$f(x)$ 在分布 $P(x)$ 下的方差
$\text{Cov}(f(x), g(x))$	$f(x)$ 和 $g(x)$ 在分布 $P(x)$ 下的协方差
$H(x)$	随机变量 x 的香侬熵
$D_{\text{KL}}(P \ Q)$	概率分布 P 和 Q 的 Kullback-Leibler 距离
$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	均值为 $\boldsymbol{\mu}$ 协方差为 $\boldsymbol{\Sigma}$, 向量 \mathbf{x} 的高斯分布

函数

$f : \mathbb{A} \rightarrow \mathbb{B}$ 定义域为 \mathbb{A} 值域为 \mathbb{B} 的函数 f

$f \circ g$	函数 f 和 g 构成的复合函数
$f(\mathbf{x}; \boldsymbol{\theta})$	含参量 $\boldsymbol{\theta}$, 关于 \mathbf{x} 的函数
$\log x$	x 的自然对数
$\sigma(x)$	Logistic sigmoid, $\frac{1}{1 + \exp(-x)}$
$\ \mathbf{x}\ _p$	\mathbf{x} 的 L^p 范数
$\ \mathbf{x}\ $	\mathbf{x} 的 L^2 范数
x^+	x 的正数部分, 即 $\max(0, x)$

$\mathbf{1}_{\text{condition}}$ 如果条件为真则为 1, 否则为 0

有时我们将参数类型为标量的函数 f 应用于向量, 矩阵或张量: $f(\mathbf{x})$, $f(\mathbf{X})$, or $f(\mathbf{X})$. 这时表示将函数 f 逐个应用于数组中的元素. 例如, 假设 $\mathbf{C} = \sigma(\mathbf{X})$, 那么对所有索引 i, j 和 k 处的元素有 $C_{i,j,k} = \sigma(X_{i,j,k})$.

数据集及分布

p_{data} 数据生成分布

\hat{p}_{data} 由训练集定义的经验分布

\mathbb{X} 训练样本的集合

$\mathbf{x}^{(i)}$ 数据集的第 i 个样本（输入）

$y^{(i)}$ or $\mathbf{y}^{(i)}$ $\mathbf{x}^{(i)}$ 的标注用于有监督学习

\mathbf{X} $m \times n$ 矩阵，输入样本 $\mathbf{x}^{(i)}$ 为该矩阵行切片 $\mathbf{X}_{i,:}$

目 录

符号定义	i
1 微积分	1
1.1 导数	1
1.1.1 导数定义	1
1.1.2 常用公式	1
1.2 偏导数	2
1.2.1 偏导数定义	2
1.2.2 Hessian 矩阵	2
1.3 梯度	2
1.3.1 梯度定义	2
1.3.2 梯度垂域等高线(面)证明	3
1.4 方向导数	3
1.4.1 方向导数定义	3
1.4.2 最速提升方向	4
1.5 泰勒级数	4
1.5.1 单变量函数近似	4
1.5.2 多变量函数近似	4
2 线性代数	5
2.1 基本概念	5
2.2 向量	5
2.2.1 向量定义	5
2.2.2 向量内积	6
2.2.3 向量夹角及不等式	6
2.2.4 向量外积	7
2.3 矩阵	7
2.3.1 矩阵定义	7
2.3.2 矩阵内积	7
2.3.3 矩阵乘法	7
2.4 矩阵操作和属性	10
2.4.1 单位阵和对角阵	10
2.4.2 线性无关和秩	11
2.4.3 矩阵的逆	11
2.4.4 正交矩阵	12
2.4.5 矩阵的 Range 及 Nullspace	12

2.4.6 行列式	14
2.5 特征值和特征向量 (Eigenvalue and Eigenvector)	15
2.5.1 特征值及特征向量	15
2.5.2 对称矩阵	17
2.5.3 SVD 分解	17
2.6 范数	20
2.6.1 向量范数	20
2.6.2 矩阵范数	21
2.7 向量化导数	25
2.7.1 梯度 ($f : \mathbb{R}^n \rightarrow \mathbb{R}$)	25
2.7.2 雅可比矩阵 ($f : \mathbb{R}^n \rightarrow \mathbb{R}^m$)	25
2.7.3 广义雅可比矩阵 ($f : \mathbb{R}^{n_1 \times \dots \times n_{d_x}} \rightarrow \mathbb{R}^{n_1 \times \dots \times n_{d_y}}$)	26
3 概率论	29
3.1 概率基础	29
3.1.1 基础定义	29
3.1.2 条件概率	30
3.2 随机变量	30
3.2.1 基本定义	30
3.2.2 累积分布函数	31
3.2.3 概率质量函数	31
3.2.4 概率密度函数	31
3.2.5 期望	32
3.2.6 方差	32
3.2.7 常用随机变量	33
3.3 两个随机变量	34
3.3.1 联合和边缘累积分布函数	35
3.3.2 联合和边缘概率质量函数	35
3.3.3 联合和边缘概率密度函数	35
3.3.4 条件分布	36
3.3.5 贝叶斯法则	36
3.3.6 独立性	37
3.3.7 期望及协方差	37
3.4 多个随机变量	38
3.4.1 基本属性	38
3.4.2 随机向量	39
3.5 参数估计	40
3.5.1 最大似然估计	41

3.5.2 最大后验估计	41
3.6 结构化概率模型	42
3.6.1 有向图模型	42
3.6.2 无向图模型	43
3.7 主成分分析	44
4 数值计算	47
4.1 凸集	47
4.1.1 凸集定义	47
4.1.2 例子	47
4.2 凸函数	48
4.2.1 凸函数定义	49
4.2.2 凸性的一阶条件	49
4.2.3 凸性的二阶条件	50
4.2.4 Jensen 不等式	50
4.2.5 次水平集	50
4.2.6 例子	51
4.3 优化问题	52
4.3.1 无约束优化	52
4.3.2 等式约束优化问题	52
4.3.3 不等式约束优化问题	55
4.4 一阶近似优化	59
4.4.1 SGD	59
4.4.2 Adagrad	60
4.4.3 Adam	60
5 神经网络及反向传播	62
5.1 神经网络	62
5.2 计算图	63
5.3 反向传播算法	64
5.3.1 链乘法则	64
5.3.2 反向传播算法	64
5.4 向量化反向传播	65
5.4.1 线性函数节点	65
5.4.2 非线性函数节点	68
5.5 梯度爆炸或消散	71
6 激活函数	75
6.1 Sigmoid	75

6.2	Tanh	75
6.3	ReLU	76
6.4	Leaky ReLU	77
6.5	PReLU	77
6.6	ELU	78
6.7	GELU	79
7	参数优化	80
7.1	零值初始化	80
7.2	随机初始化	80
7.3	Xavier 初始化	81
7.4	He 初始化	85
7.5	批归一化	87
7.5.1	Internal Covariate Shift	87
7.5.2	Batch Normalization	88
7.6	层归一化	91
8	损失函数	93
8.1	0-1 损失函数	93
8.2	平方错误损失	94
8.3	交叉熵损失	96
8.4	Hinge 损失	97
8.5	CRF 损失	98
8.6	KL-divergence 损失	100
8.7	RankNet 损失	102
8.8	LambdaRank 损失	102
9	正则化	104
9.1	L_1 正则化	104
9.2	L_2 正则化	105
9.3	Dropout	106
9.4	数据增强	106
9.5	样本标注引入噪声	107
9.6	半监督学习	107
9.7	多任务学习	107
9.8	提前终止	108
10	网络架构	109
10.1	前向网络	109
10.1.1	网络结构	109

10.1.2 语言模型	110
10.1.3 排序问题	111
10.1.4 推荐问题	113
10.2 卷积网络	117
10.2.1 网络结构	117
10.2.2 图像分类	121
10.2.3 文本分类	123
10.3 循环网络	124
10.3.1 网络架构	124
10.3.2 应用案例	130
10.4 Transformer	137
10.4.1 Transformer	137
10.4.2 BERT	140
10.4.3 ELECTRA	142

第1章 微积分

本部分主要回顾了一元变量下的导数定义研究自变量和因变量之间的变化关系，及在此基础之上推广到多元变量场景下的偏导数定义研究因变量和多元变量中某一变量之间的变化关系，及梯度定义和方向导数的定义研究因变量和多元变量之间的变化关系。在此基础上回顾了函数逼近所涉及的泰勒级数定义。

1.1 导数

本部分回顾导数的定义及导数的常用公式

1.1.1 导数定义

导数

单变量函数的导数定义公式如下：

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (1.1)$$

其几何解释为函数在点 x 处切线的斜率，物理解释为函数在点 x 处的变化率。当导数为零时，点 x 为函数 f 可能的极值点/鞍点。

二阶导

单变量函数二阶导定义如下：

$$f''(x) = \frac{d^2 f}{dx^2} = \lim_{h \rightarrow 0} \frac{f'(x+h) - f'(x)}{h} \quad (1.2)$$

可以看到二阶导数为导数在某点的变化率。

1.1.2 常用公式

求导法则

常量法则: $f' = 0$, 若 $f(x)$ 为常量; 和法则: $(\alpha f + \beta g)' = \alpha f' + \beta g'$

乘法则: $(fg)' = f'g + fg'$; 商法则: $(\frac{f}{g})' = \frac{f'g - fg'}{g^2}$

链乘法则: 若 $f(x) = h(g(x))$, 则 $f'(x) = h'(g(x))g'(x)$

基本函数求导法则

幂函数的导数: $(x^r)' = rx^{r-1}, r \in \mathbb{R}$

指数函数的导数: $(e^x)' = e^x, (a^x)' = a^x \ln a$. 注: $a^x = (e^{\ln a})^x = e^{x \ln(a)}$

对数函数的导数: $(\ln x)' = \frac{1}{x}, (\log_a x)' = \frac{1}{x \ln a}$,

1.2 偏导数

本部分回顾偏导数的定义及 Hessian 矩阵

1.2.1 偏导数定义

多变量函数的偏导数定义和单变量函数的导数定义类似，在某个变量的偏导数时固定其他变量，因此多变量函数的偏导数的数学记号如下：

$$\frac{\partial f}{\partial x_i}(x_1, \dots, x_n) = \lim_{h \rightarrow 0} \frac{f(x_1, \dots, x_i + h, \dots, x_n) - f(x_1, \dots, x_i, \dots, x_n)}{h} \quad (1.3)$$

当点 \mathbf{x} 处的所有偏导数为零时，点 \mathbf{x} 为函数 f 可能的极值点/鞍点。其中

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (1.4)$$

1.2.2 Hessian 矩阵

若多变量函数所有二阶偏导数存在，且在函数的定义域连续。多变量函数的所有二阶导组成 $n \times n$ 的 Hessian 矩阵，其定义如下：

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \quad (1.5)$$

1.3 梯度

本节回顾梯度的定义及梯度垂直于等高线(面)的证明，梯度垂直于等高线(面)定性给出了函数最速提升方向，方向导数进一步给出了理论证明。

1.3.1 梯度定义

多元函数的所有偏导数构成的向量为梯度

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad (1.6)$$

1.3.2 梯度垂域等高线(面)证明

多元函数 $f(\mathbf{x})$, 其中:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (1.7)$$

x_i 为第 i 维的变量, 根据等高线(面)定义可知, 对应该等高线(面)上的所有点, 函数 $f(\mathbf{x})$ 取得常量 c 。因此有

$$f(\mathbf{x}) = c$$

假设如下曲线为等高线(面)上的曲线

$$\mathbf{r}(t) = \langle x_1(t), x_2(t), \dots, x_n(t) \rangle$$

可知:

$$g(t) = f(x_1(t), x_2(t), \dots, x_n(t)) = c \quad (1.8)$$

$$\Rightarrow \frac{dg}{dt} = \frac{\partial f}{\partial x_1} \frac{dx_1}{dt} + \frac{\partial f}{\partial x_2} \frac{dx_2}{dt} + \dots + \frac{\partial f}{\partial x_n} \frac{dx_n}{dt} = 0 \quad (1.9)$$

因此:

$$\nabla f(\mathbf{x}) \cdot \mathbf{r}'(t) = 0 \quad (1.10)$$

上述公式中 $\mathbf{r}'(t)$ 为等高线(面)在 t 时刻对应的点 \mathbf{x} 处的切线, 因此对于等高面(线)上的任意曲线, 梯度垂直于该曲线在某点的切线方向, 因此梯度垂直于等高面(线)。

1.4 方向导数

本小节回顾方向导数的定义及由此定义引出的最速提升方向

1.4.1 方向导数定义

方向导数定义了点 \mathbf{x} 处沿向量 \mathbf{v} 方向变化时, 对应的函数的瞬时变化率

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \quad (1.11)$$

$$\nabla_{\mathbf{v}} f(\mathbf{x}) = \mathbf{v}^T \cdot \nabla f(\mathbf{x}) \quad (1.12)$$

1.4.2 最速提升方向

由方向导数公式可知，在点 \mathbf{x} 处变化方向 \mathbf{v} 和梯度方向一致时函数值增加最快，此时 $\mathbf{v} = \eta \nabla f(\mathbf{x})$, η 为正的标量。当 \mathbf{v} 和梯度方向相反时函数值减小最快，此时 $\mathbf{v} = -\eta \nabla f(\mathbf{x})$ 。

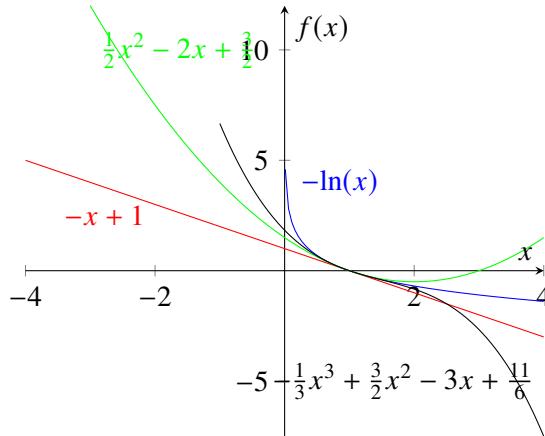
1.5 泰勒级数

本节讨论任意可导函数在某个点附近通过泰勒级数近似的一般化形式

1.5.1 单变量函数近似

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(x_k)}{2!}(x - x_k)^2 + \cdots + \frac{f^n(x_k)}{n!}(x - x_k)^n \quad (1.13)$$

上述泰勒级数为可导函数 $f(x)$ 在点 x_k 附近的多项式逼近形式。这里给出 $f(x) = -\ln(x)$ 在点 $(1, 0)$ 处的泰勒级数近似的图形化描述



其中 蓝色 表示 $f(x)$, 红色 为 $f(x)$ 在点 $(1, 0)$ 处的泰勒级数一阶近似, 绿色 为 $f(x)$ 在点 $(1, 0)$ 处的泰勒级数二阶近似, 黑色 为 $f(x)$ 在点 $(1, 0)$ 处的泰勒级数三阶近似。可以看到随着多项式逼近的阶次增大, 在该点处会有更好的近似。实际中常常使用二阶近似, 即可取得较好的效果。

1.5.2 多变量函数近似

$$f(\mathbf{x}) = f(\mathbf{x}_k) + (\mathbf{x} - \mathbf{x}_k)^T \nabla f(\mathbf{x}_k) + \frac{1}{2!} (\mathbf{x} - \mathbf{x}_k)^T \nabla^2 f(\mathbf{x}_k) (\mathbf{x} - \mathbf{x}_k) + \dots \quad (1.14)$$

上述公式中的 $\nabla f(\mathbf{x}_k)$ 为函数在点 \mathbf{x}_k 处的梯度, $\nabla^2 f(\mathbf{x}_k)$ 为函数在点 \mathbf{x}_k 处的二阶偏导数构成的 Hessian 矩阵。

第2章 线性代数

本部分主要回尙向量及矩阵基本运算，及在此基础上的空间定义，及空间中元素之间的关系引入投影定义，最后回顾了为研究线性系统凸性及稳定性所需的行列式、特征值和特征向量，及正定矩阵、奇异值分解、范数和伪逆等重要概念。

2.1 基本概念

线性代数为线性方程组的运算提供了紧凑的表示。例如考慮如下线性方程组：

$$\begin{aligned} 4x_1 - 5x_2 &= -13 \\ -2x_1 + 3x_2 &= 9. \end{aligned}$$

借助于线性代数，可以将上述方程组表示如下

$$Ax = b$$

其中

$$A = \begin{bmatrix} 4 & -5 \\ -2 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} -13 \\ 9 \end{bmatrix}.$$

变为如上表示方案后，可以借助于线性代数提供的工具分析方程组是否存在精确解还是近似解，以及相应解的求取。

2.2 向量

本节回尙向量的定义及向量的基本运算

2.2.1 向量定义

通常我们通过 $x \in \mathbb{R}^n$ 表示含有 n 个元素的向量。默认情况下， n 维向量为列向量，即含有 n 行和 1 列的矩阵。如果我们期望采用行向量形式 – 即 1 行 n 列的矩阵，一般表示为 x^\top (即向量 x 的转置)。向量 x 的第 i 个元素表示为 x_i :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

2.2.2 向量内积

给定两个向量 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\langle \mathbf{x}, \mathbf{y} \rangle$ 被称为向量内积或点积

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y} \in \mathbb{R}^n = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^n x_i y_i. \quad (2.1)$$

通过上述定义可知向量内积满足如下引理

- $\mathbf{x}^\top \mathbf{x} \geq 0, \mathbf{x}^\top \mathbf{x} = 0 \Leftrightarrow \mathbf{x} = 0$
- $\mathbf{x}^\top \mathbf{y} = \mathbf{y}^\top \mathbf{x}$
- $(\mathbf{x} + \mathbf{y})^\top \mathbf{z} = \mathbf{x}^\top \mathbf{z} + \mathbf{y}^\top \mathbf{z}$
- $(r\mathbf{x}^\top) \mathbf{y} = r(\mathbf{x}^\top \mathbf{y}) \quad \forall r \in \mathbb{R}$

2.2.3 向量夹角及不等式

结合余弦定理, 可推知如下定义

$$\mathbf{x}^\top \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta \quad (2.2)$$

其中 $\|\mathbf{x}\|$ 和 $\|\mathbf{y}\|$ 表示向量的长度, θ 表示向量的夹角。

公式2.2证明, 根据余弦定理可知:

$$\|\mathbf{x} - \mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\|\mathbf{x}\| \|\mathbf{y}\| \cos \theta$$

同时根据点积的定义和向量长度定义可知:

$$\begin{aligned} \|\mathbf{x} - \mathbf{y}\|^2 &= (\mathbf{x} - \mathbf{y})^\top (\mathbf{x} - \mathbf{y}) \\ &= \mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{x} + \mathbf{y}^\top \mathbf{y} \\ &= \|\mathbf{x}\|^2 - 2\mathbf{x}^\top \mathbf{y} + \|\mathbf{y}\|^2 \end{aligned}$$

因此有:

$$\begin{aligned} \|\mathbf{x}\|^2 - 2\mathbf{x}^\top \mathbf{y} + \|\mathbf{y}\|^2 &= \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\|\mathbf{x}\| \|\mathbf{y}\| \cos \theta \\ \mathbf{x}^\top \mathbf{y} &= \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta \end{aligned}$$

由于 $-1 \leq \cos \theta \leq 1$, 从上述余弦公式可知

$$\mathbf{x}^\top \mathbf{y} \leq |\mathbf{x}^\top \mathbf{y}| \leq \|\mathbf{x}\| \|\mathbf{y}\|.$$

对上述不等式两边取平方，可推知如下 Cauchy-Schwarz 不等式

$$|\mathbf{x}^\top \mathbf{y}|^2 \leq \|\mathbf{x}\|^2 \|\mathbf{y}\|^2. \quad (2.3)$$

2.2.3.1 正交向量

正交向量表示向量 \mathbf{x} 和 \mathbf{y} 之间的夹角为 90° ，通过上述向量点积的余弦定义可知正交向量之间的点积为 0

$$\mathbf{x}^\top \mathbf{y} = 0$$

2.2.4 向量外积

向量外积表示列向量乘以行向量，其运算结果为矩阵。假设 $\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m$ ，如下为两个向量进行外积运算的结果：

$$\mathbf{x}\mathbf{y}^\top \in \mathbb{R}^{n \times m} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_m \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_m \\ \vdots & \vdots & \ddots & \vdots \\ x_n y_1 & x_n y_2 & \cdots & x_n y_m \end{bmatrix}$$

2.3 矩阵

本节回顾矩阵的定义及矩阵的基本运算

2.3.1 矩阵定义

通常我们通过 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 表示含 m 行和 n 列的矩阵，同时矩阵的每个元素为实数

2.3.2 矩阵内积

矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 和 $\mathbf{B} \in \mathbb{R}^{n \times p}$ 的内积结果为实数

$$\langle \mathbf{A}, \mathbf{B} \rangle = \text{tr}(\mathbf{A}^\top \mathbf{B}) = \sum_i \sum_j A_{i,j} B_{i,j} \quad (2.4)$$

上述公式中 $\text{tr}(\mathbf{Z})$ 为矩阵的 trace， $\text{tr}(\mathbf{Z}) = \sum_i Z_{i,i}$

2.3.3 矩阵乘法

矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 和 $\mathbf{B} \in \mathbb{R}^{n \times p}$ 的结果依然是矩阵

$$\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p},$$

且

$$C_{i,j} = \sum_{k=1}^n A_{i,k} B_{kj}. \quad (2.5)$$

这里需要注意的是，仅当矩阵 \mathbf{A} 的列数和矩阵 \mathbf{B} 的行数一致时，矩阵间的相乘才是合法的。

2.3.3.1 矩阵乘向量

给定矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 和向量 $\mathbf{x} \in \mathbb{R}^n$ ，矩阵和向量相乘的结果为向量 $\mathbf{y} = \mathbf{Ax} \in \mathbb{R}^m$ 。我们将从不同角度回顾矩阵和向量的乘法运算。

假设我们将矩阵 \mathbf{A} 表示为行向量形式，那么 \mathbf{Ax} 运算表示为如下形式

$$\mathbf{y} = \mathbf{Ax} = \begin{bmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ \vdots & & \\ - & \mathbf{a}_m^\top & - \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{x} \\ \mathbf{a}_2^\top \mathbf{x} \\ \vdots \\ \mathbf{a}_m^\top \mathbf{x} \end{bmatrix}$$

从上述计算方式可以看出，向量 \mathbf{y} 的第 i 个元素为矩阵 \mathbf{A} 的第 i 行和向量 \mathbf{x} 的点积， $y_i = \mathbf{a}_i^\top \mathbf{x}$ 。

另外我们也可以将矩阵 \mathbf{A} 表示为列向量形式，在该表示形式下，有

$$\mathbf{y} = \mathbf{Ax} = \begin{bmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_n \end{bmatrix} x_1 + \begin{bmatrix} \mathbf{a}_2 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_n \end{bmatrix} x_2 + \cdots + \begin{bmatrix} \mathbf{a}_n \\ \mathbf{a}_n \\ \vdots \\ \mathbf{a}_n \end{bmatrix} x_n.$$

从该形式来看，向量 \mathbf{y} 为矩阵 \mathbf{A} 的列向量的线性组合，线性组合的系数由向量 \mathbf{x} 确定。

目前为止我们均是在矩阵的右边乘以列向量，但同时在矩阵的左边乘以行向量也是合法的，该运算的结果仍为行向量。 $\mathbf{y}^\top = \mathbf{x}^\top \mathbf{A}$ ，其中 $\mathbf{A} \in \mathbb{R}^{m \times n}$ ， $\mathbf{x} \in \mathbb{R}^m$ ， $\mathbf{y} \in \mathbb{R}^n$ 。和之前的矩阵列向量乘法运算类似，同样可以将运算表示为向量点积或行向量的线性组合。向量点积的运算方式表示为如下所示形式：

$$\mathbf{y}^\top = \mathbf{x}^\top \mathbf{A} = \mathbf{x}^\top \begin{bmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ | & | & & | \end{bmatrix} = [\mathbf{x}^\top \mathbf{a}_1 \quad \mathbf{x}^\top \mathbf{a}_2 \quad \cdots \quad \mathbf{x}^\top \mathbf{a}_n]$$

从该形式看向量 \mathbf{y}^\top 的第 i 个元素为向量 \mathbf{x} 和矩阵 \mathbf{A} 的第 i 列的点积。向量线性组合的

运算方式表示为如下所示形式:

$$\begin{aligned} \mathbf{y}^\top &= \mathbf{x}^\top \mathbf{A} \\ &= \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ \vdots & & \\ - & \mathbf{a}_m^\top & - \end{bmatrix} \\ &= x_1 \begin{bmatrix} - & \mathbf{a}_1^\top & - \end{bmatrix} + x_2 \begin{bmatrix} - & \mathbf{a}_2^\top & - \end{bmatrix} + \cdots + x_n \begin{bmatrix} - & \mathbf{a}_n^\top & - \end{bmatrix} \end{aligned}$$

可以看到向量 \mathbf{y}^\top 是矩阵 \mathbf{A} 的行向量的线性组合, 线性组合的系数由向量 \mathbf{x} 给定。

2.3.3.2 矩阵乘矩阵

有了矩阵向量相乘不同角度的运算基础, 不同于本节开始给出的矩阵运算算法, 现在我们可以四个不同的角度来看矩阵和矩阵的相乘运算 $\mathbf{C} = \mathbf{AB}$ 。

首先我们可以将矩阵和矩阵的相乘运算看成一系列的向量点积运算。根据本节给出矩阵相乘运算的定义, 可以发现矩阵 \mathbf{C} 的第 (i, j) 个元素等于矩阵 \mathbf{A} 的第 i 行和矩阵 \mathbf{B} 的第 j 列的点积。符号表示如下:

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ \vdots & & \\ - & \mathbf{a}_m^\top & - \end{bmatrix} \begin{bmatrix} | & | & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_p \\ | & | & | \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{b}_1 & \mathbf{a}_1^\top \mathbf{b}_2 & \cdots & \mathbf{a}_1^\top \mathbf{b}_p \\ \mathbf{a}_2^\top \mathbf{b}_1 & \mathbf{a}_2^\top \mathbf{b}_2 & \cdots & \mathbf{a}_2^\top \mathbf{b}_p \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_m^\top \mathbf{b}_1 & \mathbf{a}_m^\top \mathbf{b}_2 & \cdots & \mathbf{a}_m^\top \mathbf{b}_p \end{bmatrix}$$

由于 $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times p}$, $\mathbf{a}_i \in \mathbb{R}^n$, $\mathbf{b}_j \in \mathbb{R}^n$, 因此上述的点积均合法。上述将矩阵 \mathbf{A} 看成行向量, 矩阵 \mathbf{B} 看成列向量为最自然的表示形式。另外我们也可以将矩阵 \mathbf{A} 表示为列向量, 将矩阵 \mathbf{B} 表示为行向量, 采用该表示形式时, 可以将 \mathbf{AB} 看成所有向量向量外积的和。符号表示如下:

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} | & | & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ | & | & | \end{bmatrix} \begin{bmatrix} - & \mathbf{b}_1^\top & - \\ - & \mathbf{b}_2^\top & - \\ \vdots & & \\ - & \mathbf{b}_n^\top & - \end{bmatrix} = \sum_{i=1}^n \mathbf{a}_i \mathbf{b}_i^\top.$$

可以看出 $\mathbf{a}_i \in \mathbb{R}^m$, $\mathbf{b}_i^\top \in \mathbb{R}^p$, 因此 $\mathbf{a}_i \mathbf{b}_i^\top \in \mathbb{R}^{m \times p}$ 。该维度和矩阵 \mathbf{C} 一致, 上述公式中最后的等式可能看上去不是那么直观, 可以通过简单的形式来进行验证并一般化推广。

另外我们也可以将矩阵和矩阵的乘法运算看成一系列的矩阵和向量的乘法运算。假设我们将矩阵 \mathbf{B} 表示为列向量形式。可以将矩阵 \mathbf{C} 的列表示为矩阵 \mathbf{A} 和矩阵 \mathbf{B} 的列向

量的乘积。符号表示如下：

$$\mathbf{C} = \mathbf{AB} = A \begin{bmatrix} | & | & \cdots & | \\ \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_p \\ | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & | & \cdots & | \\ A\mathbf{b}_1 & A\mathbf{b}_2 & \cdots & A\mathbf{b}_p \\ | & | & \cdots & | \end{bmatrix}.$$

上述表示形式中 $\mathbf{c}_i = A\mathbf{b}_i$, 可以根据上节给出给出的两种不同视角进行矩阵列向量乘法运算。

最后和矩阵向量运算类似，我们也可以将矩阵 \mathbf{A} 表示为行向量，将矩阵 \mathbf{C} 的行向量看成矩阵 \mathbf{A} 的行向量和矩阵 \mathbf{B} 的乘法运算，符号表示如下：

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ \vdots & & \\ - & \mathbf{a}_m^\top & - \end{bmatrix} \mathbf{B} = \begin{bmatrix} - & \mathbf{a}_1^\top \mathbf{B} & - \\ - & \mathbf{a}_2^\top \mathbf{B} & - \\ \vdots & & \\ - & \mathbf{a}_m^\top \mathbf{B} & - \end{bmatrix}.$$

上述表示形式中 $\mathbf{c}_i^\top = \mathbf{a}_i^\top \mathbf{B}$ 。

最后矩阵乘法运算满足如下定律：

- 矩阵乘法满足结合律: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$
- 矩阵乘法满足分配率: $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$
- 矩阵乘法一般不满足交换律: $\mathbf{AB} \neq \mathbf{BA}$

2.4 矩阵操作和属性

本节回顾线性代数中矩阵的类型及矩阵的属性

2.4.1 单位阵和对角阵

单位阵 $\mathbf{I}_n \in \mathbb{R}^{n \times n}$, 为方阵, 对角线上的元素为 1, 其他元素为 0。即:

$$I_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad (2.6)$$

对任意矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$, 存在如下等式

$$\mathbf{AI}_n = \mathbf{A} = \mathbf{I}_m \mathbf{A}$$

对角阵的所有非对角线元素为 0, 对角阵一般表示为 $\mathbf{D} = \text{diag}(d_1, d_2, \dots, d_n)$, 存在如下定义:

$$D_{i,j} = \begin{cases} d_i & i = j \\ 0 & i \neq j \end{cases} \quad (2.7)$$

从如上定义可见， $\mathbf{I} = \text{diag}(1, 1, \dots, 1)$ 。

2.4.2 线性无关和秩

若集合 $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\} \subset \mathbb{R}^m$ 中的任意向量无法表示为集合中的其他向量的线性组合，则称该向量集合为线性无关，相反若集合中存在某个向量可以线性表示为其余向量的线性组合，则称集合为线性相关，即

$$\mathbf{x}_i = \sum_{j \neq i} \alpha_j \mathbf{x}_j$$

其中 $\alpha_j \in \mathbb{R}$ 。譬如，如下向量集合为线性相关

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} 4 \\ 1 \\ 5 \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} 2 \\ -3 \\ -1 \end{bmatrix}$$

因为 $\mathbf{x}_3 = -2\mathbf{x}_1 + \mathbf{x}_2$ 。

矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 的列秩为矩阵 \mathbf{A} 列向量构成的最大的线性无关的子集的大小，或者简单描述为矩阵 \mathbf{A} 的线性无关的列向量个数。类似地行秩为行向量构成的最大的线性无关的子集的大小。对任意矩阵来说列秩和行秩相等，统称为矩阵的秩，即 $\text{rank}(\mathbf{A})$ ，这一点可以通过对矩阵进行 Gauss-Jordan 消元运算看出。矩阵的 rank 具有如下性质：

- 对任意矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\text{rank}(\mathbf{A}) \leq \min(m, n)$ 。若 $\text{rank}(\mathbf{A}) = \min(m, n)$ ，则称矩阵为满秩矩阵
- 对任意矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}^\top)$
- 对任意矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times p}$, $\text{rank}(\mathbf{AB}) \leq \min(\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B}))$
- 对任意矩阵 $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, $\text{rank}(\mathbf{A} + \mathbf{B}) \leq \text{rank}(\mathbf{A}) + \text{rank}(\mathbf{B})$

2.4.3 矩阵的逆

方阵 $\mathbf{A} \in \mathbb{R}^{n \times n}$ 的逆表示为 \mathbf{A}^{-1} ，为唯一矩阵，且存在如下计算公式：

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} = \mathbf{AA}^{-1}. \quad (2.8)$$

并非所有矩阵均存在逆，若矩阵不是方阵根据定义则没有逆（该情况下可以求矩阵的伪逆）。即使矩阵为方阵，也可能不存在逆，若矩阵 \mathbf{A} 存在逆阵 \mathbf{A}^{-1} ，我们称矩阵 \mathbf{A} 为可逆矩阵或非奇异矩阵，否则称为奇异矩阵。

矩阵 \mathbf{A} 存在逆阵 \mathbf{A}^{-1} 的前提是 \mathbf{A} 必须满秩。如下为求逆操作的性质，其中 $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$

- $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
- $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$
- $(\mathbf{A}^{-1})^\top = (\mathbf{A}^\top)^{-1}$

求逆操作可以应用于求解本章开头的线性方程组 $\mathbf{A}\mathbf{x} = \mathbf{b}$, 其中 $\mathbf{A} \in \mathbb{R}^{n \times n}$, 且 $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ 。若 \mathbf{A} 为非奇异矩阵, 则有 $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ 。

2.4.4 正交矩阵

若 $\mathbf{x}^\top \mathbf{y} = 0$, 则两个向量 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ 彼此正交。若 $\|\mathbf{x}\| = 1$ 则称向量 $\mathbf{x} \in \mathbb{R}^n$ 为单位向量。若方阵 $\mathbf{U} \in \mathbb{R}^{n \times n}$ 的所有列向量彼此正交且为单位向量, 则称方阵为正交矩阵。由方阵的定义可知存在如下公式

$$\mathbf{U}^\top \mathbf{U} = \mathbf{I} = \mathbf{U}\mathbf{U}^\top. \quad (2.9)$$

从上述公式可以看出正交矩阵的逆为该矩阵的转置。正交矩阵的另外一个优良性质是对某向量 \mathbf{x} 进行线性变换将不改变向量的二范数, 该性质对线性系统的稳态至关重要

$$\|\mathbf{U}\mathbf{x}\| = \|\mathbf{x}\| \quad (2.10)$$

其中 $\mathbf{x} \in \mathbb{R}^n, \mathbf{U} \in \mathbb{R}^{n \times n}$ 。上述公式较容易证明:

$$\|\mathbf{U}\mathbf{x}\| = (\mathbf{U}\mathbf{x})^\top \mathbf{U}\mathbf{x} = \mathbf{x}^\top \mathbf{U}^\top \mathbf{U}\mathbf{x} = \mathbf{x}^\top \mathbf{I}\mathbf{x} = \mathbf{x}^\top \mathbf{x} = \|\mathbf{x}\|$$

2.4.5 矩阵的 Range 及 Nullspace

向量集合 $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ 的 *span* 为该集合中所有向量线性组合生成的向量构成的集合。表示如下:

$$\text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\}) = \left\{ \mathbf{v} : \mathbf{v} = \sum_{i=1}^n \alpha_i \mathbf{x}_i, \quad \alpha_i \in \mathbb{R} \right\}. \quad (2.11)$$

若集合 $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ 中的 n 个向量彼此线性无关, 其中 $\mathbf{x}_i \in \mathbb{R}^n$, 则 $\text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\}) = \mathbb{R}^n$ 。也就是说, 任意向量 $\mathbf{v} \in \mathbb{R}^n$ 可以表示为向量 \mathbf{x}_1 到向量 \mathbf{x}_n 的线性组合。向量 $\mathbf{y} \in \mathbb{R}^m$ 在空间 $\text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\})$ (这里 $\mathbf{x}_i \in \mathbb{R}^m$) 上的投影为距离向量 \mathbf{y} 最近的向量 $\mathbf{v} \in \text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\})$, 向量距离由欧几里得距离衡量 $\|\mathbf{v} - \mathbf{y}\|$ 。该投影操作表示为如下形式:

$$\text{Proj}(\mathbf{y}; \{\mathbf{x}_1, \dots, \mathbf{x}_n\}) = \underset{\mathbf{v} \in \text{span}(\{\mathbf{x}_1, \dots, \mathbf{x}_n\})}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{v}\|.$$

矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 的 *range* (也称为列空间), 表示为 $\mathcal{R}(\mathbf{A})$, 为矩阵 \mathbf{A} 的列向量构成的 *span*。

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{v} \in \mathbb{R}^m : \mathbf{v} = \mathbf{A}\mathbf{x}, \mathbf{x} \in \mathbb{R}^n\}.$$

这里假设矩阵 \mathbf{A} 为满秩矩阵且 $n < m$, 此时如下方程:

$$\mathbf{A}\mathbf{x} = \mathbf{y}$$

不存在解析解，仅存在近似解（其中 $\mathbf{x} \in \mathbb{R}^m$ ）。假设在矩阵 A 的的 **range** 中存在某个和 \mathbf{y} 距离最小的向量 \mathbf{v} 。此时有：

$$A\hat{\mathbf{x}} = \mathbf{v}$$

其中向量 \mathbf{v} 为向量 \mathbf{y} 在 $\mathcal{R}(A)$ 上的投影，因此当向量 $\mathbf{y} - \mathbf{v}$ 垂直（正交）于 $\mathcal{R}(A)$ 的任意向量时， $\|\mathbf{y} - \mathbf{v}\|$ 取得最小值。此时有：

$$\begin{aligned} A^\top(\mathbf{y} - \mathbf{v}) &= \mathbf{0} \\ \Rightarrow A^\top(\mathbf{y} - A\hat{\mathbf{x}}) &= \mathbf{0} \\ \Rightarrow A^\top A\hat{\mathbf{x}} &= A^\top \mathbf{y} \\ \Rightarrow \hat{\mathbf{x}} &= (A^\top A)^{-1} A^\top \mathbf{y} \\ \Rightarrow \mathbf{v} &= A\hat{\mathbf{x}} = A(A^\top A)^{-1} A^\top \mathbf{y} \end{aligned}$$

因此向量 \mathbf{y} 在 $\mathcal{R}(A)$ 上的投影如下：

$$\text{Proj}(\mathbf{y}; A) = \underset{\mathbf{v} \in \mathcal{R}(A)}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{v}\| = A\hat{\mathbf{x}} = A(A^\top A)^{-1} A^\top \mathbf{y}. \quad (2.12)$$

当矩阵 A 仅包含一列 $\mathbf{a} \in \mathbb{R}^m$ 时， \mathbf{y} 在 $\mathcal{R}(A)$ 上的投影等价于 \mathbf{y} 在 \mathbf{a} 上的投影。假设该投影为 $\hat{\mathbf{x}}\mathbf{a}$, $\hat{\mathbf{x}} \in \mathbb{R}$ ，类似地当 $\mathbf{y} - \hat{\mathbf{x}}\mathbf{a}$ 垂直于向量 \mathbf{a} 时 $\hat{\mathbf{x}}\mathbf{a}$ 为向量 \mathbf{y} 在向量 \mathbf{a} 上最近似投影。因此有：

$$\begin{aligned} \mathbf{a}^\top(\mathbf{y} - \hat{\mathbf{x}}\mathbf{a}) &= 0 \\ \Rightarrow \hat{\mathbf{x}} &= \frac{\mathbf{a}^\top \mathbf{y}}{\mathbf{a}^\top \mathbf{a}} \\ \Rightarrow \mathbf{v} &= \hat{\mathbf{x}}\mathbf{a} = \mathbf{a}\hat{\mathbf{x}} = \frac{\mathbf{a}\mathbf{a}^\top}{\mathbf{a}^\top \mathbf{a}} \mathbf{y} \end{aligned}$$

因此有：

$$\text{Proj}(\mathbf{y}; A) = \underset{\mathbf{v} \in \mathcal{R}(A)}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{v}\| = \frac{\mathbf{a}\mathbf{a}^\top}{\mathbf{a}^\top \mathbf{a}} \mathbf{y} \quad (2.13)$$

矩阵 $A \in \mathbb{R}^{m \times n}$ 的 **nullspace** 的表示为 $\mathcal{N}(A)$ 由所有和矩阵 A 相乘结果为零向量的向量构成，表示如下：

$$\mathcal{N}(A) = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = 0\}. \quad (2.14)$$

可以看到列空间 $\mathcal{R}(A)$ 中的所有向量的维数为 m ，而零空间 $\mathcal{N}(A)$ 中的所有向量的维数为 n ，因此列空间 $\mathcal{R}(A^\top)$ 和零空间 $\mathcal{N}(A)$ 的向量的维数均为 n ，且存在如下关系：

$$\{\mathbf{w} : \mathbf{w} = \mathbf{u} + \mathbf{v}, \mathbf{u} \in \mathcal{R}(A^\top), \mathbf{v} \in \mathcal{N}(A)\} = \mathbb{R}^n \quad \text{且} \quad \mathcal{R}(A^\top) \cap \mathcal{N}(A) = \emptyset. \quad (2.15)$$

也就是说集合 $\mathcal{R}(A^\top)$ 和 $\mathcal{N}(A)$ 没有交集，其并集构成了整个集合 \mathbb{R}^n 。存在如上关系的两个集合称为正交互补集合，表示为 $\mathcal{R}(A^\top) = \mathcal{N}(A)^\perp$

2.4.6 行列式

矩阵 A 的行列式表示为 $\det(A) = |A|$ ，在看如何具体计算矩阵的行列式之前，我们首先从几何角度理解行列式的意义。给定如下矩阵

$$\begin{bmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ \vdots & & \\ - & \mathbf{a}_n^\top & - \end{bmatrix}$$

集合 $S \subset \mathbb{R}^n$ 由矩阵 A 的所有行向量 $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{R}^n$ 的线性组合构成，线性组合的所有系数属于闭区间 $[0, 1]$ ；也就是说集合 S 定义如下：

$$S = \{\mathbf{v} \in \mathbb{R}^n : \mathbf{v} = \sum_{i=1}^n \alpha_i \mathbf{a}_i \quad \text{where } 0 \leq \alpha_i \leq 1, i = 1, \dots, n\}.$$

行列式的绝对值为集合 S 的基数。

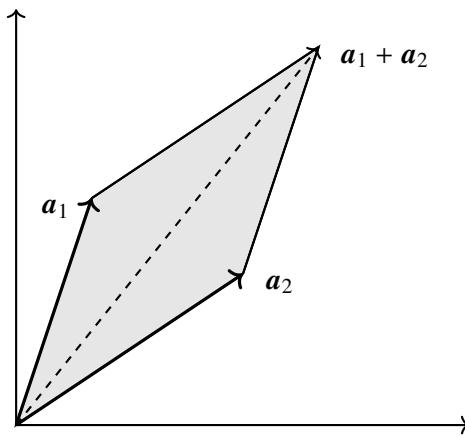
例如考虑如下的 2×2 矩阵，

$$A = \begin{bmatrix} 1 & 3 \\ 3 & 2 \end{bmatrix}$$

该矩阵的行向量为：

$$\mathbf{a}_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \mathbf{a}_2 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

由这两个向量构成的集合 S ，如下图所示：



对于二维矩阵来说，集合 S 一般对应为平行四边形 (parallelogram)，上面例子中，矩阵 A 的行列式的值为 $|A| = -7$ ，因此该四边形的面积为 7。对于三维矩阵来说，集合 S 一般对应为平行六面体 (parallelipiped)，矩阵 $A \in \mathbb{R}^{3 \times 3}$ 的行向量定义了集合 S ，该集合的基数即为平行六面体的体积，对应为行列式的绝对值。对于更高维的矩阵来说，集合 S 对应为 n 维的超平行体 (parallelotope)。对于任意矩阵 $A \in \mathbb{R}^{n \times n}$, $A_{\setminus i, \setminus j} \in \mathbb{R}^{(n-1) \times (n-1)}$ 表示删除

了矩阵 A 的 i 行和 j 列后构成的矩阵。行列式的一般 (递归) 定义公式如下:

$$|A| = \sum_{i=1}^n (-1)^{i+j} A_{i,j} |A_{\setminus i, \setminus j}| \quad (\text{for any } j \in 1, \dots, n) \quad (2.16)$$

$$= \sum_{j=1}^n (-1)^{i+j} A_{i,j} |A_{\setminus i, \setminus j}| \quad (\text{for any } i \in 1, \dots, n) \quad (2.17)$$

初始值 $|A| = A_{1,1}$, $A \in \mathbb{R}^{1 \times 1}$ 。若将上述公式完全展开, 对于 $A \in \mathbb{R}^{n \times n}$ 来说, 将有 $n!$ 不同项。因此我们很少显式写出超过 3×3 维数的矩阵的行列式公式。

行列式具有如下性质

1. 单位矩阵的行列式为 1, 即 $\det(A) = 1$
2. 交换矩阵的任意两行行列式符号修改为相反值

$$\det(A) = \begin{cases} |A| & \text{偶数次行交换} \\ -|A| & \text{奇数次行交换} \end{cases}$$

3. 给定矩阵 $A \in \mathbb{R}^{n \times n}$, 若对矩阵 A 的某一行乘以标量 $t \in \mathbb{R}$, 那么新矩阵的行列式为 $t|A|$
4. 存在相同行的矩阵 $A \in \mathbb{R}^{n \times n}$, 其行列式为 0, 该性质可以由性质 2 推知
5. 若矩阵 $A, B \in \mathbb{R}^{n \times n}$, 则有 $|AB| = |A||B|$
6. 矩阵的转置的行列式等于矩阵的行列式, 即: $\det(A^\top) = \det(A)$, 该性质可以通过对矩阵 A 进行 LU 分解推知
7. 若矩阵 $A \in \mathbb{R}^{n \times n}$, 当且仅当矩阵 A 为奇异矩阵 (即不可逆) 时, 其行列式 $|A| = 0$ (若矩阵 A 为奇异矩阵, 矩阵 A 为非满秩矩阵, 因此其列向量线性相关, 在此情况下, 集合 S 对应 n 维空间中的一个平的薄片 (二维情况下为一条线), 因此其容量为 0)。
8. 若矩阵 $A \in \mathbb{R}^{n \times n}$ 且矩阵 A 为非奇异矩阵, 此时有 $|A^{-1}| = 1/|A|$

2.5 特征值和特征向量 (Eigenvalue and Eigenvector)

本节回顾线性代数中涉及矩阵的最重要的特征值及特征向量的概念, 及在此基础上涉及的矩阵分解等重要概念

2.5.1 特征值及特征向量

给定方阵 $A \in \mathbb{R}^{n \times n}$, 我们称 $\lambda \in \mathbb{R}$ 为该矩阵的特征值, $x \in \mathbb{R}^n$ 为相应的特征向量, 若其满足如下公式:

$$Ax = \lambda x, \quad x \neq \mathbf{0}. \quad (2.18)$$

直观上看, 该定义意味着对向量 x 乘以矩阵 A 生成一个新的向量, 该向量和向量 x 指向同一个方向, 只是对向量采用因子 λ 进行了缩放。同时也可以看到, 对任意特征向量

$\mathbf{x} \in \mathbb{R}^n$, 及标量 $c \in \mathbb{R}$, 有 $\mathbf{A}(c\mathbf{x}) = c\mathbf{A}\mathbf{x} = c\lambda\mathbf{x} = \lambda(c\mathbf{x})$, 因此 $c\mathbf{x}$ 也是特征向量。基于该原因, 当我们谈及特征值 λ 的特征向量时, 我们一般假设为归一化特征向量, 即其长度为 1。(即使这样, 依然存在某些歧义, 因为 \mathbf{x} 和 $-\mathbf{x}$ 均为特征向量)。

我们可以将等式2.18为如下形式, 表明 (λ, \mathbf{x}) 为矩阵 \mathbf{A} 的某个特征值-特征向量对, 若存在如下公式:

$$(\lambda\mathbf{I} - \mathbf{A})\mathbf{x} = 0, \quad \mathbf{x} \neq 0$$

但当且仅当 $(\lambda\mathbf{I} - \mathbf{A})$ 存在非空的零空间时, 公式 $(\lambda\mathbf{I} - \mathbf{A})\mathbf{x} = 0$ 才存在非零解 \mathbf{x} , 此时 $(\lambda\mathbf{I} - \mathbf{A})$ 为奇异矩阵, 即

$$|(\lambda\mathbf{I} - \mathbf{I})| = 0.$$

现在我们可以使用之前给出的行列式定义公式将该表达式展开为 λ 的多项式形式, λ 的最高阶为 n , 然后我们可以解出该表达式的 n 个根(可能为复数), 生成特征值 $\lambda_1, \dots, \lambda_n$ 。有了特征 λ_i 后, 可以通过解线性方程 $(\lambda_i\mathbf{I} - \mathbf{A})\mathbf{x} = 0$ 生成对应的特征向量。有必要说明的是, 该方法并非实际数值方法计算特征值和特征向量真实所用的方法。

特征值和特征向量具有如下属性(其中 $\mathbf{A} \in \mathbb{R}^{n \times n}$, 相应的特征值为 $\lambda_1, \dots, \lambda_n$, 特征值对应的特征向量为 $\mathbf{x}_1, \dots, \mathbf{x}_n$):

- 矩阵 \mathbf{A} 的 Trace 为所有特征值之和

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^n \lambda_i.$$

- 矩阵 \mathbf{A} 的行列式为所有特征值乘积

$$|\mathbf{A}| = \prod_{i=1}^n \lambda_i.$$

- 矩阵 \mathbf{A} 的秩为非零特征值的个数
- 若矩阵 \mathbf{A} 为非奇异矩阵, 则 $1/\lambda_i$ 为矩阵 \mathbf{A}^{-1} 的特征值, 其对应的特征向量为 \mathbf{x}_i , 也就是说 $\mathbf{A}^{-1}\mathbf{x}_i = (1/\lambda_i)\mathbf{x}_i$
- 对角矩阵 $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$ 的特征值为对角线元素 d_1, \dots, d_n 。

我们可以将所有特征向量等式写为如下形式

$$\mathbf{AX} = \mathbf{A} \begin{bmatrix} | & \cdots & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & \cdots & | \\ \lambda_1\mathbf{x}_1 & \cdots & \lambda_n\mathbf{x}_n \\ | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & \cdots & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & 0 \\ 0 & \cdots & \lambda_n \end{bmatrix} = \mathbf{X}\Lambda$$

其中 $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ 。若矩阵 \mathbf{A} 的特征向量彼此线性无关, 则矩阵 \mathbf{X} 可逆, 因此 $\mathbf{A} = \mathbf{X}\Lambda\mathbf{X}^{-1}$, 可以书写为该形式的矩阵称为可对角化矩阵。

2.5.2 对称矩阵

对于对称矩阵 (Symmetric Matrix) $A \in \mathbb{R}^{n \times n}$ 来说，特征值和特征向量具有如下两个优良性质：

- 对称矩阵的所有特征值均为实数
- 对称矩阵的所有特征向量均为正交单位向量

因此由特征向量构建而成的矩阵 $X \in \mathbb{R}^{n \times n}$ 为正交矩阵（因此可以用 U 表示特征向量构成的矩阵），因此矩阵 A 可以表示为 $A = U\Lambda U^\top$ 。基于上述的表示形式，可以发现矩阵 A 的定性 (definiteness) 完全依赖于其特征值的符号。假设 $A \in \mathbb{R}^{n \times n} = U\Lambda U^\top$ ，那么有

$$\mathbf{x}^\top A \mathbf{x} = \mathbf{x}^\top U \Lambda U^\top \mathbf{x} = \mathbf{y}^\top \Lambda \mathbf{y} = \sum_{i=1}^n \lambda_i y_i^2 \quad (2.19)$$

其中 $\mathbf{y} = U^\top \mathbf{x}$ （由于 U 为满秩矩阵，因此任意 $\mathbf{y} \in \mathbb{R}^n$ 均可以表示为该形式）。由于 y_i^2 总是正的，因此该表达式的符号完全依赖于 λ_i 。若所有的 $\lambda_i > 0$ ，则矩阵为正定矩阵；若所有的 $\lambda_i \geq 0$ ，则为半正定矩阵。类似地，若所有 $\lambda_i < 0$ 或 $\lambda_i \leq 0$ ，则矩阵 A 为负定或半负定矩阵。最后，若 A 同时有正和负的特征值，则该矩阵为不定矩阵。

特征值和特征向量常用于求解和矩阵相关的最大值或最小值，特别当矩阵为对称矩阵时，即 $A \in \mathbb{R}^{n \times n}$ ，考虑如下的最大值问题，

$$\max_{\mathbf{x} \in \mathbb{R}^n} \mathbf{x}^\top A \mathbf{x} \quad \text{subject to } \|\mathbf{x}\|^2 = 1$$

也就说，我们希望求得某个向量（其二范数为 1），该向量最大化上述二次型。假设将特征值排序为形式 $\lambda_1 \geq \lambda_2 \geq \dots \lambda_n$ ，该优化问题最优的解则为特征向量 \mathbf{x}_1 ，其对应的特征值为 λ_1 ，因此该二次型的最大值为 λ_1 。类似的，求解如下二次型的最小值

$$\min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{x}^\top A \mathbf{x} \quad \text{subject to } \|\mathbf{x}\|^2 = 1$$

其对应的解为特征向量 \mathbf{x}_n ，相应的特征值为 λ_n ，该二次型的最小值为 λ_n 。可以通过将 \mathbf{x} 表示为特征向量的线性组合，即 $\mathbf{x} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$ ， $\sum_{i=1}^n \alpha_i = 1$ ，且结合对称矩阵的特征值和特征向量很容易进行证明，当 $\alpha_1 = 1$ 时，二次型取得最大值，当 $\alpha_n = 1$ 时，二次型取得最小值。

2.5.3 SVD 分解

对于方阵 $A \in \mathbb{R}^{n \times n}$ 我们可以将其分解为如下形式：

$$A = X \Lambda X^{-1}$$

若矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$, 则存在正交矩阵 $\mathbf{U} \in \mathbb{R}^{m \times m}$ 及正交矩阵 $\mathbf{V} \in \mathbb{R}^{n \times n}$, 此时矩阵 \mathbf{A} 满足如下等式:

$$\mathbf{A} = \mathbf{U} \mathbf{D} \mathbf{V}^\top. \quad (2.20)$$

上述等式中 $\mathbf{D} \in \mathbb{R}^{m \times n}$ 为对角矩阵。对角矩阵 \mathbf{D} 的对角线元素为矩阵 \mathbf{A} 的奇异值。正交矩阵 \mathbf{U} 的列向量称为左奇异向量, 这部分向量为 $\mathbf{A}\mathbf{A}^\top$ 的特征向量; 正交矩阵 \mathbf{V} 的列向量称为右奇异向量, 这部分向量为 $\mathbf{A}^\top\mathbf{A}$ 的特征向量, 矩阵 \mathbf{A} 的非零奇异值为 $\mathbf{A}^\top\mathbf{A}$ 的特征值的平方根。下面给出 SVD 分解的证明。

奇异值定义: 给定矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$, 则 $\mathbf{A}^\top\mathbf{A}$ 为对称矩阵, 因此可以被正交对角化。假设 $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ 为空间 \mathbb{R}^n 的单位正交基, 包含 $\mathbf{A}^\top\mathbf{A}$ 的特征向量, 且 $\lambda_1, \dots, \lambda_n$ 为 $\mathbf{A}^\top\mathbf{A}$ 的相应的特征值。因此对 $1 \leq i \leq n$, 有:

$$\begin{aligned} \|\mathbf{A}\mathbf{v}_i\|^2 &= (\mathbf{A}\mathbf{v}_i)^\top \mathbf{A}\mathbf{v}_i = \mathbf{v}_i^\top \mathbf{A}^\top \mathbf{A}\mathbf{v}_i \\ &= \mathbf{v}_i^\top (\lambda_i \mathbf{v}_i) \\ &= \lambda_i \end{aligned}$$

因此 $\mathbf{A}^\top\mathbf{A}$ 的特征值均非负。另外我们可以假设特征值满足如下定义:

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$$

矩阵 \mathbf{A} 的奇异值为矩阵 $\mathbf{A}^\top\mathbf{A}$ 的特征值的平方根, 表示为: $\sigma_1, \dots, \sigma_n$, 且满足降序排列, 因此有: $\sigma_i = \sqrt{\lambda_i}, \quad 1 \leq i \leq n$ 。根据上述定义可知, 矩阵 \mathbf{A} 的奇异值为向量 $\mathbf{A}\mathbf{v}_1, \dots, \mathbf{A}\mathbf{v}_n$ 的长度。

定理 1: 设 $\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ 是由 $\mathbf{A}^\top\mathbf{A}$ 的特征向量所构成的 \mathbb{R}^n 的一组标准正交基, 该集合中特征向量根据其相应的特征值排序, 满足 $\lambda_1 \geq \dots \geq \lambda_n$, 假设矩阵 \mathbf{A} 有 r 个非零奇异值, 那么 $\{\mathbf{A}\mathbf{v}_1, \dots, \mathbf{A}\mathbf{v}_r\}$ 为矩阵 \mathbf{A} 的列向量空间一组正交基, 且 $\text{rank}(\mathbf{A}) = r$ 。

证明: 由于当 $i \neq j$ 时, \mathbf{v}_i 和 $\lambda_j \mathbf{v}_j$ 正交, 所以:

$$(\mathbf{A}\mathbf{v}_i)^\top (\mathbf{A}\mathbf{v}_j) = \mathbf{v}_i^\top \mathbf{A}^\top \mathbf{A}\mathbf{v}_j = \mathbf{v}_i^\top (\lambda_j \mathbf{v}_j) = 0$$

因此 $\{\mathbf{A}\mathbf{v}_1, \dots, \mathbf{A}\mathbf{v}_n\}$ 为正交集合。更进一步, 由于向量 $\mathbf{A}\mathbf{v}_1, \dots, \mathbf{A}\mathbf{v}_n$ 的长度是 \mathbf{A} 的奇异值, 并且存在 r 个非零奇异值, 所以 $\mathbf{A}\mathbf{v}_i \neq 0$ 当且仅当 $1 \leq i \leq r$ 。因此 $\mathbf{A}_1, \dots, \mathbf{A}_r$ 线性无关, 并且属于 \mathbf{A} 的列空间, 对于矩阵 \mathbf{A} 的列空间中任意向量 \mathbf{y} , $\mathbf{y} = \mathbf{A}\mathbf{x}$, 其中向量 \mathbf{x} 可以表示为 $\mathbf{x} = c_1\mathbf{v}_1 + \dots + c_n\mathbf{v}_n$, 此时有:

$$\begin{aligned} \mathbf{y} &= \mathbf{A}\mathbf{x} = c_1\mathbf{A}\mathbf{v}_1 + \dots + c_r\mathbf{A}\mathbf{v}_r + c_{r+1}\mathbf{A}\mathbf{v}_{r+1} + \dots + c_n\mathbf{A}\mathbf{v}_n \\ &= c_1\mathbf{A}\mathbf{v}_1 + \dots + c_r\mathbf{A}\mathbf{v}_r + 0 + \dots + 0 \end{aligned}$$

因此 \mathbf{y} 属于空间 $\text{span}\{\mathbf{A}\mathbf{v}_1, \dots, \mathbf{A}\mathbf{v}_r\}$, 这表明 $\{\mathbf{A}\mathbf{v}_1, \dots, \mathbf{A}\mathbf{v}_r\}$ 是矩阵 \mathbf{A} 的列空间一组正交基。

奇异值分解：设矩阵 $A \in \mathbb{R}^{m \times n}$ 的秩为 r ，则存在如下对角阵

$$D = \begin{bmatrix} \sigma_1 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & 0 \\ 0 & \cdots & \sigma_r & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}$$

其中 D 的对角线元素是 A 的前 r 个奇异值： $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ ，且存在正交矩阵 $U \in \mathbb{R}^{m \times m}$ 及正交矩阵 $V \in \mathbb{R}^{n \times n}$ ，使得：

$$A = UDV^\top.$$

上述分解式中 U 为 A 的左奇异向量， V 为 A 的右奇异向量。证明：设 λ_i 及 v_i 同定理 1，因此 $\{Av_1, \dots, Av_r\}$ 为矩阵 A 的列向量空间一组正交基。对每个正交基向量 Av_i 进行标准化，可以得到标准正交基 $\{u_1, \dots, u_r\}$ ，其中：

$$u_i = \frac{1}{\|Av_i\|} Av_i = \frac{1}{\sigma_i} Av_i$$

因此有：

$$Av_i = \sigma_i u_i \quad (1 \leq i \leq r)$$

现在将 $\{u_1, \dots, u_r\}$ 扩充成 \mathbb{R}^m 的一组标准正交基 $\{u_1, \dots, u_m\}$ ，并且令：

$$U = \begin{bmatrix} | & \cdots & | \\ u_1 & \cdots & u_m \\ | & \cdots & | \end{bmatrix} \quad \text{及} \quad V = \begin{bmatrix} | & \cdots & | \\ v_1 & \cdots & v_n \\ | & \cdots & | \end{bmatrix}$$

有上述构造可知 U 和 V 为正交矩阵，且：

$$AV = \begin{bmatrix} | & \cdots & | & | & \cdots & | \\ Av_1 & \cdots & Av_r & \mathbf{0} & \cdots & 0 \\ | & \cdots & | & | & \cdots & | \end{bmatrix} = \begin{bmatrix} | & \cdots & | & | & \cdots & | \\ \sigma_1 u_1 & \cdots & \sigma_r u_r & \mathbf{0} & \cdots & \mathbf{0} \\ | & \cdots & | & | & \cdots & | \end{bmatrix}$$

因此：

$$UD = \begin{bmatrix} | & \cdots & | \\ u_1 & \cdots & u_m \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} \sigma_1 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & 0 \\ 0 & \cdots & \sigma_r & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} | & \cdots & | & | & \cdots & | \\ \sigma_1 u_1 & \cdots & \sigma_r u_r & \mathbf{0} & \cdots & \mathbf{0} \\ | & \cdots & | & | & \cdots & | \end{bmatrix} = AV$$

由于 V 为正交矩阵，因此 $UDV^\top = AVV^\top = A$ 。SVD 分解可以用于求解矩阵的伪逆及

矩阵二范数，这里我们看一下 SVD 求解伪逆的应用，矩阵二范数的求解留待矩阵范数小节回顾。

伪逆求解：对于非方阵，矩阵的逆是未定义的。假设我们想对如下线性方程中的矩阵 \mathbf{A} 求解其左逆 \mathbf{B} ：

$$\mathbf{Ax} = \mathbf{y}$$

通过对上述方程中两边同时左乘 \mathbf{B} ，生成解 \mathbf{x} ：

$$\mathbf{x} = \mathbf{B}\mathbf{y}.$$

根据矩阵 \mathbf{A} 的形态，有可能无法设计从 \mathbf{A} 到 \mathbf{B} 的唯一映射。若 $\mathbf{A} \in \mathbb{R}^{m \times n}$ ，其中 $m < n$ ，则线程方程可能无解；若 $m > n$ ，则可能存在若干解。可以通过下方程求解 \mathbf{A} 的伪逆：

$$\mathbf{A}^+ = \mathbf{V}\mathbf{D}^+\mathbf{U}^\top \quad (2.21)$$

其中 $\mathbf{U}, \mathbf{D}, \mathbf{V}$ 为矩阵 \mathbf{A} 的 SVD 分解，伪逆 \mathbf{D}^+ 通过对矩阵 \mathbf{D} 的非零元素求倒数后求转置生成。当矩阵 \mathbf{A} 的列数大于行数时，使用伪逆求解线性方程时提供了若干解中某个解。特别地是，解 $\mathbf{x} = \mathbf{A}^+\mathbf{y}$ 在所有可能解中二范数最小；当矩阵 \mathbf{A} 的行数大于列数时，线性方程可能无解。在该情况下，使用伪逆求解得到的 \mathbf{x} 对应的 \mathbf{Ax} 和 \mathbf{y} 的欧几里得距离最小。

2.6 范数

本节回顾向量范数及在其基础上衍生出的矩阵范数的概念，矩阵范数对研究线性系统的稳态具有重要意义，为研究深度学习中的反向传播算法所涉及的梯度爆炸和梯度消散奠定了良好的基础

2.6.1 向量范数

本节我们回顾向量的 p 范数，并重点考察 1 范数、2 范数及 ∞ 范数，范数描述了向量的大小。向量的 p 范数 L^p 定义如下：

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}} \quad p \in \mathbb{R}, p \geq 1. \quad (2.22)$$

直观上看，向量 \mathbf{x} 的范数衡量了原点到点 \mathbf{x} 的距离。严格定义上看，范数，包含 L^p 为满足如下属性的任意函数 f ：

- $f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = 0$
- $f(\mathbf{x} + \mathbf{y}) < f(\mathbf{x}) + f(\mathbf{y})$ (三角不等式)
- $\forall \alpha \in \mathbb{R}, f(\alpha\mathbf{x}) = |\alpha|f(\mathbf{x})$

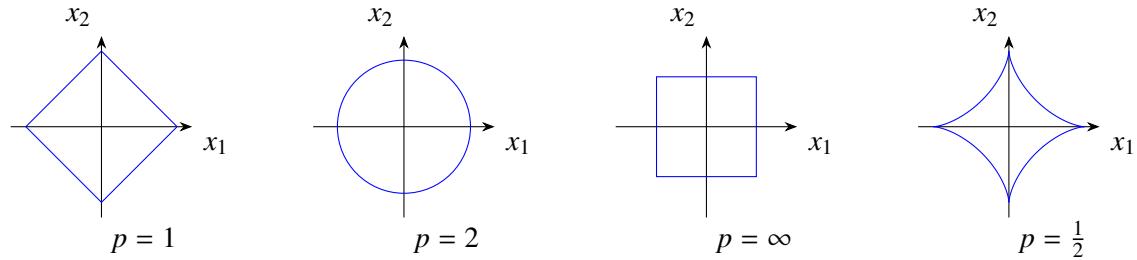
从如上定义可见二范数 $L^2 = \mathbf{x}^\top \mathbf{x}$ 。向量的 1 范数 L^1 为如下形式：

$$L^1 = \sum_i |x_i|. \quad (2.23)$$

向量的 ∞ 范数为向量中绝对值最大的元素的绝对值

$$\|\mathbf{x}\|_\infty = \max_i |x_i|. \quad (2.24)$$

这里给出二维向量下，向量 1 范数，2 范数， ∞ 及 $\frac{1}{2}$ 范数的等高线形态：



2.6.2 矩阵范数

矩阵范数的定义由向量范数延伸而来，矩阵范数的定义如下所示：

$$\|A\|_p = \sup_{\mathbf{x} \neq \mathbf{0}} \frac{\|A\mathbf{x}\|_p}{\|\mathbf{x}\|_p} \quad (2.25)$$

上述公式中的 sup 表示上界。可以看到矩阵范数表示对任意向量进行变换后的最大放大系数。由于 $\|\mathbf{x}\|_p$ 为标量，因此公式 2.25 可以改写为如下形式：

$$\|A\|_p = \sup_{\mathbf{x} \neq \mathbf{0}} \left\| A \frac{\mathbf{x}}{\|\mathbf{x}\|_p} \right\|_p = \sup_{\|\mathbf{x}\|_p=1} \|A\mathbf{x}\|_p. \quad (2.26)$$

矩阵 1 范数：由向量的 1 范数定义

$$\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|.$$

可以推知矩阵的 1 范数为如下形式：

$$\|A\|_1 = \max_j \left(\sum_i |A_{i,j}| \right) \quad (2.27)$$

也就是说矩阵的 1 范数为所有列向量的 1 范数的最大值。这里给出具体的证明，假设矩

阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$, 表示为如下列向量形式

$$\mathbf{A} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ | & | & \cdots & | \end{bmatrix}$$

因此

$$\mathbf{Ax} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{k=1}^n \mathbf{a}_k x_k$$

有三角不等式可知:

$$\begin{aligned} \|\mathbf{Ax}\|_1 &= \left\| \sum_{k=1}^n \mathbf{a}_k x_k \right\| \leq \sum_{k=1}^n |x_k| \|\mathbf{a}_k\|_1 \\ &\leq \max_j \|\mathbf{a}_j\|_1 \left(\sum_{k=1}^n |x_k| \right) \\ &= \max_j \|\mathbf{a}_j\|_1 \|\mathbf{x}\|_1. \end{aligned}$$

因此当

$$\mathbf{x} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

其中 \mathbf{x} 的第 j 个位置为 1, 其他位置元素为 0 时, 有 $\|\mathbf{x}\|_1 = 1$, 此时有

$$\|\mathbf{A}\|_1 = \max_j \left(\sum_i |A_{i,j}| \right).$$

矩阵 ∞ 范数: 由向量的 ∞ 范数定义

$$\|\mathbf{x}\|_\infty = \max_k |x_k|$$

可知为向量 \mathbf{x} 最大项。类似可以推知, 矩阵的 ∞ 范数为:

$$\|\mathbf{A}\|_\infty = \max_i \left(\sum_j |A_{i,j}| \right).$$

从上述定义可见，矩阵的 ∞ 范数为行向量 1 范数的最大值。假设 $A \in \mathbb{R}^{m \times n}$ ，因此有：

$$\begin{aligned}\|Ax\|_\infty &= \left\| \begin{bmatrix} \sum_k A_{1,k}x_k \\ \sum_k A_{2,k}x_k \\ \vdots \\ \sum_k A_{m,k}x_k \end{bmatrix} \right\|_\infty = \max_i \left| \sum_k A_{i,k}x_k \right| \leq \max_i \left(\sum_k |A_{i,k}x_k| \right) \\ &\leq \max_i \left(\sum_k |A_{i,k}| \right) \max_k |x_k| \\ &= \max_i \left(\sum_k |A_{i,k}| \right) \|x\|_\infty\end{aligned}$$

可以选择如下向量 x 求得矩阵的 A 的 ∞ 范数

$$x_k = \begin{cases} \frac{A_{i,k}}{|A_{i,k}|}, & A_{i,k} \neq 0, 1 \leq k \leq n \\ 1 & A_{i,k} = 0 \end{cases}$$

其中 i 为行向量最大 1 范数对应的行下标。

矩阵 2 范数：由向量的 2 范数定义

$$\|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2} = \sqrt{\langle x, x \rangle}.$$

由向量的二范数定义，可以推知矩阵的 2 范数为

$$\|A\|_2 = \sqrt{\text{largest eigenvalue of } A^*A} \quad (2.28)$$

上述公式中 A^* 为 A 的共轭转置，当 A 为实数矩阵时， $A^* = A^\top$ 。对于任意矩阵 $A \in \mathbb{R}^{m \times n}$ ， $A^*A \in \mathbb{R}^{n \times n}$ 为 Hermitian 矩阵¹。由对称矩阵的性质可知 A^*A 的特征值为实数值，且 A^*A 至少为半正定矩阵，因为：

$$x^*(A^*A)x = (Ax)^*(Ax) \geq 0 \quad \text{for all } x$$

因此 A^*A 的特征值均为实数且非负，假设将其特征值按如下顺序排序：

$$\sigma_1^2 \geq \sigma_2^2 \geq \sigma_3^2 \geq \dots \geq \sigma_n^2 \geq 0$$

可以看到上述特征值的顺序满足从大到小，其中 σ_1^2 为最大的特征值，结合 SVD 分解可以看到这些特征值的平方根即为矩阵 A 的奇异值。和这些特征值对应的是 n 个标准正交

¹https://en.wikipedia.org/wiki/Hermitian_matrix

的特征向量 $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n$, 因此满足如下公式:

$$(\mathbf{A}^* \mathbf{A}) \mathbf{u}_i = (\sigma_i^2) \mathbf{u}_i \quad 1 \leq i \leq n$$

这 n 个标准正交向量构成了 Unitary 矩阵 ² $\mathbf{U} \in \mathbb{R}^{n \times n}$, 有对称矩阵的性质可知

$$\mathbf{U}^* \mathbf{A}^* \mathbf{A} \mathbf{U} = \Lambda$$

由于 n 个标准正交特征向量 $\mathbf{u}_1, \dots, \mathbf{u}_n$ 彼此线性无关, 因此可以作为空间 \mathbb{R}^n 的一组基向量, 因此任意向量 \mathbf{x} 可以由基向量表示, 即存在如下形式:

$$\mathbf{x} = \sum_{i=1}^n c_i \mathbf{u}_i.$$

其中 c_i 为 \mathbf{x} 相关的标量。因此对 \mathbf{x} 进行矩阵 $\mathbf{A}^* \mathbf{A}$ 变换, 有:

$$\mathbf{A}^* \mathbf{A} \mathbf{x} = \mathbf{A}^* \mathbf{A} \sum_{i=1}^n c_i \mathbf{u}_i = \sum_{i=1}^n c_i \sigma_i^2 \mathbf{u}_i$$

因此对任意向量 \mathbf{x} 有:

$$\begin{aligned} \|\mathbf{A}\mathbf{x}\|^2 &= (\mathbf{A}\mathbf{x})^* \mathbf{A}\mathbf{x} = \mathbf{x}^* (\mathbf{A}^* \mathbf{A} \mathbf{x}) \\ &= \left(\sum_{i=1}^n c_i \mathbf{u}_i^* \right) \left(\sum_{j=1}^n c_j \sigma_j^2 \mathbf{u}_j \right) = \sum_{i=1}^n c_i^2 \sigma_i^2 \\ &\leq \sigma_1^2 \left(\sum_{i=1}^n c_i^2 \right) = \sigma_1^2 \|\mathbf{x}\|^2 \end{aligned}$$

可以看到当我们取 $\mathbf{x} = \mathbf{u}_1$ 时, 矩阵的二范数取得最大值 σ_1 。

通过如上矩阵二范数求解过程可以看出, 某个输入量 (向量 \mathbf{x}) 经过线性系统 (矩阵 \mathbf{A}) 线性变换后向量的大小 (二范数) 最大变为原来的 σ_1 倍, 该性质对研究如下线性系统的稳态具有重要作用:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$$

可以看出当 $\sigma_1 > 1$ 时经过若干次变换后, 向量的大小将被指数放大; 当 $\sigma_1 < 1$ 时, 经过若干次变换后, 向量的大小将被指数衰减; 当 $\sigma_1 = 1$ 时, 线性系统将处于稳态。该性质对理解深度神经网络构成的动态系统涉及的反向传播算法至关重要, 通过观察反向传播算法的计算过程 (若干次线性变换), 我们可以理解为何要研究权值初始化, 激活函数或网络结构 (譬如 LSTM 替代 Elman network), 这些内容后续我们会有专门的章节一起回顾经典的工作及这些工作是如何解决反向传播中遇到的梯度爆炸或梯度消散问题的。

²https://en.wikipedia.org/wiki/Unitary_matrix

2.7 向量化导数

本节回顾不同维度输入及输出下的导数形式

2.7.1 梯度 ($f : \mathbb{R}^n \rightarrow \mathbb{R}$)

假设函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 接受向量作为输入，输出为标量。函数 f 在点 $\mathbf{x} \in \mathbb{R}^n$ 处的导数称为梯度，定义为如下形式：

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \lim_{\mathbf{h} \rightarrow \mathbf{0}} \frac{f(\mathbf{x} + \mathbf{h}) - f(\mathbf{x})}{\|\mathbf{h}\|}$$

从上述公式可以看出梯度 $\nabla_{\mathbf{x}} f(\mathbf{x}) \in \mathbb{R}^n$ 为向量，和输入输出均为标量的类似，假设我们设定 $y = f(\mathbf{x})$ ，存在如下公式：

$$\mathbf{x} \rightarrow \mathbf{x} + \Delta \mathbf{x} \Rightarrow y \rightarrow y + \frac{\partial y}{\partial \mathbf{x}} \cdot \Delta \mathbf{x}$$

上述公式和输入输出均为标量时稍微有所不同的是，这里的 $\mathbf{x}, \Delta \mathbf{x}$ 及 $\frac{\partial y}{\partial \mathbf{x}}$ 均为 \mathbb{R}^n 空间中的向量，而 y 为标量。同时对 $\frac{\partial y}{\partial \mathbf{x}}$ 乘以 $\Delta \mathbf{x}$ 时，我们使用了点积，将两个向量的结合在一起产生一个标量。

上述公式带来的好处是，其给梯度 $\frac{\partial y}{\partial \mathbf{x}}$ 的每个元素赋予了相应的意义。假设 $\Delta \mathbf{x}$ 为第 i 个基向量 $\mathbf{e}^{(i)}$ ，因此 $\mathbf{e}^{(i)}$ 的第 i 维坐标为 1，所有其他的坐标为 0。因此点积 $\frac{\partial y}{\partial \mathbf{x}} \cdot \Delta \mathbf{x}$ 结果即为偏导度 $\frac{\partial y}{\partial \mathbf{x}}$ 的第 i 维坐标，因此偏导的第 i 维坐标显示了向量 \mathbf{x} 第 i 维坐标变化时， y 的近似变化量。这意味着，我们也可以将梯度看成由偏导构成的向量：

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix} \quad (2.29)$$

其中 x_i 为向量 \mathbf{x} 的第 i 维坐标，该值为标量，因此每个偏导数 $\frac{\partial y}{\partial x_i}$ 也是标量。

2.7.2 雅可比矩阵 ($f : \mathbb{R}^n \rightarrow \mathbb{R}^m$)

现在假设 $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 接受一个向量作为输入，产生一个向量作为输出。此时函数 f 在点 \mathbf{x} 处的梯度也称为雅可比矩阵，其为 $m \times n$ 维的偏导数矩阵。令 $\mathbf{y} = f(\mathbf{x})$ ，则有：

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \quad (2.30)$$

上述雅可比矩阵显示了向量 \mathbf{x} 的每个元素和向量 \mathbf{y} 的每个元素的关系：偏导数 $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ 的 (i, j) 维元素等于 $\frac{\partial y_i}{\partial x_j}$ ，因此雅可比矩阵显示了 x_j 的微小变化量会对 y_i 带来多大的变化。和前面梯度类似，雅可比矩阵显示输入变化和输出变化之间的关系：

$$\mathbf{x} \rightarrow \mathbf{x} + \Delta \mathbf{x} \Rightarrow \mathbf{y} \rightarrow \approx \mathbf{y} + \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \Delta \mathbf{x}$$

上述公式中的 $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ 为 $m \times n$ 的矩阵， $\Delta \mathbf{x}$ 为 n 维的向量，因此乘积 $\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \Delta \mathbf{x}$ 为矩阵和向量相乘，其结果为 m 的向量。链乘法则可以通过雅可比矩阵拓展至向量场景。假设 $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ ， $g : \mathbb{R}^m \rightarrow \mathbb{R}^k$ 。令 $\mathbf{x} \in \mathbb{R}^n$ $\mathbf{y} \in \mathbb{R}^m$ 且 $\mathbf{z} \in \mathbb{R}^k$ ，且 $\mathbf{y} = f(\mathbf{x})$ $\mathbf{z} = g(\mathbf{y})$ ，因此和标量计算类似，存在如下计算图：

$$\mathbf{x} \xrightarrow{f} \mathbf{y} \xrightarrow{g} \mathbf{z}$$

链乘法则形式和标量类似：

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

和标量计算有所不同的是，上述公式中每一项为雅可比矩阵： $\frac{\partial \mathbf{z}}{\partial \mathbf{y}}$ 为 $k \times m$ 的矩阵， $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ 为 $m \times n$ 的矩阵， $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ 为 $k \times n$ 的矩阵。

2.7.3 广义雅可比矩阵 ($f : \mathbb{R}^{n_1 \times \dots \times n_{d_x}} \rightarrow \mathbb{R}^{m_1 \times \dots \times m_{d_y}}$)

我们知道向量是一维的数列表，这一维的大小确定了列表的长度，矩阵是二维的数网格，每一维的长度，确定了网格在相应维度上的长度。在向量及矩阵基础上进一步进行概念拓展，张量是 d 维的数网格。深度学习中很多运算的输入是一个张量，输出也是一个张量。例如，一张图片通常表示为一个三维的数网格，三个维度分布对应于图片的高度、宽度及色彩通道（红、绿、蓝）。因此，我们需要给出处理张量的函数的导数。现在假设 $f : \mathbb{R}^{n_1 \times \dots \times n_{d_x}} \rightarrow \mathbb{R}^{m_1 \times \dots \times m_{d_y}}$ ，可知函数 f 的输入为 d_x 维的张量，其形状为 $n_1 \times \dots \times n_{d_x}$ ，输出为 d_y 维的张量，其形状为 $m_1 \times \dots \times m_{d_y}$ 。若 $\mathbf{Y} = f(\mathbf{X})$ ，则 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 为广义雅可比矩阵，其形状为：

$$(m_1 \times \dots \times m_{d_y}) \times (n_1 \times \dots \times n_{d_x})$$

从上述形状表示方式可以看出， $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 的维度被分成了两组：第一组和 \mathbf{Y} 的维度一致，第二组和 \mathbf{X} 的维度一致。通过该分组方法，我们可以将广义雅可比看成矩阵的推广，其中每“行”的形状和 \mathbf{Y} 一致，每“列”的形状和 \mathbf{X} 一致。假设，我们令 $i \in \mathbb{Z}^{d_y}$ 及 $j \in \mathbb{Z}^{d_x}$ 为整数索引构成的向量，因此存在如下等式：

$$\left(\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \right)_{i,j} = \frac{\partial \mathbf{Y}_i}{\partial \mathbf{X}_j}$$

上述公式中 \mathbf{Y}_i 及 \mathbf{X}_j 均为标量，因此 $\frac{\mathbf{Y}_i}{\mathbf{X}_j}$ 也是标量。通过该表示方式，我们可以看出和标准雅可比矩阵类似，广义雅可比矩阵显示了张量 \mathbf{X} 所有元素的变化和张量 \mathbf{Y} 所有元素的

变化间的关系。因此和梯度、及雅可比类似，广义雅可比矩阵同样给出了如下输入和输出间的关系式：

$$\mathbf{X} \rightarrow \mathbf{X} + \Delta \mathbf{X} \Rightarrow \mathbf{Y} \rightarrow \approx \mathbf{Y} + \frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \Delta \mathbf{X}$$

和之前所不同的是，现在 $\Delta \mathbf{X}$ 为形状 $n_1 \times \cdots \times n_{d_x}$ 的张量，且 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 为形状为 $(m_1 \times \cdots \times m_{d_y}) \times (n_1 \times \cdots \times n_{d_x})$ 的广义矩阵。因此 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \Delta \mathbf{X}$ 的乘积为广义的矩阵向量相乘，其结果为形状为 $m_1 \times \cdots \times m_{d_y}$ 的张量。这里广义的矩阵向量相乘和普通矩阵向量相乘一样，遵循同样代数法则：

$$\left(\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \Delta \mathbf{X} \right)_i = \sum_j \left(\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \right)_{i,j} (\Delta \mathbf{X})_i = \left(\frac{\partial \mathbf{Y}}{\partial \mathbf{X}_{j,:}} \right) \cdot \Delta \mathbf{X}$$

唯一不同的是这里的索引 i 和 j 不是标量，而是索引构成的向量。上述等式中的 $\left(\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \right)_{j,:}$ 为广义矩阵 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 的第 j "行"，其结果为形状和 \mathbf{X} 一致的张量。我们同时应用了法则：同样形状的两个张量间的点积为相应元素的乘积后求和，和向量的点积一致。链乘法则对于张量值函数来说看上去也类似，假设 $\mathbf{Y} = f(\mathbf{X})$ 及 $\mathbf{Z} = g(\mathbf{Y})$ ，其中 \mathbf{X} 和 \mathbf{Y} 和先前提交的形状一致， \mathbf{Z} 的形状为 $k_1 \times \cdots \times k_{d_z}$ 。现在链乘法则看上去和先前一致：

$$\frac{\partial \mathbf{Z}}{\partial \mathbf{X}} = \frac{\partial \mathbf{Z}}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$$

不同是现在广义矩阵 $\frac{\partial \mathbf{Z}}{\partial \mathbf{Y}}$ 其形状为 $(k_1 \times \cdots \times k_{d_z}) \times (m_1 \times \cdots \times m_{d_y})$ ，广义矩阵 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 的形状为 $(m_1 \times \cdots \times m_{d_y}) \times (n_1 \times \cdots \times n_{d_x})$ ，乘积 $\frac{\partial \mathbf{Z}}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 为广义矩阵矩阵乘积，产生的对象形状为 $(k_1 \times \cdots \times k_{d_z}) \times (n_1 \times \cdots \times n_{d_x})$ 。和先前定义的广义矩阵向量相乘类似，广义矩阵矩阵相乘和传统矩阵矩阵相乘遵循同样的代数法则：

$$\left(\frac{\partial \mathbf{Z}}{\partial \mathbf{X}} \right)_{i,j} = \sum_k \left(\frac{\partial \mathbf{Z}}{\partial \mathbf{Y}} \right)_{i,k} \left(\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \right)_{k,j} = \left(\frac{\partial \mathbf{Z}}{\partial \mathbf{Y}} \right)_{i,:} \cdot \left(\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \right)_{:,j} \quad (2.31)$$

上述方程中的索引 i, j, k 均是向量索引，且代数项 $\left(\frac{\partial \mathbf{Z}}{\partial \mathbf{Y}} \right)_{i,:}$ 及 $\left(\frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \right)_{:,j}$ 为 $\frac{\partial \mathbf{Z}}{\partial \mathbf{Y}}$ 的第 i 行，及 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 的第 j 列。

这里给出一个张量展开为向量进行求导的例子，对张量的求导链乘法则理解会有相应的帮助假设存在如下张量

$$\mathbf{X} \in \mathbb{R}^{2 \times 2 \times 2} \quad \mathbf{Y} \in \mathbb{R}^{3 \times 1 \times 2} \quad \mathbf{Z} \in \mathbb{R}^{2 \times 1 \times 2}$$

将其分别展开为向量形式，则有：

$$\mathbf{x} = \begin{bmatrix} X_{1,1,1} \\ X_{1,1,2} \\ X_{1,2,1} \\ X_{1,2,2} \\ X_{2,1,1} \\ X_{2,1,2} \\ X_{2,2,1} \\ X_{2,2,2} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} Y_{1,1,1} \\ Y_{1,1,2} \\ Y_{2,1,1} \\ Y_{2,1,2} \\ Y_{3,1,1} \\ Y_{3,1,2} \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} Z_{1,1,1} \\ Z_{1,1,2} \\ Z_{2,1,1} \\ Z_{2,1,2} \end{bmatrix}$$

因此有如下雅可比矩阵：

$$\frac{\partial \mathbf{z}}{\partial \mathbf{y}} = \begin{bmatrix} \frac{\partial Z_{1,1,1}}{\partial Y_{1,1,1}} & \frac{\partial Z_{1,1,1}}{\partial Y_{1,1,2}} & \cdots & \frac{\partial Z_{1,1,1}}{\partial Y_{3,1,2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Z_{2,1,2}}{\partial Y_{1,1,1}} & \frac{\partial Z_{2,1,2}}{\partial Y_{1,1,2}} & \cdots & \frac{\partial Z_{2,1,2}}{\partial Y_{3,1,2}} \end{bmatrix} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial Y_{1,1,1}}{\partial X_{1,1,1}} & \cdots & \frac{\partial Y_{1,1,1}}{\partial X_{2,2,2}} \\ \frac{\partial Y_{1,1,2}}{\partial X_{1,1,1}} & \cdots & \frac{\partial Y_{1,1,2}}{\partial X_{2,2,2}} \\ \vdots & \ddots & \vdots \\ \frac{\partial Y_{3,1,2}}{\partial X_{1,1,1}} & \cdots & \frac{\partial Y_{3,1,2}}{\partial X_{2,2,2}} \end{bmatrix}$$

从上述导数，可以较容易验证公式2.31成立。

第3章 概率论

本部分主要回顾概率论研究的样本空间，事件及概率公理和由公理可以推知的概率属性，及条件概率和独立性定义。在此基础上我们进一步回顾了随机变量的定义，及其对应的概率质量函数 (Probability Mass Function) 和概率密度函数及累积分布函数 (Cumulative Distribution Function) 定义，并进一步回顾了随机变量期望和方差的定义。在随机变量定义基础之上我们进一步回顾了两个随机变量及多个随机变量的联合分布和边缘分布及贝叶斯定理和期望及协方差定义。在上述概念基础之上，我们回顾了两类参数估计方法，最大似然估计和最大后验估计。最后我们回顾了两类结构化概率模型，分别是有向图概率模型和无向图概率模型。

3.1 概率基础

本小节回顾样本空间、概率公理及条件概率等定义

3.1.1 基础定义

概率论是研究不确定性的理论，譬如研究抛硬币这样的随机实验，出现正反面是不确定的。为了研究随机实验可能的结果及其对应的概率我们需要如下基本元素：

样本空间 Ω : 样本空间由随机实验所有可能的结果构成的集合。集合中的每个元素 $\omega \in \Omega$ 可以被看出某次随机实验后真实世界的结果状态。

事件空间 \mathcal{F} : 事件空间 (\mathcal{F}) 由事件 (A) 构成，即 $A \in \mathcal{F}$ 。事件由实验的可能结果构成，为样本空间的子集 $A \subseteq \Omega$ 。

概率公理: 概率公理为事件空间到实数的函数映射: $P : \mathcal{F} \rightarrow \mathbb{R}$, 其满足如下属性:

- 对于事件空间中的所有事件，均有 $P(A) \geq 0$
- 对于样本空间有 $P(\Omega) = 1$
- 若 A_1, A_2, \dots 彼此之间没有交集 (即当 $i \neq j$ 时, $A_i \cap A_j = \emptyset$)，则有:

$$P(\bigcup_i A_i) = \sum_i P(A_i) \quad (3.1)$$

这里举一个掷骰子(六面)的随机实验的例子。该随机实验的样本空间为 $\Omega = \{1, 2, 3, 4, 5, 6\}$ 。可以在该样本空间上定义不同的事件空间，譬如 $\mathcal{F} = \{\emptyset, \Omega\}$ ，另外一个可能的事件空间譬如是样本空间的所有子集构成的集合。对于前一个事件空间定义，唯一的满足上述概率公理定义的概率函数定义为 $P(\emptyset) = 0, P(\Omega) = 1$ 。对于后一个事件空间，一个合理的概率函数定义为，对事件空间中的每个事件，定义该事件发生的概率为 $\frac{i}{6}$ ，其中 i 为事件中的元素个数，譬如: $P(\{1, 2, 3, 4\}) = \frac{4}{6}$ 及 $P(\{1, 2, 3\}) = \frac{3}{6}$ 。

性质:

- 若 $A \subseteq B$ ，则有 $P(A) \leq P(B)$.
- $P(A \cap B) = \min(P(A), P(B))$.

- $P(A \cup B) \leq P(A) + P(B)$.
- $P(\Omega A) = 1 - P(A)$.
- (总概率法则) 若 A_1, \dots, A_k 一系列彼此之间没有交集的事件, 且 $\cup_{i=1}^k A_i = \Omega$, 则有:

$$\sum_{i=1}^k P(A_i) = 1 \quad (3.2)$$

3.1.2 条件概率

假设事件 B 为非零概率事件, 在给定事件 B 的发生的条件下, 任意事件 A 的条件概率定义如下:

$$P(A|B) \triangleq \frac{P(A \cap B)}{P(B)} \quad (3.3)$$

也就是说, $P(A|B)$ 衡量了观测到事件 B 发生的条件下, 事件 A 发生的概率。两个事件为独立事件当且仅当 $P(A \cap B) = P(A)P(B)$ (或者 $P(A|B) = P(A)$)。因此事件独立等价于说观察到事件 B 对事件 A 的概率没有任何影响。

3.2 随机变量

为了更有效研究随机实验对应的样本空间中元素的概率分布性质, 引入随机变量概念, 本节回顾随机变量及离散和连续随机变量相关的分布函数及随机变量的期望和方差, 最后我们将回顾常用随机变量。

3.2.1 基本定义

在给出随机变量的正式定义之前, 首先让我们来看抛 10 次硬币这样一个随机实验, 我们想考察结果为正面的硬币个数。该随机实验的样本空间 Ω 为长度为 10 的正面反面序列。譬如, $\omega_0 = < H, H, T, H, T, H, H, T, T, T > \in \Omega$ 。然而, 实际上我们通常不关心得到某个具体正反面序列的概率, 我们通常对样本空间中的每个实验结果映射为一个实数值的函数更感兴趣, 譬如抛 10 次硬币出现正面的次数, 或者第一次出现正面的位置。这些函数, 被称为随机变量。

定义: 随机变量 X 为样本空间到实数空间的函数映射, 即 $X : \Omega \rightarrow \mathbb{R}$ 。通常, 我们用大写字母 $X(\omega)$ 或者简写为 X (隐含依赖于随机结果 ω), 对于随机变量的取值, 我们用小写字母 x 表示。

例 1: 在我们上面的抛硬币实验中, 假设 $X(\omega)$ 为抛硬币产生的实验结果 ω 序列, 对应的出现正面次数。假设仅仅抛了十个硬币, $X(\omega)$ 可以输出有限可数的值, 这种情况下 X 为离散随机变量。此时随机变量 X 取某个具体值 k 对应的事件集合的概率定义如下:

$$P(X = k) = P(\{\omega : X(\omega) = k\}). \quad (3.4)$$

例 2: 假设随机变量 $X(\omega)$ 表示某个放射性颗粒的衰减时间。这种情况下, $X(\omega)$ 可以输出

无限可能值，此时 X 为连续随机变量。此时随机变量 X 取区间 $[a, b]$ （其中 $a < b$ ）的概率定义如下：

$$P(a \leq X \leq b) = P(\{w : a \leq X(w) \leq b\}). \quad (3.5)$$

3.2.2 累积分布函数

研究随机变量时，我们需要确定该随机变量服从的概率分布，通常可以使用 CDF, PMF 或 PDF 来描述随机实验的概率分布，本节给出累积分布函数 (Cumulative Distribution Function) 的定义。

累积分布函数 (CDF): 其表示为 $F_X : \mathbb{R} \rightarrow [0, 1]$ ，对应的概率度量定义如下：

$$F_X(x) \triangleq P(X \leq x).$$

通过该函数，我们可以计算事件空间 \mathcal{F} 中的任意事件的概率。

性质:

- $0 \leq F_X(x) \leq 1$.
- $\lim_{x \rightarrow -\infty} F_X(x) = 0$.
- $\lim_{x \rightarrow \infty} F_X(x) = 1$.
- $x \leq y \Rightarrow F_X(x) \leq F_X(y)$.

3.2.3 概率质量函数

当随机变量为离散随机变量时，指定该随机变量的概率度量的更简单的形式是，直接指定随机变量取各个值的概率，这类概率度量称为概率质量函数 (Probability Mass Function) : $p_X : \Omega \rightarrow \mathbb{R}$ ，其定义如下：

$$p_X(x) \triangleq P(X = x).$$

假设我们用 $Val(X)$ 表示离散性随机变量 X 的取值集合。例如，假设 $X(w)$ 表示抛 10 次硬币出现正面的次数，此时 $Val(X) = \{0, 1, 2, \dots, 10\}$ 。

性质:

- $0 \leq p_X(x) \leq 1$.
- $\sum_{x \in Val(x)} p_X(x) = 1$.
- $\sum_{x \in A} p_X(x) = P(X \in A)$.

3.2.4 概率密度函数

当随机变量为连续型随机变量时，若累积分布函数 (CDF) $F_X(x)$ 处处可导。我们定义概率密度函数 (Probability Density Function) 为 CDF 的导数，即：

$$f_X(x) \triangleq \frac{dF_X(x)}{dx}. \quad (3.6)$$

根据导数性质，对于非常小的 Δx ，有：

$$P(x \leq X \leq x + \Delta x) \approx f_X(x)\Delta x. \quad (3.7)$$

性质：

- $f_X(x) \geq 0.$
- $\int_{-\infty}^{\infty} f_X(x) = 1.$
- $\int_{x \in A} f_X(x)dx = P(X \in A).$

3.2.5 期望

假设 X 为离散随机变量，其服从概率质量函数 $p_X(x)$ ，函数 $g : \mathbb{R} \rightarrow \mathbb{R}$ 为任意函数。此时可以将 $g(X)$ 看成随机变量，我们定义期望 (Expectation) 或者 $g(X)$ 的期望值 (Expected value) 如下：

$$E[g(X)] \triangleq \sum_{x \in Val(X)} g(x)p_X(x). \quad (3.8)$$

若 X 为连续随机变量，服从概率密度函数 $f_X(x)$ ，此时 $g(X)$ 的期望值定义如下：

$$E[g(X)] \triangleq \int_{-\infty}^{\infty} g(x)f_X(x)dx. \quad (3.9)$$

直观上看， $g(X)$ 的期望可以看成 $g(x)$ 所有不同取值 x 的加权平均，权重由 $p_X(x)$ 或 $f_X(x)$ 指定。作为上面形式的特例，随机变量自身的期望 $E[X]$ 可以通过令 $g(x) = x$ 得到，该值也称为随机变量 X 的均值。

性质：

- 对于任意常量 $a \in \mathbb{R}$ ，有 $E[a] = a.$
- 对于任意常量 $a \in \mathbb{R}$ ，有 $E[af(X)] = aE[f(X)].$
- $E[f(X) + g(X)] = E[f(X)] + E[g(X)]$ (期望线性性) .
- 对于离散性随机变量 X ， $E[1\{X = k\}] = P(X = k).$

3.2.6 方差

随机变量的方差衡量了随机变量 X 的分布围绕其均值的聚集程度。其正式定义如下：

$$Var[X] \triangleq E[(X - E(X))^2] \quad (3.10)$$

通过使用上一小节的期望的性质，我们可推出方差的另一种表示形式：

$$\begin{aligned} E[(X - E[X])^2] &= E[X^2 - 2E[X]X + E[X]^2] \\ &= E[X^2] - 2E[X]E[X] + E[X]^2 \\ &= E[X^2] - E[X]^2 \end{aligned} \quad (3.11)$$

性质:

- 对于任意常量 $a \in \mathbb{R}$, 有 $\text{Var}[a] = 0$.
- 对于任意常量 $a \in \mathbb{R}$, 有 $\text{Var}[af(X)] = a^2\text{Var}[f(X)]$.

例 1: 计算服从如下均匀分布的随机变量 X 的均值和方差

$$f_X(x) = \begin{cases} 1 & \forall x \in [0, 1] \\ 0 & \text{elsewhere} \end{cases} \quad (3.12)$$

根据均值和方差的定义有:

$$E[X] = \int_{-\infty}^{\infty} xf_X(x)dx = \int_0^1 xdx = \frac{1}{2}.$$

$$E[X^2] = \int_{-\infty}^{\infty} x^2 f_X(x)dx = \int_0^1 x^2 dx = \frac{1}{3}.$$

$$\text{Var}[X] = E[X^2] - E[X]^2 = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}.$$

例 2: 假设对于 $A \subseteq \Omega$ 有 $g(x) = \{x \in A\}$, 求 $E[g(X)]$

若随机变量 X 为离散型:

$$E[g(X)] = \sum_{x \in \text{Val}(X)} 1\{x \in A\} P_X(x) = P(x \in A). \quad (3.13)$$

若随机变量 X 为连续型:

$$E[g(X)] = \int_{-\infty}^{\infty} 1\{x \in A\} f_X(x)dx = \int_{x \in A} f_X(x)dx = P(x \in A). \quad (3.14)$$

3.2.7 常用随机变量

离散随机变量

- $X \sim \text{Bernoulli}(p)$ ($0 \leq p \leq 1$): 一次抛硬币实验, 出现正面 ($x = 1$) 的概率为 p , 出现反面 ($x = 0$) 的概率为 $1 - p$.

$$p(x) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \end{cases} \quad (3.15)$$

- $X \sim \text{Binomial}(n, p)$ ($0 \leq p \leq 1$): 独立进行 n 次抛硬币实验, 出现 x 次正面的概率为:

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x} \quad (3.16)$$

- $X \sim Geometric(p)$ ($p > 0$): 抛硬币 x 次, 第 x 次为首次出现正面的概率为:

$$p(x) = p(1-p)^{x-1} \quad (3.17)$$

连续随机变量

- $X \sim Uniform(a, b)$ ($a < b$): 均匀分布表示实线上 a 和 b 之间的所有值等概率密度, 其概率密度函数如下:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \quad (3.18)$$

- $X \sim Exponential(\lambda)$ ($\lambda > 0$): 在非负实数上的概率密度逐渐衰减, 其对应于离散随机变量的几何分布:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.19)$$

- $X \sim Normal(\mu, \sigma^2)$: 正态分布, 也称为高斯分布:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \quad (3.20)$$

下表总结了不同分布的性质

Distribution	PDF or PMF	Mean	Variance
$Bernoulli(p)$	$\begin{cases} p & \text{if } x = 1 \\ 1-p & \text{if } x = 0 \end{cases}$	p	$p(1-p)$
$Binomial(n, p)$	$\binom{n}{k} p^k (1-p)^{n-k} \quad \text{for } 0 \leq k \leq n$	np	$np(1-p)$
$Geometric(p)$	$p(1-p)^{k-1} \quad \text{for } k = 1, 2, \dots$	$\frac{1}{p}$	$\frac{1-p}{p^2}$
$Uniform(a, b)$	$\frac{1}{b-a} \quad \forall x \in (a, b)$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
$Gaussian(\mu, \sigma^2)$	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	μ	σ^2
$Exponential(\lambda)$	$\lambda e^{-\lambda x} \quad x \geq 0, \lambda > 0$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

3.3 两个随机变量

目前为止, 我们主要回顾单随机变量的性质。然而, 很多情况下, 我们会对随机实验的多个随机变量感兴趣。譬如, 随机实验抛一个硬币 10 次, 我们可能会同时对出现正面的次数 $X(\omega)$ 及最大连续出现正面的长度 $Y(\omega)$ 这两个随机感兴趣。本小节, 我们将回顾两个随机变量的性质。

3.3.1 联合和边缘累积分布函数

假设我们有两个随机变量 X 及 Y , 研究这两个随机变量的一种方法是分别研究, 此时我们仅需要累积分布函数 $F_X(x)$ 及 $F_Y(y)$ 。若我们希望研究某次随机实验同时产生随机变量 X 和 Y 的性质, 我们需要随机变量 X 及 Y 的联合累积分布函数, 定义如下:

$$F_{XY}(x, y) = P(X \leq x, Y \leq y) \quad (3.21)$$

给定联合分布函数后, 即可计算出随机变量 X 及 Y 关联的任意事件的概率。联合分布及边缘分布的关系如下:

$$F_X(x) = \lim_{y \rightarrow \infty} F_{XY}(x, y) \quad (3.22)$$

$$F_Y(y) = \lim_{x \rightarrow \infty} F_{XY}(x, y) \quad (3.23)$$

上述公式中 $F_X(x)$ 及 $F_Y(y)$ 为联合分布 $F_{XY}(x, y)$ 的边缘累积分布函数。

性质:

- $0 \leq F_{XY}(x, y) \leq 1$.
- $\lim_{x, y \rightarrow \infty} F_{XY}(x, y) = 1$.
- $\lim_{x, y \rightarrow -\infty} F_{XY}(x, y) = 0$.
- $F_X(x) = \lim_{y \rightarrow \infty} F_{XY}(x, y)$.

3.3.2 联合和边缘概率质量函数

若 X 和 Y 为离散型随机变量, 则联合概率质量函数 $p_{XY} : \mathbb{R} \times \mathbb{R} \rightarrow [0, 1]$ 定义如下:

$$p_{XY}(x, y) = P(X = x, Y = y). \quad (3.24)$$

上述公式中, $0 \leq p_{XY}(x, y) \leq 1 \quad \forall x, y$, 且 $\sum_{x \in Val(X)} \sum_{y \in Val(Y)} p_{XY}(x, y) = 1$ 。给定联合概率质量函数后, 边缘概率质量函数定义如下:

$$p_X(x) = \sum_y p_{XY}(x, y). \quad (3.25)$$

$p_Y(y)$ 定义类似。

3.3.3 联合和边缘概率密度函数

若 X 和 Y 为连续型随机变量, 其联合累积分布函数为 F_{XY} , 且 F_{XY} 处处可导, 则联合概率密度函数为:

$$f_{XY}(x, y) = \frac{\partial^2 F_{XY}(x, y)}{\partial x \partial y}. \quad (3.26)$$

和离散随机变量类似，边缘概率密度函数定义如下：

$$f_X(x) = \int_{-\infty}^{\infty} f_{XY}(x, y) dy \quad (3.27)$$

$f_Y(y)$ 定义类似。

3.3.4 条件分布

条件分布研究在给定随机变量 X 取值 x 时，随机变量 Y 的概率分布。对于离散型随机变量，随机变量 X 取某个具体值 x 时，随机变量 Y 的概率质量函数定义如下：

$$p_{Y|X}(y|x) = \frac{p_{XY}(x, y)}{p_X(x)} \quad (3.28)$$

其中 $p_X(x) \neq 0$ 。

对于连续型随机变量，和离散型随机变量类似，条件概率密度函数定义如下：

$$f_{Y|X}(y|x) = \frac{f_{XY}(x, y)}{f_X(x)} \quad (3.29)$$

其中 $f_X(x) \neq 0$ 。

3.3.5 贝叶斯法则

贝叶斯法则用于推导给定某个随机变量的条件下当前随机变量的条件概率

贝叶斯法则的四个版本：

- 离散型随机变量 X ，离散型随机变量 Y

$$p_{X|Y}(x|y) = \frac{p_X(x)p_{Y|X}(y|x)}{p_Y(y)} \quad (3.30)$$

- 离散型随机变量 X ，连续性随机变量 Y

$$p_{X|Y}(x|y) = \frac{p_X(x)f_{Y|X}(y|x)}{f_Y(y)} \quad (3.31)$$

- 连续型随机变量 X ，离散型随机变量 Y

$$f_{X|Y}(x|y) = \frac{f_X(x)p_{Y|X}(y|x)}{p_Y(y)} \quad (3.32)$$

- 连续型随机变量 X ，连续型随机变量 Y

$$f_{X|Y}(x|y) = \frac{f_X(x)f_{Y|X}(y|x)}{f_Y(y)} \quad (3.33)$$

3.3.6 独立性

两个随机变量 X 和 Y , 若对所有 x, y 均有 $F_{XY}(x, y) = F_X(x)F_Y(y)$, 则 X 和 Y 相互独立, 如下相互独立的等价声明:

- 若 X 和 Y 为相互独立的离散型随机变量, 则有 $p_{XY}(x, y) = p_X(x)p_Y(y) \quad \forall x \in Val(X) \quad \forall y \in Val(Y)$.
- 若 X 和 Y 为相互独立的离散型随机变量, 则当 $p_X(x) \neq 0$ 时, 有 $p_{Y|X}(y|x) = p_Y(y) \quad \forall y \in Val(Y)$.
- 若 X 和 Y 为相互独立的连续型随机变量, 则有 $f_{XY}(x, y) = f_X(x)f_Y(y) \quad \forall x, y \in \mathbb{R}$.
- 若 X 和 Y 为相互独立的连续型随机变量, 则当 $f_X(x) \neq 0$ 时, 有 $f_{Y|X}(y|x) = f_Y(y) \quad \forall y \in \mathbb{R}$.

直观来看, 两个随机变量 X 和 Y 相互独立, 若给定某个随机变量的取值后, 对另外一个随机变量取值的概率分布没有任何影响。

3.3.7 期望及协方差

给定离散型随机变量 X, Y 及关于随机变量的函数 $g : \mathbb{R}^2 \rightarrow \mathbb{R}$, 则函数 g 的期望值定义如下:

$$E[g(X, Y)] \triangleq \sum_{x \in Val(X)} \sum_{y \in Val(Y)} g(x, y)p_{XY}(x, y). \quad (3.34)$$

若随机变量 X 及 Y 为连续型随机变量, 则有:

$$E[g(X, Y)] \triangleq \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y)f_{XY}(x, y)dx dy. \quad (3.35)$$

我们可以借助于期望的定义来研究两个随机变量彼此间的关系。两个随机变量 X 及 Y 的协方差定义如下:

$$Cov[X, Y] \triangleq E[(X - E[X])(Y - E[Y])] \quad (3.36)$$

对上述公式进行进一步化简:

$$\begin{aligned} Cov[X, Y] &= E[(X - E[X])(Y - E[Y])] \\ &= E[XY - XE[Y] - YE[X] + E[X]E[Y]] \\ &= E[XY] - E[X]E[Y] - E[Y]E[X] + E[X]E[Y] \\ &= E[XY] - E[X]E[Y] \end{aligned} \quad (3.37)$$

当 $Cov[X, Y] = 0$ 时, 我们称 X 和 Y 相互独立。

性质:

- (期望线性性) $E[f(X, Y) + g(X, Y)] = E[f(X, Y)] + E[g(X, Y)]$.

- $\text{Var}[X \pm Y] = \text{Var}[X] + \text{Var}[Y] \pm 2\text{Cov}[X, Y]$.

$$\begin{aligned}
\text{Var}[X \pm Y] &= E[((X \pm Y) - E[X \pm Y])^2] \\
&= E[((X - E[X]) \pm (Y - E[Y]))^2] \\
&= E[(X - E[X])^2 \pm 2(X - E[X])(Y - E[Y]) + (Y - E[Y])^2] \\
&= \text{Var}[X] + \text{Var}[Y] \pm 2\text{Cov}[X, Y]
\end{aligned} \tag{3.38}$$

- 若 X 和 Y 彼此独立, 则有 $\text{Cov}[X, Y] = 0$.
- 若 X 和 Y 彼此独立, 则有 $E[f(X)g(Y)] = E[f(X)]E[g(Y)]$.
- 若 X 和 Y 彼此独立, 则有 $\text{Var}[XY] = E[X]^2\text{Var}[Y] + E[Y]^2\text{Var}[X] + \text{Var}[X]\text{Var}[Y]$

$$\begin{aligned}
\text{Var}[XY] &= E[(XY)^2] - E[XY]^2 \\
&= E[X^2]E[Y^2] - E[X]^2E[Y]^2 \\
&= (\text{Var}[X] + E[X]^2)(\text{Var}[Y] + E[Y]^2) - E[X]^2E[Y]^2 \\
&= E[X]^2\text{Var}[Y] + E[Y]^2\text{Var}[X] + \text{Var}[X]\text{Var}[Y]
\end{aligned} \tag{3.39}$$

3.4 多个随机变量

上一小节介绍的符号和思想可以推广至多个随机变量 (大于两个)。假设我们有 n 连续随机变量: $X_1(\omega), X_2(\omega), \dots, X_n(\omega)$, 为简单起见, 本节仅考虑连续性随机变量, 对离散型随机变量存在类似的一般化推广。

3.4.1 基本属性

我们可以定义随机变量 X_1, X_2, \dots, X_n 的联合概率分布函数及概率密度函数, 及 X_1 的边缘概率密度函数, 及在给定 X_2, \dots, X_n 的条件下, X_1 的条件概率密度函数如下:

$$F_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n) = P(X_1 \leq x_1, X_2 \leq x_2, \dots, X_n \leq x_n) \tag{3.40}$$

$$f_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n) = \frac{\partial^n F_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n)}{\partial x_1 \dots \partial x_n} \tag{3.41}$$

$$f_{X_1}(x_1) = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} f_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n) dx_2 \dots dx_n \tag{3.42}$$

$$f_{X_1|X_2, \dots, X_n}(x_1, x_2, \dots, x_n) = \frac{f_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n)}{f_{X_2, \dots, X_n}(x_1, x_2, \dots, x_n)} \tag{3.43}$$

如下为某个事件 $A \subseteq \mathbb{R}^n$ 发生的概率:

$$P((x_1, x_2, \dots, x_n) \in A) = \int_{(x_1, x_2, \dots, x_n) \in A} f_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n) dx_1 dx_2 \dots dx_n \tag{3.44}$$

链乘法则：由上述多元随机变量条件概率密度函数定义可知：

$$f(x_1, x_2, \dots, x_n) = f(x_n | x_1, x_2, \dots, x_{n-1}) f(x_1, x_2, \dots, x_{n-1}) \quad (3.45)$$

$$= f(x_n | x_1, x_2, \dots, x_{n-1}) f(x_{n-1} | x_1, x_2, \dots, x_{n-2}) f(x_1, x_2, \dots, x_{n-2}) \quad (3.46)$$

$$= \dots = f(x_1) \prod_{i=2}^n f(x_i | x_1, \dots, x_{i-1}). \quad (3.47)$$

独立性：对于多个事件， A_1, \dots, A_k ，我们称 A_1, \dots, A_k 为互相独立，若对任意子集 $\mathbb{S} \subseteq 1, 2, \dots, k$ ，如下等式成立：

$$P(\cap_{i \in \mathbb{S}} A_i) = \prod_{i \in \mathbb{S}} P(A_i). \quad (3.48)$$

类似地，我们称随机变量 X_1, \dots, X_n 为相互独立，若如下等式成立：

$$f(x_1, \dots, x_n) = f(x_1) f(x_2) \dots f(x_n). \quad (3.49)$$

独立随机变量常常出现于机器学习算法中，我们一般假设训练集中的训练样本为属于某个未知概率分布的独立样本。下面通过一个例子来体现独立性的重要性，考虑构建这样的一个差训练集：我们首先从未知概率分布中抽样出一个样本 $(x^{(1)}, y^{(1)})$ 加入训练集，然后将该样本重复 $m - 1$ 次添加到训练集中，在这种情况下，我们有：

$$P((x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})) \neq \prod_{i=1}^m P(x^{(i)}, y^{(i)}). \quad (3.50)$$

即使训练集的大小为 m ，但是样本并不独立。虽然上述构建训练集的方法不正确，实际上，非独立样本常常出现，因此这类样本将降低训练集的有效大小。

3.4.2 随机向量

假设我们有 n 个随机变量，当同时处理这些随机变量时，将这些随机变量表示为向量常常更加方便： $X = [X_1, X_2, \dots, X_n]^\top$ 。我们称 X 为随机向量，更正式的定义是：随机向量为 Ω 到 \mathbb{R}^n 的函数映射。显而易见，随机向量为处理 n 个随机变量的替表示符号，因此概率密度函数和概率分布函数同样可以应用于随机向量。**期望：**考虑函数 $g : \mathbb{R}^n \rightarrow \mathbb{R}$ 。该函数的期望定义如下：

$$E[g(X)] = \int_{\mathbb{R}^n} g(x_1, x_2, \dots, x_n) f_{X_1, X_2, \dots, X_n}(x_1, x_2, \dots, x_n) dx_1 dx_2 \dots dx_n. \quad (3.51)$$

其中 $\int_{\mathbb{R}^n}$ 为 n 次 $-\infty$ 到 ∞ 连续积分。若 g 为 \mathbb{R}^n 到 \mathbb{R}^m 的函数映射，则函数 g 的期望值为输出向量的各元素的期望值：

$$g(x) = \begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_m(x) \end{bmatrix} \quad (3.52)$$

此时 g 的期望定义如下：

$$E[g(x)] = \begin{bmatrix} E[g_1(x)] \\ E[g_2(x)] \\ \vdots \\ E[g_m(x)] \end{bmatrix} \quad (3.53)$$

协方差矩阵：对于给定的随机向量 $X : \Omega \rightarrow \mathbb{R}^n$ ，其协方差矩阵 Σ 为 $n \times n$ 的方阵， $\Sigma_{i,j} = Cov[X_i, X_j]$ 。根据协方差定义有：

$$\Sigma = \begin{bmatrix} Cov[X_1, X_1] & \dots & Cov[X_1, X_n] \\ \vdots & \ddots & \vdots \\ Cov[X_n, X_1] & \dots & Cov[X_n, X_n] \end{bmatrix} \quad (3.54)$$

$$= \begin{bmatrix} E[X_1^2] - E[X_1]E[X_1] & \dots & E[X_1X_n] - E[X_1]E[X_n] \\ \vdots & \ddots & \vdots \\ E[X_nX_1] - E[X_n]E[X_1] & \dots & E[X_n^2] - E[X_n]E[X_n] \end{bmatrix} \quad (3.55)$$

$$= \begin{bmatrix} E[X_1^2] & \dots & E[X_1X_n] \\ \vdots & \ddots & \vdots \\ E[X_nX_1] & \dots & E[X_n^2] \end{bmatrix} \quad (3.56)$$

$$- \begin{bmatrix} E[X_1]E[X_1] & \dots & E[X_1]E[X_n] \\ \vdots & \ddots & \vdots \\ E[X_n]E[X_1] & \dots & E[X_n]E[X_n] \end{bmatrix} \quad (3.56)$$

$$= E[XX^\top] - E[X]E[X]^\top = \dots = E[(X - E[X])(X - E[X])^\top]. \quad (3.57)$$

协方差矩阵具有如下性质：

- Σ 为半正定矩阵
- $\Sigma = \Sigma^\top$ ，即 Σ 为对角阵

3.5 参数估计

本节回顾参数估计的两类主要方法，最大似然估计和最大后验估计

3.5.1 最大似然估计

最大似然估计 (Maximum Likelihood Estimation) 目标是求解的参数 θ 最大化观测到事件 (随机变量) 的概率, 形式如下:

$$\hat{\theta}_{MLE} = \arg \max_{\theta} p(X|\theta) \quad (3.58)$$

假设事件 $X = (x_1, x_2, \dots, x_n)$, 且 x_1, x_2, \dots, x_n 为独立同分布 (i.i.d) 的一组元素。因此有:

$$\begin{aligned} \hat{\theta}_{MLE} &= \arg \max_{\theta} p(X|\theta) = \arg \max_{\theta} \prod_{i=1}^n p(x_i|\theta) \\ &= \arg \max_{\theta} \log \prod_{i=1}^n p(x_i|\theta) = \arg \max_{\theta} \sum_{i=1}^n \log p(x_i|\theta) \\ &= \arg \min_{\theta} - \sum_{i=1}^n \log p(x_i|\theta) \end{aligned} \quad (3.59)$$

3.5.2 最大后验估计

最大似然估计没有考虑参数的先验分布支持, 假设给定参数的先验分布后, 我们期望在给定观察到事件和先验概率的情况下, 概率最大的参数, 形式如下:

$$\hat{\theta} = \arg \max_{\theta} p(\theta|X) \quad (3.60)$$

由贝叶斯法则可知:

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} \propto p(X|\theta)p(\theta) \quad (3.61)$$

可以看到上述公式中的左边为后验概率, 结合上述公式, 可知该参数估计方法为最大后验估计 (Maximum a Posteriori Estimation), 形式如下:

$$\begin{aligned} \hat{\theta}_{MAP} &= \arg \max_{\theta} p(\theta|X) = \arg \max_{\theta} p(X|\theta)p(\theta) \\ &= \arg \min_{\theta} - \sum_{i=1}^n \log p(x_i|\theta) - \log p(\theta) \end{aligned} \quad (3.62)$$

因此若 $\Theta \sim Normal(0, 1)$, 即:

$$f(\theta) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\theta^2}{2}}$$

此时最大后验估计的形式如下:

$$\hat{\theta}_{MAP} = \arg \min_{\theta} - \sum_{i=1}^n \log p(x_i|\theta) + \frac{1}{2}\theta^2 \quad (3.63)$$

可以看到该形式为机器学习中常用的 L_2 正则化形式。

若 $\Theta \sim Laplace(0, 1)$, 即:

$$f(\theta) = \frac{1}{2}e^{-|x|}$$

此时最大后验估计的形式如下:

$$\hat{\theta}_{MAP} = \arg \min_{\theta} - \sum_{i=1}^n \log p(x_i | \theta) + |\theta| \quad (3.64)$$

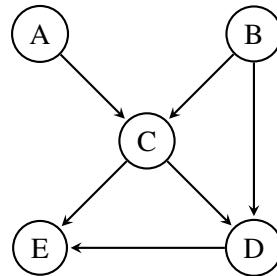
可以看到该形式为机器学习中常用的 L_1 正则化形式。

3.6 结构化概率模型

本小节回顾两类结构化概率模型：有向图模型和无向图模型。前一类模型对应贝叶斯网络，后一类模型对应马尔科夫随机场。

3.6.1 有向图模型

贝叶斯网络 (Bayesian Network) 通过有向无环图 (Directed Acyclic Graph) 表示，有向无环图直观表示了变量间的依赖关系和条件独立性。譬如:



$$p(A, B, C, D, E) = p(A)p(B)p(C|A, B)p(D|B, C)p(E|C, D)$$

一般化形式为:

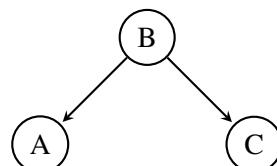
$$p(x_1, \dots, x_n) = \prod_i p(x_i | x_{parents(i)}) \quad (3.65)$$

通过有向无环图可以将变量的联合概率分布表示为每个变量的概率分布的乘积，其中每个变量的概率分布仅依赖一部分变量，即依赖于有向图表示的变量对应节点的父节点。通过这种表示方案，意味着在联合概率分布模型中引入某些变量之间的独立性假设。

有向无环图蕴含的条件独立性假设

1) 公共父节点

若图的形式为



观察到变量 B 时，则 $A \perp C | B$ 。若未观察到变量 B ，则 A 和 C 不独立。直观解释是 B 包含了所有决定 A 和 C 的信息，因此若观察到 B ，则没有其他信息影响变量 A 和 C 。

2) 层次依赖

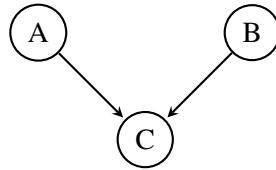
若图的形式为



观察到变量 B 时，则 $A \perp C | B$ 。若未观察到变量 B ，则 A 和 C 不独立。直观解释是 B 包含所有决定 C 的信息，因此若观察到 B ，则没有其他信息影响变量 C 。

3) V 型结构

若图的形式为



未观察到变量 C 时，则 $A \perp B$ 。若观察到变量 C 则 A 和 B 不独立。假设 C 为布尔类型，表示早上草坪是否是湿的， A 表示是否下雨， B 表示洒水装置是否打开。若草坪 C 是湿的，但洒水装置 B 没有打开，则一定下雨了 A 。

上面三种结构清晰描述了三变量贝叶斯网络可以表示的条件独立性，可以通过递归应用上述三种结构，表示任意有向图。

3.6.2 无向图模型

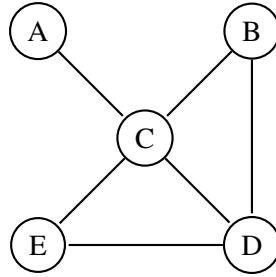
马尔科夫随机场 (Markov Random Fields) 通过无向图 (Undirected Graph) 表示。无向图基于图的最大团 (Maximum Clique) 定义变量的联合分布。这里解释一下团 (Clique) 的概念，团表示无向图中的一个子图，其中任意两个顶点直接存在一条边。最大团表示当前团不能被更大的团所有包含，也就是说，不存在一个顶点与当前团中任意顶点之间存在一条边。无向图的联合分布定义如下：

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in \mathbb{C}} \phi_c(\vec{x}_c) \quad (3.66)$$

其中 \mathbb{C} 表示特定随机变量值联合时，对应无向图的所有最大子团， $\phi_c(x_c)$ 表示相应子团的计算因子， Z 为归一化因子，用于生成所有随机变量值的联合概率：

$$Z = \sum_{x_1, \dots, x_n} \prod_{c \in \mathbb{C}} \phi_c(\vec{x}_c) \quad (3.67)$$

如下是一个无向图：



其对应的联合概率分布定义为：

$$p(A, B, C, D, E) = \frac{1}{Z} \phi_c(A, C) \phi_c(B, C, D) \phi_c(C, D, E)$$

结合参数估计方法，考虑计算简洁性，函数 $\phi = f \circ g$ ，其中 $f : \mathbb{R} \rightarrow \mathbb{R}$ 一般取指数函数，即：

$$f(x) = e^x \quad (3.68)$$

子团内所有顶点影响权重，可以表示为线性加权函数的形式，即：

$$g(\{x_i : x_i \in \mathbb{C}\}) = \sum_{x_i \in \mathbb{C}} \theta_i x_i \quad (3.69)$$

因此子团函数为：

$$\phi(\{x_i : x_i \in \mathbb{C}\}) = e^{\sum_{x_i \in \mathbb{C}} \theta_i x_i} \quad (3.70)$$

其中 θ 为待估计参数。

3.7 主成分分析

主成分分析（Principle Component Analysis）可以用于分析数据中的主要信息，并利用主要信息对数据进行降维处理，这样可以起到数据压缩的效果。

步骤 1：准备数据

假设我们收集了 n 个学生的信息，每个学生收集了 m 个维度的信息，譬如：身高、体重、平均成绩等。因此所有数据可以表示为如下矩阵形式：

$$\mathbf{X} = \begin{bmatrix} x_{11} & \dots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nm} \end{bmatrix}$$

步骤 2：减去均值

基于 n 个学生信息，计算 m 维上各维度的均值，得到均值向量 $\bar{\mathbf{x}}$ ，对每个学生对应的向

量执行减去均值操作，得到如下矩阵：

$$\mathbf{B} = \begin{bmatrix} (x_{11} - \bar{x}_1) & \dots & (x_{1m} - \bar{x}_m) \\ \vdots & \ddots & \vdots \\ (x_{n1} - \bar{x}_1) & \dots & (x_{nm} - \bar{x}_m) \end{bmatrix}$$

步骤 3：计算协方差矩阵

$$\mathbf{C} = \begin{bmatrix} cov(\mathbf{b}_1, \mathbf{b}_1) & \dots & cov(\mathbf{b}_1, \mathbf{b}_m) \\ \vdots & \ddots & \vdots \\ cov(\mathbf{b}_m, \mathbf{b}_1) & \dots & cov(\mathbf{b}_m, \mathbf{b}_m) \end{bmatrix} = \frac{1}{m} \mathbf{B}^\top \mathbf{B}$$

步骤 4：计算协方差矩阵的特征值和特征向量

有协方差矩阵 \mathbf{C} 计算过程可见其为 $m \times m$ 的对称矩阵，结合线性代数一章的知识，可知协方差矩阵可以通过特征分解 (Eigendecomposition) 表示为如下形式：

$$\mathbf{C} = \mathbf{V} \Lambda \mathbf{V}^\top$$

其中 $\mathbf{V} \in \mathbb{R}^{m \times m}$ ，为所有特征向量构成的矩阵， $\Lambda \in \mathbb{R}^{m \times m}$ 为特征值构成的对角阵。由协方差矩阵定义可知，若 $\mathbf{x}_i, \mathbf{x}_j$ 两两相互独立，则有：

$$cov(\mathbf{b}_i, \mathbf{b}_j) = 0, \quad \text{if } i \neq j$$

此时协方差矩阵即为如下形式：

$$\mathbf{C} = \begin{bmatrix} var(\mathbf{x}_1) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & var(\mathbf{x}_m) \end{bmatrix} = \frac{1}{m} \mathbf{B}^\top \mathbf{B}$$

此时特征向量即为 $\mathbb{R}^{m \times m}$ 的基向量，特征值即为相应 \mathbf{x}_i 的方差。

步骤 5：

选取特征值最大的 k 个特征向量，并构建特征向量 (\mathbf{e}) 矩阵

$$\mathbf{F} = \begin{bmatrix} | & \dots & | \\ \mathbf{e}_1 & \dots & \mathbf{e}_k \\ | & \dots & | \end{bmatrix}$$

这里选取的 k 个特征向量为我们希望在数据中保留的主成分。

步骤 6：生成压缩后的数据集

该步骤为主成分分析的最后步骤，也是最容易的步骤。一旦我们构建好主成分矩阵 $\mathbf{F} \in$

$\mathbb{R}^{m \times k}$ 后，并右乘原来的数据矩阵 $\mathbf{X} \in \mathbb{R}^{n \times m}$ ，则得到压缩后的数据 \mathbf{X}^* ，计算如下：

$$\mathbf{X}^* = \mathbf{X}\mathbf{F}$$

第4章 数值计算

本章首先回顾凸集及凸函数的定义及性质，在此基础上回顾无约束优化及约束优化问题求解策略，最后回顾目前应用较多的一阶近似优化的不同算法。

4.1 凸集

本节首先回顾凸集的定义，在此基础上给出一些凸集例子

4.1.1 凸集定义

定义：集合 \mathbb{C} 为凸集，则对于 $\forall x, y \in \mathbb{C}$ 及 $\theta \in \mathbb{R}$ 其中 $0 \leq \theta \leq 1$ ，有：

$$\theta x + (1 - \theta)y \in \mathbb{C}. \quad (4.1)$$

直观来看，上述公式意味着若我们取集合 \mathbb{C} 中的两个元素，并在两个元素之间画一条线段，则该线段上的所有元素也属于 \mathbb{C} 。下图分布显示了凸集和非凸集：点 $\theta x + (1 - \theta)y$ 称

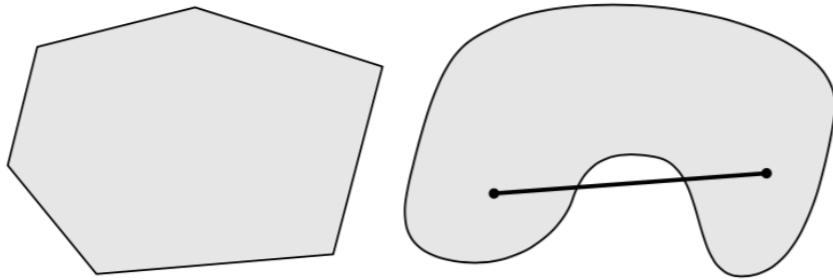


图 4.1: 左图: 凸集 右图: 非凸集

为点 x 和 y 的凸组合。

4.1.2 例子

下面给出一些凸集的例子

- 所有 \mathbb{R}^n : 若 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ ，则 $\theta\mathbf{x} + (1 - \theta)\mathbf{y} \in \mathbb{R}^n$ 。
- 非负象限 \mathbb{R}_+^n : 非负象限包含所有元素非负的 \mathbb{R}^n 维向量: $\mathbb{R}_+^n = \{\mathbf{x} : x_i \geq 0 \quad \forall i = 1, \dots, n\}$ 。为了说明该集合为凸集，仅需注意对于任意 $\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^n$ ，及 $0 \leq \theta \leq 1$ ，有:

$$(\theta\mathbf{x} + (1 - \theta)\mathbf{y})_i = \theta x_i + (1 - \theta)y_i \geq 0 \quad \forall i. \quad (4.2)$$

- 范数球 (Norm balls): 令 $\|\cdot\|$ 为 \mathbb{R}^n 上的范数(譬如: 欧几里得范数, $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$)。集合 $\{\mathbf{x} : \|\mathbf{x}\| \leq 1\}$ 为凸集。假设 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ ，其中 $\|\mathbf{x}\| \leq 1, \|\mathbf{y}\| \leq 1$ 及 $0 \leq \theta \leq 1$ ，则有:

$$\|\theta\mathbf{x} + (1 - \theta)\mathbf{y}\| \leq \theta\|\mathbf{x}\| + (1 - \theta)\|\mathbf{y}\| \leq 1 \quad (4.3)$$

上述公式中应用了三角不等式

- **仿射子空间及多面体:** 给定矩阵 $A \in \mathbb{R}^{m \times n}$ 及向量 $b \in \mathbb{R}^m$, 仿射子空间为集合 $\{x \in \mathbb{R}^n : Ax = b\}$ (注意: 该集合可能为空, 若 b 不属于 A 的列空间)。类似地, 多面体为集合 $\{x \in \mathbb{R}^n : Ax \leq b\}$, 其中 \leq 表示 Ax 的所有元素均小于等于 b 中的对应元素。为了证明该集合为凸集, 首先考虑 $x, y \in \mathbb{R}^n$, 满足 $Ax = Ay = b$, 对于 $0 \leq \theta \leq 1$, 有:

$$A(\theta x + (1 - \theta)y) = \theta Ax + (1 - \theta)Ay = \theta b + (1 - \theta)b = b. \quad (4.4)$$

类似地对于, 对于 $x, y \in \mathbb{R}^n$, 其满足 $Ax \leq b$ 及 $Ay \leq b$ 及 $0 \leq \theta \leq 1$, 有:

$$A(\theta x + (1 - \theta)y) = \theta Ax + (1 - \theta)Ay \leq \theta b + (1 - \theta)b = b. \quad (4.5)$$

- **凸集的交集:** 假设 C_1, C_2, \dots, C_k 为凸集, 则其交集:

$$\cap_{i=1}^k C_i = x : x \in C_i \quad \forall i = 1, \dots, k \quad (4.6)$$

也是凸集。为了证明其为凸集, 考虑 $x, y \in \cap_{i=1}^k C_i$ 及 $0 \leq \theta \leq 1$, 此时有:

$$\theta x + (1 - \theta)y \in C_i \quad \forall i = 1, \dots, k \quad (4.7)$$

根据凸集的定义, 因此有:

$$\theta x + (1 - \theta)y \in \cap_{i=1}^k C_i. \quad (4.8)$$

注意: 凸集的并集通常不是凸集

- **半正定矩阵:** 所有的半正定矩阵的集合, 通常称为半正定圆锥体, 表示为 S_+^n 为凸集 (通常, $S \subset \mathbb{R}^{n \times n}$ 表示 $n \times n$ 的对称矩阵)。矩阵 $A \in \mathbb{R}^{n \times n}$ 为对称半正定矩阵, 当且仅当 $A = A^\top$ 且对任意 $x \in \mathbb{R}^n$, 有 $x^\top Ax \geq 0$ 。现在考虑两个对称半正定矩阵 $A, B \in S_+^n$ 及 $0 \leq \theta \leq 1$, 对于任意 $x \in \mathbb{R}^n$, 有:

$$x^\top (\theta A + (1 - \theta)B)x = \theta x^\top Ax + (1 - \theta)x^\top Bx \geq 0. \quad (4.9)$$

该过程同样可以应用于所有正定矩阵, 及负定矩阵和半负定矩阵集合, 这些集合同样为凸集。

4.2 凸函数

本部分讨论凸函数定义, 并给出凸性的条件, 在此基础上给出一些凸函数例子

4.2.1 凸函数定义

凸函数为凸优化的中心问题，本节给出凸函数的定义。**定义：**函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 为凸函数，若其定义域（表示为 $\mathcal{D}(f)$ ）为凸集，且对于所有 $x, y \in \mathcal{D}(f)$ 及 $\theta \in \mathbb{R}$, $0 \leq \theta \leq 1$, 有：

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y). \quad (4.10)$$

直观来看，若我们在凸函数的图形上任意选择两点，并在两点之间画一条线段，则函数上两点之间的部分将处于线段之下。如下图所示：我们称函数为严格凸函数，若对于任

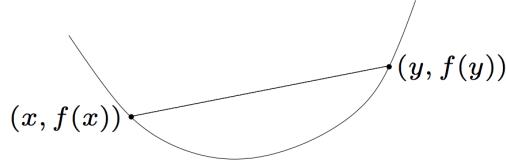


图 4.2: 凸函数：根据定义连接两点的线段处于函数之上

意 $x \neq y$ 及 $0 < \theta < 1$ ，凸函数定义中不等式严格成立。若 $-f$ 为凸函数，则 f 为凹函数，类似地，若 $-f$ 为严格凸函数，则 f 为严格凹函数。

4.2.2 凸性的一阶条件

若函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 可导（即：函数 f 定义域上的所有点 \mathbf{x} 均存在梯度 $\nabla_{\mathbf{x}} f(\mathbf{x})$ ），函数为凸函数当且仅当 $\mathcal{D}(f)$ 为凸集，且对 $\forall \mathbf{x}, \mathbf{y} \in \mathcal{D}(f)$ ，有：

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla_{\mathbf{x}} f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}). \quad (4.11)$$

函数 $f(\mathbf{x}) + \nabla_{\mathbf{x}} f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x})$ 为函数 f 在点 \mathbf{x} 附近的一阶近似。直观来，上述近似可以看成用函数在点 \mathbf{x} 处的切线近似函数 f 。上述凸性的一阶条件声明函数为凸函数，当且仅当其切线为函数的全局下界估计。也就是说，若我们在函数的任意点作切线，该线上的所有点均位于函数上相应点下方。和凸性的定义类似，若 4.11 为严格不等式，则函数为严格凸函数，下图为凸函数的一阶条件示意图：

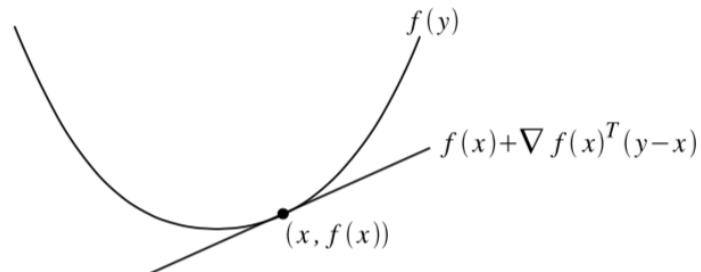


图 4.3: 凸函数：一阶条件

4.2.3 凸性的二阶条件

若函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 为二阶可导（即：函数 f 定义域上的所有点 \mathbf{x} 均存在 Hessian 矩阵 $\nabla_{\mathbf{x}}^2 f(\mathbf{x})$ ）。则函数 f 为凸函数，当且仅当 $\mathcal{D}(f)$ 为凸集且 Hessian 矩阵为半正定矩阵，即 $\forall \mathbf{x} \in \mathcal{D}(f)$ ，有：

$$\nabla_{\mathbf{x}}^2 f(\mathbf{x}) \succeq 0. \quad (4.12)$$

上述公式中符号 \succeq 表示矩阵为半正定矩阵。当 \mathbf{x} 为一维时，上述不等式意味着二阶导数 $f''(x)$ 总是非负。类似地，若 Hessian 矩阵为正定矩阵，则 f 为严格凸函数。

4.2.4 Jensen 不等式

假设给定如下凸函数不等式：

$$f(\theta \mathbf{x} + (1 - \theta) \mathbf{y}) \leq \theta f(\mathbf{x}) + (1 - \theta) f(\mathbf{y}) \quad \text{for } 0 \leq \theta \leq 1. \quad (4.13)$$

通过归纳法，上述不等式可以较容易地推广至多个点组合的情况：

$$f\left(\sum_{i=1}^k \theta_i \mathbf{x}_i\right) \leq \sum_{i=1}^k \theta_i f(\mathbf{x}_i) \quad \text{for } \sum_{i=1}^k \theta_i = 1, \theta_i \geq 0 \forall i. \quad (4.14)$$

实际上，上述不等式可以被推广至无限和及积分情形。在后一种情况下，不等式可以被写为：

$$f\left(\int p(\mathbf{x}) \mathbf{x} d\mathbf{x}\right) \leq \int p(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \quad \text{for } \int p(\mathbf{x}) d\mathbf{x} = 1, \quad p(\mathbf{x}) \geq 0 \quad \forall \mathbf{x}. \quad (4.15)$$

由于 $p(\mathbf{x})$ 积分为 1，函数 f 可以被看出概率密度函数，因此上述不等式等价于如下不等式：

$$f(E[X]) \leq E[f_X(\mathbf{x})]. \quad (4.16)$$

该不等式又称为 Jensen 不等式。

4.2.5 次水平集

凸函数产生了一类重要的凸集称为 α -次水平集 (α -sublevel set)。给定凸函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 及实数 $\alpha \in \mathbb{R}$ ， α 次水平集定义如下：

$$\{\mathbf{x} \in \mathcal{D}(f) : f(\mathbf{x}) \leq \alpha\}. \quad (4.17)$$

也就是说， α -次水平集为所有满足不等式 $f(\mathbf{x}) \leq \alpha$ 的点的集合。为了显示该集合为凸集，考虑 $\mathbf{x}, \mathbf{y} \in \mathcal{D}(f)$ 满足 $f(\mathbf{x}) \leq \alpha$ 及 $f(\mathbf{y}) \leq \alpha$ ，则有：

$$f(\theta \mathbf{x} + (1 - \theta) \mathbf{y}) \leq \theta f(\mathbf{x}) + (1 - \theta) f(\mathbf{y}) \leq \theta \alpha + (1 - \theta) \alpha = \alpha. \quad (4.18)$$

4.2.6 例子

下面首先给出一些单变量凸函数的例子，然后给出多变量凸函数的例子

- **指数函数.** 令函数 $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x) = e^{\alpha x} \quad \forall \alpha \in \mathbb{R}$ 。为了说明 f 为凸函数，求函数的二阶导 $f''(x) = \alpha^2 e^{\alpha x}$ ，对所有 x 该二阶导为正。
- **负对数函数.** 令函数 $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x) = -\log(x)$ ，其中定义域 $\mathcal{D}(f) = \mathbb{R}_{++}$ (这里 \mathbb{R}_{++} 表示严格正实数集合, $\{x : x > 0\}$)。求函数的二阶导 $f''(x) = 1/x^2 > 0 \quad \forall x$ 。
- **仿射函数.** 令函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $f(\mathbf{x}) = \mathbf{b}^\top \mathbf{x} + c \quad \mathbf{b} \in \mathbb{R}^n, c \in \mathbb{R}$ 。此时 Hessian 矩阵 $\nabla_{\mathbf{x}}^2 f(\mathbf{x}) = 0 \forall \mathbf{x}$ 。由于零矩阵同时为半正定和半负定，因此 f 既是凸函数又是凹函数。实际上，该形式的仿射函数为唯一的同时是凸函数及凹函数的函数。
- **二次函数.** 令函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^\top \mathbf{A} \mathbf{x} + \mathbf{b}^\top \mathbf{x} + c$ ，其中矩阵 $\mathbf{A} \in \mathbb{R}^{n \times n}$ 为对称矩阵， $\mathbf{b} \in \mathbb{R}^n$ 及 $c \in \mathbb{R}$ 。该函数的 Hessian 矩阵为：

$$\nabla_{\mathbf{x}}^2 f(\mathbf{x}) = \mathbf{A}. \quad (4.19)$$

因此，函数 f 是否为凸函数完全看矩阵 \mathbf{A} 是否是半正定矩阵，若 \mathbf{A} 为半正定矩阵，则函数为凸函数，若 \mathbf{A} 为不定矩阵，则 f 既非凸函数又非凹函数。注意到欧几里得函数平方 $f(x) = \|\mathbf{x}\|_2^2 = \mathbf{x}^\top \mathbf{x}$ 为特殊形式的二次函数，其中 $\mathbf{A} = \mathbf{I}, \mathbf{b} = 0, c = 0$ ，因此其为严格凸函数。

- **范数.** 令函数 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 为定义在 \mathbb{R}^n 上的范数。根据三角不等式及范数的正齐次性可知，对任意 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ 及 $0 \leq \theta \leq 1$ ，有：

$$f(\theta \mathbf{x} + (1 - \theta) \mathbf{y}) \leq f(\theta \mathbf{x}) + f((1 - \theta) \mathbf{y}) = \theta f(\mathbf{x}) + (1 - \theta) f(\mathbf{y}). \quad (4.20)$$

该例凸函数为不可通过一阶条件或二阶条件证明的凸函数，因为范数通常并非所有点均可导。(譬如， L_1 -范数, $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$ ，在所有含 x_i 为 0 的点 \mathbf{x} 上均不可导)

- **凸函数非负加权平均.** 令 f_1, f_2, \dots, f_k 为凸函数及 $\omega_1, \omega_2, \dots, \omega_k$ 为非负实数，则：

$$f(\mathbf{x}) = \sum_{i=1}^k \omega_i f_i(\mathbf{x}) \quad (4.21)$$

为凸函数，因为：

$$f(\theta \mathbf{x} + (1 - \theta) \mathbf{y}) = \sum_{i=1}^k \omega_i f_i(\theta \mathbf{x} + (1 - \theta) \mathbf{y}) \quad (4.22)$$

$$\leq \sum_{i=1}^k \omega_i (\theta f_i(\mathbf{x}) + (1 - \theta) f_i(\mathbf{y})) \quad (4.23)$$

$$= \theta \sum_{i=1}^k \omega_i f_i(\mathbf{x}) + (1 - \theta) \sum_{i=1}^k \omega_i f_i(\mathbf{y}) \quad (4.24)$$

$$= \theta f(\mathbf{x}) + (1 - \theta) f(\mathbf{y}). \quad (4.25)$$

4.3 优化问题

本部分会首先讨论无约束优化问题，进一步讨论等式约束优化问题，及拉格朗日乘子对该问题的解决方案，最后讨论不等式约束优化问题，及 KKT 条件对该问题的解决方案。

4.3.1 无约束优化

由凸函数定义可知，凸函数 $f(\mathbf{x})$ 在点 \mathbf{x}_k 处取得最小值的必要条件是

$$\nabla_{\mathbf{x}} f(\mathbf{x}_k) = 0 \quad (4.26)$$

由第一章微积分方向导数部分可知，梯度方向为最速上升方向，因此我们可以沿着负梯度方向变化既可找到凸函数的最小值。

4.3.2 等式约束优化问题

在给定约束函数 $h(\mathbf{x}) = 0$ 时，求函数 $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ 的最小值，即：

$$\arg \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \quad \text{subject to} \quad h(\mathbf{x}) = 0 \quad (4.27)$$

这里假设函数分别为如下形式：

$$f(\mathbf{x}) = x_1 + x_2, \quad h(\mathbf{x}) = x_1^2 + x_2^2 - 2 \quad (4.28)$$

因此函数 $f(\mathbf{x})$ 有如下等高线：

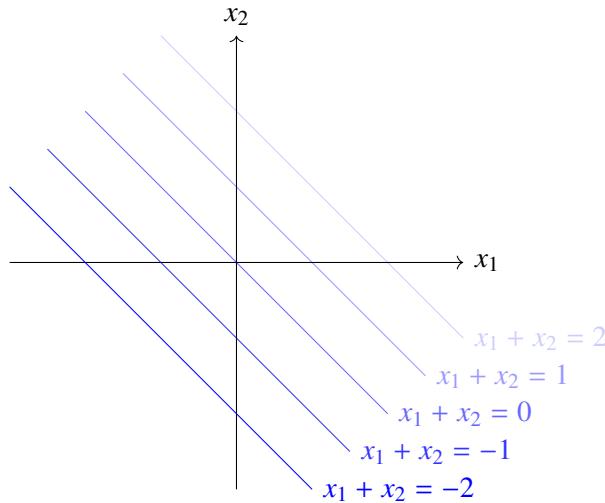
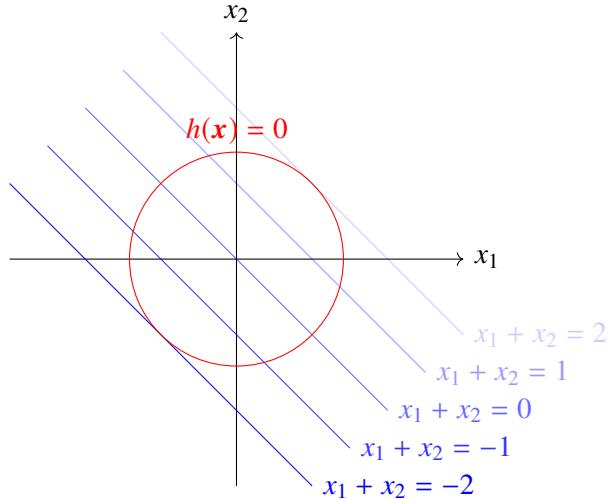
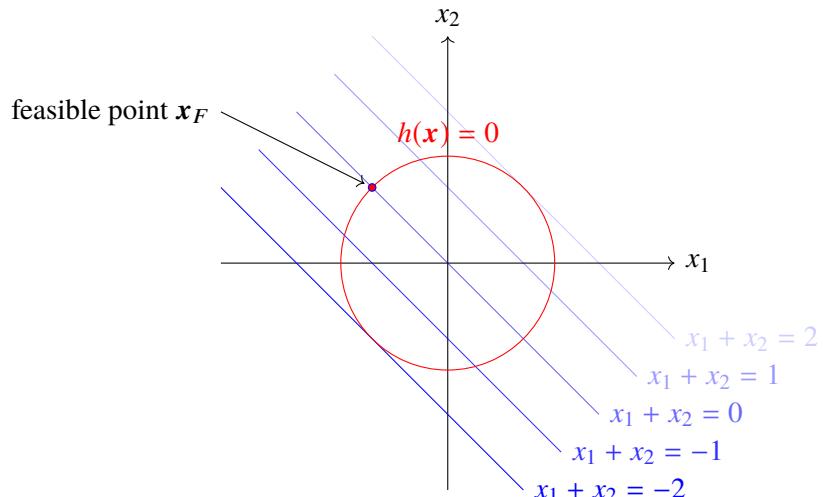


图 4.4: 函数 $f(\mathbf{x})$ 的等高线

在上图中加入约束函数 $h(\mathbf{x})$ 即可以知道满足约束函数的点集：

图 4.5: 函数 $f(\mathbf{x})$ 及 $h(\mathbf{x})$ 的等高线

譬如下图中的点 \mathbf{x}_F 即为约束函数上一个可取点

图 4.6: 约束 $h(\mathbf{x})$ 可取点 \mathbf{x}_F

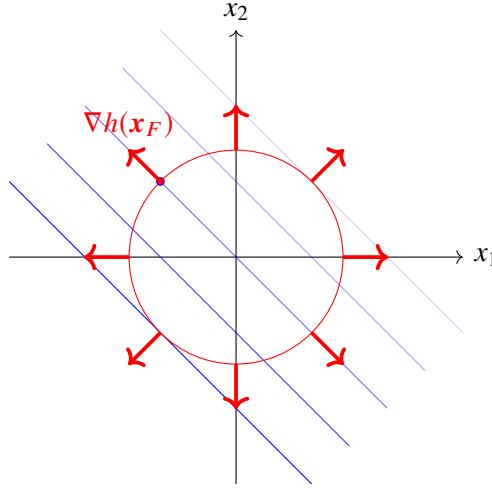
为了求得满足约束函数 $h(\mathbf{x})$ 且使得 $f(\mathbf{x})$ 取得最小值的点, 可知点 \mathbf{x}_F 增量 $\Delta\mathbf{x}$ 需满足如下条件:

$$h(\mathbf{x}_F + \Delta\mathbf{x}) = 0 \quad \text{且} \quad f(\mathbf{x}_F + \Delta\mathbf{x}) < f(\mathbf{x}_F) \quad (4.29)$$

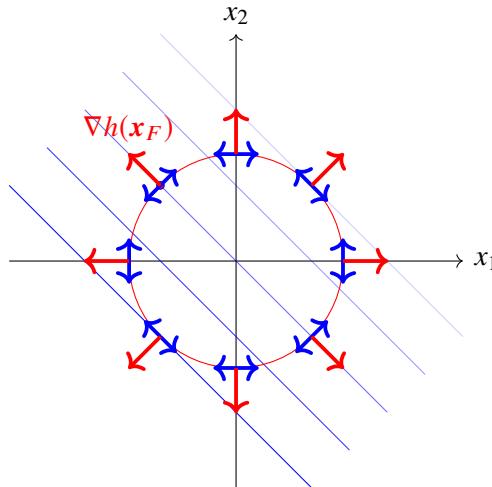
由于在任意点 \mathbf{x} 处 $f(\mathbf{x})$ 的负梯度方向为 $-\nabla f(\mathbf{x})$ 为最速下降方向, 因此从点 \mathbf{x} 处移动 $\Delta\mathbf{x}$ 使得 $f(\mathbf{x}_F + \Delta\mathbf{x}) < f(\mathbf{x}_F)$, $\Delta\mathbf{x}$ 需满足如下条件:

$$\Delta\mathbf{x} \cdot (-\nabla f(\mathbf{x})) > 0 \quad (4.30)$$

另外可知函数 $h(\mathbf{x})$ 在点 \mathbf{x}_F 处的法线方向为 $\nabla h(\mathbf{x}_F)$ 或 $-\nabla h(\mathbf{x}_F)$, 如下图所示:

图 4.7: 函数 $h(\mathbf{x})$ 的法线方向

因此从任意点 \mathbf{x} 处移动微小的 $\Delta\mathbf{x}$ 使得移动后的点仍然在约束函数 $h(\mathbf{x})$ 的表明上，移动方向 $\Delta\mathbf{x}$ 必须垂直于法线方向，如下图所示：

图 4.8: \mathbf{x} 可移动方向

因此若 \mathbf{x}_F 处于约束函数的表面，移动方向 $\Delta\mathbf{x}$ 需满足如下条件：

- $\Delta\mathbf{x}$ 垂直于约束函数 $h(\mathbf{x})$ 在点 \mathbf{x}_F 处的法线法线方向 $\nabla h(\mathbf{x}_F)$
- 仅当 $\Delta\mathbf{x} \cdot (-\nabla f(\mathbf{x}_F)) > 0$ 时 $f(\mathbf{x}_F + \Delta\mathbf{x}) < f(\mathbf{x}_F)$

因此当：

$$\nabla f(\mathbf{x}_F) = \mu \nabla h(\mathbf{x}_F) \quad (4.31)$$

时，其中 μ 为常数。由于 $\Delta\mathbf{x}$ 垂直于 $\nabla h(\mathbf{x}_F)$ ，因此：

$$\Delta\mathbf{x} \cdot (-\nabla f(\mathbf{x}_F)) = -\Delta\mathbf{x} \cdot \mu \nabla h(\mathbf{x}_F) = 0 \quad (4.32)$$

此时，在点 \mathbf{x}_F 处移动 Δ 时，将无法增加或减小函数 $f(\mathbf{x})$ 。因此约束优化的极值点发生

在点 \mathbf{x}^* 处，该点处梯度 $\nabla f(\mathbf{x}^*)$ 平行于 $\nabla h(\mathbf{x}^*)$ ，即：

$$\nabla f(\mathbf{x}^*) = \mu \nabla h(\mathbf{x}^*) \quad (4.33)$$

综上所述，为了求解4.27约束优化问题，可以定义如下拉格朗日函数（ μ 为拉格朗日乘子）：

$$\mathcal{L}(\mathbf{x}, \mu) = f(\mathbf{x}) + \mu h(\mathbf{x}) \quad (4.34)$$

当 \mathbf{x}^* 为最小值时，必要条件有：

- $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \mu) = 0$ ，该方程蕴含着 $\nabla f(\mathbf{x}^*) = \mu \nabla h(\mathbf{x}^*)$
- $\nabla_{\mu} \mathcal{L}(\mathbf{x}^*, \mu) = 0$ ，该方程蕴含着 $h(\mathbf{x}^*) = 0$

4.3.3 不等式约束优化问题

在如上等式约束优化讨论的基础之上，这里我们讨论不等式约束优化问题。在给定约束函数 $g(\mathbf{x}) \leq 0$ 时，求函数 $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ 最小值，即：

$$\arg \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \quad \text{subject to} \quad g(\mathbf{x}) \leq 0 \quad (4.35)$$

这里假设函数分别为如下形式：

$$f(\mathbf{x}) = x_1^2 + x_2^2, \quad g(\mathbf{x}) = x_1^2 + x_2^2 - 1 \quad (4.36)$$

因此函数 $f(\mathbf{x})$ 等高线如下图所示：

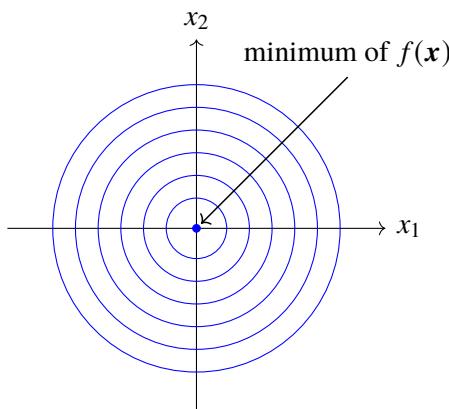
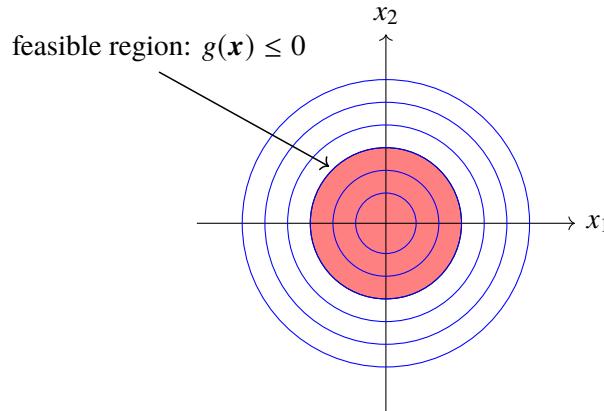
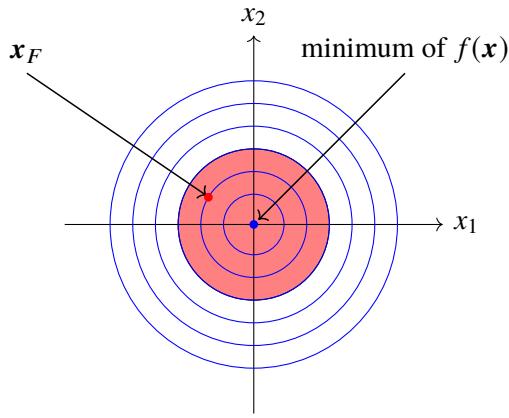


图 4.9: 函数 $f(\mathbf{x})$ 等高线

在上图中加入约束函数 $g(\mathbf{x})$ 的等高线后，如下图所示：

图 4.10: 函数 $f(\mathbf{x})$ 等高线及约束函数 $g(\mathbf{x}) \leq 0$

譬如下图中的点 \mathbf{x}_F 即为约束函数上一个可取点:

图 4.11: 约束函数上可取点 \mathbf{x}_F

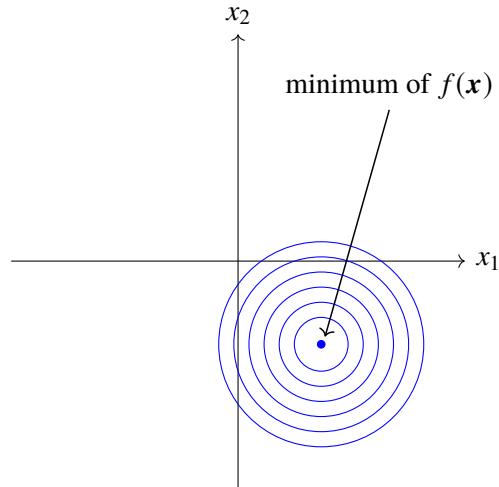
从上图可见函数 $f(\mathbf{x})$ 的无约束最小值点处于不等式约束可取点集中, 因此, 在该情况下凸函数的不等式约束求最小值问题, 等价于无约束最小值求解问题, 其所需条件为:

$$\nabla f(\mathbf{x}_F) = 0 \quad (4.37)$$

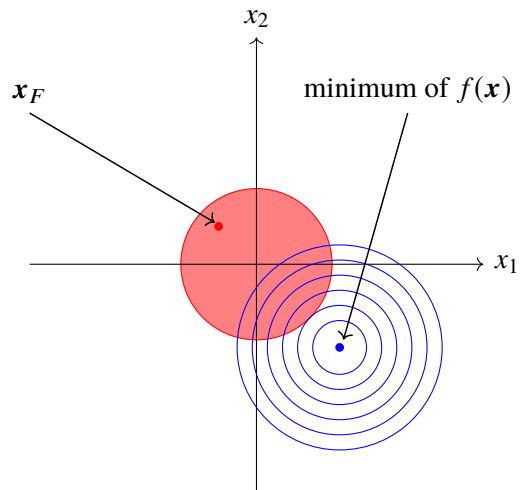
让我们再来看另一个不等式约束优化问题, 这里假设函数分别为如下形式:

$$f(\mathbf{x}) = (x_1 - 1.1)^2 + (x_2 + 1.1)^2, \quad g(\mathbf{x}) = x_1^2 + x_2^2 - 1 \quad (4.38)$$

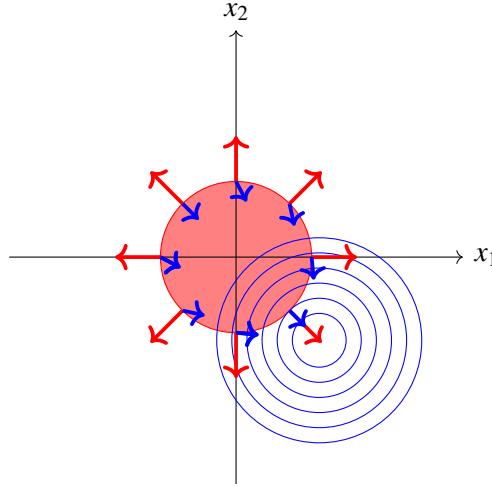
如下是函数 $f(\mathbf{x})$ 的等高线及无约束下的最小值点:

图 4.12: 函数 $f(\mathbf{x})$ 及最小值点

在上图中加入不等式约束函数可取点集合, 图如下所示:

图 4.13: 约束函数上可取点 \mathbf{x}_F

若 \mathbf{x}_F 为满足约束不等式的点且在该点处函数 $f(\mathbf{x})$ 取得最小值, 则点 \mathbf{x}_F 可取的点集处于约束不等式对应的球体表面, 如下图所示:

图 4.14: x_F 处于约束不等式球体表面

该情形下，不等式约束优化问题等价于等式约束优化问题，此时最小值点 \mathbf{x}^* 满足如下条件：

$$-\nabla f(\mathbf{x}^*) = \lambda \nabla g(\mathbf{x}^*), \quad \lambda > 0 \quad (4.39)$$

因此给定约束优化问题4.35式，若 \mathbf{x}^* 为约束条件下的最小值点，则存在两种情形：

- 内部解：函数 $f(\mathbf{x})$ 的无约束最小值点处于约束不等式 $g(\mathbf{x}) \leq 0$ 定义域中，这时有：

$$g(\mathbf{x}^*) < 0 \quad \nabla f(\mathbf{x}^*) = 0 \quad \text{且} \quad \lambda = 0 \quad (4.40)$$

- 边界解：函数 $f(\mathbf{x})$ 的无约束最小值点处于约束不等式 $g(\mathbf{x}) \leq 0$ 定义域外，这时有：

$$g(\mathbf{x}^*) = 0 \quad \text{且} \quad -\nabla f(\mathbf{x}^*) = \lambda \nabla g(\mathbf{x}^*), \quad \lambda > 0 \quad (4.41)$$

因此无论是内部解或边界解， $\lambda g(\mathbf{x}^*) = 0$ 恒成立。考虑如上两类情形，定义如下拉格朗日函数：

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}), \quad (\lambda \geq 0) \quad (4.42)$$

当 \mathbf{x}^* 为最小值点时，其必要条件包括：

- KKT 条件 1: $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \lambda) = 0$
- KKT 条件 2: $g(\mathbf{x}^*) \leq 0$
- KKT 条件 3: $\lambda \geq 0$
- KKT 条件 4: $\lambda g(\mathbf{x}^*) = 0$

上述这些条件合称为 Karush-Kuhn-Tucker (KKT) 条件。上面的结果可推广至多个约束等

式和和约束不等式的情况。考虑如下约束优化问题：

$$\arg \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \quad (4.43)$$

$$\text{subject to } g_i(\mathbf{x}) = 0, \quad i = 1, \dots, k \quad (4.44)$$

$$h_j(\mathbf{x}) \leq 0, \quad j = 1, \dots, m \quad (4.45)$$

$$(4.46)$$

定义如下拉格朗日函数：

$$\mathcal{L}(\mathbf{x}, \{\lambda_i\}, \{\mu_j\}) = f(\mathbf{x}) + \sum_{i=1}^k \lambda_i g_i(\mathbf{x}) + \sum_{j=1}^m \mu_j h_j(\mathbf{x}) \quad (4.47)$$

其中 λ_i 为 $g_i(\mathbf{x}) = 0$ 的拉格朗日乘子， μ_j 为 $h_j(\mathbf{x}) \leq 0$ 的拉格朗日乘子。此时 KKT 条件包括：

- KKT 条件 1: $\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \{\lambda_i\}, \{\mu_j\}) = 0$
- KKT 条件 2: $g_i(\mathbf{x}^*) = 0, \quad i = 1, \dots, k$
- KKT 条件 3: $h_j(\mathbf{x}^*) \leq 0, \quad j = 1, \dots, m$
- KKT 条件 4: $\mu_j \geq 0$
- KKT 条件 5: $\mu_j h_j(\mathbf{x}^*) = 0$

4.4 一阶近似优化

本节回顾采用函数的一阶导数信息结合函数的泰勒展开式近似，迭代求解函数最小值点数值方法

4.4.1 SGD

可导函数 $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ 在任意点 \mathbf{x}_k 附近可以用如下一阶泰勒展开式近似：

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + (\mathbf{x} - \mathbf{x}_k)^T \nabla f(\mathbf{x}_k) \quad (4.48)$$

若点 \mathbf{x}_k 处连续但不可导，此时可以用次梯度近似梯度。考虑到函数 $f(\mathbf{x})$ 的一阶近似为线性函数，求解器最小值，我们仅需沿着负梯度方向移动一个较小量，即：

$$\mathbf{x} - \mathbf{x}_k = -\mu_k \nabla f(\mathbf{x}_k), \quad (\mu_k > 0) \quad (4.49)$$

将该解代入一阶泰勒展开式近似可得：

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) - \mu_k \overbrace{\nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k)}^{>0} < f(\mathbf{x}_k) \quad (4.50)$$

因此迭代公式如下：

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mu_k \nabla f(\mathbf{x}_k) \quad (4.51)$$

上述公式即为最小值点搜索的梯度下降数值解。

- 梯度下降算法：若每次计算时采用全量样本，则上述方法称为梯度下降算法 (Batch Gradient Descent)
- 随机梯度下降算法：若每次计算时采用一个样本则上述公式称为随机梯度下降算法 (Stochastic Gradient Descent)
- 小批量随机梯度下降算法：若采用一部分样本，上述算法又称为小批量随机梯度下降算法 (Mini-Batch Stochastic Gradient Descent)

公式4.51中 μ_k 称为学习率，常常设定为固定值，或者根据迭代次数做相应的衰减，譬如采用公式 $1/\sqrt{k}$ 。

4.4.2 Adagrad

上述梯度下降算法对每个变量设定了统一的步长，Adagrad (adaptive gradient) 算法通过累积各变量的历史偏导数，对历史累积偏导数比较大的变量设定较小的学习率，对历史累积偏导数较小的变量设定较大的偏导数。

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \nabla f(\mathbf{x}_k)^2 \quad (4.52)$$

因此有如下迭代公式：

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\mu}{\sqrt{\mathbf{z}_{k+1} + \epsilon}} \nabla f(\mathbf{x}_k) \quad (4.53)$$

上述公式 ϵ 一般取值为 10^{-8} 。可以看到随着迭代次数增加，梯度 \mathbf{z}_{k+1} 各维度值将逐渐变大，最终导致 \mathbf{x}_{k+1} 逐渐收敛。

4.4.3 Adam

Adam 算法有效综合了 Momentum 算法和 RMSprop 算法的优点，同时考虑了梯度和梯度的平方动态平均信息。

$$\mathbf{m}_{k+1} = \beta_1 \mathbf{m}_k + (1 - \beta_1) \nabla f(\mathbf{x}_k) \quad (4.54)$$

$$\mathbf{v}_{k+1} = \beta_2 \mathbf{v}_k + (1 - \beta_2) \nabla f(\mathbf{x}_k)^2 \quad (4.55)$$

$$\hat{\mathbf{m}}_{k+1} = \frac{\mathbf{m}_{k+1}}{1 - \beta_1^{k+1}} \quad (4.56)$$

$$\hat{\mathbf{v}}_{k+1} = \frac{\mathbf{v}_{k+1}}{1 - \beta_2^{k+1}} \quad (4.57)$$

迭代公式如下：

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\mu}{\sqrt{\hat{\mathbf{v}}_{k+1} + \epsilon}} \hat{\mathbf{m}}_{k+1} \quad (4.58)$$

上述公式中 β_1 一般取值为 0.9, β_2 取值为 0.9999, ϵ 取值为 10^{-8} 。关于 Momentum 算法和 RMSprop 算法, 若需要可进一步阅读 Ruder 的综述¹。

¹<http://ruder.io/optimizing-gradient-descent/index.html#momentum>

第5章 神经网络及反向传播

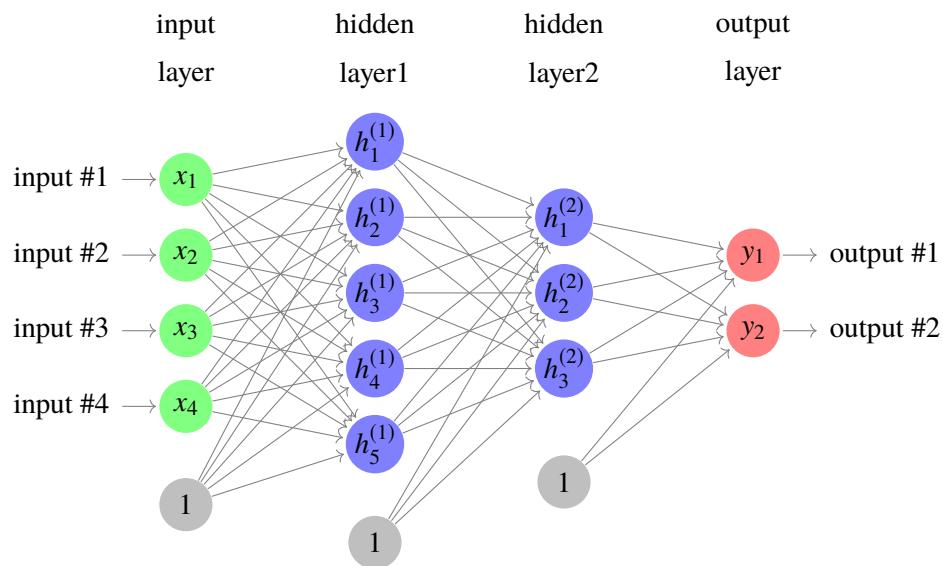
本章首先简要回顾神经网络的基本概念，在此基础上引入计算图的神经网络表示方案，计算图的表示方案理解神经网络在计算机内部的表示形式至关重要，有了计算图之后，我们可以更好的理解神经网络的前向传播计算及反向传播计算，其各自对应的向量化计算形式。最后我们进一步讨论了梯度爆炸或梯度消散问题，理解了梯度爆炸或消散的原因，会明白为何我们需要优化权值初始化算法或选择某个特定的激活函数。

5.1 神经网络

神经网络由若干如下所示的类神经处理单元构成：

$$a = \phi\left(\sum_i \omega_i x_i + b\right).$$

其中 x_i 为处理单元的输入， ω_i 为输入权重， b 为先验偏置项， ϕ 为非线性激活函数， a 为处理单元的输出。可以看到某个神经元的工作都很简单，但是将这些神经元组合后将可以执行复杂的任务。关于多层神经网络可以拟合任意复杂函数的问题，这里不做具体的讨论。另外这里及本章后面的算法讨论将以多层感知器神经网络作为讨论对象。最简单的前向神经网络为为多层感知器（Multilayer Perceptron），如下图所示：



上面多层感知机包含一个输入层，两个隐藏层，一个输出层，除输入层外，每一层的包含若干定义一致的处理单元（神经元）。定义如下：

$$h_i^{(1)} = \phi^{(1)}\left(\sum_j \omega_{ij}^{(1)} x_j + b_i^{(1)}\right)$$

$$h_i^{(2)} = \phi^{(2)}\left(\sum_j \omega_{ij}^{(2)} h_j^{(1)} + b_i^{(2)}\right)$$

$$y_i = \phi^{(3)}\left(\sum_j \omega_{ij}^{(3)} h_j^{(2)} + b_i^{(3)}\right)$$

可以看到上述定义对不同层的激活函数 $\phi^{(k)}$ 进行了区分，实际应用中，常常对隐藏层选用统一的激活函数，譬如 Relu，输出层根据要解决的具体问题选择相应的激活函数。

5.2 计算图

上一小节给出了一个多层感知机神经网络的架构图，神经网络在计算机内部表示时一般采用计算图的表示方案，图的节点和边的定义：

- 节点：表示函数，譬如创建输入张量或矩阵、向量及标量的函数，或接受两个输入产生一个输出的函数，譬如加法函数，或接受一个输入参数产生一个输出的函数，譬如 sigmoid 函数。每个节点定义了如何求取函数输出，及函数对于输入的导数
- 边：表示张量或矩阵、向量及标量，即节点产生的输出

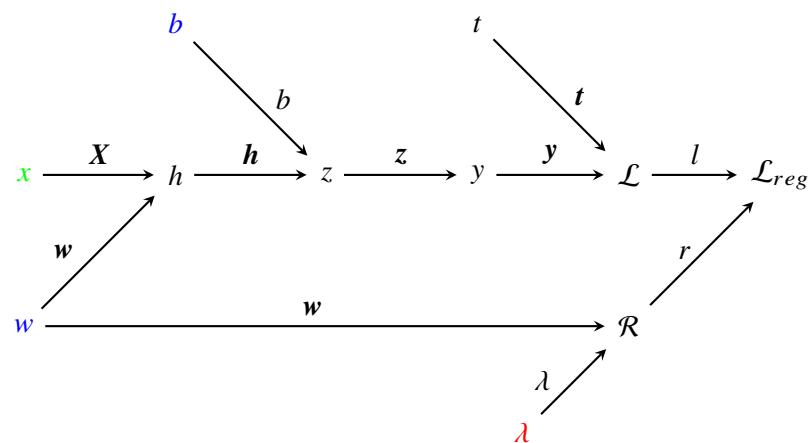
由节点和边共同构成了整个计算图。从上述定义可知计算图描述了函数在计算机内部的表达形式，以及给定输入后如何对函数进行求解。基于 MLE 或 MAP 参数估计方法，以及给定数据集，即可以进行函数参数估计。这里给出一个简单的例子（一个神经元）来描述计算图如何用于神经网络表示：

$$z(w, X, b) = Xw + b, \quad y(z) = \frac{1}{1 + e^{-z}}$$

$$\mathcal{L}(y, t) = -(t \cdot \log(y) + (1 - t) \cdot \log(1 - y))$$

$$\mathcal{R}(w, \lambda) = \lambda ||w||, \quad \mathcal{L}_{reg}(l, r) = l + r$$

上述函数表达式稍显复杂，下面我们通过计算图来描述该函数表达式：



可以看到上述计算图有四类节点：数据函数，参数函数，常量函数及计算函数，节点与节点之间的边表示相互之间的数据依赖，边上是对应的数据，上述计算图中有三类数据：矩阵，向量和标量。计算图中的每个节点（函数）知道如何求得其输出，及相对其输入的导数。因此通过计算图的表示方式，我们可以将复合函数通过分而治之的方式表达为若干函数的组合，借助于每个函数内部完成输出计算和导数计算，完成整体复合函数的输出计算和导数计算。下面我们将分别回顾神经网络前向传播计算和反向传播计算如何借助于计算图完成。

5.3 反向传播算法

本小节首先回顾一下链乘法则，在此基础上结合前一小节给出的计算图定义，给出前向反向传播算法如何进行导数计算。

5.3.1 链乘法则

给定复合函数 $f(x(t), y(t))$ ，根据链式法则，复合函数对 t 的导数如下：

$$\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}.$$

假设函数 \mathcal{L} 相对其参数 v 的偏导数已经计算好，并表示为：

$$\bar{v} = \frac{\partial \mathcal{L}}{\partial v}$$

则导数 $\frac{df}{dt}$ 可以表示为：

$$\bar{t} = \bar{x} \frac{dx}{dt} + \bar{y} \frac{dy}{dt}.$$

因此，为了计算 \bar{t} ，我们需要计算的导数为 dx/dt 和 dy/dt ， \bar{x} 和 \bar{y} 为之前已经计算好的导数。

5.3.2 反向传播算法

给定数据集和神经网络函数对应的计算图架构定义后，我们的任务主要有两个：

- 参数估计，譬如上一小节中的 w 及 b
- 超参优化，譬如上一小节中的 λ

这里我们重点讨论参数估计方法，超参优化留待后续章节讨论。参数估计可以通过 *MLE* 或 *MAP* 完成。上一小节定义的计算图对应 *MAP* 参数估计方法。下面我们看看如何通过反向传播算法完成求导，结合随机梯度下降通过数值迭代即可完成参数估计。

令 v_1, \dots, v_N 表示计算图上的所有节点，节点满足拓扑排序（拓扑排序的输出中父节点排在子节点之前，可以通过深度优先遍历结合栈及集合的方式得到）。我们希望计算各节点相对其输入的所有导数 \bar{v}_i ，虽然并非所有导数对解决问题来说均是需要的。为了计算各节点的导数，我们首先通过前向过程计算图上的所有节点，生成边上的值（张量等），然

后通过反向过程计算所有导数 \bar{v}_i 。对于最后一个节点 v_N , 其输出为自身, 由于自身相对自身的导数为 1, 因此有 $\bar{v}_N = 1$ 。反向传播算法定义如下:

Algorithm 1 Backpropagation Algorithm

```

procedure FORWARD PASS
  for  $i = 1, \dots, N$  do
    Compute  $v_i$  as a function of  $Pa(v_i)$ 
   $\bar{v}_N = 1$ 
procedure BACKWARD PASS
  for  $i = N - 1, \dots, 1$  do
     $\bar{v}_i = \sum_{j \in Ch(v_i)} \bar{v}_j \frac{\partial v_j}{\partial v_i}$ 

```

上述算法中 $Pa(v_i)$ 和 $Ch(v_i)$ 表示 v_i 的父节点和子节点。下面我们将该算法应用于上一小节给出的计算图, 求解导数:

$$\begin{aligned}
 \overline{\mathcal{L}_{reg}} &= 1 \\
 \overline{\mathcal{L}} &= \overline{\mathcal{L}_{reg}} \frac{\partial \mathcal{L}_{reg}}{\partial \mathcal{L}} = \overline{\mathcal{L}_{reg}} \\
 \overline{\mathcal{R}} &= \overline{\mathcal{L}_{reg}} \frac{\partial \mathcal{L}_{reg}}{\partial \mathcal{R}} = \overline{\mathcal{L}_{reg}} \\
 \bar{y} &= \overline{\mathcal{L}} \frac{\partial \mathcal{L}}{\partial y} = \overline{\mathcal{L}} \left(-t \odot \frac{1}{y} + (1-t) \odot \frac{1}{1-y} \right) \\
 \bar{z} &= \bar{y} \frac{\partial y}{\partial z} = \bar{y} \odot \sigma'(z) \\
 \bar{b} &= \bar{z} \frac{\partial z}{\partial b} = [1, \dots, 1] \cdot \bar{z}^\top \\
 \bar{h} &= \bar{z} \frac{\partial z}{\partial h} = \bar{z} \\
 \bar{w} &= \bar{h} \frac{\partial h}{\partial w} + \overline{\mathcal{R}} \frac{\partial \mathcal{R}}{\partial w} = X^\top \bar{h} + \overline{\mathcal{R}} \lambda
 \end{aligned}$$

计算符号 \odot 为 hadamard 乘, 表示张量相应元素相乘。从反向传播算法的计算过程可见, 正确求解计算图上每个节点的导数至关重要, 下一小节我们将讨论反向传播过程中如何正确求解每个节点的导数。

5.4 向量化反向传播

本节主要回顾反向传播过程中如何进行向量化求导, 我们会首先考察线性函数节点, 在其基础上进一步考察非线性函数节点

5.4.1 线性函数节点

假设节点有两个输入, 分别为输入数据 $X \in \mathbb{R}^{N \times D}$ 及权重矩阵 $W \in \mathbb{R}^{D \times M}$, 节点为线性函数 $Y = XW$, 因此 $Y \in \mathbb{R}^{N \times M}$ 。考虑到让问题更加具象, 这里我们假设 $N = 2, D = 2, M = 3$ 。

我们可以基于输入元素写出当前节点的前向过程，具体如下：

$$\mathbf{X} = \begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \end{bmatrix} \quad (5.1)$$

$$\mathbf{Y} = \mathbf{X}\mathbf{W} = \begin{bmatrix} X_{1,1}W_{1,1} + X_{1,2}W_{2,1} & X_{1,1}W_{1,2} + X_{1,2}W_{2,2} & X_{1,1}W_{1,3} + X_{1,2}W_{2,3} \\ X_{2,1}W_{1,1} + X_{2,2}W_{2,1} & X_{2,1}W_{1,2} + X_{2,2}W_{2,2} & X_{2,1}W_{1,3} + X_{2,2}W_{2,3} \end{bmatrix} \quad (5.2)$$

当前节点经过前向过程后，假设节点的输出将被用于计算图中后续节点的输入，最终将被应用于输出为标量的损失函数节点 \mathcal{L} 。在反向传播过程中，假设损失函数节点对当前节点的导数 $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$ 已经完成计算（函数 \mathbf{Y} 对应的输出为 \mathbf{Y} ）。譬如，若当前节点为线性分类器的一部分，此时矩阵 \mathbf{Y} 为样本在各个类别上的得分，将这些得分输入给损失函数（譬如 softmax 或者多类别 SVM loss）其计算输出标量损失值 L 及损失函数相对得分的导数 $\frac{\partial L}{\partial \mathbf{Y}}$ 。由于 L 为标量， \mathbf{Y} 为 $N \times M$ 的矩阵，因此梯度 $\frac{\partial L}{\partial \mathbf{Y}}$ 为和 \mathbf{Y} 的维度一致的矩阵， $\frac{\partial L}{\partial \mathbf{Y}}$ 的每个元素为损失 L 相对矩阵 \mathbf{Y} 的每个元素的导数：

$$\frac{\partial L}{\partial \mathbf{Y}} = \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix} \quad (5.3)$$

在反向过程中，我们的目标是通过使用 $\frac{\partial L}{\partial \mathbf{Y}}$ 计算出 $\frac{\partial L}{\partial \mathbf{X}}$ 及 $\frac{\partial L}{\partial \mathbf{W}}$ （注：函数 \mathbf{X} 对应的输出为 \mathbf{X} ，函数 \mathbf{W} 对应的输出为 \mathbf{W} ）。同样，由于 L 为标量，因此我们知道 $\frac{\partial L}{\partial \mathbf{X}}$ 为和 $\mathbf{X} \in \mathbb{R}^{N \times D}$ 维度一致的矩阵， $\frac{\partial L}{\partial \mathbf{W}}$ 为和 $\mathbf{W} \in \mathbb{R}^{D \times M}$ 维度一致的矩阵。根据链乘法则，我们知道：

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \quad \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{W}} \quad (5.4)$$

由第二章向量化导数计算小节可知，上述公式中的 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 及 $\frac{\partial \mathbf{Y}}{\partial \mathbf{W}}$ 为广义雅可比矩阵，包含了矩阵 \mathbf{Y} 的每个元素相对输入 \mathbf{X} 和 \mathbf{W} 的导数。然而我们并不希望显式构建雅可比矩阵 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 及 $\frac{\partial \mathbf{Y}}{\partial \mathbf{W}}$ ，原因是广义雅可比矩阵太大了。在一个典型的神经网络架构中，给定维度可能为 $N = 64$ 及 $M = D = 4096$ ，此时 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 包含 $64 \cdot 4096 \cdot 64 \cdot 4096$ 个标量值，超过 680 亿，若每个元素用 32 位的浮点数表示，该广义雅可比矩阵将需要 256GB 的内存空间。因此显式存储和操作该广义雅可比矩阵时完全没有希望的。幸运的是，对于大部分神经网络来说，我们可以不用显式构建 $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 来完成 $\frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ 计算。很多时候我们可以通过解决一个小的问题，在其基础上完成一般化推广。我们首先来看，在 $N = 2, D = 2, M = 3$ 时，如何计算 $\frac{\partial L}{\partial \mathbf{X}}$ 。同样，我们知道 $\frac{\partial L}{\partial \mathbf{X}}$ 和 \mathbf{X} 的维度一致：

$$\mathbf{X} = \begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{bmatrix} \Rightarrow \frac{\partial L}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial L}{\partial X_{1,1}} & \frac{\partial L}{\partial X_{1,2}} \\ \frac{\partial L}{\partial X_{2,1}} & \frac{\partial L}{\partial X_{2,2}} \end{bmatrix} \quad (5.5)$$

我们可以一次求解一个元素，首先我们将求解 $\frac{\partial L}{\partial X_{1,1}}$ 。根据链乘法则，我们知道：

$$\frac{\partial L}{\partial X_{1,1}} = \sum_{i=1}^N \sum_{j=1}^M \frac{\partial L}{\partial Y_{i,j}} \frac{\partial Y_{i,j}}{\partial X_{1,1}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial X_{1,1}} \quad (5.6)$$

上述公式中 L 和 $X_{1,1}$ 均为标量，因此 $\frac{\partial L}{\partial X_{1,1}}$ 也是标量。若我们不是将 \mathbf{Y} 看成矩阵，而是看出函数 Y ，对于矩阵 \mathbf{Y} 的每个元素为标量值函数，因此我们可以应用链乘法则将 $\frac{\partial L}{\partial X_{1,1}}$ 表示为标量导数的链乘形式。

为了避免处理求和，通常将 $\frac{\partial L}{\partial Y_{i,j}}$ 的所有元素放到一个矩阵 $\frac{\partial L}{\partial Y}$ 中，这里 L 为标量，矩阵维度和 $\mathbf{Y} \in \mathbb{R}^{N \times M}$ 一致，矩阵中的每个元素为 L 相对 \mathbf{Y} 中每个元素的导数。类似地，我们也可以将所有 $\frac{\partial Y_{i,j}}{\partial X_{1,1}}$ 放到矩阵 $\frac{\partial Y}{\partial X_{1,1}}$ 中，由于 Y 为对应的矩阵 $\mathbf{Y} \in \mathbb{R}^{N \times M}$ ， $X_{1,1}$ 为标量，因此 $\frac{\partial Y}{\partial X_{1,1}}$ 为和矩阵维度一致的矩阵。由于 $\frac{\partial L}{\partial X_{1,1}}$ 为标量，因此 $\frac{\partial L}{\partial Y}$ 和 $\frac{\partial Y}{\partial X_{1,1}}$ 的乘积为标量，通过观察前面仅通过标量导数链乘的表达式5.6，我们知道 $\frac{\partial L}{\partial Y}$ 和 $\frac{\partial Y}{\partial X_{1,1}}$ 的乘积运算为点击运算。

在反向过程中，已经给定导数 $\frac{\partial L}{\partial Y}$ ，因此我们仅需要计算 $\frac{\partial L}{\partial X_{1,1}}$ 。通过等式5.2，我们可以很容易的求出：

$$\frac{\partial Y}{\partial X_{1,1}} = \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ 0 & 0 & 0 \end{bmatrix} \quad (5.7)$$

现在结合等式5.6及5.7，可得：

$$\frac{\partial L}{\partial X_{1,1}} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial X_{1,1}} \quad (5.8)$$

$$= \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix} \cdot \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ 0 & 0 & 0 \end{bmatrix} \quad (5.9)$$

$$= \frac{\partial L}{\partial Y_{1,1}} W_{1,1} + \frac{\partial L}{\partial Y_{1,2}} W_{1,2} + \frac{\partial L}{\partial Y_{1,3}} W_{1,3} \quad (5.10)$$

我们可以重复该过程，计算 $\frac{\partial L}{\partial X}$ 的其他元素：

$$\frac{\partial L}{\partial X_{1,2}} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial X_{1,2}} \quad (5.11)$$

$$= \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix} \cdot \begin{bmatrix} W_{2,1} & W_{2,2} & W_{2,3} \\ 0 & 0 & 0 \end{bmatrix} \quad (5.12)$$

$$= \frac{\partial L}{\partial Y_{1,1}} W_{2,1} + \frac{\partial L}{\partial Y_{1,2}} W_{2,2} + \frac{\partial L}{\partial Y_{1,3}} W_{2,3} \quad (5.13)$$

$$\frac{\partial L}{\partial X_{2,1}} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial X_{2,1}} \quad (5.14)$$

$$= \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ W_{1,1} & W_{1,2} & W_{1,3} \end{bmatrix} \quad (5.15)$$

$$= \frac{\partial L}{\partial Y_{2,1}} W_{1,1} + \frac{\partial L}{\partial Y_{2,2}} W_{1,2} + \frac{\partial L}{\partial Y_{2,3}} W_{1,3} \quad (5.16)$$

$$\frac{\partial L}{\partial X_{2,2}} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial X_{2,2}} \quad (5.17)$$

$$= \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ W_{2,1} & W_{2,2} & W_{2,3} \end{bmatrix} \quad (5.18)$$

$$= \frac{\partial L}{\partial Y_{2,1}} W_{2,1} + \frac{\partial L}{\partial Y_{2,2}} W_{2,2} + \frac{\partial L}{\partial Y_{2,3}} W_{2,3} \quad (5.19)$$

最后结合上述等式，我们可以得到如下 $\frac{\partial L}{\partial X}$ 的表达式：

$$\frac{\partial L}{\partial X} = \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} W_{1,1} + \frac{\partial L}{\partial Y_{1,2}} W_{1,2} + \frac{\partial L}{\partial Y_{1,3}} W_{1,3} & \frac{\partial L}{\partial Y_{1,1}} W_{2,1} + \frac{\partial L}{\partial Y_{1,2}} W_{2,2} + \frac{\partial L}{\partial Y_{1,3}} W_{2,3} \\ \frac{\partial L}{\partial Y_{2,1}} W_{1,1} + \frac{\partial L}{\partial Y_{2,2}} W_{1,2} + \frac{\partial L}{\partial Y_{2,3}} W_{1,3} & \frac{\partial L}{\partial Y_{2,1}} W_{2,1} + \frac{\partial L}{\partial Y_{2,2}} W_{2,2} + \frac{\partial L}{\partial Y_{2,3}} W_{2,3} \end{bmatrix} \quad (5.20)$$

$$= \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix} \begin{bmatrix} W_{1,1} & W_{2,1} \\ W_{1,2} & W_{2,2} \\ W_{1,3} & W_{2,3} \end{bmatrix} \quad (5.21)$$

$$= \boxed{\frac{\partial L}{\partial Y} \mathbf{W}^\top} \quad (5.22)$$

等式5.22中， $\frac{\partial L}{\partial Y} \in \mathbb{R}^{N \times M}$ ， $\mathbf{W} \in \mathbb{R}^{D \times M}$ ，因此 $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \mathbf{W}^\top \in \mathbb{R}^{N \times D}$ ，其和 X 的维度一致。针对具体值 $N = D = 2, M = 3$ ，我们推出的等式5.22对任意 N, D, M 均成立，该等式让我们可以基于 $\frac{\partial L}{\partial Y}$ 及 \mathbf{W} 高效计算 $\frac{\partial L}{\partial X}$ ，而无需显式构建 $\frac{\partial Y}{\partial X}$ 的广义雅可比形式。使用同样的策略，我们可以求得 $\frac{\partial L}{\partial \mathbf{W}}$ ，而无需显式构建 $\frac{\partial Y}{\partial \mathbf{W}}$ 的广义雅可比形式：

$$\boxed{\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^\top \frac{\partial L}{\partial Y}} \quad (5.23)$$

该等式中 $\frac{\partial L}{\partial \mathbf{W}}$ 需要和 $\mathbf{W} \in \mathbb{R}^{D \times M}$ 具有同样的维度。等式的右边 $\mathbf{X}^\top \in \mathbb{R}^{D \times N}$ ， $\frac{\partial L}{\partial Y} \in \mathbb{R}^{N \times M}$ ，因此 $\frac{\partial L}{\partial \mathbf{W}} \in \mathbb{R}^{D \times M}$ 。一次考虑一个元素的策略可以帮助我们在反向传播过程推出当节点的输入和输出均为任意维度的张量对应的导数，譬如该策略对反向传播过程中求解卷积函数的导数非常有效。

5.4.2 非线性函数节点

上述推导过程中我们考虑的节点为线性函数，即：

$$\mathbf{Y} = \mathbf{X}\mathbf{W}$$

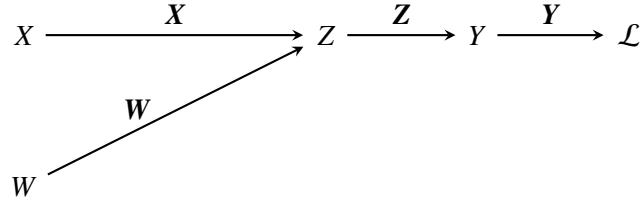
若节点为非线性函数，譬如如下形式：

$$\mathbf{Y} = \sigma(\mathbf{Z}), \quad \mathbf{Z} = \mathbf{X}\mathbf{W}$$

因此有

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial Z} \frac{\partial Z}{\partial X} \quad \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial Z} \frac{\partial Z}{\partial \mathbf{W}} \quad (5.24)$$

此时其对应的计算图如下所示：



同样我们假设 $N = 2, D = 2, M = 3$ ，我们可以基于输入元素写出计算图中各节点的前向过程，具体如下：

$$\mathbf{Z} = \mathbf{X}\mathbf{W} = \begin{bmatrix} X_{1,1}W_{1,1} + X_{1,2}W_{2,1} & X_{1,1}W_{1,2} + X_{1,2}W_{2,2} & X_{1,1}W_{1,3} + X_{1,2}W_{2,3} \\ X_{2,1}W_{1,1} + X_{2,2}W_{2,1} & X_{2,1}W_{1,2} + X_{2,2}W_{2,2} & X_{2,1}W_{1,3} + X_{2,2}W_{2,3} \end{bmatrix} \quad (5.25)$$

$$\mathbf{Y} = \sigma(\mathbf{Z}) = \begin{bmatrix} \sigma(Z_{1,1}) & \sigma(Z_{1,2}) & \sigma(Z_{1,3}) \\ \sigma(Z_{2,1}) & \sigma(Z_{2,2}) & \sigma(Z_{2,3}) \end{bmatrix} \quad (5.26)$$

同样，我们知道 $\frac{\partial L}{\partial Z}$ 和 \mathbf{Z} 的维度一致：

$$\mathbf{Z} = \begin{bmatrix} Z_{1,1} & Z_{1,2} & Z_{1,3} \\ Z_{2,1} & Z_{2,2} & Z_{2,3} \end{bmatrix} \Rightarrow \frac{\partial L}{\partial \mathbf{Z}} = \begin{bmatrix} \frac{\partial L}{\partial Z_{1,1}} & \frac{\partial L}{\partial Z_{1,2}} & \frac{\partial L}{\partial Z_{1,3}} \\ \frac{\partial L}{\partial Z_{2,1}} & \frac{\partial L}{\partial Z_{2,2}} & \frac{\partial L}{\partial Z_{2,3}} \end{bmatrix} \quad (5.27)$$

我们可以一次求解一个元素，首先我们将求解 $\frac{\partial L}{\partial Z_{1,1}}$ 。根据链乘法则，我们知道：

$$\frac{\partial L}{\partial Z_{1,1}} = \sum_{i=1}^N \sum_{j=1}^M \frac{\partial L}{\partial Y_{i,j}} \frac{\partial Y_{i,j}}{\partial Z_{1,1}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial Z_{1,1}} \quad (5.28)$$

上述公式中 L 和 $Z_{1,1}$ 均为标量，因此 $\frac{\partial L}{\partial Z_{1,1}}$ 也是标量。若我们不是将 \mathbf{Y} 看成矩阵，而是看成函数 Y ，对于矩阵 \mathbf{Y} 的每个元素为标量值函数，因此我们可以应用链乘法则将 $\frac{\partial L}{\partial Z_{1,1}}$ 表示为标量导数的链乘形式。

和上述线性函数求导过程类似， $\frac{\partial L}{\partial Y}$ 的每个元素为损失 L 相对矩阵 \mathbf{Y} 的每个元素的导数：

$$\frac{\partial L}{\partial Y} = \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix}$$

在反向过程中，已经给定导数 $\frac{\partial L}{\partial Y}$ 因此我们仅需要计算 $\frac{\partial Y}{\partial Z_{1,1}}$ 。通过等式5.26，我们可以很容易的求出：

$$\frac{\partial Y}{\partial Z_{1,1}} = \begin{bmatrix} \sigma'(Z_{1,1}) & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (5.29)$$

现在结合等式5.28及5.29, 可得:

$$\frac{\partial L}{\partial Z_{1,1}} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial Z_{1,1}} \quad (5.30)$$

$$= \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix} \cdot \begin{bmatrix} \sigma'(Z_{1,1}) & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (5.31)$$

$$= \frac{\partial L}{\partial Y_{1,1}} \sigma'(Z_{1,1}) \quad (5.32)$$

我们可以重复该过程, 计算 $\frac{\partial L}{\partial Z}$ 的其他元素:

$$\frac{\partial L}{\partial Z_{1,2}} = \frac{\partial L}{\partial Y_{1,2}} \sigma'(Z_{1,2}) \quad (5.33)$$

$$\frac{\partial L}{\partial Z_{1,3}} = \frac{\partial L}{\partial Y_{1,3}} \sigma'(Z_{1,3}) \quad (5.34)$$

$$\frac{\partial L}{\partial Z_{2,1}} = \frac{\partial L}{\partial Y_{2,1}} \sigma'(Z_{2,1}) \quad (5.35)$$

$$\frac{\partial L}{\partial Z_{2,2}} = \frac{\partial L}{\partial Y_{2,2}} \sigma'(Z_{2,2}) \quad (5.36)$$

$$\frac{\partial L}{\partial Z_{2,3}} = \frac{\partial L}{\partial Y_{2,3}} \sigma'(Z_{2,3}) \quad (5.37)$$

考虑后续表达方便, 这里定义如下变量:

$$\mathbf{Z}' = \begin{bmatrix} \sigma'(Z_{1,1}) & \sigma'(Z_{1,2}) & \sigma'(Z_{1,3}) \\ \sigma'(Z_{2,1}) & \sigma'(Z_{2,2}) & \sigma'(Z_{2,3}) \end{bmatrix}$$

最后结合上述等式, 我们可以得到如下 $\frac{\partial L}{\partial Z}$ 的表达式:

$$\frac{\partial L}{\partial Z} = \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} \sigma'(Z_{1,1}) & \frac{\partial L}{\partial Y_{1,2}} \sigma'(Z_{1,2}) & \frac{\partial L}{\partial Y_{1,3}} \sigma'(Z_{1,3}) \\ \frac{\partial L}{\partial Y_{2,1}} \sigma'(Z_{2,1}) & \frac{\partial L}{\partial Y_{2,2}} \sigma'(Z_{2,2}) & \frac{\partial L}{\partial Y_{2,3}} \sigma'(Z_{2,3}) \end{bmatrix} \quad (5.38)$$

$$= \begin{bmatrix} \frac{\partial L}{\partial Y_{1,1}} & \frac{\partial L}{\partial Y_{1,2}} & \frac{\partial L}{\partial Y_{1,3}} \\ \frac{\partial L}{\partial Y_{2,1}} & \frac{\partial L}{\partial Y_{2,2}} & \frac{\partial L}{\partial Y_{2,3}} \end{bmatrix} \odot \begin{bmatrix} \sigma'(Z_{1,1}) & \sigma'(Z_{1,2}) & \sigma'(Z_{1,3}) \\ \sigma'(Z_{2,1}) & \sigma'(Z_{2,2}) & \sigma'(Z_{2,3}) \end{bmatrix} \quad (5.39)$$

$$= \boxed{\frac{\partial L}{\partial Y} \odot \mathbf{Z}'} \quad (5.40)$$

因此 $\frac{\partial L}{\partial Z} \in \mathbb{R}^{N \times M}$, 在求得 $\frac{\partial L}{\partial Z}$ 的基础上, 求解 $\frac{\partial L}{\partial X}$ 及 $\frac{\partial L}{\partial W}$ 简化为如下形式:

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial X} \quad \frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \frac{\partial Z}{\partial W} \quad (5.41)$$

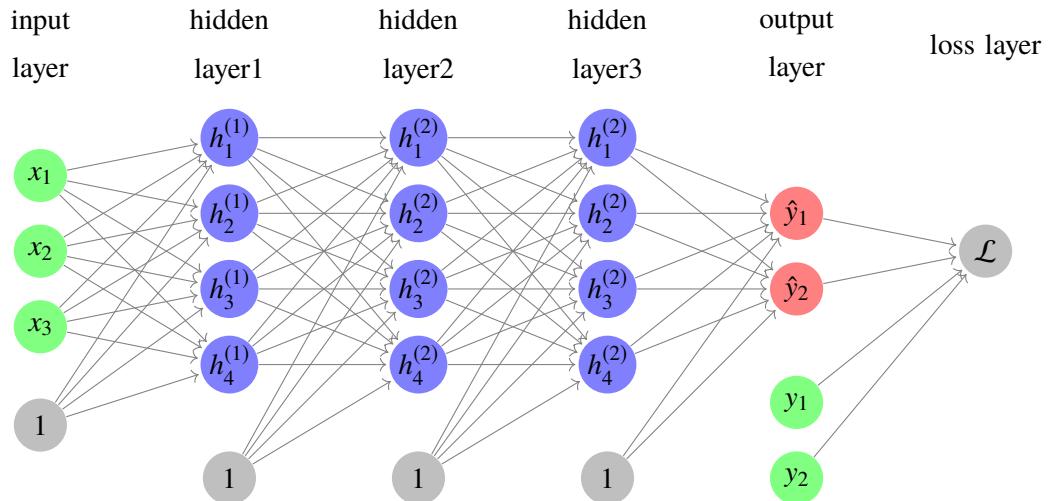
因此后续的计算和线性函数节点一致，这里不再赘述。因此有：

$$\frac{\partial L}{\partial X} = \boxed{\left(\frac{\partial L}{\partial Y} \odot Z' \right) W^\top} \quad (5.42)$$

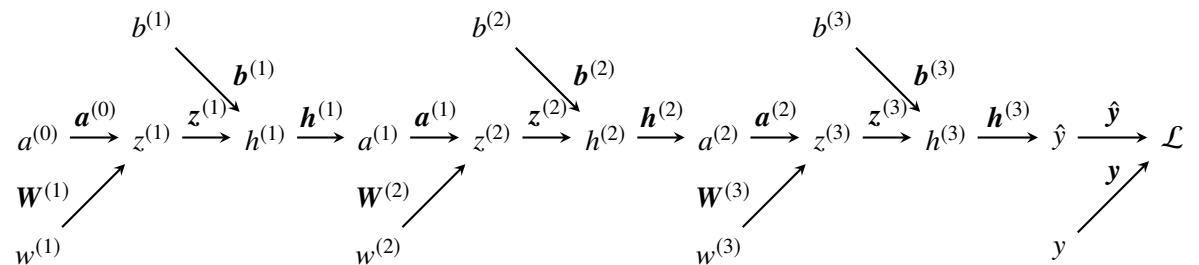
$$\frac{\partial L}{\partial W} = \boxed{X^\top \left(\frac{\partial L}{\partial Y} \odot Z' \right)} \quad (5.43)$$

5.5 梯度爆炸或消散

考虑如下神经网络架构



假设每一层的激活函数为 σ 。其对应的计算图如下所示：



上述计算图中， $a^{(0)}$ 表示输入层， $a^{(0)}$ 表示输入向量。因此， $a^{(i-1)} \rightarrow a^{(i)}$ 表示神经网络中第 i 层。基于计算图和上一小节介绍的反向传播算法，假设完成前向过程计算，反向过程计算如下：

$$\bar{\mathcal{L}} = 1 \quad (5.44)$$

$$\bar{y} = \bar{\mathcal{L}} \frac{\partial \mathcal{L}}{\partial \hat{y}} \quad (\text{令 } \hat{y}' = \frac{\partial \mathcal{L}}{\partial \hat{y}}) \quad (5.45)$$

$$= \bar{\mathcal{L}} \hat{y}' \quad (5.46)$$

$$(5.47)$$

$$\overline{h^{(3)}} = \overline{\hat{y}} \frac{\partial \hat{y}}{\partial h^{(3)}} \quad (\text{令 } \mathbf{h}^{(3)'} = \frac{\partial \hat{y}}{\partial h^{(3)}}) \quad (5.48)$$

$$= \overline{\hat{y}} \odot \mathbf{h}^{(3)'} \quad (5.49)$$

$$\overline{b^{(3)}} = \overline{h^{(3)}} \frac{\partial h^{(3)}}{\partial b^{(3)}} \quad (5.50)$$

$$= \overline{h^{(3)}} \quad (5.51)$$

$$\overline{z^{(3)}} = \overline{h^{(3)}} \frac{\partial h^{(3)}}{\partial z^{(3)}} \quad (5.52)$$

$$= \overline{h^{(3)}} \quad (5.53)$$

$$\overline{w^{(3)}} = \overline{z^{(3)}} \frac{\partial z^{(3)}}{\partial w^{(3)}} \quad (5.54)$$

$$= \overline{z^{(3)}} \mathbf{a}^{(2)\top} \quad (5.55)$$

$$\overline{a^{(2)}} = \overline{z^{(3)}} \frac{\partial z^{(3)}}{\partial a^{(2)}} \quad (5.56)$$

$$= \mathbf{W}^{(3)\top} \overline{z^{(3)}} \quad (5.57)$$

$$\overline{h^{(2)}} = \overline{a^{(2)}} \frac{\partial a^{(2)}}{\partial h^{(2)}} \quad (\text{令 } \mathbf{h}^{(2)'} = \frac{\partial a^{(2)}}{\partial h^{(2)}}) \quad (5.58)$$

$$= \overline{a^{(2)}} \odot \mathbf{h}^{(2)'} \quad (5.59)$$

$$\overline{b^{(2)}} = \overline{h^{(2)}} \frac{\partial h^{(2)}}{\partial b^{(2)}} \quad (5.60)$$

$$= \overline{h^{(2)}} \quad (5.61)$$

$$\overline{z^{(2)}} = \overline{h^{(2)}} \frac{\partial h^{(2)}}{\partial z^{(2)}} \quad (5.62)$$

$$= \overline{h^{(2)}} \quad (5.63)$$

$$\overline{w^{(2)}} = \overline{z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(2)}} \quad (5.64)$$

$$= \overline{z^{(2)}} \mathbf{a}^{(1)\top} \quad (5.65)$$

$$\overline{a^{(1)}} = \overline{z^{(2)}} \frac{\partial z^{(2)}}{\partial a^{(1)}} \quad (5.66)$$

$$= \mathbf{W}^{(2)\top} \overline{z^{(2)}} \quad (5.67)$$

$$\overline{h^{(1)}} = \overline{a^{(1)}} \frac{\partial a^{(1)}}{\partial h^{(1)}} \quad (\text{令 } \mathbf{h}^{(1)'} = \frac{\partial a^{(1)}}{\partial h^{(1)}}) \quad (5.68)$$

$$= \overline{a^{(1)}} \odot \mathbf{h}^{(1)'} \quad (5.69)$$

$$\overline{b^{(1)}} = \overline{h^{(1)}} \frac{\partial h^{(1)}}{\partial b^{(1)}} \quad (5.70)$$

$$= \overline{h^{(1)}} \quad (5.71)$$

$$\overline{z^{(1)}} = \overline{h^{(1)}} \frac{\partial h^{(1)}}{\partial z^{(1)}} \quad (5.72)$$

$$= \overline{h^{(1)}} \quad (5.73)$$

$$(5.74)$$

$$\overline{w^{(1)}} = \overline{z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}} \quad (5.75)$$

$$= \overline{z^{(1)}} \mathbf{a}^{(0)\top} \quad (5.76)$$

综合如上等式，可得 $\frac{\partial L}{\partial w^{(1)}}$ 计算公式如下：

$$\frac{\partial L}{\partial w^{(1)}} = \left(\left(\mathbf{W}^{(2)\top} \left((\mathbf{W}^{(3)\top} (\hat{\mathbf{y}}' \odot \mathbf{h}^{(3)'})) \odot \mathbf{h}^{(2)'} \right) \right) \odot \mathbf{h}^{(1)'} \right) \mathbf{a}^{(0)\top} \quad (5.77)$$

令 $\mathbf{v}^{(3)} = \hat{\mathbf{y}}' \odot \mathbf{h}^{(3)'} = \text{diag}(\mathbf{h}^{(3)'} \hat{\mathbf{y}}')$ ，因此有：

$$\frac{\partial L}{\partial w^{(1)}} = \left(\left(\mathbf{W}^{(2)\top} \left((\mathbf{W}^{(3)\top} \mathbf{v}^{(3)}) \odot \mathbf{h}^{(2)'} \right) \right) \odot \mathbf{h}^{(1)'} \right) \mathbf{a}^{(0)\top} \quad (5.78)$$

令 $\mathbf{v}^{(2)} = (\mathbf{W}^{(3)\top} \mathbf{v}^{(3)}) \odot \mathbf{h}^{(2)'} = \text{diag}(\mathbf{h}^{(2)}) \mathbf{W}^{(3)\top} \mathbf{v}^{(3)}$ ，因此有：

$$\frac{\partial L}{\partial w^{(1)}} = \left(\left(\mathbf{W}^{(2)\top} \mathbf{v}^{(2)} \right) \odot \mathbf{h}^{(1)'} \right) \mathbf{a}^{(0)\top} \quad (5.79)$$

令 $\mathbf{v}^{(1)} = (\mathbf{W}^{(2)\top} \mathbf{v}^{(2)}) \odot \mathbf{h}^{(1)'} = \text{diag}(\mathbf{h}^{(1)}) \mathbf{W}^{(2)\top} \mathbf{v}^{(2)}$ ，因此有：

$$\frac{\partial L}{\partial w^{(1)}} = \mathbf{v}^{(1)} \mathbf{a}^{(0)\top} \quad (5.80)$$

将上述公式展开后即为：

$$\frac{\partial L}{\partial w^{(1)}} = \left(\text{diag}(\mathbf{h}^{(1)}) \mathbf{W}^{(2)\top} \text{diag}(\mathbf{h}^{(2)}) \mathbf{W}^{(3)\top} \text{diag}(\mathbf{h}^{(3)'} \hat{\mathbf{y}}') \right) \mathbf{a}^{(0)\top} \quad (5.81)$$

对上述等式进行一般化推广，若神经网络隐藏层为 k 层，则有：

$$\frac{\partial L}{\partial w^{(1)}} = \left(\text{diag}(\mathbf{h}^{(1)}) \mathbf{W}^{(2)\top} \text{diag}(\mathbf{h}^{(2)}) \dots \mathbf{W}^{(k)\top} \text{diag}(\mathbf{h}^{(k)'} \hat{\mathbf{y}}') \right) \mathbf{a}^{(0)\top} \quad (5.82)$$

假设 $\|\text{diag}(\mathbf{h}^{(i)'})\| = \beta$, $\|\mathbf{W}^{(i)\top}\| = \gamma$ 。其中 β 为激活函数对应的对角矩阵的最大的奇异值， γ 为权重矩阵最大的奇异值。矩阵范数的定义可以参考第二章矩阵范数部分内容，由矩阵范数不等式可知：

$$\|\text{diag}(\mathbf{h}^{(i)'}) \mathbf{W}^{(i)\top}\| \leq \|\text{diag}(\mathbf{h}^{(i)'})\| \|\mathbf{W}^{(i)\top}\| = \beta \gamma \quad (5.83)$$

因此有：

$$\frac{\partial L}{\partial w^{(1)}} \leq \left(\beta (\beta \gamma)^{k-1} \right) \hat{\mathbf{y}}' \mathbf{a}^{(0)\top} \quad (5.84)$$

从上述公式可见若 $\beta \gamma > 1$ ，随着网络的加深，即 k 值变大，将出现指数增加，此时出现梯度爆炸现象，若 $\beta \gamma < 1$ ，此时出现梯度消散现象。要保持输出层梯度 $\hat{\mathbf{y}}'$ 稳态传输，需

要确保 $\beta\gamma = 1$, 要达到该效果可以从如下两个途径来解决:

- 分别优化权值初始化和激活函数, 确保其对应的矩阵的奇异值最大值为 1.
- 优化网络结构, 譬如循环神经网络中将 Elman network 替换为 LSTM

第七章我们将考察不同的权值初始化策略, 及其如何解决梯度爆炸和消散问题。

第6章 激活函数

本章回顾用于隐藏层的不同类型的激活函数及其特性，作为隐藏层激活函数，常希望这类函数函数具有非线性特性。

6.1 Sigmoid

Sigmoid 函数是二分类领域用的最多一类函数，公式如下：

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (6.1)$$

如下是该函数的图形化描述：

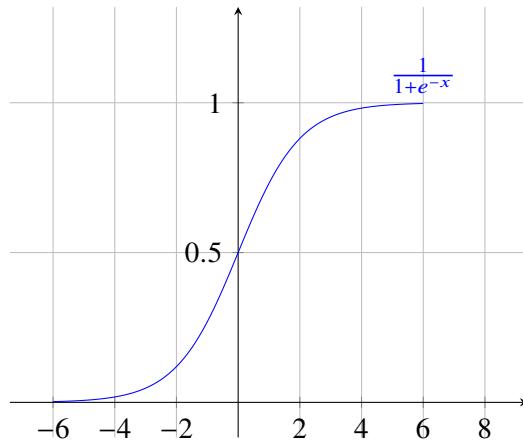


图 6.1: 函数 $\sigma(x)$ 的曲线

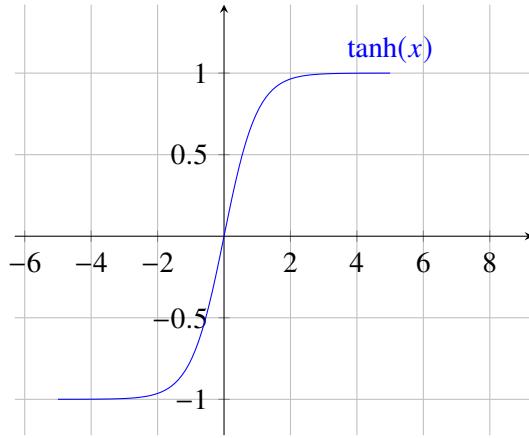
从上图可见函数 $\sigma(x)$ 值域为 $[0, 1]$ ，且非零对称，当 $|x|$ 大于某个值之后，函数 $\sigma(x)$ 进入饱和区，对应的梯度趋于 0。由于 $\sigma(x)$ 存在饱和区，且存在计算开销较高的指数运算，因此目前深度神经网络中较少将该函数作为隐藏层激活函数应用，在 LSTM 循环神经网络中用于进行门控。

6.2 Tanh

Tanh 函数公式如下：

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (6.2)$$

如下是该函数的图形化描述：

图 6.2: 函数 $\tanh(x)$ 的曲线

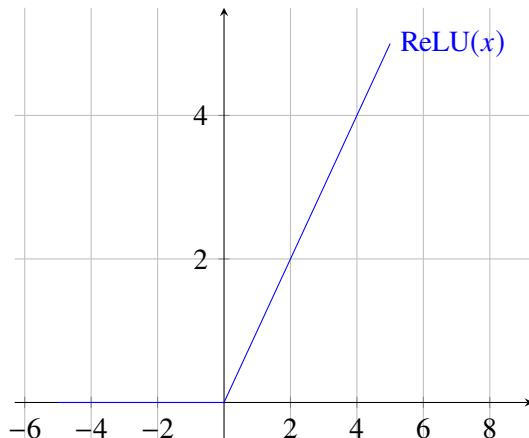
从上图可见函数 $\tanh(x)$ 的值域为 $[-1, 1]$ ，且关于零对称，和 $\sigma(x)$ 函数类似，当 $|x|$ 大于某个值之后，函数 $\tanh(x)$ 进入饱和区，对应的梯度趋于 0，且存在计算开销较高的指数运算，因此目前深度神经网络中较少将该函数作为隐藏层激活函数应用，在 LSTM 循环神经网络中用于进行 cell 更新和 cell 输出变换。

6.3 ReLU

ReLU (Rectified Linear Unit) 首先被 Vinod Nair 及 Geoffrey E. Hinton¹ 应用于 Restricted Boltzmann Machines，函数公式如下：

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (6.3)$$

如下是该函数的图形化描述：

图 6.3: 函数 $\text{ReLU}(x)$ 的曲线

从上图可见 $\text{ReLU}(x)$ 函数不像前述两类激活函数存在饱和区，且没有计算开销较高的指数运算，且当 $x \geq 0$ 时对应的梯度为 1，具有良好的梯度反向传播能力。但是 $\text{ReLU}(x)$ 函数

¹Rectified Linear Units Improve Restricted Boltzmann Machines

数存在的问题是，当 $x < 0$ 时，其输出为 0，且梯度为 0，此时激活函数对应的神经元进入死区，且永远无法跳出。

6.4 Leaky ReLU

为了解决 ReLU 激活函数当 $x < 0$ 时进入死区的问题，Andrew L. Maas 等人²提出 Leaky ReLU (LReLU) 激活函数，函数公式如下：

$$\text{LReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \frac{x}{a} & \text{if } x < 0 \end{cases} \quad (6.4)$$

上述公式中 a 为超参，定义域为 $(1, +\infty)$ ，Andrew L. Maas 等人建议 $a = 100$ 。如下是该函数的图形化描述（注：下图中 $a = 20$ ）：

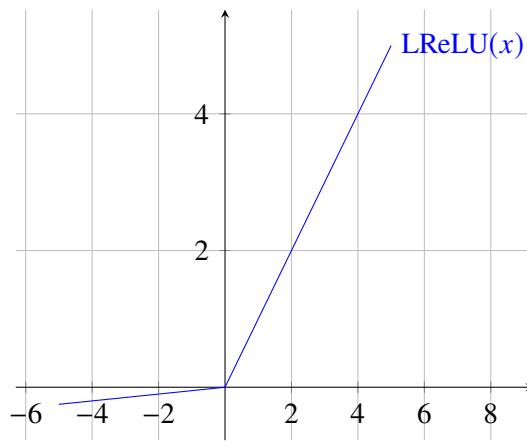


图 6.4: 函数 $\text{LReLU}(x)$ 的曲线

从上图可见 $\text{LReLU}(x)$ 函数解决了 $\text{ReLU}(x)$ 函数当 $x < 0$ 时进入死区的问题，同时良好地保留了 ReLU 函数的非线性特性。

6.5 PReLU

PReLU(Parametric Rectified Linear Unit) 和 LReLU 有所不同的是 Kaiming He 等人³在 ReLU 的基础上在定义域负区间引入可变参数 a ，公式如下：

$$f(x_i) = \begin{cases} x_i & \text{if } x_i \geq 0 \\ a_i x_i & \text{if } x_i < 0 \end{cases} \quad (6.5)$$

²Rectifier Nonlinearities Improve Neural Network Acoustic Models

³Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

考虑到其应用场景为 CNN 卷积层激活函数，因此 a_i 参数个数为输出通道数， a_i 为待学习参数，初始值设置为 0.25，并通过 momentum 方法进行更新。假设 \mathcal{L} 表示损失函数：

$$\frac{\partial \mathcal{L}}{\partial a_i} = \sum_{x_i} \frac{\partial \mathcal{L}}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial a_i} \quad (6.6)$$

$\frac{\partial \mathcal{L}}{\partial f(x_i)}$ 表示深度网络反向传播至当前层的梯度， $\frac{\partial f(x_i)}{\partial a_i}$ 定义如下：

$$\frac{\partial f(x_i)}{\partial a_i} = \begin{cases} 0, & \text{if } x_i > 0 \\ x_i, & \text{if } x_i \leq 0 \end{cases} \quad (6.7)$$

由 memontum 方法可知 a_i 的更新步长如下：

$$\Delta a_i = \mu \Delta a_i + \epsilon \frac{\partial \mathcal{L}}{\partial a_i} \quad (6.8)$$

其中 μ 为动量， ϵ 为学习率。

6.6 ELU

ELU(Exponential Linear Unit) 由 Djork-Arné Clevert 等人⁴提出。ELU 公式如下：

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases} \quad (6.9)$$

上述公式中 α 为超参，和 LReLU 及 PReLU 有所不同的是，当 $x \leq 0$ 时，随着 x 值的变小， $f(x)$ 出现饱和，因此 ELU 对输入中存在的特性进行了表示，对缺失的特性不做定量表示。根据 Djork-Arné Clevert 等人实验当网络深度超过 5 层时，ELU 相对于 ReLU 和 LReLU 学习速度更快，且有更好的泛化能力。

如下是该函数的图形化描述（注：下图中 $\alpha = 1$ ）：

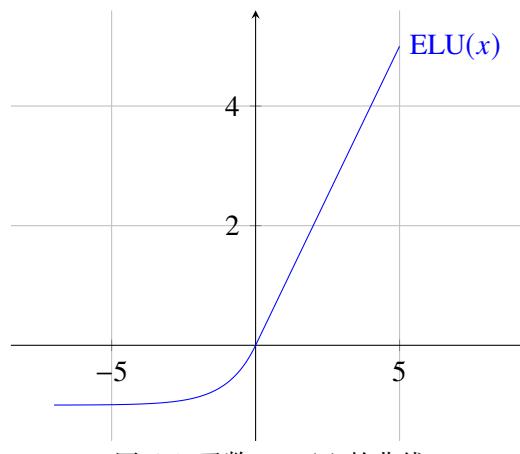


图 6.5: 函数 $\text{ELU}(x)$ 的曲线

⁴Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)

6.7 GELU

GELU(Gaussian Error Linear Unit)由 Dan Hendrycks 和 Kevin Gimpel⁵提出。GELU 是 ReLU 的函数的可导版本，同时类似于 ELU 在负区间段像具有良好的饱和特性。GELU 背后的思想是希望和 ReLU 类似在定义域正区间段类似于乘以 1，而负区间段乘以 0，达到类似于 dropout 的特性。因此 GELU 通过对输入 x 乘以 $m \sim \text{Bernoulli}(\Phi(x))$, m 服从 Bernoulli 分布⁶其取值为 1 的概率定义为 $\Phi(x)$ ，其中 $\Phi(x) = P(X \leq x)$, $X \sim \mathcal{N}(0, 1)$ 为标准整体分布的概率累积函数，因此有：

$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x). \quad (6.10)$$

上述公式可以通过如下公式近似：

$$\text{GELU}(x) = 0.5x \left(1 + \tanh \left[\sqrt{2/\pi} (x + 0.044715x^3) \right] \right) \quad (6.11)$$

或者

$$\text{GELU}(x) = x\sigma(1.702x) \quad (6.12)$$

Jacob Devlin 等人⁷在 BERT 中采用了近似公式6.11，如下是采用该近似公式对应的 GELU 的图形化描述：

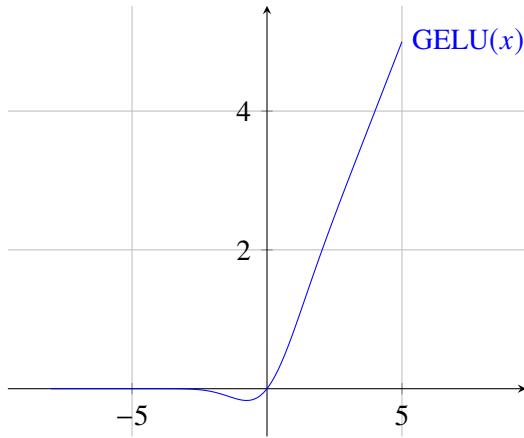


图 6.6: 函数 $\text{GELU}(x)$ 的曲线

⁵Gaussian Error Linear Units (GELUs)

⁶https://en.wikipedia.org/wiki/Bernoulli_distribution

⁷BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

第 7 章 参数优化

本章重点回顾不同参数初始化方法及其对神经网络的影响，在参数初始化基础上我们进一步回顾了 Interval Covariate Shift 问题，及两类不同归一化方法如何减小该问题，并加速网络训练。

7.1 零值初始化

神经网络权值初始化中一个常见的缺陷是将所有的权值初始化为零，假设神经网络的第 i 层为满足如下方程：

$$\begin{aligned} z^{[l]} &= \mathbf{W}^{[l]} \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \\ \mathbf{a}^{[l]} &= \tanh(z^{[l]}) \end{aligned}$$

若将所有权值均初始化为零，则 $z^{[l]}$ 所有维度均为零，因此 $\mathbf{a}^{[l]}$ 所有维度均为零。由反向传播求导可知， $\mathbf{W}^{[l]}$ 和 $\mathbf{b}^{[l]}$ 所有维度的导数均一致，因此反向传播更新后所有维度权值均一致，此时输出层 $\mathbf{a}^{[l]} \in \mathbb{R}^{n^{[l]}}$ 退化为一个神经元。

7.2 随机初始化

考虑到零值初始化带来的问题是输出多个神经元退化一个神经元，常见的做法是采用均值为 0，方差为 1 的正态分布生成初始化权值 \mathbf{W} ，并对权值乘以某个较小的常数（譬如 0.01）。偏执项 $\mathbf{b}^{[l]}$ 一般初始化为 0，原因神经网络反向传播过程中偏执项不会参与计算，且将偏执项初始化为零对变换系统初始处于线性域有帮助，可以加快梯度传播，加快网络学习。譬如如下所示：

$$\mathbf{W}^{[l]} = 0.01 * \text{np.random.randn}(n^{[l]}, n^{[l-1]}) \quad (7.1)$$

通过上述初始化方法，输出层 $\mathbf{a}^{[l]}$ 对应的不同神经元 i 对应的权值 $\mathbf{W}_{i,*}^{[l]}$ 被初始化为不同的较小值，这样解决了零值初始化存在的神经网络退化问题。但是较小的权值并不能保证神经网络一定可以工作的很好。由神经网络一章反向传播算法过程可知，若 $\mathbf{W}^{[l]}$ 较小，反向传播过程中梯度将被极大程度衰弱，因此导致网络无法进行学习。

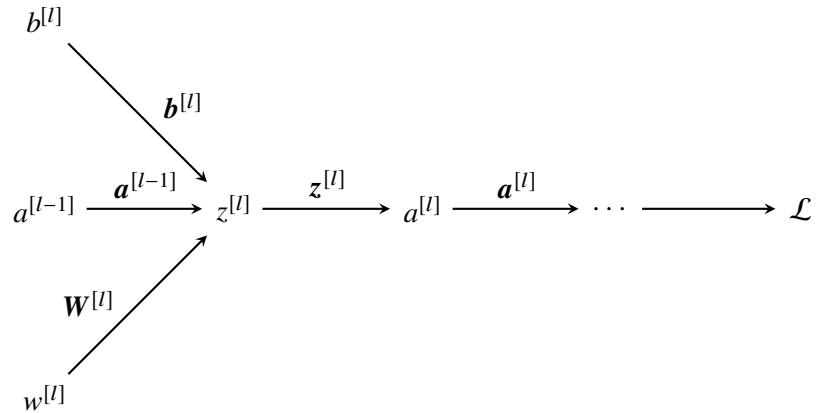
7.3 Xavier 初始化

考虑到随机初始化也无法让神经网络可以较好地学习，Xavier Glorot 及 Yoshua Bengio¹给出采用 \tanh 激活函数时对应的权值初始化方法。假设依然如下的全连接层表示：

$$z^{[l]} = \mathbf{W}^{[l]} \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \quad (7.2)$$

$$\mathbf{a}^{[l]} = \tanh(z^{[l]}) \quad (7.3)$$

考虑表达简洁，这里允许计算图的某个节点可以有两个以上的输入，其对应的计算图如下：



考虑到系统稳定性，偏执项 $\mathbf{b}^{[l]}$ 初始化为 0，其均值和方差均为 0，输入值 $\mathbf{a}^{[l-1]}$ 满足均值为 0，方差为 1。因此我们希望权值矩阵 $\mathbf{W}^{[l]}$ 的奇异值最大值为 1，此时 $z^{[l]}$ 的均值和方差等于 $\mathbf{a}^{[l-1]}$ ，考虑激活函数为 $f(\mathbf{x}) = \tanh(\mathbf{x})$ ，此时 $z^{[l]}$ 处于激活函数的线性区间，其对应的导数为 1，此时函数对应的一阶近似为线性形式，即：

$$f'(z_k^{[l]}) = [1, \dots, 1]^\top \quad (7.4)$$

$$f(z^{[l]}) \approx f(z_k^{[l]}) + f'(z_k^{[l]})^\top (z^{[l]} - z_k^{[l]}) \quad (7.5)$$

$$= f(z_k^{[l]}) + (z^{[l]} - z_k^{[l]}) \quad (7.6)$$

¹Understanding the difficulty of training deep feedforward neural networks

因此有：

$$\mathbf{a}^{[l]} = f(z_k^{[l]}) + (z^{[l]} - z_k^{[l]}) \quad (7.7)$$

$$\Rightarrow \text{Var}[\mathbf{a}^{[l]}] = \text{Var}[f(z_k^{[l]}) + (z^{[l]} - z_k^{[l]})] \quad (7.8)$$

$$= \text{Var}\left[\underbrace{(f(z_k^{[l]}) - z_k^{[l]})}_{\text{constant}} + z^{[l]}\right] \quad (7.9)$$

$$= \text{Var}[z^{[l]}] \quad (7.10)$$

$$= \text{Var}[\mathbf{a}^{[l-1]}] \quad (\mathbf{a}^{[l-1]} \text{ 和 } \mathbf{b}^{[l]} \text{ 彼此线性无关}) \quad (7.11)$$

可见当权值矩阵 $\mathbf{W}^{[l]}$ 的奇异值最大值为 1，偏执项 $\mathbf{b}^{[l]}$ 初始化为 0，及输入项 $\mathbf{a}^{[l-1]}$ 满足矩阵为 0，方差为 1 时，有：

$$\text{Var}[\mathbf{a}^{[l]}] = \text{Var}[z^{[l]}] \quad (7.12)$$

$$\text{Var}[\mathbf{a}^{[l]}] = \text{Var}[\mathbf{a}^{[l-1]}] \quad (7.13)$$

下面我们将具体看如何初始化权值矩阵 $\mathbf{W}^{[l]}$ 使得上述等式成立。由 $z^{[l]}$ 公式可知：

$$z^{[l]} = \mathbf{W}^{[l]} \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} = \left[z_1^{[l]}, z_2^{[l]}, \dots, z_{n^{[l]}}^{[l]} \right]^T \quad (7.14)$$

其中：

$$z_k^{[l]} = \sum_{j=1}^{n^{[l-1]}} W_{k,j}^{[l]} a_j^{[l-1]} + b_k^{[l]} \quad (7.15)$$

考虑到偏执项 $\mathbf{b}^{[l]}$ 的均值和方差为 0，且和 $\mathbf{W}^{[l]} \mathbf{a}^{[l-1]}$ 线性无关，因此下面方差中的推导略去该项。因此有：

$$\text{Var}[a_k^{[l]}] = \text{Var}[z_k^{[l]}] = \text{Var}\left[\sum_{j=1}^{n^{[l-1]}} W_{k,j}^{[l]} a_j^{[l-1]}\right] \quad (7.16)$$

假设权重矩阵 $\mathbf{W}^{[l]}$ 及输入向量 $\mathbf{a}^{[l-1]}$ 满足如下假设：

- 假设 1：权重矩阵中的每个变量期望为 0 且彼此独立同分布
- 假设 2：输入向量中的每个变量期望为 0 且彼此独立同分布
- 假设 3：权重矩阵中的变量和输入向量中的变量彼此线性无关

因此有：

$$\text{Var}[a_k^{[l]}] = \sum_{j=1}^{n^{[l-1]}} \text{Var}[W_{k,j}^{[l]} a_j^{[l-1]}] \quad (7.17)$$

由第三章概率论，两个相互独立的随机变量的联合方差可知

$$\text{Var}[XY] = E[X]^2 \text{Var}[Y] + \text{Var}[X]E[Y]^2 + \text{Var}[X]\text{Var}[Y] \quad (7.18)$$

令 $X = W_{k,j}^{[l]}$ 及 $Y = a_j^{[l-1]}$, 则有:

$$\begin{aligned} \text{Var}\left[W_{k,j}^{[l]} a_j^{[l-1]}\right] &= E\left[W_{k,j}^{[l]}\right]^2 \text{Var}\left[a_j^{[l-1]}\right] \\ &\quad + \text{Var}\left[W_{k,j}^{[l]}\right] E\left[a_j^{[l-1]}\right]^2 + \text{Var}\left[W_{k,j}^{[l]}\right] \text{Var}\left[a_j^{[l-1]}\right] \end{aligned} \quad (7.19)$$

考慮假设 1 滿足 $E\left[W_{k,j}^{[l]}\right]^2 = 0$, 假設 2 滿足 $E\left[a_j^{[l-1]}\right]^2 = 0$, 因此有:

$$\text{Var}\left[z_k^{[l]}\right] = \sum_{j=1}^{n^{[l-1]}} \text{Var}\left[W_{k,j}^{[l]}\right] \text{Var}\left[a_j^{[l-1]}\right] \quad (7.20)$$

由假设 1 可知:

$$\text{Var}\left[W_{k,j}^{[l]}\right] = \text{Var}\left[W_{1,1}^{[l]}\right] = \text{Var}\left[W_{2,2}^{[l]}\right] = \cdots = \text{Var}\left[\mathbf{W}^{[l]}\right] \quad (7.21)$$

有假设 2 可知:

$$\text{Var}\left[a_j^{[l-1]}\right] = \text{Var}\left[a_1^{[l-1]}\right] = \text{Var}\left[a_2^{[l-1]}\right] = \cdots = \text{Var}\left[\mathbf{a}^{[l-1]}\right] \quad (7.22)$$

因此有:

$$\text{Var}\left[z_k^{[l]}\right] = \sum_{j=1}^{n^{[l-1]}} \text{Var}\left[\mathbf{W}^{[l]}\right] \text{Var}\left[\mathbf{a}^{[l-1]}\right] = n^{[l-1]} \text{Var}\left[\mathbf{W}^{[l]}\right] \text{Var}\left[\mathbf{a}^{[l-1]}\right] \quad (7.23)$$

考慮公式7.12有:

$$\text{Var}\left[\mathbf{a}^{[l]}\right] = n^{[l-1]} \text{Var}\left[\mathbf{W}^{[l]}\right] \text{Var}\left[\mathbf{a}^{[l-1]}\right] \quad (7.24)$$

结合公式7.13有:

$$\text{Var}\left[\mathbf{W}^{[l]}\right] = \frac{1}{n^{[l-1]}} \quad (7.25)$$

現在考慮反向传播时梯度传播对权值 $\mathbf{W}^{[l]}$ 初始化的要求, 由计算图可知:

$$\frac{\partial \mathcal{L}}{\partial a^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial a^{[l]}} \frac{\partial a^{[l]}}{\partial z^{[l]}} \frac{\partial z^{[l]}}{\partial a^{[l-1]}} \quad (7.26)$$

令：

$$\mathbf{g}^{[l-1]} = \frac{\partial \mathcal{L}}{\partial a^{[l-1]}}, \quad \mathbf{g}^{[l]} = \frac{\partial \mathcal{L}}{\partial a^{[l]}} \quad (7.27)$$

因此有：

$$\mathbf{g}^{[l-1]} = \mathbf{W}^{[l]\top} \text{diag}\left(a_1^{[l]\top}, \dots, a_{n^{[l]}}^{[l]\top}\right) \mathbf{g}^{[l]} \quad (7.28)$$

考虑激活函数为 $f(\mathbf{x}) = \tanh(\mathbf{x})$, 此时 $z^{[1]}$ 处于激活函数的线性区间, 其对应的导数为 1, 因此有：

$$\mathbf{g}^{[l-1]} = \mathbf{W}^{[l]\top} \mathbf{g}^{[l]} \quad (7.29)$$

其中：

$$g_k^{[l-1]} = \sum_{j=1}^{n^{[l]}} W_{j,k}^{[l]} g_j^{[l]} \quad (7.30)$$

因此有：

$$\text{Var}\left[g_k^{[l-1]}\right] = \text{Var}\left[\sum_{j=1}^{n^{[l]}} W_{j,k}^{[l]} g_j^{[l]}\right] \quad (7.31)$$

考虑权值矩阵 $\mathbf{W}^{[l]}$ 及输入向量 $\mathbf{a}^{[l-1]}$ 满足假设 1-3, 因此有如下假设成立：

- 假设 4：梯度向量中每个变量期望为 0, 且彼此独立同分布
- 假设 5：权值矩阵中的变量和梯度向量中的变量彼此线性无关

因此和之前的推导类似, 有::

$$\text{Var}\left[\mathbf{g}^{[l-1]}\right] = n^{[l]} \text{Var}\left[\mathbf{W}^{[l]}\right] \text{Var}\left[\mathbf{g}^{[l]}\right] \quad (7.32)$$

考虑线性系统的稳定性, 即要求 \mathbf{W} 的最大奇异值为 1, 此时有:

$$\text{Var}\left[\mathbf{g}^{[l-1]}\right] = \text{Var}\left[\mathbf{g}^{[l]}\right] \quad (7.33)$$

因此有:

$$\text{Var}\left[\mathbf{W}^{[l]}\right] = \frac{1}{n^{[l]}} \quad (7.34)$$

结合公式7.25及7.34, 可知权值矩阵 $\mathbf{W}^{[l]}$ 的方差可以设为两个公式的调和平均数, 即:

$$\text{Var}\left[\mathbf{W}^{[l]}\right] = \frac{2}{n^{[l-1]} + n^{[l]}} \quad (7.35)$$

上述公式即为 **Xavier** 初始化方法, 实际中常常采用均值为 0, 方差为 $\frac{2}{n^{[l-1]} + n^{[l]}}$ 的正态分布进行权值初始化。

7.4 He 初始化

当激活函数不是 \tanh 而是 ReLU 时，Kaiming He 等人²指出 **Xavier** 初始化方法无法进行较好地工作，因此在 **Xavier** 初始化方法的基础之上，其提出了 **He** 初始化方法。下面让我们来看看采用 ReLU 激活函数时如何进行权重矩阵初始化，假设采用如下的卷积层表示：

$$z^{[l]} = \mathbf{W}^{[l]} \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \quad (7.36)$$

$$\mathbf{a}^{[l]} = \text{ReLU}(z^{[l]}) \quad (7.37)$$

上述公式中， $\mathbf{a}^{[l-1]} \in \mathbb{R}^{k^2 c}$ 的向量，表示 c 个输入通道中 $k \times k$ 的卷积核， k 为卷积核的 filter size。令 $n^{[l-1]} = k^2 c$ 表示某个相应的连接数，则 $\mathbf{W}^{[l]} \in \mathbb{R}^{d \times n^{[l-1]}}$ ，其中 d 为卷积核的数量， $\mathbf{W}^{[l]}$ 的每一行表示一个卷积核的权重。 \mathbf{b} 为向量偏执， y 为输出中某个像素的响应。计算图和上一小节类似，这里不再重复给出。和 Xavier 初始化类似，假设权重矩阵 $\mathbf{W}^{[l]}$ 及输入向量 $\mathbf{a}^{[l-1]}$ 满足如下假设：

- 假设 1：权重矩阵中的每个变量期望为 0 且彼此独立同分布
- 假设 2：输入向量中的每个变量期望为 0 且彼此独立同分布
- 假设 3：权重矩阵中的变量和输入向量中的变量彼此线性无关

因此和 Xavier 推导过程类似，存在如下等式：

$$\text{Var}[z^{[l]}] = n^{[l-1]} \text{Var}[\mathbf{W}^{[l]} \mathbf{a}^{[l-1]}] \quad (7.38)$$

和 Xavier 类似，令 $\mathbf{W}^{[l]}$ 的均值为 0，则上述独立变量乘积的方差为：

$$\text{Var}[z^{[l]}] = n^{[l-1]} \text{Var}[\mathbf{W}^{[l]}] E[\mathbf{a}^{[l-1]2}] \quad (7.39)$$

上述公式中 $E[\mathbf{a}^{[l-1]2}]$ 为 $\mathbf{a}^{[l-1]}$ 的平方的期望。值得注意的是除非 $\mathbf{a}^{[l-1]}$ 的均值为 0，否则 $E[\mathbf{a}^{[l-1]2}] \neq \text{Var}[\mathbf{a}^{[l-1]}]$ 。对于 ReLU 激活函数来说， $\mathbf{a}^{[l]} = \max(0, z^{[l]})$ ，因此其均值不为 0，因此若采用该激活函数，对应的权值初始化方法将和 Xavier 初始化方法有所不同。假设令 $\mathbf{W}^{[l-1]}$ 满足均值为 0 的对称分布，偏执项 $\mathbf{b}^{[l-1]}$ 为 0，则 $z^{[l-1]}$ 满足均值为 0 的对

²Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

称分布，因此有：

$$E\left[\mathbf{a}^{[l-1]2}\right] = \int \mathbf{a}^{[l-1]2} f(\mathbf{a}^{[l-1]}) d\mathbf{a}^{[l-1]} \quad (7.40)$$

$$= \frac{1}{2} \int \mathbf{z}^{[l-1]2} f(\mathbf{z}^{[l-1]}) d\mathbf{z}^{[l]} \quad (7.41)$$

$$= \frac{1}{2} E\left[\mathbf{z}^{[l-1]2}\right] \quad (7.42)$$

$$= \frac{1}{2} \left(E\left[\mathbf{z}^{[l-1]2}\right] - \underbrace{\overbrace{E\left[\mathbf{z}^{[l-1]}\right]}^2}_{=0} \right) \quad (7.43)$$

$$= \frac{1}{2} Var\left[\mathbf{z}^{[l-1]}\right] \quad (7.44)$$

因此有：

$$Var\left[\mathbf{z}^{[l]}\right] = \frac{1}{2} n^{[l-1]} Var\left[\mathbf{W}^{[l]}\right] Var\left[\mathbf{z}^{[l-1]}\right] \quad (7.45)$$

考虑线性系统的稳定性，即要求 \mathbf{W} 的最大奇异值为 1，此时有：

$$Var\left[\mathbf{z}^{[l]}\right] = Var\left[\mathbf{z}^{[l-1]}\right] \quad (7.46)$$

因此有：

$$Var\left[\mathbf{W}^{[l]}\right] = \frac{2}{n^{[l-1]}} \quad (7.47)$$

其中 $n^{[l-1]} = k^2 c$ 表示相应的连接数。

现在考虑反向传播时梯度传播对权值 $\hat{\mathbf{W}}^{[l]} \in \mathbb{R}^{c \times n^{[l]}}$ 初始化的要求，其中 $n^{[l]} = k^2 d$ ，因此 $\hat{\mathbf{W}}^{[l]}$ 可以由 $\mathbf{W}^{[l]}$ 变换得到。由计算图可知：

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{z}^{[l]}} \frac{\partial \mathbf{z}^{[l]}}{\partial \mathbf{a}^{[l-1]}} \quad (7.48)$$

令：

$$\mathbf{g}^{[l-1]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l-1]}}, \quad \mathbf{g}^{[l]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \quad (7.49)$$

因此有：

$$\mathbf{g}^{[l-1]} = \hat{\mathbf{W}}^{[l]} \text{diag}\left(\mathbf{a}_1^{[l]'}, \dots, \mathbf{a}_{n^{[l]}}'\right) \mathbf{g}^{[l]} \quad (7.50)$$

令：

$$\mathbf{y}^{[l-1]} = \text{diag}\left(\mathbf{a}_1^{[l]'}, \dots, \mathbf{a}_{n^{[l]}}'\right) \mathbf{g}^{[l]} \quad (7.51)$$

上述公式中 $a_k^{[l]'} \text{ 为 ReLU 激活函数的导数，考虑输入为对称分布，因此该激活函数取值}$

为 0 和 1 的概率一致。假设 $\mathbf{a}^{[l]}'$ 和 $\mathbf{g}^{[l]}$ 彼此相互独立。因此有：

$$E\left[\mathbf{y}^{[l-1]}\right] = \frac{1}{2}E\left[\mathbf{g}^{[l]}\right] = 0 \quad (7.52)$$

因此有：

$$E\left[(\mathbf{y}^{[l-1]})^2\right] = Var\left[\mathbf{y}^{[l-1]}\right] = \frac{1}{2}Var\left[\mathbf{g}^{[l]}\right] \quad (7.53)$$

因此有：

$$\begin{aligned} Var\left[\mathbf{g}^{[l-1]}\right] &= \hat{n}^{[l]}Var\left[\hat{\mathbf{W}}^{[l]}\right]Var\left[\mathbf{y}^{[l-1]}\right] \\ &= \frac{1}{2}n^{[l]}Var\left[\hat{\mathbf{W}}^{[l]}\right]Var\left[\mathbf{g}^{[l]}\right] \end{aligned} \quad (7.54)$$

考虑线性系统的稳定性，即要求 \mathbf{W} 的最大奇异值为 1，此时有：

$$Var\left[\mathbf{g}^{[l-1]}\right] = Var\left[\mathbf{g}^{[l]}\right] \quad (7.55)$$

因此有：

$$Var\left[\hat{\mathbf{W}}^{[l]}\right] = \frac{1}{n^{[l]}} \quad (7.56)$$

实际应用时，无论采用公式7.47还是公式7.56初始化权值矩阵 $\mathbf{W}^{[l]}$ 均可以。

7.5 批归一化

前几小节我们讨论了如何对权值矩阵进行初始化的方法，本节我们首先回顾深度网络中存在的 Internal Covariate Shift 问题，在此基础上进一步介绍 Sergey Ioffe 和 Christian Szegedy³如何通过 Batch Normalization 算法来解决该问题。

7.5.1 Internal Covariate Shift

多层神经网络中每一层输入的分布由于前一层参数的调整不停发生变化，当学习系统的输入分布发生变化时，该学习系统面临 covariate shift 问题。考虑如下网络结构：

$$\mathcal{L} = F_2(F_1(\mathbf{u}, \Theta_1), \Theta_2) \quad (7.57)$$

其中 F_1 及 F_2 表示任意变换，需要优化参数 Θ_1 及 Θ_2 以便最小化损失 \mathcal{L} 。优化参数 Θ_2 可以看出输入为 $\mathbf{x} = F_1(\mathbf{u}, \Theta_1)$ 如下子网络：

$$\mathcal{L} = F_2(\mathbf{x}, \Theta_2) \quad (7.58)$$

³Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

譬如如下是一次梯度下降参数优化过程：

$$\Theta_2 = \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^m \frac{\partial F_2(\mathbf{x}_i, \Theta_2)}{\partial \Theta_2} \quad (7.59)$$

其中批大小为 m , 学习率为 α 。输入 \mathbf{x} 的在训练和预测阶段的分布一致对子网络学习效率提升很重要, 因此固定 \mathbf{x} 的分布将带来帮助, 此时参数 Θ_2 无需不断调整来弥补输入 \mathbf{x} 分布发生的变化。

固定子网络输入的分布对其他层的网络同样有正向帮助。考虑某层子网络, 其激活函数为 sigmoid 函数:

$$z = g(\mathbf{W}\mathbf{u} + \mathbf{b}) \quad (7.60)$$

上述子网络的输入为 \mathbf{u} , 参数矩阵 \mathbf{W} 及偏执向量 \mathbf{b} 为当前子网络需要学习的参数。其中:

$$g(x) = \frac{1}{1 + \exp(-x)}. \quad (7.61)$$

当 $|x|$ 增加时, 导数 $g'(x)$ 将趋向于 0, 这意味着对于 $\mathbf{x} = \mathbf{W}\mathbf{u} + \mathbf{b}$ 的所有维度除了那些拥有较小绝对值的维度, 用于更新 \mathbf{u} 的梯度将消失, 此时训练将变得缓慢。由于 \mathbf{x} 受到参数 \mathbf{W} , \mathbf{b} 及所有前驱层参数的影响, 训练过程中修改这些参数有可能让 \mathbf{x} 的更多维移向非线性激活函数的饱和区, 使得学习过程收敛速度变慢。该效应随着网络的深度加深被进一步放大。应用中, 一般采用 $\text{ReLU}(x) = \max(x, 0)$ 激活函数解决饱和问题及因此导致的梯度消散问题, 采用 He 初始化方法初始化权值矩阵 \mathbf{W} , 并采用小的学习率。然而, 若我们可以保证非线性函数的输入在网络训练过程中更加稳定, 则优化器陷入饱和区概率将变小, 因此训练速度将得到提升。

我们称深度网络中内部节点随着训练过程而发生的分布变化为 Internal Covariate Shift, 消除该现象将加速训练过程。Batch Normalization 通过对输入进行归一化以便固定每一层输入的均值和方差, 来减小 Interval Covariate Shift, 以便极大程度上加速深度网络训练。另外可以看出该归一化过程, 对梯度在网络中反向传播也有帮助, 该过程降低了梯度对参数大小或初始值大小的依赖, 因此可以采用更大的学习率, 同时 Batch Normalization 对模型进行了正则化, 因此降低了对 Dropout 的依赖, 最后 Batch Normalization 使得采用存在饱和线性的非线性激活函数成为可能。下一小节我们将具体看如何在深度网络中引入 Batch Normalization。

7.5.2 Batch Normalization

令 d 维向量 \mathbf{x} 表示深度网络中层的输入:

$$\mathbf{x} = [x_1, \dots, x_d]^\top \quad (7.62)$$

对输入的每个维度采用如下方法进行归一化：

$$\hat{x}_k = \frac{x_k - E[x_k]}{\sqrt{Var[x_k]}} \quad (7.63)$$

经过归一化后，输入向量的每个维度的均值为 0，方差为 1。另外注意到，如果仅仅对每层输入的每个维度进行上述归一化，将改变每层可表示范围。例如，对 sigmoid 函数的输入进行归一化，将限制该函数处于非线性的线性区间。为了解决该问题，我们希望上述在网络中引入的归一化策略对每个维度可以进行 Identity 变换。因此对每个激活维度 x_k ，我们引入一对参数 γ_k 及 β_k ，分别对归一化值进行缩放和平移：

$$\hat{y}_k = \gamma_k \hat{x}_k + \beta_k. \quad (7.64)$$

上述两个参数和初始的模型参数一起在训练过程中进行学习，用于恢复网络的表示能力。实际上当 $\gamma_k = \sqrt{Var[x_k]}$, $\beta_k = E[x_k]$ 时，上述变换可以恢复到初始的输入。

考虑到在全量数据上各层输入的每个维度进行归一化成本较高，在估计输入的各维度的均值和方差时，采用 mini-batch 中输入的进行均值和方差估计，因此 Batch Normalization 变换如下：

Algorithm 2 Batch Normalization Transformation

Input: Values of \mathbf{x} over a mini-batch: $\mathcal{B} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$;

Parameters to be learned: γ, β

Output: $\{\mathbf{y}^{(i)} = BN_{\gamma, \beta}(\mathbf{x}^{(i)})\}$

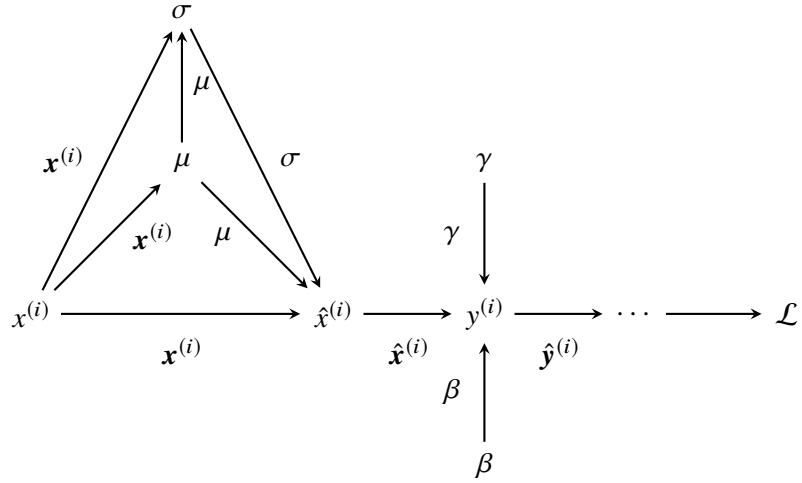
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m \mathbf{x}^{(i)} \quad \triangleright \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (\mathbf{x}^{(i)} - \mu_{\mathcal{B}})^2 \quad \triangleright \text{mini-batch variance}$$

$$\hat{\mathbf{x}}^{(i)} \leftarrow \frac{\mathbf{x}^{(i)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad \triangleright \text{normalize}$$

$$\mathbf{y}^{(i)} \leftarrow \gamma \hat{\mathbf{x}}^{(i)} + \beta \equiv BN_{\gamma, \beta}(\mathbf{x}^{(i)}) \quad \triangleright \text{scale and shift}$$

上述 BN 变换应用于网络中任意的激活函数函数输出。上述算法中的从上述变换过程可见， $\hat{\mathbf{x}}$ 的期望为 0，方差为 1， $\hat{\mathbf{x}}^{(k)}$ 可以看成线性变换子网络 $\mathbf{y}^{(k)} = \gamma(\hat{x})^{(k)} + \beta$ 的输入，后接原始网络中处理逻辑。所有线性变换子网络的输入的均值均为 0 和方差为 1，虽然 $\hat{\mathbf{x}}^{(k)}$ 的联合分布在训练过程中可能会发生变化，我们希望通过引入归一化后的输入加速子网络训练，从而加速整个网络的训练。训练过程中，我们需要将损失函数 \mathcal{L} 的梯度通过反向传播流过孩子网络，同时计算 BN 变换的参数的梯度。为了较容易的理解梯度反向传播链式法则，下面给出孩子网络的计算图：



基于计算图可以得到该子网络的反向传播梯度如下：

$$\frac{\partial \mathcal{L}}{\partial \hat{x}_i} = \frac{\partial \mathcal{L}}{\partial y_i} \odot \gamma \quad (7.65)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \odot (x^{(i)} - \mu_{\mathcal{B}}) \odot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2} \quad (7.66)$$

$$\frac{\partial \mathcal{L}}{\partial \mu} = \left(\sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \odot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \odot \frac{\sum_{i=1}^m -2(x_i - \mu_{\mathcal{B}})}{m} \quad (7.67)$$

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \odot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \odot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \mathcal{L}}{\partial \mu} \odot \frac{1}{m} \quad (7.68)$$

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \odot \hat{x}_i \quad (7.69)$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = \sum_{i=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \quad (7.70)$$

因此 BN 为可导变换，该变换对网络中的激活进行归一化操作，其确保了在模型训练过程中，层输入的分布的 Internal Covariate Shift 被降低，并加速训练过程。由 BN 变换可见训练过程中需要估计 $\mu_{\mathcal{B}}$ 和 $\sigma_{\mathcal{B}}^2$ ，同时预测过程也需要由训练过程得到这两个参数并固定住，训练过程中一般使用移动平均方法得到这两个参数的估计，并在预测过程中直接应用该移动平均值。

这里有必要说明一下 CNN 网络的 Batch Normalization 策略，考虑经过 Convolution 阶段对输出数据的每个通道应用了同样的 kernel 进行卷积计算，因此和上面的 Batch Normal 类似，假设输出数据的维度为：[batch_size, height, width, channel]，则输出数据的每个通道对应一个均值和方差，均值和方差估计的样本量为：batch_size \times height \times width。

在非线性变换前，我们对 $Wu + b$ 应用 Batch Normalization，偏执项 b 可以去掉，因为该偏执项会在后续的归一化过程中由于减均值操作被抵消掉。因此 $z = g(BN(Wu + b))$ 可被替换为：

$$z = g(BN(Wu)) \quad (7.71)$$

其中 BN 变换被独立应用到 $\mathbf{x} = \mathbf{W}\mathbf{u}$ 的各个维度上，每个维度有独立的学习到的缩放系数 γ_k 及偏移系数 β_k 。

最后让我们来看一下为何引入 Batch Normalization 之后可以加大学习率。传统深度网络中，过高的学习率可能导致梯度爆炸或消散，并陷入较差的局部最小值。Batch Normalization 可以解决这些问题。通过对网络中非线性变换函数的输入值进行归一化，该操作防止了权重参数上较小的变化放大为激活函数上梯度较大的次优的变化，譬如，通过归一化可以防止训练陷入非线性变换的饱和区。Batch Normalization 同时让训练对权值参数的大小更加有弹性。通常，大的学习率会增加每层权重参数的大小，权值参数大小的增加进一步在反向传播过程中加大了梯度并导致梯度爆炸。然而，通过 Batch Normalization，反向传播时，将不收权重参数的大小的影响。譬如，考虑缩放系数 a ，则有：

$$\text{BN}(\mathbf{W}\mathbf{u}) = \text{BN}((a\mathbf{W})\mathbf{u}) \quad (7.72)$$

这里简单对上述等式进行一下推导，假设 $\mathbf{W}\mathbf{u}$ 对应的均值和方差为： $\mu_{\mathcal{B}}$ 及 $\sigma_{\mathcal{B}}^2$ ，因此 $(a\mathbf{W})\mathbf{u}$ 对应的均值和方差为： $a\mu_{\mathcal{B}}$ 及 $a^2\sigma_{\mathcal{B}}^2$ ，考虑 Batch Normalization 过程：

$$\text{BN}((a\mathbf{W})\mathbf{u}) = \gamma \frac{(a\mathbf{W})\mathbf{u} - a\mu_{\mathcal{B}}}{a\sigma_{\mathcal{B}}} + \beta = \gamma \frac{(\mathbf{W}\mathbf{u}) - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} + \beta = \text{BN}(\mathbf{W}\mathbf{u}) \quad (7.73)$$

可见经过 Batch Normalization 后权值参数大小对非线性变换函数没有影响。另外：

$$\frac{\partial \text{BN}((a\mathbf{W})\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}} \left(\gamma \frac{(a\mathbf{W})\mathbf{u} - a\mu_{\mathcal{B}}}{a\sigma_{\mathcal{B}}} + \beta \right) = \gamma \frac{\mathbf{W}}{\sigma_{\mathcal{B}}} = \frac{\partial \text{BN}(\mathbf{W}\mathbf{u})}{\partial \mathbf{u}} \quad (7.74)$$

$$\frac{\partial \text{BN}((a\mathbf{W})\mathbf{u})}{\partial (a\mathbf{W})} = \frac{\partial}{\partial (a\mathbf{W})} \left(\gamma \frac{(a\mathbf{W})\mathbf{u} - a\mu_{\mathcal{B}}}{a\sigma_{\mathcal{B}}} + \beta \right) = \frac{1}{a} \gamma \frac{\mathbf{u}}{\sigma_{\mathcal{B}}} = \frac{1}{a} \frac{\partial \text{BN}(\mathbf{W}\mathbf{u})}{\partial \mathbf{W}} \quad (7.75)$$

由公式7.74可知经过 Batch Normalization 之后，权重参数的大小对梯度传播没有影响，因此 Batch Normalization 对权值初始化的要求不太严格，由公式7.75可知大的权值参数将导致小的梯度，因此 Batch Normalization 具有稳定权重参数增长的作用。

7.6 层归一化

由上一小节可知 Batch Normalization 需要对每一层存储均值和方差这两个移动平均统计量。对于有固定深度的前向网络来说，对每一层分别存储各自的上述两个统计量是没有问题的，然而对于循环神经网络来说，我们需要计算和保存序列中不同时间步骤的上述两个统计量，若测试序列比所有训练序列均要长，则 Batch Normalization 将会遇到问题。另外 Batch Normalization 无法应用到在线学习任务或者非常大的分布式模型任务上，此时训练的 batch 较小，因此基于较小的 batch 的计算出的均值和方差统计量无法有效表示全局样本的统计量。

考虑到 Batch Normalization 存在的上述问题，Jimmy Lei Ba 等人引入 Layer Normalization⁴策略。和 Batch Normalization 有所不同的，Layer Normalization 假设在非线性激

⁴Layer Normalization

活前的输入的随机变量的分布接近，因此可以直接基于每层的所有非线性激活前的输入估计均值和方差，并基于这两个统计量对输入进行均值为 0，方差为 1 的归一化。因此 Layer Normalization 的均值和方差公式如下：

$$\mu^l = \frac{1}{H} \sum_{i=1}^H h_i^l, \quad \sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^H (h_i^l - \mu^l)^2} \quad (7.76)$$

上述公式中 H 表示隐藏层 l 的神经元个数，和 Batch Normalization 有所不同的是，这里 μ^l 及 σ^l 为标量，因此隐藏层 l 的所有神经元公用统一的统计量 μ^l 及 σ^l 。在此基础上 Batch Normalization 的公式如下：

$$\mathbf{h}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l, \quad \text{LN}(\mathbf{h}^l) = \frac{\mathbf{g}^l}{\sigma^l} \odot (\mathbf{h}^l - \mu^l) + \mathbf{b}^l, \quad \mathbf{x}^l = g(\text{LN}(\mathbf{h}^l)) \quad (7.77)$$

考虑到训练神经网络不同时序时采用的权重参数一致，因此上述公式变为如下形式：

$$\mathbf{a}^t = \mathbf{W}[\mathbf{x}^t, \mathbf{h}^{t-1}] + \mathbf{b}, \quad \text{LN}(\mathbf{a}^t) = \frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b}, \quad \mathbf{h}^t = g(\text{LN}(\mathbf{a}^t)) \quad (7.78)$$

上述公式中 $[\mathbf{x}^t, \mathbf{h}^{t-1}]$ 表示向量拼接。

考虑到 Layer Normalization 假设在非线性激活前的输入的随机变量的分布接近，而 CNN 网络中图像边缘对应的 kernel 由于由大量的隐藏单元未被激活，因此该假设不成立，因此在 CNN 网络中 Layzer Normalization 效果没有 Batch Normalization 效果好。

第8章 损失函数

本章回顾不同类型的损失函数定义及其应用场景

8.1 0-1 损失函数

假定要处理的问题为二分类问题，譬如点击率预估，用户在特定场景下是否会点击某广告。假定标注样本集为 $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$ ，其中 $\mathbf{x}^{(i)}$ 为样本 i 特征表示向量， $t^{(i)}$ 为样本 i 对应的标注。假定分类器为如下线性分类器：

$$z = \mathbf{w}^\top \mathbf{x} + b \quad (8.1)$$

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (8.2)$$

我们最小化错误分类的样本数，因此可以定义如下 0-1 损失函数：

$$\mathcal{L}_{0-1}(y, t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{otherwise.} \end{cases} \quad (8.3)$$

给定上述损失函数后，我们希望通过梯度下降法最小化损失函数，因此需要求解偏导 $\partial \mathcal{L}_{0-1} / \partial w_j$ ，回到偏导的定义，偏导描述了若对 w_j 引入某个微小的变化量， \mathcal{L}_{0-1} 会产生多大的变化，由 0-1 损失函数可知，只要当前样本没有处于线性分类器的分界面处，对 w_j 的微小变化对 \mathcal{L}_{0-1} 没有影响，因此只要样本没有处于线性分类器的分界面处，则 $\partial \mathcal{L}_{0-1} / \partial w_j = 0$ ，因此此时基于梯度下降方法，损失函数不会下降，若样本处于线性分界面上，由于损失函数不连续，偏导不存在。因此我们无法通过梯度下降方法优化 0-1 损失函数。

下图是 0-1 损失函数对应 $t = 1$ 时的图形化描述：

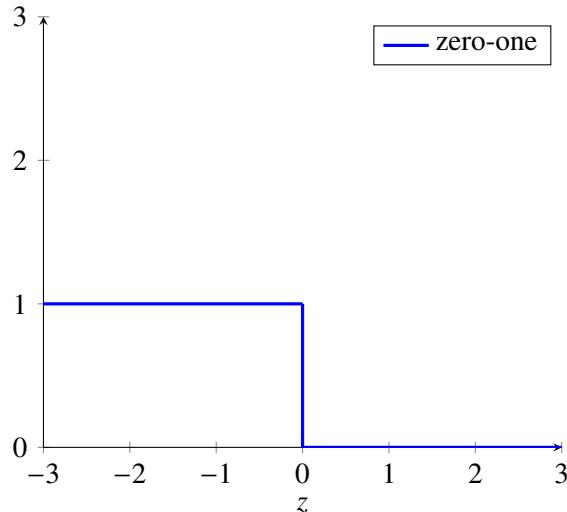


图 8.1: 损失函数曲线

8.2 平方错误损失

平方错误损失 (Squared Error Loss) 一般用于回归任务，譬如预测房价，预测股价等。假设模型为线性模型，其对应的损失函数公式如下

$$y = \mathbf{w}^\top \mathbf{x} + b \quad (8.4)$$

$$\mathcal{L}_{SE}(y, t) = \frac{1}{2}(y - t)^2 \quad (8.5)$$

若将该损失函数应用于 0-1 损失小节定义的分类问题，则 y 对应公式8.1 中的 z 。预测阶段可以设定阈值 $y = 1/2$ 。考虑到平方错误损失可导，因此可以用基于梯度下降算法对损失函数进行优化，虽然平方错误损失不像 0-1 损失函数直接对目标建模，考虑到 0-1 损失函数优化困难，我们可以考虑用平方错误损失替代 0-1 损失函数，这里平方错误损失起到了间接达到原损失函数的目标，因此该损失函数又称为**代理损失函数 (surrogate loss function)**。将平方错误损失函数应用于上述二分类问题存在的问题是，若当前样本为正样本，即 $t = 1$ ，训练过程中对该样本为正样本非常确信，并给出了预测 $y = 9$ ，遗憾的是此时损失为 $\frac{1}{2}(9 - 1)^2 = 32$ ，因此训练过程中算法将在这类现象上付出较大开销。

下图是平方错误损失函数对应 $t = 1$ 时的图形化描述，叠加至 0-1 损失，并将 y 替换为 z ：

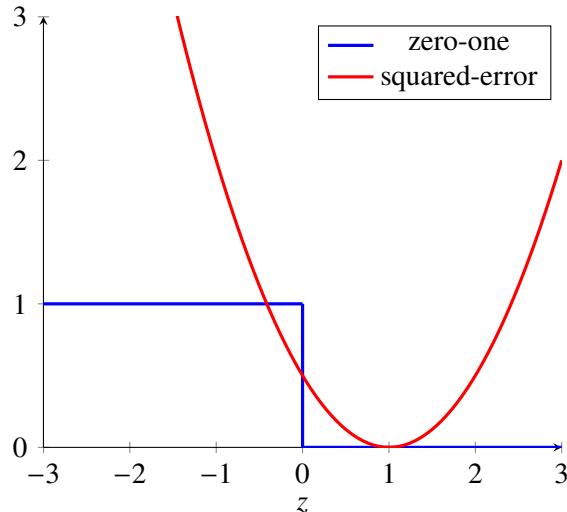


图 8.2: 损失函数曲线

考虑二分类问题中直接将线性函数的输出 z 应用于平方错误损失函数存在对正负样本中高置信预测算法需要花费较大开销进行优化的问题，考虑对输出 z 进行 sigmoid 变换，将变换后的输出应用于平方错误损失函数，公式如下：

$$z = \mathbf{w}^\top \mathbf{x} + b \quad (8.6)$$

$$y = \sigma(z) \quad (8.7)$$

$$\mathcal{L}_{SE}(y, t) = \frac{1}{2}(y - t)^2. \quad (8.8)$$

此时对应的损失函数如下图所示：

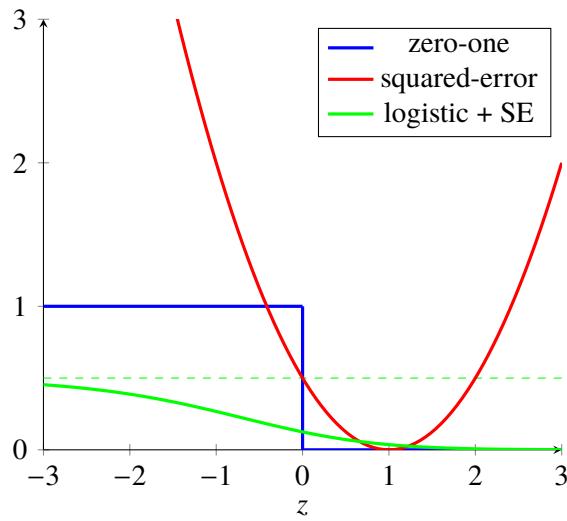


图 8.3: 损失函数曲线

从上述损失函数曲线可见，对输出引入 sigmoid 变换后，对应的损失函数曲线对 0-1 损失的拟合要好于不做 sigmoid 变换的损失，存在的问题是负区间存在饱和问题，该问题导致对正样本，即使 z 取较大的赋值，在损失上最多为 $\frac{1}{2}$ ，下面我们来看下一小节引入的交叉熵损失如何解决该问题。

8.3 交叉熵损失

交叉熵 (Cross-Entropy)¹ 定义如下：

$$H(p, q) = E_p[-\log q]. \quad (8.9)$$

对于离散概率分布 p 及 q 即为如下形式：

$$H(p, q) = - \sum_{x \in \mathcal{X}} p(x) \log q(x). \quad (8.10)$$

对于二分类问题， $p(t)$ 对应样本 t 的标注概率分布， $q(t)$ 对应样本的预测概率分布，交叉熵损失定义如下：

$$\mathcal{L}_{CE}(y, t) = \begin{cases} -\log(y) & \text{if } t = 1 \\ -\log(1 - y) & \text{if } t = 0 \end{cases} \quad (8.11)$$

其中 $y = \sigma(z)$ ，上述公式可以合并为如下形式：

$$\mathcal{L}_{CE}(y, t) = -t \log y - (1 - t) \log(1 - y). \quad (8.12)$$

对于多分类问题，根据定义类推有如下损失：

$$\mathcal{L}_{CE}(y, t) = -\log y_t \quad (8.13)$$

其中 y_t 表示对应标注类别的概率，因为样本 t 对应的 $p(t)$ 在其他类别的概率为 0。二分类时，交叉熵对应正类别的损失函数如下图所示：

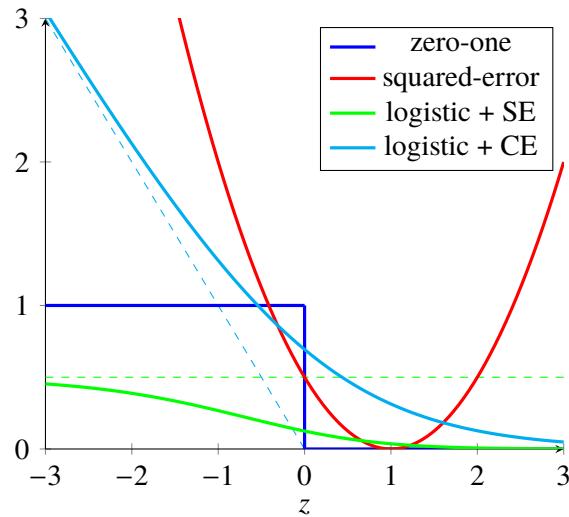


图 8.4: 损失函数曲线

从交叉熵损失函数曲线可见，相对于 sigmoid 平方错误损失，对于正样本当预测值 z 较

¹https://en.wikipedia.org/wiki/Cross_entropy

小时，不会不出现之前的饱和现象，导致偏导数接近于 0，sigmoid 交叉熵损失随着 z 减小，偏导无限逼近-1。

8.4 Hinge 损失

另外一类我们常见的损失函数为 Hinge Loss，假设为二分类问题，对应的标注为 $t \in \{-1, 1\}$ 。其中 -1 代表负样本，1 代表正样本， y 是实数预测值，则如下是 Hinge 损失函数定义：

$$\mathcal{L}_H(y, t) = \max(0, 1 - ty) \quad (8.14)$$

若应用线性模型且采用 Hinge Loss，则该类模型称为 Support Vector Machine (SVM)。如下是公式定义：

$$y = \mathbf{w}^\top \mathbf{x} + b \quad (8.15)$$

$$\mathcal{L}_H = \max(0, 1 - ty) \quad (8.16)$$

可见上述公式中 y 对应之前小节的 z 。对于多分类问题，SVM 定义为如下形式：

$$y = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (8.17)$$

$$\mathcal{L}_H = \sum_{j \neq t} \max(0, y_j - y_t + \Delta) \quad (8.18)$$

其中 t 为标注类别的编号， $\mathbf{x} \in \mathbb{R}^n$ 为特征向量， $\mathbf{W} \in \mathbb{R}^{k \times n}$ 为权值矩阵，其中 k 为类别数。二分类时，SVM 对应正类别的损失函数如下图所示：

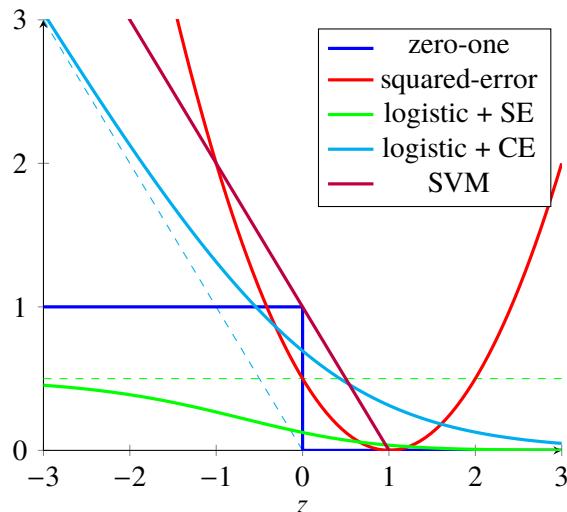


图 8.5: 损失函数曲线

从 SVM 曲线可见，其效果和 logistic 交叉熵损失函数基本近似。

8.5 CRF 损失

在处理序列标注任务时常常会使用 CRF 模型 + 交叉熵损失，由于该类问题建模和之前分类问题建模有所区别，因此独立作为一个小节进行回顾。CRF 模型全称为 Conditional Random Fields，首次由 John Lafferty 等人²引入用于序列标注任务。CRF 模型为无向图模型，其目标是给定序列 \vec{x} 生成其对应的标注序列 \vec{y} ，譬如词性标注任务，命名实体识别任务等。应用于序列标注问题时，常采用的形式为 Linear-chain CRF，其无向图和相应的因子图形如下所示³：

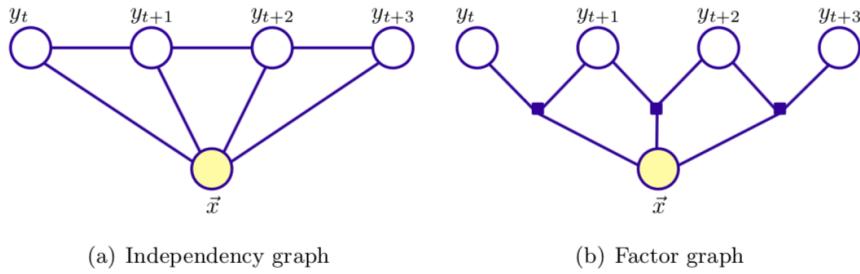


图 8.6: A Linear Chain Conditional Random Field

考虑到前述章节未对因子图 (factor graph) 做相应的解释，这里有必要对其做一定的解释。因子图中 \circ 和无向图一样表示随机变量，因子图中包含因子节点 \blacksquare ，表示因子 $\phi_c(\vec{x}_c)$ 。因子图中边均是无向的，连接了因子和其关联的所有随机变量。相对于无向图，因子图对概率分布的表示更加直观。上述因子图对应的 Linear-Chain CRF 对给定序列 \vec{x} 其对应的某个标注序列 \vec{y} 概率分布公式表示如下：

$$\begin{aligned}
 p(\vec{y} | \vec{x}) &= \frac{1}{Z_{\vec{w}}(\vec{x})} \prod_{c \in \mathbb{C}} \phi_c(\vec{x}_c, \vec{y}_c) \\
 &= \frac{1}{Z_{\vec{w}}(\vec{x})} \prod_{j=1}^n \exp \left(\sum_{i=1}^m w_i f_i(y_{j-1}, y_j, \vec{x}, j) \right) \\
 &= \frac{1}{Z_{\vec{w}}(\vec{x})} \exp \left(\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y_{j-1}, y_j, \vec{x}, j) \right)
 \end{aligned} \tag{8.19}$$

其中 $Z_{\vec{w}}(\vec{x})$ 为归一化项，公式如下：

$$Z_{\vec{w}}(\vec{x}) = \sum_{\vec{y}' \in \mathcal{Y}} \exp \left(\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y'_{j-1}, y'_j, \vec{x}, j) \right) \tag{8.20}$$

这里有必要做一些解释，结合因子图可见上述公式中 f_i 为特征， w_i 为权值。可以采用交

²Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data

³Classical Probabilistic Models and Conditional Random Fields

叉熵损失函数优化 CRF 模型。

$$\begin{aligned}
 \mathcal{L} &= -\log p(\vec{y} | \vec{x}) \\
 &= -\log \left[\frac{\exp \left(\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y_{j-1}, y_j, \vec{x}, j) \right)}{\sum_{\vec{y}' \in \mathcal{Y}} \exp \left(\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y'_{j-1}, y'_j, \vec{x}, j) \right)} \right] \\
 &= \underbrace{-\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y_{j-1}, y_j, \vec{x}, j)}_{\mathcal{A}} + \underbrace{\log \left[\sum_{\vec{y}' \in \mathcal{Y}} \exp \left(\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y'_{j-1}, y'_j, \vec{x}, j) \right) \right]}_{\mathcal{B}} \quad (8.21)
 \end{aligned}$$

因此对损失函数求 w_k 的偏导，可以通过分别对 \mathcal{A} 和 \mathcal{B} 求偏导并求和得到。 \mathcal{A} 部分的偏导如下：

$$\frac{\partial}{\partial w_k} \left[-\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y_{j-1}, y_j, \vec{x}, j) \right] = -\sum_{j=1}^n f_k(y_{j-1}, y_j, \vec{x}, j) \quad (8.22)$$

上述公式比较容易计算，只需要遍历一遍序列累加相应的特征值即可。

\mathcal{B} 部分对应归一化项，其偏导如下：

$$\begin{aligned}
 \frac{\partial}{\partial w_k} \log Z_{\vec{w}}(\vec{x}) &= \frac{1}{Z_{\vec{w}}(\vec{x})} \frac{\partial Z_{\vec{w}}(\vec{x})}{\partial w_k} \\
 &= \frac{1}{Z_{\vec{w}}(\vec{x})} \sum_{\vec{y}' \in \mathcal{Y}} \left[\exp \left(\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y'_{j-1}, y'_j, \vec{x}, j) \right) \cdot \sum_{j=1}^n f_k(y'_{j-1}, y'_j, \vec{x}, j) \right] \\
 &= \sum_{\vec{y}' \in \mathcal{Y}} \left[\underbrace{\frac{1}{Z_{\vec{w}}(\vec{x})} \exp \left(\sum_{j=1}^n \sum_{i=1}^m w_i f_i(y'_{j-1}, y'_j, \vec{x}, j) \right)}_{p(\vec{y}' | \vec{x})} \cdot \sum_{j=1}^n f_k(y'_{j-1}, y'_j, \vec{x}, j) \right] \\
 &= \sum_{\vec{y}' \in \mathcal{Y}} p(\vec{y}' | \vec{x}) \sum_{j=1}^n f_k(y'_{j-1}, y'_j, \vec{x}, j) \quad (8.23)
 \end{aligned}$$

考虑到到序列空间随着序列的变长成指数增长，因此直接计算上述 $p(\vec{y}' | \vec{x})$ 几乎不可行，因此考虑采用前向后向算法。定义函数 $T_j(s)$ ，该函数将位置 j 状态 s 映射至位置 $j+1$ 处所有可能的后续状态。定义函数 $T_j^{-1}(s)$ ，该函数将位置 j 状态 s 映射至位置 $j-1$ 处所有可能的前驱状态。定义特殊状态 \perp 表示序列的开始状态， \top 表示序列的结束状态。定义前向得分 α 和后向得分 β ，用于表示信息沿着网络传输的过程，假设为线性链网络，则

有：

$$\alpha_j(s|\vec{x}) = \sum_{s' \in T_j^{-1}(s)} \alpha_{j-1}(s'|\vec{x}) \phi_j(\vec{x}, s', s) \quad (8.24)$$

$$\beta_j(s|\vec{x}) = \sum_{s' \in T_j(s)} \beta_{j+1}(s'|\vec{x}) \phi_{j+1}(\vec{x}, s, s') \quad (8.25)$$

其中：

$$\phi_j(\vec{x}, s', s) = \sum_{i=1}^m w_i f_i(y_{j-1} = s', y_j = s, \vec{x}, j) \quad (8.26)$$

函数 α 记录了从网络开始位置发送网络结束的信息，函数 β 记录了从网络结束位置到开始位置发送的信息。初始化如下：

$$\alpha_0(\perp|\vec{x}) = 1 \quad (8.27)$$

$$\beta_{|\vec{x}|+1}(\top|\vec{x}) = 1. \quad (8.28)$$

基于 α 和 β 即可高效计算公式8.23，其形式如下：

$$\frac{\partial}{\partial w_k} \log Z_{\vec{w}}(\vec{x}) = \frac{1}{Z_{\vec{w}}(\vec{x})} \sum_{j=1}^n \sum_{s \in \mathcal{S}} \sum_{s' \in T_j(s)} f_i(s, s', \vec{x}, j) \alpha_{j-1}(s|\vec{x}) \phi_j(\vec{x}, s, s') \beta_j(s'|\vec{x}). \quad (8.29)$$

归一化项公式如下：

$$Z_{\vec{w}}(\vec{x}) = \beta_0(\perp|\vec{x}) \quad (8.30)$$

有上述计算过程可见，前向后向算法的时间复杂度为 $O(|\mathcal{S}|^2 n)$ 。

8.6 KL-divergence 损失

KL-divergence (Kullback-Leibler divergence)⁴又称为相对熵，用于衡量概率分布差异，考虑我们的应用场景为分类问题，该场景对应离散型概率分布，因此 KL-divergence 公式如下：

$$D_{\text{KL}}(P||Q) = - \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{Q(x)}{P(x)} \right) \quad (8.31)$$

KL-divergence 损失可以用于 semi-supervised 学习范式，用于未标注数据。譬如 Qizhe Xie 等人⁵提出的 UDA 方法在少量标注基础上，对大量未标注数据应用和任务相关的数据增强方法，并期望增强前后的样本在类别上概率分布一致，因此有如下公式：

$$\mathcal{L}_{\text{UDA}}(\theta) = E_{x \in \mathbb{U}} E_{\hat{x} \sim q(\hat{x}|x)} [D_{\text{KL}}(p_{\hat{\theta}}(y|x)||p_{\theta}(y|\hat{x}))] \quad (8.32)$$

⁴https://en.wikipedia.org/wiki/Kullback–Leibler_divergence

⁵Unsupervised Data Augmentation for Consistency Training

上述公式中 $q(\hat{x}|x)$ 为数据增强变换， $\hat{\theta}$ 为参数 θ 的拷贝。将上述损失和标注数据损失结合到一起作为最终的目标函数：

$$\mathcal{L} = E_{x,y^* \in \mathbb{L}} [p_\theta(y^*|x)] + \lambda \mathcal{L}_{\text{UDA}}(\theta) \quad (8.33)$$

通过最小化上述损失函数，UDA 使得标注信息可以从标注数据传递至非标注数据。可以看到上述方法的关键在数据增强方法 $q(\hat{x}|x)$ ，有效的数据增强方法可以建立标注数据和非标注数据的关联关系。同时考虑到标注数据的数据量往往远小于非标注数据的数据量，为了防止标注数据过拟合问题，Qizhe Xie 等人提出了 TSA(Training Signal Annealing) 方法，用于缓慢释放训练样本，即将标注数据的交叉熵损失函数修改为如下形式：

$$\frac{1}{Z} \sum_{x,y^* \in \mathbb{B}} [-I(p_\theta(y^*|x) < \eta_t) \log p_\theta(y^*|x)] \quad (8.34)$$

其中 $I(\cdot)$ 为指示函数， $Z = \sum_{x,y^* \in \mathbb{B}} I(p_\theta(y^*|x))$ 为归一化因子。 η_t 用于防止模型对置信的样本进行过训练。假设类别数为 K ，训练过程中，当 η_t 逐渐从 $\frac{1}{K}$ 增加至 1 时，模型可以缓慢地接受标注样本标注信号，极大程度降低了过拟合问题。假设 T 为总的训练步数， t 为当前的训练步数，Qizhe Xie 等给出了如下三种 η_t 设定方案：

- **log-schedule**: 该设定下，在训练开始阶段 η_t 增长较快

$$\eta_t = \left(1 - \exp\left(-\frac{t}{T} * 5\right)\right) * \left(1 - \frac{1}{K}\right) + \frac{1}{K} \quad (8.35)$$

- **linear-schedule**: 该设定下， η_t 随着训练过程线性增长

$$\eta_t = \frac{t}{T} * \left(1 - \frac{1}{K}\right) + \frac{1}{K} \quad (8.36)$$

- **exp-schedule**: 该设定下， η_t 在接近训练结束阶段快速增长

$$\eta_t = \exp\left(\left(\frac{t}{T} - 1\right) * 5\right) * \left(1 - \frac{1}{K}\right) + \frac{1}{K} \quad (8.37)$$

上述三类设定方法可以统一表示为：

$$\eta_t = \frac{1}{K} + \lambda_t * \left(1 - \frac{1}{K}\right) \quad (8.38)$$

其中 λ_t 对应不同的设定策略如下所示：

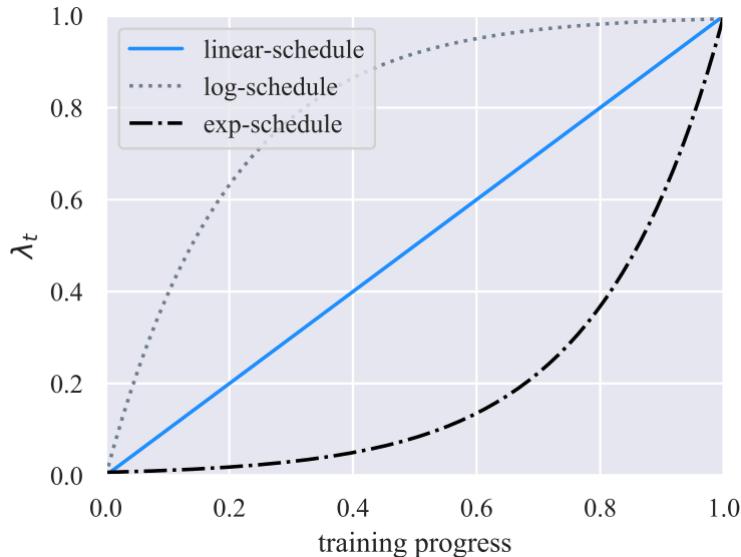


图 8.7: training progress

8.7 RankNet 损失

RankNet 损失由 Chris Burges 等人⁶提出用于网页排序任务。这里有必要对网页排序任务背景进行简要地介绍，网页排序中训练数据基于 query 进行分组，每个 query(Q) 下存在若干文档(urls)，可以对若干文档两两进行分组生成 pair 对 (U_i, U_j) ，训练网络时仅将存在偏序关系的 pair 对用于训练任务，考虑到偏序的对称关系，因此训练时可以考虑仅保留正序的偏序对，即 $U_i > U_j$ ，假设 s_i 为模型对 U_i 的打分， s_j 为模型对 U_j 的打分，RankNet 采用如下方法将 U_i 需要排在 U_j 映射为概率：

$$P_{ij} \equiv P(U_i > U_j) \equiv \frac{1}{1 + e^{-\lambda(s_i - s_j)}} \quad (8.39)$$

结合交叉熵公式推出如下损失函数：

$$\mathcal{L} = \log \left(1 + e^{-\lambda(s_i - s_j)} \right) \quad (8.40)$$

上述公式中 s_i 和 s_j 可以应用深度学习模型给出 $\langle Q, U_i \rangle$ 和 $\langle Q, U_j \rangle$ 的打分， $\lambda > 0$ 为超参。

8.8 LambdaRank 损失

RankNet 损失存在的问题是仅考虑了给定 query 下文档间的偏序关系，拟合偏序关系和实际的信息检索指标存在一定的差异。这里考虑采用 NDCG (Normalized Discounted Cu-

⁶Learning to Rank using Gradient Descent

mulative Gain)⁷指标，对给定 query 和 url 列表，NDCG 计算公式如下：

$$DCG@T \equiv \sum_{i=1}^T \frac{2^{l_i} - 1}{\log(1 + i)} \quad (8.41)$$

$$NDCG@T \equiv \frac{DCG@T}{maxDCG@T} \quad (8.42)$$

其中 T 为截断等级（譬如仅考虑第一页的返回列表，此时 $T = 10$ ）， l_i 为返回 URL 列表中第 i 个文档的标注，一般考虑五档相关得分 $l_i \in \{0, 1, 2, 3, 4\}$ 。考虑下图⁸中给定 query 下的一系列 url，其中浅灰色条表示和 query 不相关的 url，深蓝色的条表示和 query 相关的文档。左图中总的错误对为 13 个，右图中：通过将头部的 url 往下移到第四的排序位置，底部的相关文档往上移动 5 位，总的错误对降至 11 个。然而采用信息检索衡量指标譬如 NDCG 指标，这类指标强调头部的几条结果，因此右图的变化不是信息检索中期望的结果。下图中左边黑色的箭头表示 RankNet 梯度（梯度随着错误对增加而增加），而我们实际想要的是 红色 箭头表示的变化。



图 8.8: A set of urls ordered for a given query using a binary relevance measure.

结合上述现象，及 NDCG 指标，因此有必要对 RankNet 损失进行优化，以便逼近 NDCG 指标。考虑到 NDCG 指标不可导，Chris Burges 等人在 RankNet 基础上提出 LambdaRank⁹，即在 RankNet 的梯度基础上引入 $|\Delta_{NDCG}|$ ，公式如下：

$$\frac{\partial \mathcal{L}_{\text{LambdaRank}}}{\partial s_i} = \frac{-\lambda}{1 + e^{\lambda(s_i - s_j)}} |\Delta_{NDCG}| \quad (8.43)$$

考虑到 NDCG 不可导， $\mathcal{L}_{\text{LambdaRank}}$ 为未知损失函数。 $|\Delta_{NDCG}|$ 通过调换 U_i 和 U_j 的位置即可计算得到 NDCG 变化量。

⁷https://en.wikipedia.org/wiki/Discounted_cumulative_gain

⁸From RankNet to LambdaRank to LambdaMART: An Overview

⁹Learning to Rank with Nonsmooth Cost Functions

第9章 正则化

本章回顾不同正则化方法，降低模型过拟合风险

9.1 L_1 正则化

本节介绍 L_1 正则化，这里介绍 L_1 正则化方法主要从直观角度进行介绍，详细的理论分析，可参考深度学习一书的正则化章节¹。 L_1 正则化通过在原损失函数定义上引入 L_1 范数项，达到模型稀疏的效果，从参数估计的角度看， L_1 正则化项相当于 Laplace 先验，因此 L_1 正则化项引入相当于采用基于 Laplace 先验的最大后验估计。

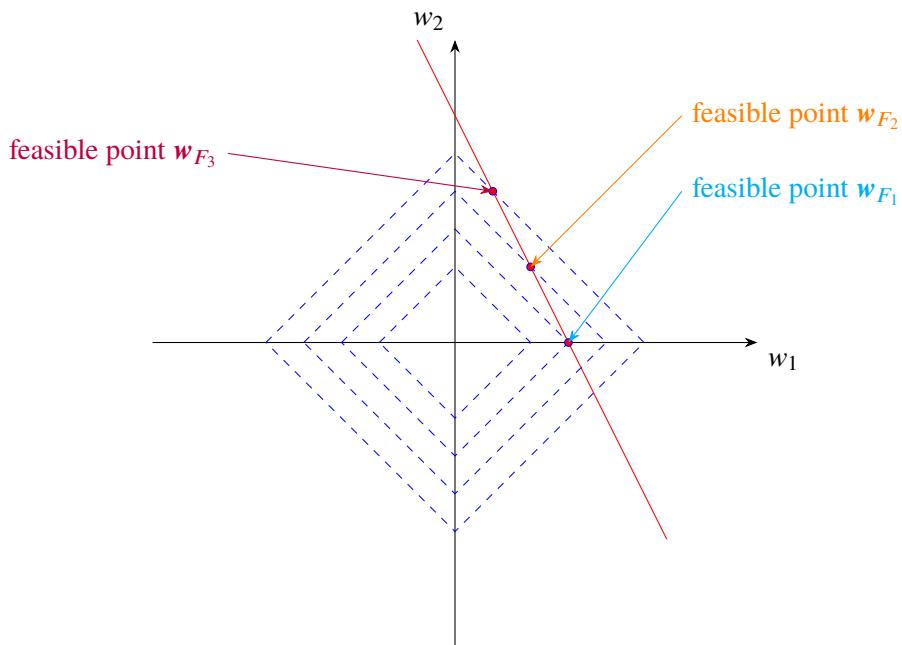
引入 L_1 正则化项后的损失函数如下所示：

$$\tilde{\mathcal{L}}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \alpha \|\mathbf{w}\|_1 \quad (9.1)$$

假设损失函数 $\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y})$ 为连续可导函数，任意点 \mathbf{w}^* 处附近，损失函数存在一阶近似：

$$\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}) \approx \mathcal{L}(\mathbf{w}^*; \mathbf{X}, \mathbf{y}) + (\mathbf{w} - \mathbf{w}^*)^\top \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^*; \mathbf{X}, \mathbf{y}) \quad (9.2)$$

假设参数向量 \mathbf{w} 的参数个数为 2，则损失函数 $\mathcal{L}(\mathbf{w}, \mathbf{X}, \mathbf{y})$ 近似函数和 L_1 正则化项的等高线如下图所示：



上图中红色直线表示 $\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y})$ 的一阶近似等高线，蓝色虚线表示 L_1 正则化项等高线。由上图可见在给定损失函数等高线时，上图中显示了三个可能解 \mathbf{w}_{F_1} 、 \mathbf{w}_{F_2} 及 \mathbf{w}_{F_3} ，由 L_1 正则化项等高线可知解 \mathbf{w}_{F_1} 对应的 $\tilde{\mathcal{L}}(\mathbf{w}; \mathbf{X}, \mathbf{y})$ 取得最小值，从上图可知 \mathbf{w}_{F_1} 为稀疏解，

¹<https://www.deeplearningbook.org/contents/regularization.html>

其中 w_2 为 0。

9.2 L_2 正则化

L_2 正则化和 L_1 正则化类似，在原损失函数基础之上引入 L_2 正则化项，从参数估计角度看， L_2 正则化项相对于 Gaussian 先验，因此 L_2 正则化项引入相当于采用基于 Gaussian 先验的最大后验估计。

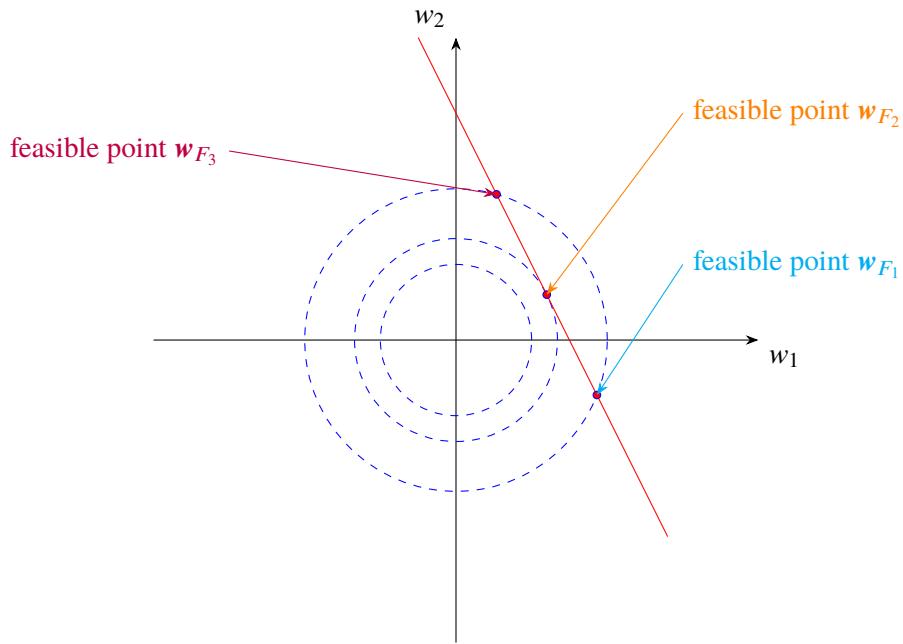
引入 L_2 正则化项后的损失函数如下所示：

$$\tilde{\mathcal{L}}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \frac{\alpha}{2} \|\mathbf{w}\|_2^2 \quad (9.3)$$

假设损失函数 $\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y})$ 为连续可导函数，任意点 \mathbf{w}^* 处附近，损失函数存在一阶近似：

$$\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}) \approx \mathcal{L}(\mathbf{w}^*; \mathbf{X}, \mathbf{y}) + (\mathbf{w} - \mathbf{w}^*)^\top \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^*; \mathbf{X}, \mathbf{y}) \quad (9.4)$$

假设参数向量 \mathbf{w} 的参数个数为 2，则损失函数 $\mathcal{L}(\mathbf{w}, \mathbf{X}, \mathbf{y})$ 近似函数和 L_2 正则化项的等高线如下图所示：



上图中红色直线表示 $\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y})$ 的一阶近似等高线，蓝色虚线表示 L_2 正则化项等高线。由上图可见在给定损失函数等高线时，上图中显示了三个可能解 \mathbf{w}_{F_1} 、 \mathbf{w}_{F_2} 及 \mathbf{w}_{F_3} ，由 L_2 正则化项等高线可知解 \mathbf{w}_{F_2} 对应的 $\tilde{\mathcal{L}}(\mathbf{w}; \mathbf{X}, \mathbf{y})$ 取得最小值，从上图可知 \mathbf{w}_{F_2} 非稀疏解。

9.3 Dropout

Dropout 由 Nitish Srivastava 等人²提出，用于防止神经网络过拟合。其思想是训练阶段随机将部分神经元输出置为 0，如下图所示：

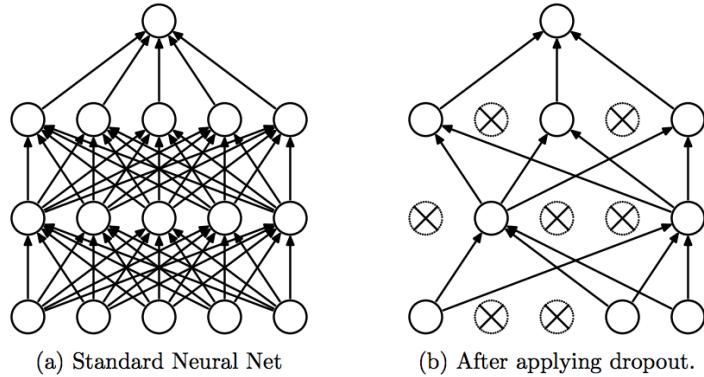


图 9.1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

通过在训练阶段应用上述策略，训练阶段相当于对若干子网络进行集成学习，最终输出的整体网络由若干子网络融合而成。Dropout 相当于对每个神经元进行 Bernoulli 实验，假设将神经元输出置 0 的概率为 P ，则保留原输出的概率为 $1 - P$ ，因此有：

$$\hat{X} = f_{\text{dropout}}(X) = \begin{cases} X^* & \text{with prob: } (1 - P) \\ 0 & \text{with prob: } P \end{cases} \quad (9.5)$$

考虑到输入期望不变性，即：

$$E[\hat{X}] = E[X] \quad (9.6)$$

因此有：

$$(1 - P) \cdot X^* + P \cdot 0 = X \quad \Rightarrow \quad X^* = \frac{1}{1 - P} X \quad (9.7)$$

9.4 数据增强

增强机器学习模型泛化能力最好的方法是在更大规模的数据集上进行训练。而实际上，我们拥有的标注数据往往量较少。解决该问题的一种方法是自动构建数据，并将自动构建的数据加入训练集。自动构建人工数据往往是和具体任务相关的，譬如图像分类领域可以采用对图像进行旋转和较小的高斯噪声方法，NLP 机器翻译或文本分类可以通过采用 Back Translation 或随机替换不重要词³方法达到扩充训练数据的方法。甚至可以研究具体的问题的数据分布，根据问题本身提出数据增强方案，以便有效达成扩充训练集的目标。

²Dropout: A Simple Way to Prevent Neural Networks from Overfitting

³Unsupervised Data Augmentation for Consistency Training

9.5 样本标注引入噪声

考虑到大多数标注数据中标注 y 存在部分错误，若 y 为错误标注，则最大化 $\log p(y|x)$ 对最终效果起到副作用。减弱该问题的一种方法是明确对标注噪声建模。譬如给定某个较小常量 ϵ ，训练数据中标注 y 为正确标注的概率为 $1 - \epsilon$ ，为其他标注的概率之和为 ϵ 。譬如标注平滑⁴通过将 k 分类输出标注的 0 及 1 概率分布函数替换为 $\frac{\epsilon}{k-1}$ 及 $1 - \epsilon$ 分布达到模型正则化效果，上述替换较容易应用于交叉熵损失函数。

9.6 半监督学习

半监督学习任务中未标注样本来自概率分布 $P(X)$ 及标注样本来自概率分布 $P(X, Y)$ 均用于预测 $P(Y|X)$ 。深度学习中半监督学习常常指的是学习出输入的表示 $\mathbf{h} = f(\mathbf{x})$ ，目标是对于来自相同类别的样本有相似的表示 \mathbf{h} 。无监督学习提供了有效的方法对相似的样本进行聚合，在此基础上应用监督学习方法，以便达到半监督学习的效果。另外一种方法是对生成模型 $P(X)$ 和判别模型 $P(Y|X)$ 进行联合学习，模型共享参数，以便达到半监督学习的效果。譬如 BERT pretrain 任务⁵，下一句预测对应判别模型，语言模型任务对应生成模型。

9.7 多任务学习

多任务学习是另一种改进模型泛化能力的方法。通过对每个训练样本同时进行若干任务学习，通过该方法增加多任务之间共享参数部分参数的泛化性。多任务学习的架构如下所示：

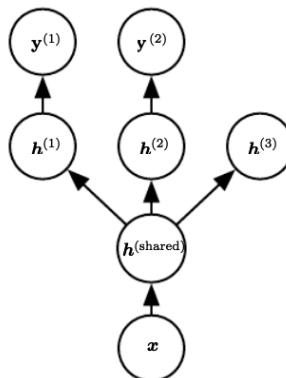


图 9.2: Multitask learning example. $\mathbf{h}^{(1)}$ 和 $\mathbf{h}^{(2)}$ 是任务特定表示， $\mathbf{h}^{(shared)}$ 是共享表示

上图中 $\mathbf{h}^{(3)}$ 对应非监督任务。通过联合学习监督任务 $\mathbf{y}^{(1)}$ 及 $\mathbf{y}^{(2)}$ 和非监督任务达到增强 $\mathbf{h}^{(shared)}$ 表示泛化能力的作用。

⁴Rethinking the Inception Architecture for Computer Vision

⁵BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

9.8 提前终止

当进行大模型且模型表达能力足以在具体任务训练集上出现过拟合时，一种方法是观察模型在验证集上的表现，若模型在验证集上经过若干步训练后，检验指标，譬如 loss 或者 F1-measure 并未出现改善，则可以提前退出训练返回在验证集上表现最好的模型参数。

第 10 章 网络架构

本章回顾不同类型的网络架构及相关应用架构

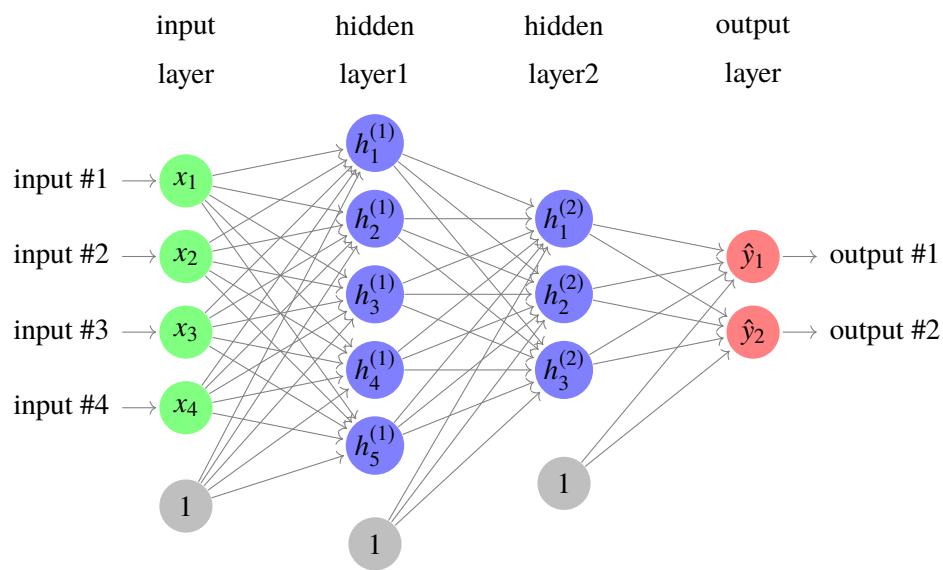
10.1 前向网络

本节首先回顾前向网络结构，在此基础上进一步回顾该类网络架构在语言模型、排序问题及推荐问题上的应用。

10.1.1 网络结构

前向神经网络由若干层隐藏层和一层输出层构成，网络中层和层之间为全连接方式，连接权重参数及偏执权值即为要训练的参数。隐藏层由若干神经元组成，每层一般采用统一的激活函数，实际上应用中所有隐藏层常常采用统一的激活函数。训练过程中在输出层之后加入损失函数即可进行网络中涉及参数的训练。

如下为一个包含两层隐藏层的前向网络：



输入层为向量 $\mathbf{x} \in \mathbb{R}^4$ ，第一层隐藏层输出向量为 $\mathbf{h}^{(1)} \in \mathbb{R}^5$ ，输入层和隐藏层间的连接权重参数为 $\mathbf{W}^{(1)} \in \mathbb{R}^{5 \times 4}$ ，对应的偏执项为 $\mathbf{b}^{(1)} \in \mathbb{R}^5$ ，第二层隐藏层输出向量为 $\mathbf{h}^{(2)} \in \mathbb{R}^3$ ，和前一层的连接权重参数为 $\mathbf{W}^{(2)} \in \mathbb{R}^{3 \times 4}$ ，对应的偏执项为 $\mathbf{b}^{(2)} \in \mathbb{R}^3$ ，输出层输出向量为 $\hat{\mathbf{y}}$ ，和前一层的连接权重参数为 $\mathbf{W}^{(3)} \in \mathbb{R}^{2 \times 3}$ ，对应的偏执项为 $\mathbf{b}^{(3)} \in \mathbb{R}^2$ 。假设隐藏层统一采用 ReLU 激活函数，输出层采用 Identity 映射，若问题为分类问题，则对输出层统一应用 Softmax 归一化操作，则输出 \hat{y}_i 为当前输入输入属于该类别的概率。该网络结构对

应的数据变换如下：

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \quad (10.1)$$

$$\mathbf{h}^{(1)} = \text{ReLU}(\mathbf{z}^{(1)}) \quad (10.2)$$

$$\mathbf{z}^{(2)} = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)} \quad (10.3)$$

$$\mathbf{h}^{(2)} = \text{ReLU}(\mathbf{z}^{(2)}) \quad (10.4)$$

$$\mathbf{z}^{(3)} = \mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)} \quad (10.5)$$

$$\hat{\mathbf{y}} = \text{Softmax}(\mathbf{z}^{(3)}) \quad (10.6)$$

要训练该网络只需要在网络的最后一层引入 Cross-Entropy Loss，及标注类别 \mathbf{y} ：

$$\arg \min_{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}, i=1,2,3} \frac{1}{N} \sum_{j=1}^N -\mathbf{y}^{(j)} \cdot \log(\hat{\mathbf{y}}^{(j)}) \quad (10.7)$$

通过随机梯度下降算法优化权重参数及偏执项，最小化损失函数。在前向网络的基础上，下面我们来回顾该网络类型在不同类型的问题上的应用。

10.1.2 语言模型

语言模型的任务是预测一句话发生的概率，即：

$$P(w_1 w_2 \dots w_n) \quad (10.8)$$

由贝叶斯法则可知，上述公式可变换为：

$$\begin{aligned} P(w_1^T) &= P(w_1)p(w_2|w_1)P(w_3|w_1^2)\dots P(w_T|w_1^{T-1}) \\ &= \prod_{t=1}^T P(w_t|w_1^{t-1}) \end{aligned} \quad (10.9)$$

由上述公式可知，若直接估计上述公式中概率 $P(w_i|w_1^{i-1})$ ，则序列会变得极其稀疏，以致在训练样本中对应的统计量不够充分，一般采用 n-gram（譬如 n=3），及假设每个词发生的概率仅依赖于历史 n-1 个词，在该假设下：

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-n+1} w_{t-1}) \quad (10.10)$$

若采用传统的语言模型模型，则可以通过若干平滑算法估计上述 n-gram 的概率。这里我们看一下如何通过前向神经网络估计给定历史 n 个词时，当前词发生的概率。

Yoshua Bengio 等人于 2003 年提出了¹基于前向神经网络的语言模型架构如下所示：

¹A Neural Probabilistic Language Model

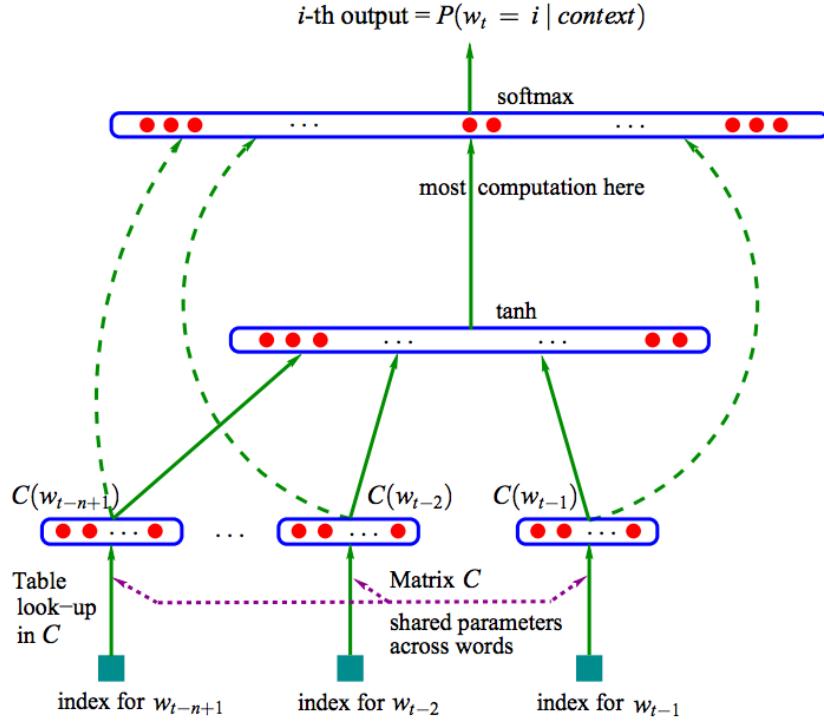


图 10.1: Neural architecture: $f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i -th word feature vector.

该网络架对应的公式如下:

$$f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1})) = P(w_t = i | w_1^{t-1}) \quad (10.11)$$

其中 C 表示将词映射为词向量操作, 可以表示为 $|V| \times m$ 的矩阵, 该矩阵的第 i 行特征向量 $C(i)$ 表示词 w_i , 函数 g 表示词向量操作之上的网络架构, 对应的变换如下:

$$\mathbf{x} = [C(w_{t-1}), C(w_{t-2}), \dots, C(w_{t-n+1})] \quad (\text{concatenation}) \quad (10.12)$$

$$\mathbf{z}^{(1)} = \mathbf{b}^{(1)} + \mathbf{H}\mathbf{x} \quad (10.13)$$

$$\mathbf{h}^{(1)} = \mathbf{U}\tanh(\mathbf{z}^{(1)}) \quad (10.14)$$

$$\mathbf{z}^{(2)} = \mathbf{b}^{(2)} + \mathbf{W}\mathbf{x} + \mathbf{h}^{(1)} \quad (10.15)$$

$$\hat{\mathbf{y}} = \text{Softmax}(\mathbf{z}^{(2)}) \quad (10.16)$$

其中 $\mathbf{x} \in \mathbb{R}^{(n-1)m}$, $\mathbf{H} \in \mathbb{R}^{h \times (n-1)m}$, $\mathbf{b}^{(1)} \in \mathbb{R}^h$, $\mathbf{U} \in \mathbb{R}^{|V| \times h}$, $\mathbf{W} \in \mathbb{R}^{|V| \times (n-1)m}$, $\mathbf{b}^{(2)} \in \mathbb{R}^{|V|}$, 训练该网络架构中的参数可以通过最小化 Cross-Entropy Loss 完成。

10.1.3 排序问题

这里讨论的排序问题为网页搜索或相似问题上的排序任务, 该类问题为, 给定查询 Q , 对文档 D_1, \dots, D_2 进行排序。因此这类问题可以转化为计算 Q 和每个 D_i 的相关性得分, 基

于该得分对文档进行排序。Po-Sen Huang 等人²给出了基于用户点击数据学习查询和文档相关性的 DSSM 模型，该模型的架构如下所示：

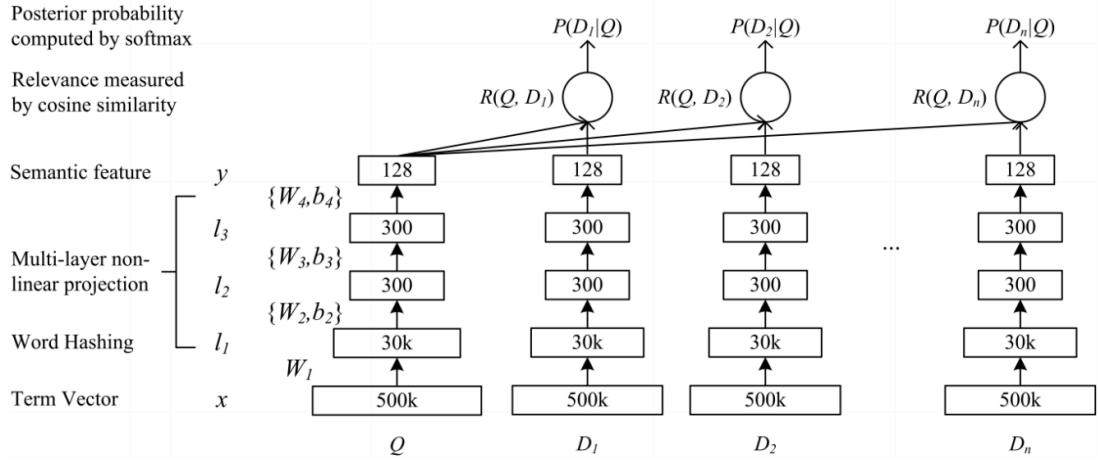


图 10.2: Illustration of the DSSM. It uses a DNN to map high-dimensional sparse text features into low-dimensional dense features in a semantic space. The first hidden layer, with 30k units, accomplishes word hashing. The word-hashed features are then projected through multiple layers of non-linear projections. The final layer's neural activities in this DNN form the feature in the semantic space.

上述架构图中 Term Vector 这一层采用的词表大小为 500k，因此对于的向量维度为 500k，即将 query 和 doc 切词后，采用 one-hot 编码方案表示为 500k 大小的词向量，若直接将该词向量应用于前向网络，则 $W_1 \in \mathbb{R}^{500k \times 30k}$ ，若采用 float32 类型存储该权重矩阵，则需要约 56GB 大小内存或显存空间，为了解决该问题， l_1 层的变换采用 Word Hashing 的方案，即将所有词（譬如：good）加上头尾表示符 #（譬如：#good#），在此基础上将所有词表示为字构成的 3 元（譬如：#go, goo, ood, od#），经过该处理后，词表大小由 500k 降低至 30k。因此查询 Q 和文档 D 最终均采用三元词表表示，该词表大小 30k。假设 x 表示输入词向量， y 表示输出向量。该网络架构对应的公式如下：

$$l_1 = W_1 x \quad (10.17)$$

$$l_i = \tanh(W_i l_{i-1} + b_i), \quad i = 2, \dots, N-1 \quad (10.18)$$

$$y = f(W_N l_{N-1} + b_N) \quad (10.19)$$

查询 Q 和文档 D 间的语义相似度为通过如下公式衡量：

$$R(Q, D) = \text{cosine}(y_Q, y_D) = \frac{y_Q^\top \cdot y_D}{\|y_Q\| \|y_D\|} \quad (10.20)$$

由网络架构图可知上述公式中的 y_Q 和 y_D 为查询和文档的语义向量表示。应用于网页搜索时，可以根据文档和当前查询的语义相关性得分进行排序。

现在我们来看如何对 DSSM 网络进行训练，假设给定查询 Q 及返回的文档列表 D_1, \dots, D_n ，假设该查询下发生点击的文档为 D^+ 为相关文档，没有发生的点击的文档为不相关文档，假设集合 \mathbb{D} 表示要排序的文档集合，集合 \mathbb{D} 由文档 D^+ 及 4 个随机抽取的未点击文档集

²Learning Deep Structured Semantic Models for Web Search using Clickthrough Data

$\{D_j^- : j = 1, \dots, 4\}$ 构成。因此给定查询 Q 和文档 D 和查询的语义相似度得分后，可以计算每个文档 D 在查询 Q 下的相似度后验概率概率如下：

$$P(D|Q) = \frac{\exp(\gamma R(Q, D))}{\sum_{D' \in \mathbb{D}} \exp(\gamma R(Q, D'))} \quad (10.21)$$

上述公式中 γ 为平滑系数，可以通过验证集调优选取。因此可以通过最小化如下 Cross-Entropy Loss 完成网络架构中的参数训练：

$$\mathcal{L}(\Lambda) = \frac{1}{N} - \log \prod_{Q, D^+} P(D^+|Q), \quad (\Lambda = \{\mathbf{W}_i, \mathbf{b}_i\}) \quad (10.22)$$

10.1.4 推荐问题

我们日常生活中不论看新闻资讯还是看长短视频或访问电子商务类网站，一般都能得到我们喜欢的信息，这背后推荐技术起到了至关重要的作用，这类技术一般涉及到对四类信息的处理，这四类信息分别为：

- 用户维度：年龄、性别、位置、标签等
- 用户行为维度：用户过去点击过的商品或新闻列表或看的视频列表等
- 场景维度：用户当前访问所用的设备、时间、语言等信息
- 条目维度：当前条目的 id，标签信息等

通过前向神经网络融合这四类信息，最终表现为用户 U (User) 对当前条目 I (Item) 感兴趣的概率。因此这类问题最终转化为二分类问题，即用户点击当前条目的概率，可以通过 logistic regression 表示，模型参数搜索可以通过最小化 Cross-Entropy Loss 完成。这里以 Youtube 视频推荐为例来看前向网络在推荐问题上的应用，Paul Covington 等人³给出了前向网络在 Youtub 推荐问题上的两阶段应用，推荐架构总体如下所示：

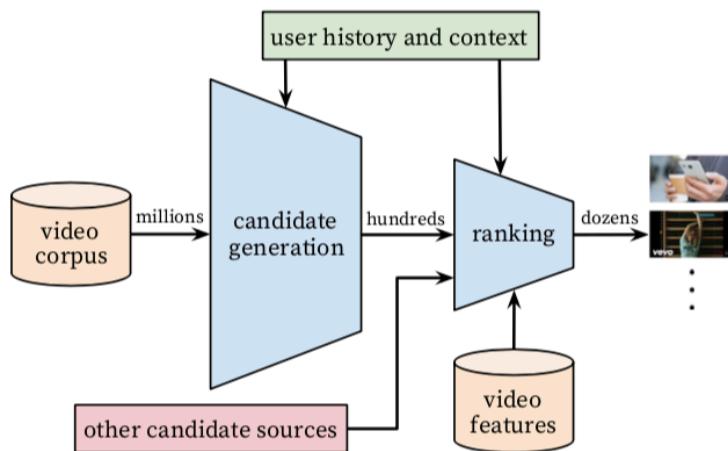


图 10.3: Recommendation system architecture demonstrating the "funnel" where candidate videos are retrieved and ranked before presenting only a few to the user.

从上述推荐架构图可见，推荐过程分为两个阶段

³Deep Neural Networks for YouTube Recommendations

- 备选生成阶段：该阶段需要从 Youtube 全量数据中生成几百个视频备选
- 排序阶段：该阶段对几百个视频备选进行精排序以便优化用户观看时长

备选生成

该阶段需要从 Youtube 全量数据中生成几百个视频备选，这篇 Paul Covington 等人将该问题看成是一个极端多分类问题，及给定用户 U 和场景 C (Context) 的情况下，用户在时间 t 观看 (w_t) 全量库 \mathbb{V} 中视频 i 的概率，公式描述如下：

$$P(w_t = i | U, C) = \frac{e^{\mathbf{v}_i^\top \cdot \mathbf{u}}}{\sum_{j \in \mathbb{V}} e^{\mathbf{v}_j^\top \cdot \mathbf{u}}} \quad (10.23)$$

其中 $\mathbf{u} \in \mathbb{R}^d$ 表示用户维度的向量表示， \mathbf{v}_j 表示每个备选视频的向量表示。考虑到全量视频库 \mathbb{V} 视频数一般为百万级，因此若直接将上述公式应用于 Cross-Entropy Loss，则学习效率低下，文章中提及了 Sebastien Jean 等人⁴在机器翻译领域解决大词表导致归一化计算开销较高问题的工作，但其紧接着提及针对每个正样本最小化正类别的交叉熵损失，同时抽样若干负类别最小化负样本交叉熵损失，因此分析其训练方法更加类似于 Tomas Mikolov 等人采用的负样本采样方法，而且问题定义类似，均是学习向量表示。实际训练过程中可以采用带权重的负采样 (Negative Sampling with Importance Weighting) 方法，对于每个正样本采样的负样本数为几千个。如下是 Tomas Mikolov 等人⁵给出的带权重负采样方法：

$$\log \sigma(\mathbf{v}_i^\top \cdot \mathbf{u}) + \sum_{k=1}^m E_{v_k \sim P_n(v)} [\log \sigma(-\mathbf{v}_k^\top \cdot \mathbf{u})] \quad (10.24)$$

上述公式中 v_i 表示正样本， v_k 表示负样本。 $P_n(v)$ 表示采样权重，Mikolov 等人在文章中建议采用一元权重，在 Youtube 视频推荐场景下可以考虑采用视频观看时长 T_{v_k} 来类似每个词发生的次数，并对该值取 $3/4$ 次方，概率质量函数如下：

$$P_n(v_k) = \frac{T_{v_k}^{3/4}}{\sum_{j=1}^m T_{v_j}^{3/4}} \quad (10.25)$$

可以看到对公式 10.24 取负，并最小化该值，即可以搜索网络架构中涉及的权重参数。

下面我们来看如何通过前向网络将用户表示为向量 \mathbf{u} ，将视频表示为向量 \mathbf{v} ，视频向量表示可以通过查 embedding 表直接生成，考虑到视频自身属性不会发生变化，因此召回阶段可以这样表示。Paul Covington 等人给出图 10.4 描述的备选生成网络架构。这里不对该前向网络进行进一步解释，具体可参考该论文。通过备选生成网络生成几百个备选视频后需要对这些视频进行进一步排序，以便优化最终的观看时长。

排序

该阶段的主要任务是预测训练样本的期望观看时长，训练样本中的正样本为有点击的视频曝光，负样本为无点击的视频曝光，每条正样本含有用户观看视频的时长信息。为了

⁴On Using Very Large Target Vocabulary for Neural Machine Translation

⁵Distributed Representations of Words and Phrases and their Compositionality

预测用户对视频的期望观看时长，Paul Covington 等人采用了加权逻辑回归方法。模型训练采用 Cross-Entropy Loss 对逻辑回归函数进行训练，训练时对正样本采用观看时长进行加权，负样本的对应的加权系数为 1。通过该方法逻辑回归函数学习到的 odds 即为用户 U 观看视频 I 的期望时长。

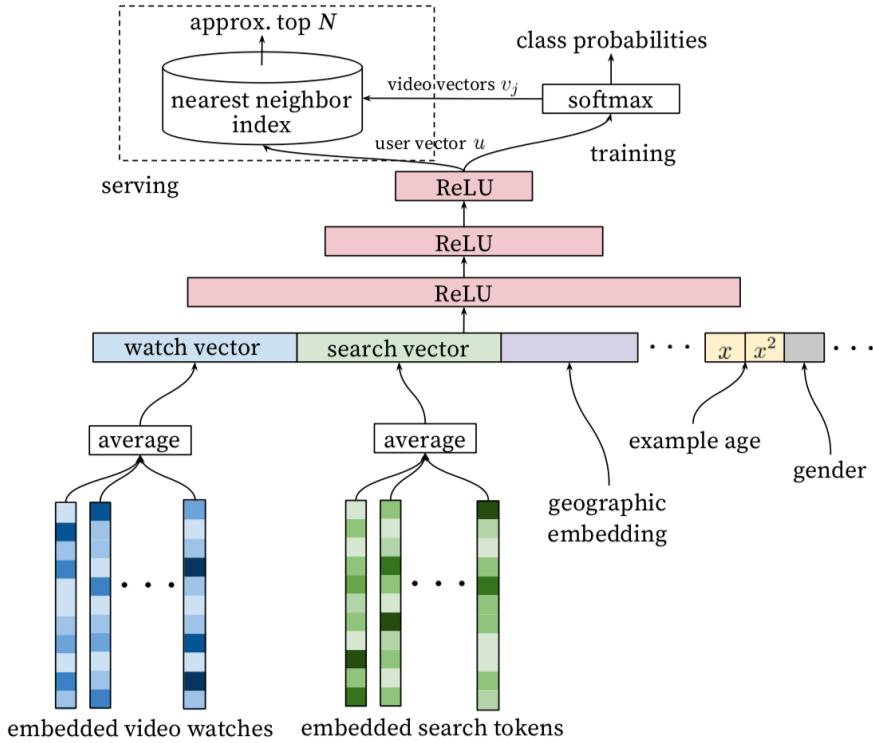


图 10.4: Deep candidate generation model architecture showing embedded sparse features concatenated with dense features. Embeddings are averaged before concatenation to transform variable sized bags of sparse IDs into fixed-width vectors suitable for input to the hidden layers. All hidden layers are fully connected. In training, a cross-entropy loss is minimized with gradient descent on the output of the sampled softmax. At serving an approximate nearest neighbor lookup is performed to generate hundreds of candidate video recommendations.

这里有必要对 odds 进行一定的解释，odds 表示样本属于某个类别的概率和不属于该类别的概率的比值，公式如下：

$$\text{odds} = \frac{P(y = \text{true}|x)}{1 - P(y = \text{true}|x)} \quad (10.26)$$

考虑到对于所有正样本均引入了观看时长 T_i 作为样本权重，因此正样本发生的概率由 $P(y = \text{true}|x_i)$ 变为 $T_i P(y = \text{true}|x_i)$ ，因此有：

$$\text{odds}_i = \frac{T_i P(y = \text{true}|x_i)}{1 - T_i P(y = \text{true}|x_i)} \quad (10.27)$$

考虑到 $P(y = \text{true}|x)$ 很小，上式可进一步简化为如下形式：

$$\text{odds}_i = \frac{T_i P(y = \text{true}|x_i)}{1 - T_i P(y = \text{true}|x_i)} \approx T_i P(y = \text{true}|x_i) = E[T_i] \quad (10.28)$$

由上述公式可见采用观看时长 T_i 对正样本进行加权后， odds_i 即为用户的观看的期望。

下面我来求取 odds 对应的计算形式，对 odds 取自然对数，即可得到公式：

$$\ln(\text{odds}) = \Theta^\top \cdot f \quad (10.29)$$

其中 Θ 表示模型参数， f 表示样本 x 对应的特征向量。上述公式中的左边部分称为 **logit** 函数：

$$\text{logit}(P(y = \text{true}|x)) = \ln(\text{odds}) = \Theta^\top \cdot f \quad (10.30)$$

对上述公式两边进行指数变换，即得到 odds 的计算形式：

$$\text{odds} = e^{\Theta^\top \cdot f} \quad (10.31)$$

上述公式即为论文中提及的用于 Youtube 线上视频排序阶段采用的排序公式，该公式对应用户 U 观看视频 I 的期望观看时长。排序阶段采用的前向网络架构如下所示：

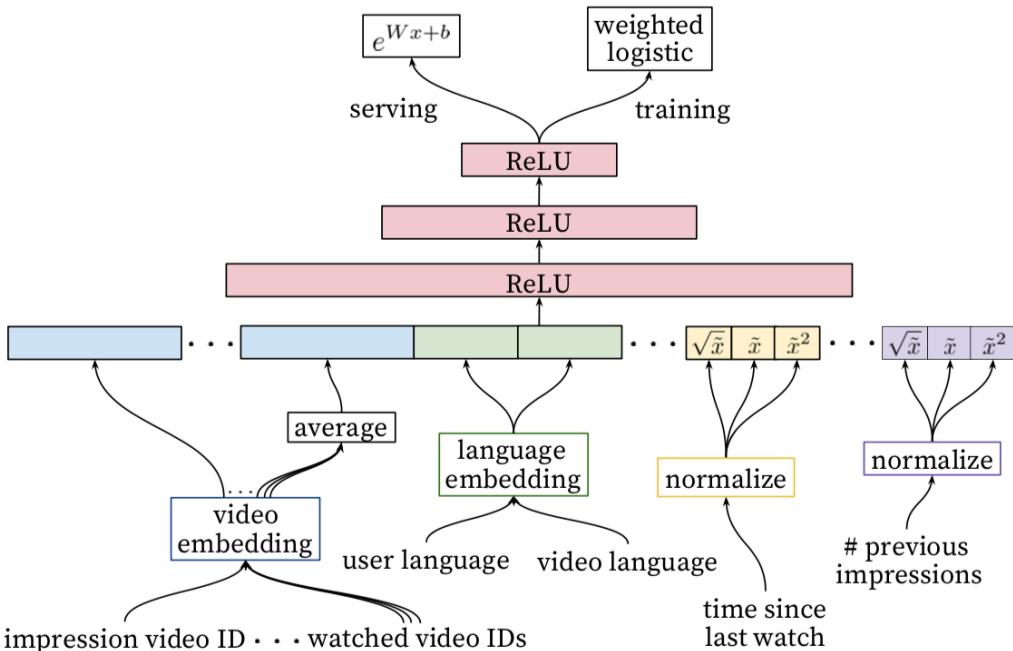


图 10.5: Deep ranking network architecture depicting embedded categorical features (both univalent and multivalent) with shared embeddings and powers of normalized continuous features. All layers are fully connected. In practice, hundreds of features are fed into the network.

这里不对该前向网络进行进一步解释，具体可参考该论文。最后需要说明的是无论是备选生成网络还是排序网络，在输入层均涉及到大量的离散特征需要进行向量化映射操作，并将映射后的向量作为上层网络的输入，考虑到离散特征的规模往往在十亿以上的级别，因此可以将上层稠密网络和输入层的向量化映射在模型定义层面分开处理，譬如在 tensorflow 中可以以 placeholder 的形式定义对应的下层网络的向量化映射的输入，这样拆分带来的好处是在训练阶段可以采用参数服务器进行模型训练，并在保存模型阶段将上层模型和稀疏映射向量分开保存，这样线上服务时可以在 tensorflow serving 中只加载上层稠密模型，并将稀疏特征对应的向量存储到 redis 这类分布式存储中。

10.2 卷积网络

本节首先回顾卷积网络结构，在此基础上进一步回顾卷积网络在图像分类和文本分类问题上的应用。

10.2.1 网络结构

卷积神经网络和前向网络有所不同是，卷积神经网络假设输入为 N 维结构（一般为 3 维）而非前向网络中的向量，卷积神经网络采用卷积（Convolution）、池化（Pooling）及全连接处理这类输入数据。如下是包含这三类操作的 VGG16 对应的卷积网络架构：

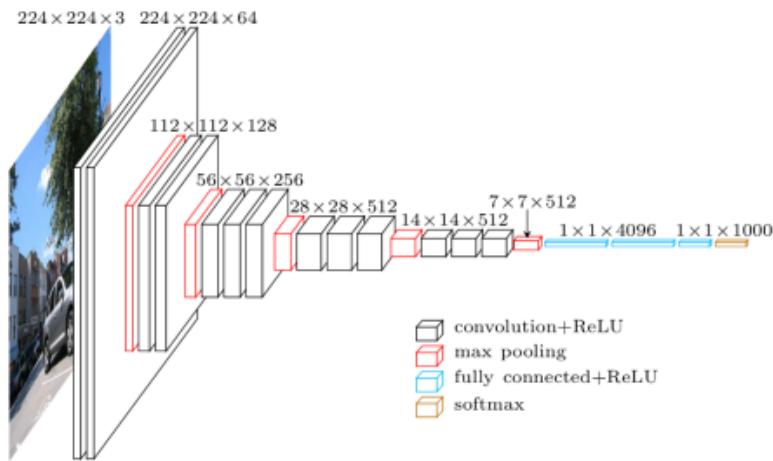


图 10.6: VGG16 网络架构

该网络架构由牛津 Visual Geometry Group 团队 Karen Simonyan 等人⁶于 2014 年提出，并在 ImageNet ILSVRC-2012 测试集上取得了 top-5 分类错误率 7.3% 的成绩。

10.2.1.1 Convolution

实际应用中卷积网络中的卷积核基本为离散卷积运算，假设输入数据的高度为 5，宽度为 5，通道数为 1，图10.7是对该数据应用 $3 \times 3 \times 1 \times 1$ 的卷积操作过程⁷。从运算过程可见，输出数据中的每个元素满足如下公式：

$$y = \mathbf{w}^\top \cdot \mathbf{x} + b \quad (10.32)$$

其中 \mathbf{w} 对应输出数据中某个通道对应的卷积核 $3 \times 3 \times 1$ 展平后的向量， \mathbf{x} 对应输入数据中所有通道对应卷积核运算区域的所有输入数据展平后的向量， b 为偏执项， y 为卷积运算后的输出，对于输出数据中的同一通道应用同样卷积核。离散卷积核的形状定义为：

⁶Very Deep Convolutional Networks for Large-Scale Image Recognition

⁷A guide to convolution arithmetic for deep learning

(k_1, \dots, k_N, m, n) , 其中:

$n \equiv$ number of output feature maps

$m \equiv$ number of input feature maps

$k_j \equiv$ kernel size along axis j.

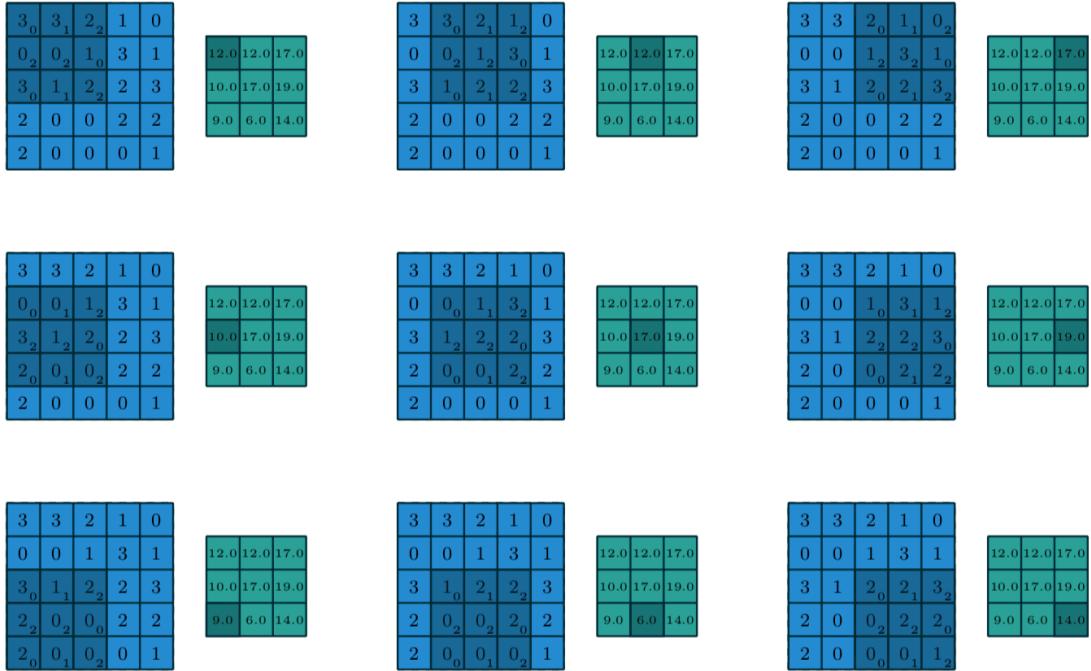


图 10.7: 计算离散卷积操作后的输出值

如下属性影响卷积层在维度 j 的输出大小 o_j :

- i_j : 输入数据在维度 j 的大小
- k_j : 卷积核在维度 j 的大小
- s_j : 沿着维度 j 的移动步长 (两个连续卷积核和之间的距离)
- p_j : 沿着维度 j 分别在该维度开始和结尾处垫的数值 0 (zero padding) 的个数

下图展示了将 $3 \times 3 \times 1$ 卷积核应用到经过 1×1 padding 处理后的 $5 \times 5 \times 1$ 输入数据的过程。

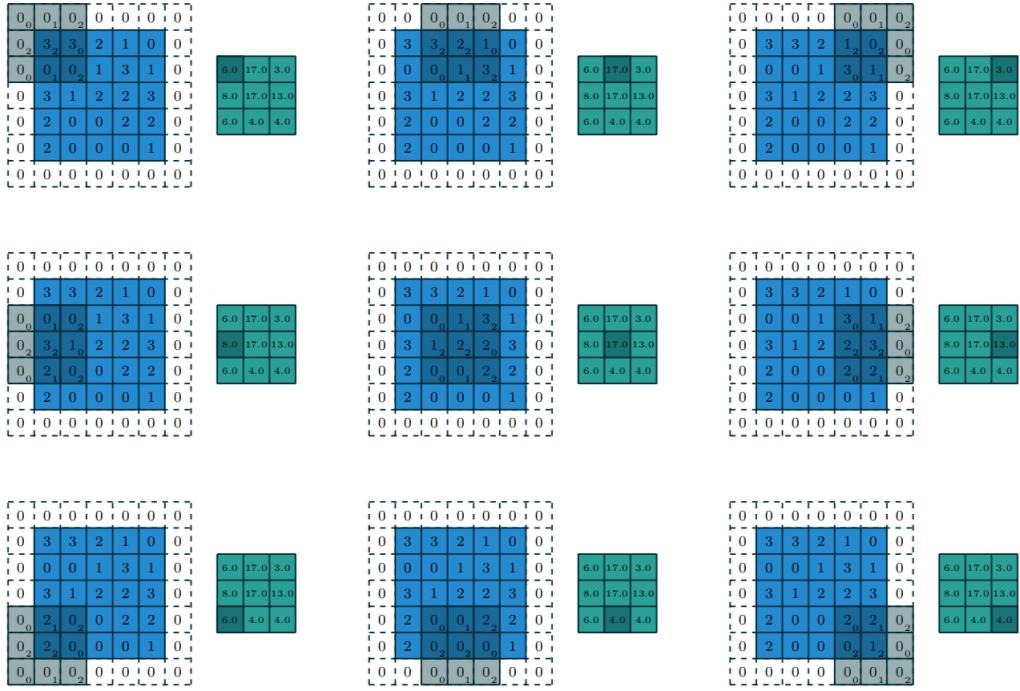


图 10.8: 对于 $i_1 = i_2 = 5, k_1 = k_2 = 3, s_1 = s_2 = 2$ 及 $p_1 = p_2 = 1$ 应用离散卷积操作后的输出值

基于上述离散卷积应用过程，输出数据 o 满足如下公式：

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1. \quad (10.33)$$

10.2.1.2 Pooling

池化操作构成了卷积神经网络的另一个重要组成部分，池化操作通过对子区域取平均值或最大值池化函数降低输入数据 (feature map) 的大小。池化操作将滑动窗口内所有输入数据的内容送给池化函数。图10.9展示了取平均值池化操作：

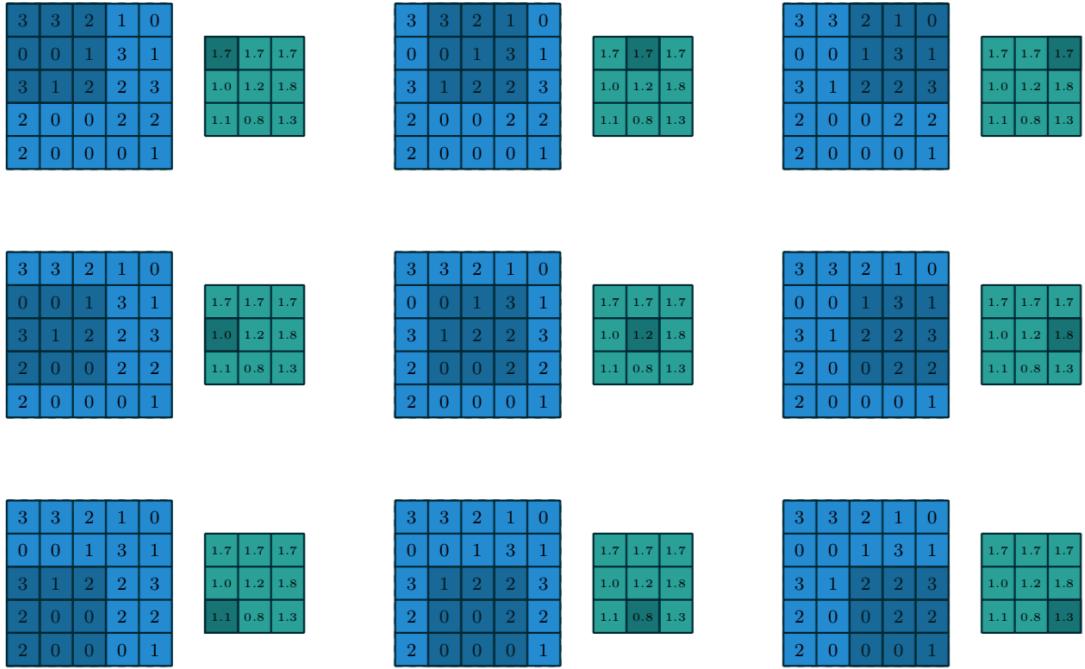


图 10.9: Computing the output values of a 3×3 average pooling operation on a 5×5 input using 1×1 strides

图10.10展示了取最大值池化操作。基于池化应用过程，输出数据大小满足如下公式：

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1. \quad (10.34)$$

10.2.1.3 Padding

卷积神经网络中有 3 类 padding 操作，分别为 valid padding, same padding 和 full padding。

Valid Padding

该模式下不对输入数据做任何的 padding 操作，此时输出数据大小满足公式10.34，图10.9和图10.10为 Valid Padding 的应用样例。

Same Padding

应用该类型 padding 操作后，若步长为 1，输出数据的大小和输入数据的大小一致。因此 padding 的大小满足如下公式：

$$p = \lfloor k/2 \rfloor \quad (10.35)$$

此时输出数据的大小满足公式10.33，图10.8为 Same Padding 的应用样例。

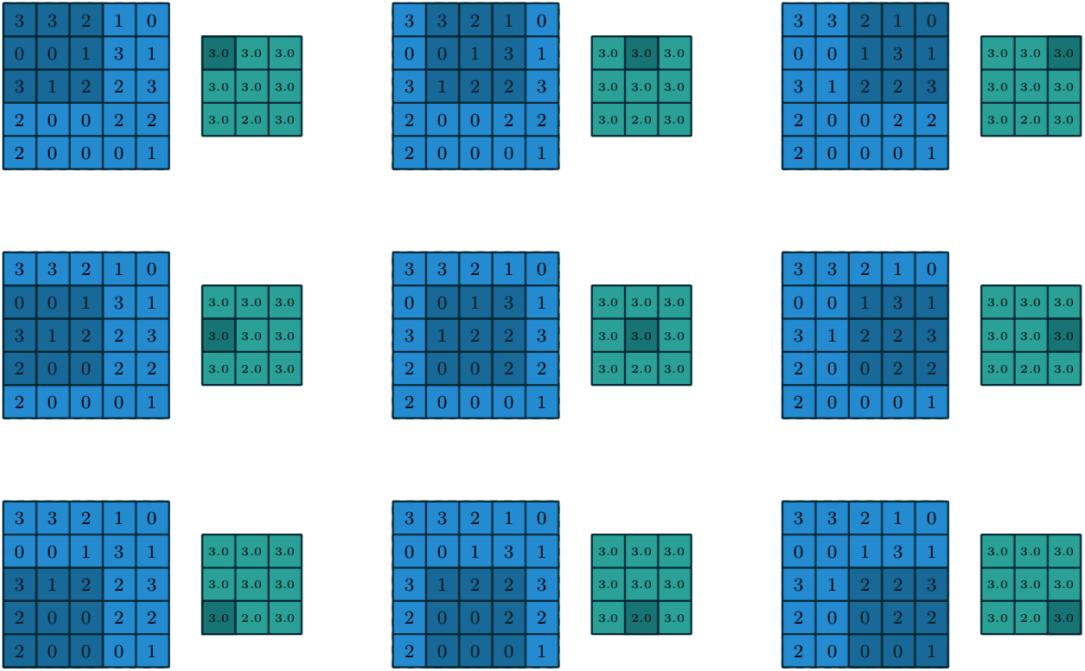


图 10.10: Computing the output values of a 3×3 max pooling operation on a 5×5 input using 1×1 strides

Full Padding

通常经过卷积运算后输出数据的大小相对输入大小常常变小，通过合适的 zero padding 也可以让输出数据大小变大。譬如若 $p = k - 1$ ，步长 $s = 1$ 时，则有：

$$\begin{aligned} o &= i + 2(k - 1) - k + 1 \\ &= i + (k - 1) \end{aligned} \tag{10.36}$$

10.2.2 图像分类

图像分类领域另一项具有里程碑意义的工作是由 KaiMing He 等人⁸提出的 Residual Network，该网络结构中通过引入短路恒等映射（Identity Mapping by Shortcuts），将网络的深度推至 152 层，并在 ILSVRC 2015 竞赛上将图像 top 5 分类错误率降至 3.57%。

10.2.2.1 短路恒等映射

Residual Network 相对于之前的卷积神经网络架构，引入如下图所示的短路恒等映射（Identity Mapping by Shortcuts）：

⁸Deep Residual Learning for Image Recognition

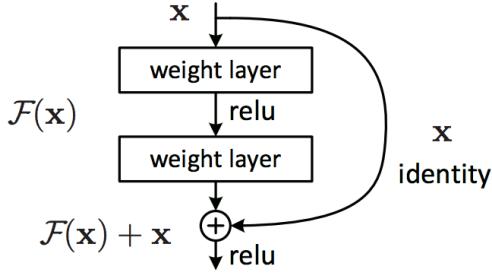


图 10.11: Residual learning: a building block.

该网络对应的公式描述如下：

$$\mathbf{Y} = \mathcal{F}(\mathbf{X}, \{\mathbf{W}_i\}) + \mathbf{X}. \quad (10.37)$$

可以看出上述公式中假设张量 \mathbf{X} 和 \mathbf{Y} 的形状一致，若不一致其对应的公式为：

$$\mathbf{Y} = \mathcal{F}(\mathbf{X}, \{\mathbf{W}_i\}) + \mathbf{W}_s \mathbf{X}. \quad (10.38)$$

\mathbf{W}_s 用于对 \mathbf{X} 进行变换使得输出形状和 \mathbf{Y} 一致。这里有必要说明的是通过引入短路恒等映射，解决了传统网络中由于网络加深导致梯度反向传播后可能出现的梯度消散问题。如下是短路恒等映射对应的变换过程：

$$\mathbf{X}_{l+1} = \mathbf{X}_l + \mathcal{F}(\mathbf{X}_l) \quad (10.39)$$

$$\begin{aligned} \mathbf{X}_{l+2} &= \mathbf{X}_{l+1} + \mathcal{F}(\mathbf{X}_{l+1}) \\ &= \mathbf{X}_l + \mathcal{F}(\mathbf{X}_l) + \mathcal{F}(\mathbf{X}_{l+1}) \end{aligned} \quad (10.40)$$

$$\Rightarrow \mathbf{X}_L = \mathbf{X}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{X}_i) \quad (10.41)$$

因此有：

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}_l} = \frac{\partial \mathcal{L}}{\partial \mathbf{X}_L} \frac{\partial \mathbf{X}_L}{\partial \mathbf{X}_l} = \frac{\partial \mathcal{L}}{\partial \mathbf{X}_L} \left(1 + \frac{\partial}{\partial \mathbf{X}_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{X}_i)\right) \quad (10.42)$$

由上述公式可见梯度 $\frac{\partial \mathcal{L}}{\partial \mathbf{X}_L}$ 可以在反向传播过程中直接传播给任意 $\frac{\partial \mathcal{L}}{\partial \mathbf{X}_l}$ ，另外 $\frac{\partial \mathcal{L}}{\partial \mathbf{X}_l}$ 对应的公式为求和形式，因此不太可能出现梯度消散。

10.2.2.2 网络结构

Residual Network 不同深度下的网络结构设置如下表所示：

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer	
conv1	112×112			7×7, 64, stride 2			
				3×3 max pool, stride 2			
conv2_x	56×56	$\left[\begin{array}{l} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	
conv3_x	28×28	$\left[\begin{array}{l} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 8$	
conv4_x	14×14	$\left[\begin{array}{l} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 6$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 6$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 23$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 36$	
conv5_x	7×7	$\left[\begin{array}{l} 3 \times 3, 512 \\ 3 \times 3, 512 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 512 \\ 3 \times 3, 512 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$	
	1×1			average pool, 1000-d fc, softmax			
	FLOPs	1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9	

图 10.12: Architectures for ImageNet. Building blocks are shown in brackets, with the numbers of blocks stacked. Downsampleing is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

Residual Network 网络中采用的短路恒等映射思路，后续被应用到了目前在文本等领域火热的 Transformer 网络架构。

10.2.3 文本分类

文本分类的目的是给定一段文本后，将给定文本分类到若干类别中的某个类别（从 m 个类别中选择一个类别）或几个类别（一般转化为 m 个二分类问题）。将卷积神经网络应用至文本分类领域具有重要影响力的工作是由 Yoon Kim⁹提出的用于句子分类的 CNN 模型，该模型结构如下所示：

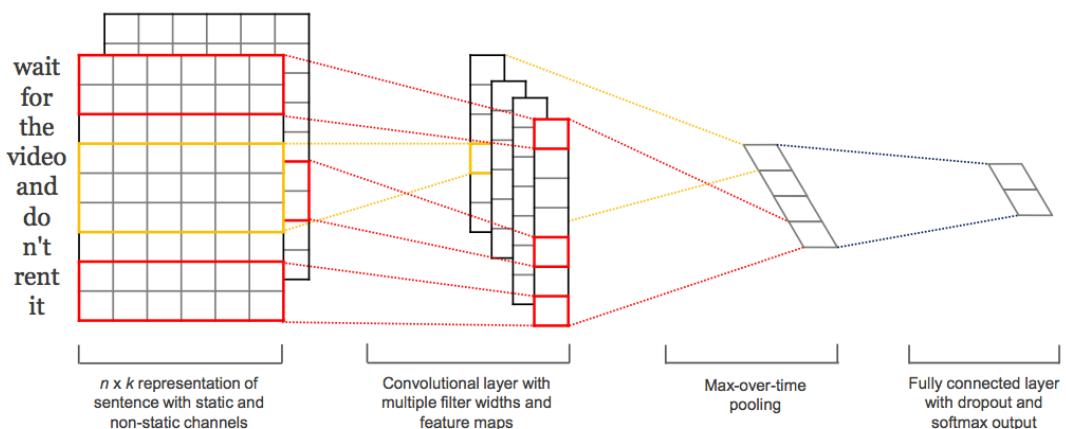


图 10.13: Model architecture with two channels for an example sentence.

从上述架构图可见，该架构中将所有词 $x_i \in \mathbb{R}^k$ 分别表示为 k 维词向量，上述架构中由于采用了双通道，因此每个词对应两个 k 维词向量，因此长度为 n 的句子（长度小于 n 需要采用零向量进行填充，长度大于 n 需要截断）表示为：

$$x_{1:n} = x_1 \oplus x_2 \oplus \cdots \oplus x_n, \quad (10.43)$$

⁹Convolutional Neural Networks for Sentence Classification

其中 \oplus 为拼接运算， $\mathbf{x}_{i:i+j}$ 表示对 $\mathbf{x}_i, \mathbf{x}_{i+1}, \dots, \mathbf{x}_{i+j}$ 执行拼接操作，这里采用的卷积核 $\mathbf{W} \in \mathbb{R}^{h \times k}$ ，该卷积核应用于窗口大小为 h 的所有词，产生特征 c_i ，如下是对窗口 $\mathbf{x}_{i:i+h-1}$ 执行卷积运算对应的特征生成公式：

$$c_i = f(\mathbf{W} \cdot \mathbf{x}_{i:i+h-1} + b). \quad (10.44)$$

其中 $b \in \mathbb{R}$ 为偏执项， f 为非线性变换函数，譬如 \tanh ，应用上述卷积时采用的步长为 1，因此对应的词窗口集合为 $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \dots, \mathbf{x}_{n-h+1:n}\}$ ，应用卷积运算后生成的特征映射 (feature map) 如下：

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \quad (10.45)$$

因此有 $\mathbf{c} \in \mathbb{R}^{n-h+1}$ 。在此基础上应用取最大值池化运算，最终当前卷积核生成一个标量 $\hat{c} = \max\{\mathbf{c}\}$ ，因此每个卷积经过上述操作后生成的 \hat{c} 表示该卷积提出出的最重要特征。实际应用中窗口 h 可以去不同值，并分别对每个类型窗口 h 取 l 个卷积，因此最终每句话对应的特征表示为 $\mathbf{z} = [\hat{c}_1, \dots, \hat{c}_m]$ 的向量，向量维度为 $m = l \times h$ 。

最后一层为全连接层，将向量 \mathbf{z} 应用于全连接层前，Yoon Kim 采用 dropout 加强模型的泛化能力，并同时在损失函数中加入对模型中除偏执项的所有权重参数的 L_2 正则化，引入 dropout 后的全连接层表示如下：

$$\mathbf{y} = \mathbf{W} \cdot (\mathbf{z} \odot \mathbf{r}) + b \quad (10.46)$$

\odot 为 hadamard 运算，表示对向量 \mathbf{z} 和向量 \mathbf{r} 执行元素依次相乘。 $\mathbf{r} \in \mathbb{R}^m$ 向量的元素为 Bernoulli 随机变量，其取值为 1 的概率为 p 。该架构在总体而言简单效果也很不错，实际应用中可以在输入层引入更多特征，并采用 Attention 机制优化总体效果。

10.3 循环网络

实际中我们常常遇到一类数据为序列数据，序列中后续的数据依赖于前面的历史。譬如当我们度一篇文章时，对文章中每个词的理解依赖于前驱词，我们并不会每理解一个词均需要从头把文章再读一遍，可见我们的思考具有持久化能力。前面介绍的前向网络和卷积网络均不具有这样的能力，譬如想象我们希望对电影中每一帧进行分类来确定具体发生什么事件，前向网络和卷积网络在如何应用其对历史事件的分类信息对当前事件进行分类上，看上去不是那么直观。本节介绍的循环神经网络 (Recurrent Neural Network) 由于引入记忆能力对这类数据具有较好的梳理能力，首先我们会介绍 Simple RNN 结构，并介绍该网络存在的问题，在此基础上介绍 LSTM RNN 网络结构，最后我们介绍 RNN 网络在机器翻译和命名实体识别问题上的应用。

10.3.1 网络架构

本节首先回顾 Simple RNN 网络架构，在此基础上介绍 LSTM 网络架构

10.3.1.1 Simple RNN

前向神经网络或卷积神经架构无法有效表示历史信息，Simple RNN（又称为 Elman network¹⁰）通过引入对历史隐含信息的环状依赖，使得信息可以持久化，如下是 Simple RNN 的架构图（注：这里涉及 Simple RNN 及后续的 LSTM 架构图均参考至¹¹）：

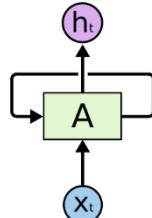


图 10.14: Recurrent Neural Networks have loops

从上述架构图可见，输出 \mathbf{h}_t 依赖于当前输入 \mathbf{x}_t ，及系统最近的历史状态 \mathbf{h}_{t-1} ，因此其对应如下计算形式：

$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c}) \quad (10.47)$$

其中 \mathbf{V} 为网络权重参数， \mathbf{c} 为偏执项， g 为激活函数。Simple RNN 某个时刻的依赖，可以展开为如下形式：

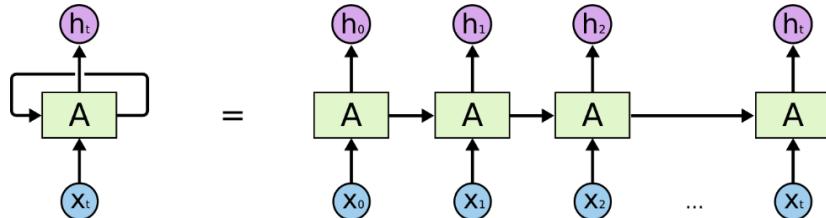


图 10.15: An unrolled Recurrent Neural Network

上述架构图清晰展现了 t 时刻的输出 \mathbf{h}_t 对历史的依赖关系。Simple RNN 通过引入环形表示解决了对历史信息持久化问题，但是 Simple RNN 表示方式决定了其在长距离依赖时，会出现梯度爆炸或梯度消散问题。假设将 Simple RNN 直接应用于语言模型任务，通过对 \mathbf{h}_t 进行如下变换，即可以求得在给定历史序列下，后续词的概率分布：

$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c}) \quad (10.48)$$

$$\hat{\mathbf{p}}_t = \text{softmax}(\mathbf{W}\mathbf{h}_t + \mathbf{b}) \quad (10.49)$$

$$\mathcal{L}_t = -\log(\hat{\mathbf{p}}_t \cdot \text{onehot}(x_{t+1})) \quad (10.50)$$

因此 T 时刻的梯度反向传播至初始时刻，满足如下关系：

$$\frac{\partial \mathcal{L}_T}{\partial h_1} = \frac{\partial \mathcal{L}_T}{\partial \hat{p}_T} \frac{\partial \hat{p}_T}{\partial h_T} \left(\prod_{t \in \{T, \dots, 2\}} \frac{\partial h_t}{\partial h_{t-1}} \right) \quad (10.51)$$

¹⁰Finding Structure in Time

¹¹<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

这里对 \mathbf{h}_t 的表示形式，进一步应用复合函数表示为如下形式：

$$\mathbf{z}_t = \mathbf{V}_x \mathbf{x}_t + \mathbf{V}_h \mathbf{h}_{t-1} + \mathbf{c} \quad (10.52)$$

$$\mathbf{h}_t = g(\mathbf{z}_t) \quad (10.53)$$

则有：

$$\begin{aligned} \frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \\ &= \mathbf{V}_h \text{diag}(g'(z_t)) \end{aligned} \quad (10.54)$$

$$\frac{\partial \mathcal{L}_T}{\partial h_1} = \left(\prod_{t \in \{T, \dots, 2\}} \mathbf{V}_h \text{diag}(g'(z_t)) \right) \left(\frac{\partial \hat{p}_T}{\partial h_T} \right)^T \frac{\partial \mathcal{L}_T}{\partial \hat{p}_T} \quad (10.55)$$

令矩阵 \mathbf{V}_h 的二范数为 α , 矩阵 $\text{diag}(g'(z_t))$ 的二范数为 β , 若 $\alpha\beta > 1$ 则出现梯度爆炸, 若 $\alpha\beta < 1$ 则出现梯度消散, 若 $\alpha\beta = 1$ 则线性系统处于稳态, 梯度可以较好传播。

10.3.1.2 LSTM

Long Short Term Memory networks 通常简称为 LSTM 为一种特殊类型的 RNN, 该网络可以学习长距离依赖, 该网络由 Hochreiter 及 Schmidhuber 提出¹²。如下是 Simple RNN 的网络展开后的依赖架构图:

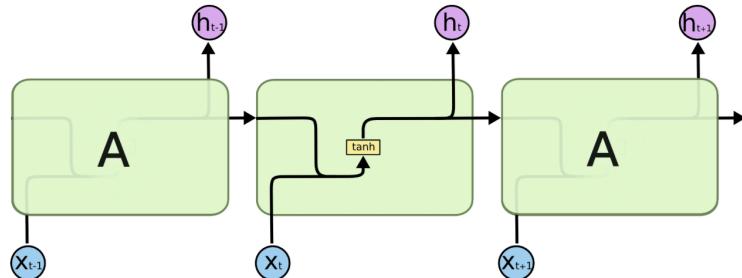


图 10.16: The repeating module in a standard RNN contains a single layer

LSTM 在 Simple RNN 基础上引入节点状态信息，并通过遗忘门和输入门更新节点状态信息，通过输出门控制更新后的节点状态信息输出。网络结构如下：

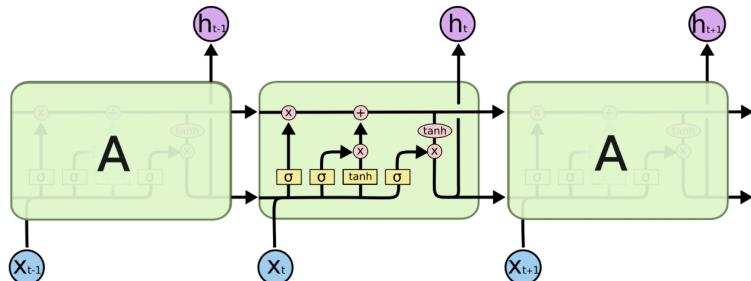


图 10.17: The repeating module in an LSTM contains four interacting layers.

¹²LONG SHORT-TERM MEMORY

下面我们分别来看遗忘门，输入门和输出门是如何工作的。

遗忘门

遗忘门在 LSTM 架构图中所处的位置如下图所示：

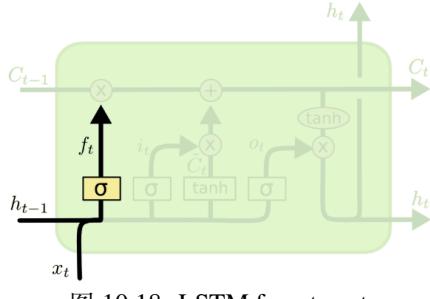


图 10.18: LSTM forget gate

遗忘门对应的变换公式如下：

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (10.56)$$

从上述变换可见遗忘门可以达到将历史节点状态信息丢弃的作用。

输入门

输入门在 LSTM 架构图中所处的位置如下图所示：

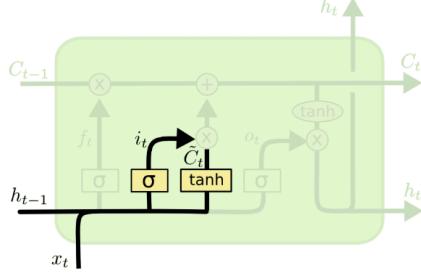


图 10.19: LSTM input gate

输入门对应的公式如下：

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (10.57)$$

$$\hat{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \quad (10.58)$$

更新状态

结合遗忘门和输入门即可对 LSTM 状态进行更新：

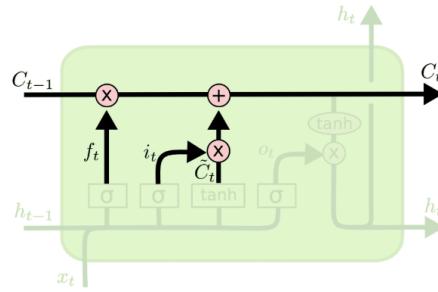


图 10.20: LSTM cell update

LSTM 状态更新对应的公式如下：

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \quad (10.59)$$

输出门

输出门在 LSTM 架构图中所处的位置如下图所示：

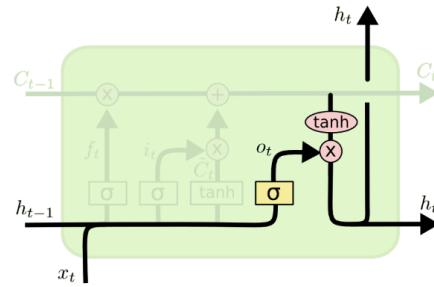


图 10.21: LSTM output gate

结合节点状态信息和输出门即可生成当前的特征表示 \mathbf{h}_t ：

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (10.60)$$

$$\mathbf{h}_t = o_t \odot \tanh(c_t) \quad (10.61)$$

下面我们来看，为何 LSTM 可以解决 Simple RNN 长距离依赖存在的梯度爆炸和梯度消散问题。假设将 LSTM 应用于语言模型任务，通过对 \mathbf{h}_t 进行如下变换，即可以求得在给定历史序列下，后续词的概率分布：

$$\hat{\mathbf{p}}_t = \text{softmax}(W\mathbf{h}_t + b) \quad (10.62)$$

$$\mathcal{L}_t = -\log(\hat{\mathbf{p}}_t \cdot \text{onehot}(x_{t+1})) \quad (10.63)$$

因此 T 时刻的梯度反向传播至初始时刻，满足如下关系：

$$\frac{\partial \mathcal{L}_T}{\partial h_1} = \frac{\partial \mathcal{L}_T}{\partial \hat{p}_T} \frac{\partial \hat{p}_T}{\partial h_T} \left(\prod_{t \in \{T, \dots, 2\}} \frac{\partial h_t}{\partial h_{t-1}} \right) \quad (10.64)$$

由 LSTM 的表示形式可知：

$$z_{o_t} = \mathbf{W}_{oh} \mathbf{h}_{t-1} + \mathbf{W}_{ox} \mathbf{x}_t + \mathbf{b}_o \quad (10.65)$$

$$\Rightarrow \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) = \sigma(z_{o_t}) \odot \tanh(\mathbf{c}_t) \quad (10.66)$$

$$\begin{aligned} \Rightarrow \frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial h_t}{\partial o_t} \frac{\partial o_t}{\partial z_{o_t}} \frac{\partial z_{o_t}}{\partial h_{t-1}} + \text{diag}(\tanh'(\mathbf{c}_t)) \frac{\partial c_t}{\partial h_{t-1}} \\ &= \mathbf{W}_{oh} \text{diag}(\tanh'(z_{o_t})) + \frac{\partial \mathbf{c}_t}{\partial \mathbf{h}_{t-1}} \text{diag}(\tanh'(\mathbf{c}_t)) \end{aligned} \quad (10.67)$$

$$\begin{aligned} \frac{\partial c_t}{\partial h_{t-1}} &= \frac{\partial c_t}{\partial f_t} \frac{\partial f_t}{\partial h_{t-1}} + \frac{\partial c_t}{\partial i_t} \frac{\partial i_t}{\partial h_{t-1}} + \frac{\partial c_t}{\partial \hat{c}_t} \frac{\partial \hat{c}_t}{\partial h_{t-1}} \\ &= \frac{\partial f_t}{\partial h_{t-1}} \text{diag}(\mathbf{c}_{t-1}) + \frac{\partial i_t}{\partial h_{t-1}} \text{diag}(\hat{\mathbf{c}}_t) + \frac{\partial \hat{c}_t}{\partial h_{t-1}} \text{diag}(\mathbf{i}_t) \end{aligned} \quad (10.68)$$

$$z_{f_t} = \mathbf{W}_{fh} \mathbf{h}_{t-1} + \mathbf{W}_{fx} \mathbf{x}_t + \mathbf{b}_f \quad (10.69)$$

$$\Rightarrow f_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) = \sigma(z_{f_t}) \quad (10.70)$$

$$\Rightarrow \frac{\partial f_t}{\partial h_{t-1}} = \mathbf{W}_{fh} \text{diag}(\sigma'(z_{f_t})) \quad (10.71)$$

$$z_{i_t} = \mathbf{W}_{ih} \mathbf{h}_{t-1} + \mathbf{W}_{ix} \mathbf{x}_t + \mathbf{b}_i \quad (10.72)$$

$$\Rightarrow i_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) = \sigma(z_{i_t}) \quad (10.73)$$

$$\Rightarrow \frac{\partial i_t}{\partial h_{t-1}} = \mathbf{W}_{ih} \text{diag}(\sigma'(z_{i_t})) \quad (10.74)$$

$$z_{c_t} = \mathbf{W}_{ch} \mathbf{h}_{t-1} + \mathbf{W}_{cx} \mathbf{x}_t + \mathbf{b}_c \quad (10.75)$$

$$\Rightarrow \hat{c}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) = \tanh(z_{c_t}) \quad (10.76)$$

$$\Rightarrow \frac{\partial \hat{c}_t}{\partial h_{t-1}} = \mathbf{W}_{ch} \text{diag}(\sigma'(z_{c_t})) \quad (10.77)$$

综合上述公式有：

$$\frac{\partial h_t}{\partial h_{t-1}} = \underbrace{\mathbf{W}_{oh} \text{diag}(\tanh'(z_{o_t}))}_A \quad (10.78)$$

$$+ \underbrace{\left(\mathbf{W}_{fh} \text{diag}(\sigma'(z_{f_t})) \right)}_B \text{diag}(\tanh'(\mathbf{c}_t)) \quad (10.79)$$

$$+ \underbrace{\left(\mathbf{W}_{ih} \text{diag}(\sigma'(z_{i_t})) \right)}_C \text{diag}(\tanh'(\mathbf{c}_t)) \quad (10.80)$$

$$+ \underbrace{\left(\mathbf{W}_{ch} \text{diag}(\sigma'(z_{c_t})) \right)}_D \text{diag}(\tanh'(\mathbf{c}_t)) \quad (10.81)$$

对比 Simple RNN 的递归公式：

$$\frac{\partial h_t}{\partial h_{t-1}} = \mathbf{V}_h \text{diag}(g'(z_t)) \quad (10.82)$$

可见上述 LSTM 递归公式通过引入四个矩阵 A, B, C, D 相加的形式，因此其特征值 (Eigen-Value) 相对 Simple RNN 递归形式，有更大的自由度使得求和后的最大特征值为 1，这样反向传播过程中系统整体达到稳态，不会出现 Simple RNN 中出现的梯度爆炸或梯度消散问题。

10.3.2 应用案例

本节重点回顾 LSTM 在自然语言处理领域的应用案例

10.3.2.1 机器翻译

机器翻译的目的是将一种类型的语言翻译成另一种类型的语言，譬如：



图 10.22: english-chinese translation

上图中显示一个英语到汉语的机器翻译的示例，其中英语为源语言，汉语为目标语言。Ilya Sutskever 等人¹³较早提出了序列到序列的模型，将源语言和目标语言分别看成两个序列，对应源语言序列，采用多层 LSTM 进行编码，编码阶段对应的模型称为编码器 (Encoder)，编码阶段最后一个词为源语言的最后一个词，譬如源语言为 ABC，则编码阶段最后一个词为 C，该词的对应目标输出为表示源语言句子结束的特殊词 <EOS>，解码阶段第一个输入为编码阶段的输出词 <EOS>，将该词表示为解码器中的 embedding 表示，及编码阶段最后一个词 C 对应的 hidden 表示。由解码器生成目标语言，解码器同样采用多层 LSTM 进行解码，解码阶段类似于语言模型任务，当解码阶段生成 <EOS>，则解码阶段结束。因此解码阶段对应的公式如下：

$$p(y_1, \dots, y'_T | x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \dots, y_{t-1}) \quad (10.83)$$

上述公式中 $x_t, t \rightarrow 1, \dots, T$ 对应源语言， $y_t, t \rightarrow 1, \dots, T'$ 对应目标语言， v 为源语言经过编码器后生成的 hidden 表示，若源语言为 ABC， v 即为词 C 输出的 hidden 表示。

如下是 Ilya Sutskever 等人用于机器翻译问题采用的序列到序列架构

¹³Sequence to Sequence Learning with Neural Networks

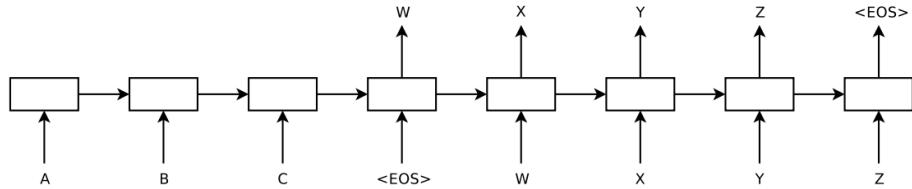


图 10.23: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Ilya Sutskever 等人提出的架构存在的问题是，需要将源语言的所有信息压缩到最后一个词的隐藏输出向量 \mathbf{v} 中，由于向量大小的限制，存在信息损失的缺陷。Kyunghyun Cho 等人¹⁴采用和 Ilya Sutskever 等类似的架构，区别的是编码阶段每个词的隐藏层和输出层同时依赖于编码阶段最后一个词的隐藏输出，同时输出层依赖于上一个词的向量表示，该架构对应的架构图如下：

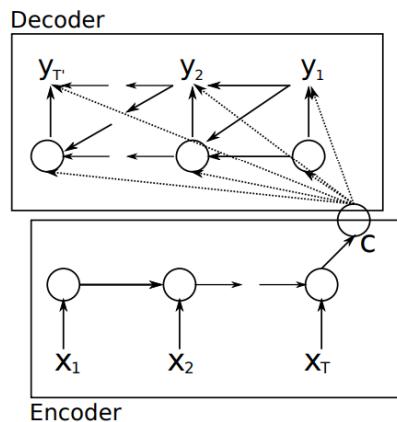


图 10.24: An illustration of the proposed RNN Encoder-Decoder.

该架构对应的公式如下：

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, y_{t-1}, \mathbf{c}) \quad (10.84)$$

$$p(y_t | y_1, \dots, y_{t-1}, \mathbf{c}) = g(\mathbf{h}_t, y_{t-1}, \mathbf{c}). \quad (10.85)$$

该架构同样无法解决由于向量大小限制，源语言信息损失的缺陷。针对该问题 Dzmitry Bahdanau 等人¹⁵在上述工作基础上在解码阶段引入注意力 (Attention) 机制使得每个输出的目标词可以在依赖于所有源语言输入词的隐藏状态表示向量，其表示表示形式如下：

$$\mathbf{s}_i = f(\mathbf{s}_{i-1}, y_{i-1}, \mathbf{c}_i) \quad (10.86)$$

$$p(y_i | y_1, \dots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, \mathbf{s}_i, \mathbf{c}_i). \quad (10.87)$$

对比前述工作可见，解码阶段每个词的隐藏层状态 \mathbf{s}_i 不再统一依赖编码阶段输出的 \mathbf{c} 表

¹⁴Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

¹⁵Neural Machine Translation by Jointly Learning to Align and Translate

示，而是依赖于 \mathbf{c}_i ，同样解码生成的每个词 y_i 概率也不再依赖统一的 \mathbf{c} 而是依赖于 \mathbf{c}_i 。这里 \mathbf{c}_i 的表示采用注意力机制，公式表示如下：

$$\mathbf{c}_i = \sum_{j=1}^{T_x} \alpha_{ij} \mathbf{h}_j. \quad (10.88)$$

其中 α_{ij} 源语言每个词对应的 \mathbf{h}_j 的权值，计算公式如下：

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \quad (10.89)$$

其中：

$$e_{ij} = a(s_{i-1}, \mathbf{h}_j) \quad (10.90)$$

上述公式中 a 为前向网络，用于对齐源语言每个词和目标语言当前词隐藏层输入，给出源语言每个词的隐藏输入词的对齐权重，并进一步得到当前目标语言词对应的 \mathbf{c}_i 表示，该机制称为注意机制，其对应的架构图如下所示：

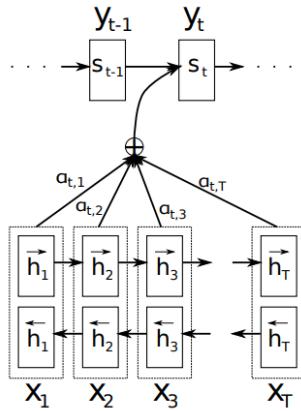


图 10.25: The graphical illustration of the proposed model trying to generate the t -th target word y_t given a source sentence (x_1, x_2, \dots, x_T) .

该架构有效解决了前述工作中由于最后一个词对应的向量表示 \mathbf{c} 的大小限制问题。

10.3.2.2 命名实体识别

命名实体识别¹⁶的主要任务是识别一段文本中的人名、地名、机构名及专有名词等。譬如：

Person Jim bought 300 shares of Organization Acme Corp. in Time 2006

这里介绍由 Guillaume Lample 等人¹⁷提出的将双向 LSTM 及 CRF 应用于命名实体识别的工作，该工作和前述命名实体识别工作有所区别的是，通常应用 CRF 进行命名实体识别时，需要给每个词人工设计一系列特征来表示给定上下文下当前词，并进一步由 CRF 融

¹⁶https://en.wikipedia.org/wiki/Named-entity_recognition

¹⁷Neural Architectures for Named Entity Recognition

合这些特征学习标注序列的概率分布。Guillaume Lample 等人采用双向 LSTM 架构学习给定上下文下每个词的特征表示，这样省去了人工设计特征的阶段，并取得了不错的效果，模型总体架构图如下所示：

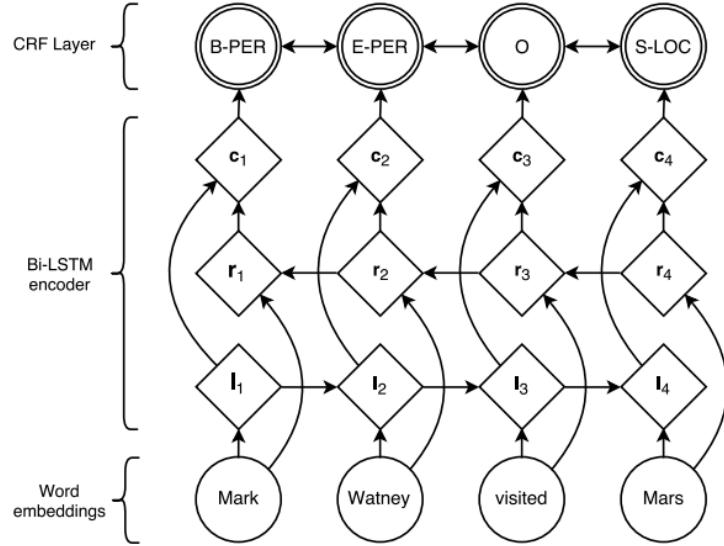


图 10.26: Main architecture of the network. Word embeddings are given to a bidirectional LSTM. \mathbf{l}_i represents the word i and its left context, \mathbf{r}_i represents the word i and its right context. Concatenating these two vectors yields a representation of the word i in its context, \mathbf{c}_i .

上述架构图中采用的 LSTM 架构和前述 LSTM 架构略有不同，具体公式如下：

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \quad (10.91)$$

$$\mathbf{c}_t = (1 - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c) \quad (10.92)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o) \quad (10.93)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \quad (10.94)$$

对比前述 LSTM 表示方式可见，这里采用了 $(1 - \mathbf{i}_t)$ 替换遗忘门，同时输入门依赖 LSTM 记忆单元历史信息 \mathbf{c}_{t-1} ，输出门依赖于更新后的当前 LSTM 记忆单元信息 \mathbf{c}_t 。给定上下文下，每个词的特征表示由双向 LSTM 各自输出的 \mathbf{h}_t 拼接而成 $\mathbf{h}_t = [\vec{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t]$ 。给定每个词当前上下文下的向量表示后，即可以用于进行命名实体识别。假设命名实体中有两类实体：人名 (PER)，地名 (LOC)，分别采用 B 和 I 两类标签表示词在命名实体中处于开始和中间位置，用 O 表示其他非命名实体词，则命名实体识别问题转化为序列标注问题，该问题可以采用 CRF 模型进行解决。给定序列：

$$\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n), \quad (10.95)$$

期望生成上述序列对应的标注序列：

$$\mathbf{y} = (y_1, y_2, \dots, y_n), \quad (10.96)$$

采用 CRF 表示方式，序列对应的矩阵 \mathbf{H} 和标注对应的向量 \mathbf{y} 联合权重表示形式如下：

$$s(\mathbf{H}, \mathbf{y}) = \sum_{i=0}^n A_{y_i, y_{i+1}} + \sum_{i=1}^n P_{i, y_i} \quad (10.97)$$

这里有必要解释一下上述公式中 $A_{y_i, y_{i+1}}$ 表示从类别 y_i 转移至类别 y_{i+1} 的权重，假设有 k 个类别，考虑开始位置和结束位置，因此矩阵 $\mathbf{A} \in \mathbb{R}^{(k+2) \times (k+2)}$ 为待学习的转移权重矩阵， P_{i, y_i} 位置 i 对应的表示 \mathbf{h}_i 和类别 y_i 的联合权重，假设 $\mathbf{h} \in \mathbb{R}^m$ ，因此每个位置对应各类别的权重计算如下：

$$P_{i,*} = \mathbf{h}\mathbf{W} \quad (10.98)$$

其中 $\mathbf{W} \in \mathbb{R}^{m \times (k+2)}$ 为待学习的参数权重。因此给定 \mathbf{H} 后，对应的标注序列 \mathbf{y} 的概率可表示为：

$$p(\mathbf{y} | \mathbf{H}) = \frac{e^{s(\mathbf{H}, \mathbf{y})}}{\sum_{\tilde{\mathbf{y}} \in \mathcal{Y}_{\mathbf{H}}} e^{s(\mathbf{H}, \tilde{\mathbf{y}})}}. \quad (10.99)$$

因此可以通过 Cross-Entropy Loss 学习上述架构中涉及的参数 \mathbf{A} 及 \mathbf{W} 及 LSTM 网络中的权重参数。

总体架构图中词的 Embedding 表示，进一步采用了双向 LSTM 将字的 Embedding 表示转成词的 Embedding 表示，及词的预训练 Embedding 表示，将这两类向量拼接到一起表示词的 Embedding，架构图如下所示：

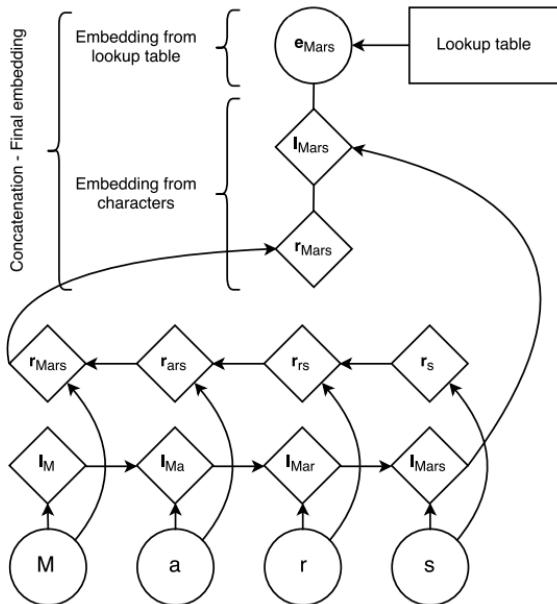


图 10.27: The character embeddings of the word "Mars" are given to a bidirectional LSTMs. We concatenate their last outputs to an embedding from a lookup table to obtain an representation for this word.

10.3.2.3 迁移学习

语言模型是自然语言处理领域进行迁移学习的较好的基石。语言模型用于学习在某个给定上下文下一个词出现的概率，我们来看如何将 LSTM 应用于语言模型问题并进行迁

移学习。考虑到语言模型任务为自回归任务 (Autoregressive)¹⁸，因此在应用深度学习的解决方案时不仅可以解决语言模型问题，同时可以学习到词在某个具体上下文下的表征，因此该任务可以成为很多其他任务的基石，语言模型输出的表征用于后续的有监督问题，该模式即为目前讨论较多的预训练 + 精调模式，预训练阶段训练语言模型，精调阶段将语言模型输出的表征知识迁移到解决当前的具体任务。

应用 LSTM 进行迁移学习取得重要影响的工作为 Matthew E. Peters 等人¹⁹提出的 **ELMo** 表征工作。该工作分为预训练和精调两阶段，预训练阶段采用双向 LSTM 输出词的表征信息，精调阶段将预训练阶段输出的双向 LSTM 以 feature 模式应用（固定 LSTM 权重参数），将 LSTM 输出的词各层表征组合得到词的 ELMo 表征，将该表征和当前任务的下的词表征进行拼接用于当前要解决的任务。下面我们分别来看预训练和精调这两个阶段。

预训练

预训练阶段采用了 Rafal Jozefowicz 等人²⁰给出的 CNN-BIG-LSTM 模型并将所有的向量表示维度降低为原来的 1/2，因此输入层首先采用 Yoon Kim²¹提出的 CNN 模型通过 2048 个字的 ngram 卷积核生成每个词的 2048 维表示，在此基础上再经过 Rupesh Kumar Srivastava 等人²²提出的两层 highway 处理得到同样维度大小的 2048 维表示，highway layer 部分程度上受到 LSTM 的启发，其变换如下所示：

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W}_H) \odot T(\mathbf{x}, \mathbf{W}_T) + \mathbf{x} \odot (1 - T(\mathbf{x}, \mathbf{W}_T)) \quad (10.100)$$

上述公式中 $\mathbf{x}, \mathbf{y}, H(\mathbf{x}, \mathbf{W}_H), T(\mathbf{x}, \mathbf{W}_T)$ 的维度必须一致， $T(\mathbf{x}, \mathbf{W}_T)$ 表示变换门，类似于 LSTM 的输入门。在此基础上再做一次矩阵线性变换将 2048 维表示变换为 512 维表示，作为双向 LSTM 的输入，LSTM 隐藏层的维度为 4096 维。预训练阶段，双向 LSTM 各自拥有独立的参数，输入层的词表征和输出层 Softmax 变换参数共享。因此损失函数如下：

$$-\sum_{k=1}^N \left(\log p(t_k | t_1, \dots, t_{k-1}; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s) + \log p(t_k | t_{k+1}, \dots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \right). \quad (10.101)$$

ELMo

ELMo 对应具体任务的精调阶段，该阶段将预训练阶段得到每个词的各层表征进行组合生成词的具体表征。对于每个词 t_k ， L 层的双向 LSTM 计算输出大小为 $2L + 1$ 的表征集

¹⁸<https://deepmind.com/blog/article/unsupervised-learning>

¹⁹Deep contextualized word representations

²⁰Exploring the Limits of Language Modeling

²¹Convolutional Neural Networks for Sentence Classification

²²Training Very Deep Networks

合

$$\mathbb{R}_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, \dots, L\} \quad (10.102)$$

$$= \{\mathbf{h}_{k,j}^{LM} | j = 0, \dots, L\}, \quad (10.103)$$

其中 $\mathbf{h}_{k,0}^{LM}$ 为 CNN 输出线性变换后的词表征层，对于双向 LSTM 的每一层来说 $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$ 。基于该集合即可得到词的 **ELMo** 表征，公式如下：

$$\mathbf{ELMo}_k^{task} = E(\mathbb{R}_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^L s_j^{task} \mathbf{h}_{k,j}^{LM}. \quad (10.104)$$

上述公式中 s^{task} 为 softmax 归一化权重， γ^{task} 允许当前任务对 ELMo 向量整体进行调权。

获得词的 **ELMo** 表征后通过在输入层和输出层将 \mathbf{ELMo}_k^{task} 拼接到相应任务表征的向量上分别得到 $[\mathbf{x}_k, \mathbf{ELMo}_k^{task}]$ 和 $[\mathbf{h}_k, \mathbf{ELMo}_k^{task}]$ ，并在损失函数中加入对 s^{task} 参数的 L_2 正则化项降低结构风险。对比未使用 \mathbf{ELMo}_k^{task} 表征，在 SQuAD, SRL, Coref, NER, SST-5 自然语言处理任务上均取得了大幅提升。如下是 Matthew E. Peters 等人²³给出的 **TagLM** 预训练语言模型向量在序列标注任务上的应用架构，和 **ELMo** 不同的是 **TagLM** 只使用了 $\mathbf{h}_{k,L}^{LM}$ ，另外在应用时，该架构中只将 **TagLM** 应用到了双层双向 LSTM 的第一输出和其拼接到一起后作为第二层的输入。

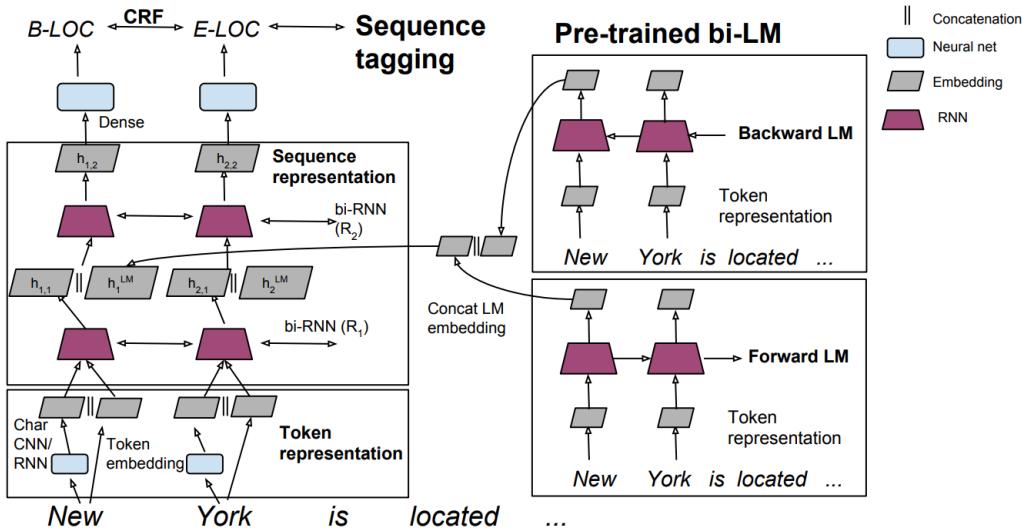


图 10.28: Overview of TagLM, our language model augmented sequence tagging architecture. The top level embeddings from a pre-trained bidirectional LM are inserted in a stacked bidirectional RNN sequence tagging model.

²³Semi-supervised sequence tagging with bidirectional language models

10.4 Transformer

本节回顾 Transformer 架构，及在 Transformer 基础上衍生出来的 BERT 模型，最后给出相应的应用案例。

10.4.1 Transformer

Transformer 网络架构由 Ashish Vaswani 等人²⁴提出并用于机器翻译任务，和前述网络架构有所区别的是，该网络架构中，编码器和解码器没有采用 RNN 或 CNN 等网络架构，而是采用完全依赖于注意力机制的架构。网络架构如下所示：

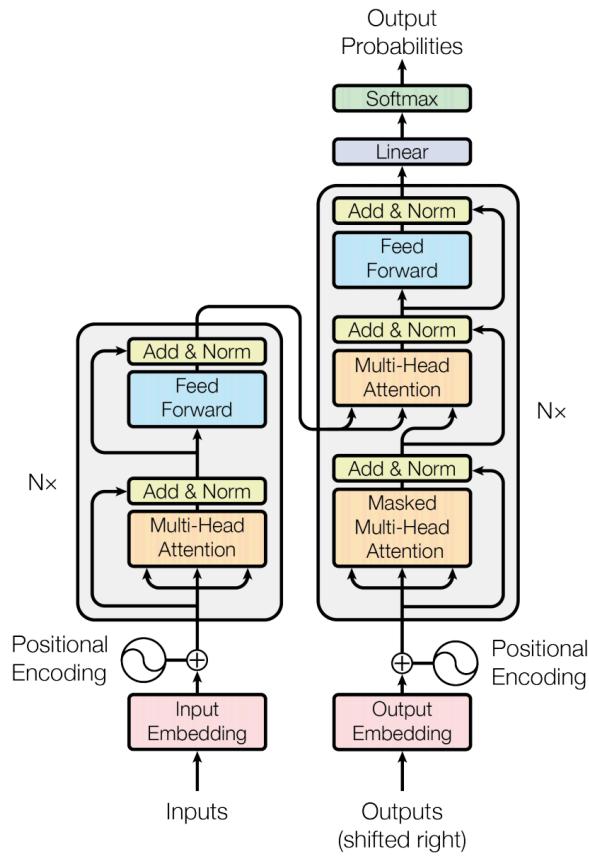


图 10.29: The Transformer - model architecture.

该网络架构中引入了多头注意力机制，该机制的网络架构如下所示：

²⁴Attention Is All You Need

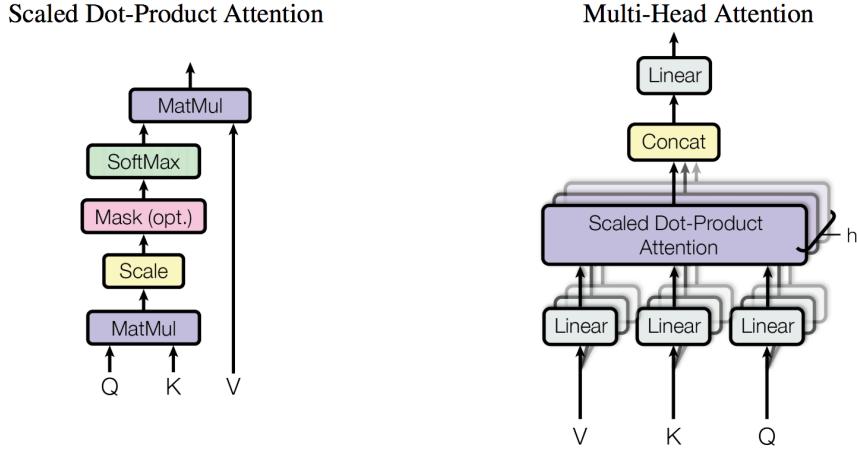


图 10.30: (left)Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel

这里有必要对多头注意力机制进行一定的解释。假设输入数据的 batch size 为 B , 输入数据的最大长度为 F , 输出数据的最大长度为 T , 共有 N 个注意力头, 每个注意力头的输出维度为 H , 则输入/输出数据中每个词的 Embedding 的维度为 $E = N \times H$, 且注意力头中每个头对应的 Q , K 和 V 矩阵的均属于 $\mathbb{R}^{E \times H}$ 。考虑到编码器和解码器涉及三个注意过程, 且输入有所不同, 这里分别来看。

10.4.1.1 编码器自注意力

考虑输入数据为 $\mathbf{X} \in \mathbb{R}^{B \times F \times E}$, 对输入数据应用如下线性变换

$$\mathbf{Q} = \mathbf{XW}^Q, \quad (\mathbf{W}^Q \in \mathbb{R}^{E \times H} \Rightarrow \mathbf{Q} \in \mathbb{R}^{B \times F \times H}) \quad (10.105)$$

$$\mathbf{K} = \mathbf{XW}^K, \quad (\mathbf{W}^K \in \mathbb{R}^{E \times H} \Rightarrow \mathbf{K} \in \mathbb{R}^{B \times F \times H}) \quad (10.106)$$

$$\mathbf{V} = \mathbf{XW}^V, \quad (\mathbf{W}^V \in \mathbb{R}^{E \times H} \Rightarrow \mathbf{V} \in \mathbb{R}^{B \times F \times H}) \quad (10.107)$$

在上述变换基础上进行如下计算, 得到输入中每个词和自身及其他词之间的关系权重

$$\mathbf{S} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{H}}\right) \quad (10.108)$$

上述变换 \mathbf{K}^\top 表示对张量的最内部矩阵进行转置, 因此 $\mathbf{K}^\top \in \mathbb{R}^{B \times H \times F}$, $\mathbf{Q}\mathbf{K}^\top$ 表示相同维度下张量 \mathbf{Q} 和张量 \mathbf{K}^\top 最内部矩阵执行矩阵乘法运算 (即 numpy.matmul 运算), 因此有 $\mathbf{S} \in \mathbb{R}^{B \times F \times F}$, 该张量表示输入数据中每个词和及其他词的关系权重, 每一行的得分之和为 1, 即

$$\forall i, j \quad \text{np.sum}(\mathbf{S}[i, j, :]) = 1 \quad (10.109)$$

基于该得分即可得到, 每个词在当前上下文下的新的向量表示, 公式如下:

$$\mathbf{x}^h = \mathbf{S}\mathbf{V}, \Rightarrow \mathbf{x}^h \in \mathbb{R}^{B \times F \times H} \quad (10.110)$$

考虑到 Transformer 采用了 N 个注意力头，因此最终产生了集合大小为 N 的注意力集合 $\{\mathbf{X}^{h_1}, \dots, \mathbf{X}^{h_N}\}$ ，将该集合中中的所有张量按照最后一个维度进行拼接，并采用矩阵 $\mathbf{W}^O \in \mathbb{R}^{E \times E}$ 进行变换，得到最终生成的自注意力输入数据，公式如下：

$$\mathbf{X}^a = \text{numpy.concatenate}((\mathbf{X}^{h_1}, \dots, \mathbf{X}^{h_N}), \text{axis} = -1) \mathbf{W}^O \quad (10.111)$$

因此有 $\mathbf{X}^a \in \mathbb{R}^{B \times F \times E}$ 。

考虑到多头注意力需要并行运算，提高计算效率，考虑充分发挥向量化计算并行效率，实际实现中往往采用如下表示方案：

$$\mathbf{X}^{par} = \text{reshape}(\mathbf{X}, \text{to_shape} = [B \times F, N \times H]) \quad (10.112)$$

$$\mathbf{Q}^{par} = \mathbf{X}^{par} \mathbf{W}^{Q^{par}} \quad (\mathbf{W}^{Q^{par}} \in \mathbb{R}^{(N \times H) \times (N \times H)} \Rightarrow \mathbf{Q}^{par} \in \mathbb{R}^{(B \times F) \times (N \times H)}) \quad (10.113)$$

$$\mathbf{K}^{par} = \mathbf{X}^{par} \mathbf{W}^{K^{par}} \quad (\mathbf{W}^{K^{par}} \in \mathbb{R}^{(N \times H) \times (N \times H)} \Rightarrow \mathbf{K}^{par} \in \mathbb{R}^{(B \times F) \times (N \times H)}) \quad (10.114)$$

$$\mathbf{V}^{par} = \mathbf{X}^{par} \mathbf{W}^{V^{par}} \quad (\mathbf{W}^{V^{par}} \in \mathbb{R}^{(N \times H) \times (N \times H)} \Rightarrow \mathbf{V}^{par} \in \mathbb{R}^{(B \times F) \times (N \times H)}) \quad (10.115)$$

在上述并行计算基础上通过如下计算得到词和自身及其他词的关系权值：

$$\mathbf{Q}^{par^r} = \text{numpy.reshape}(\mathbf{Q}^{par}, (B, F, N, H)) \quad (10.116)$$

$$\mathbf{Q}^{par^t} = \text{numpy.transpose}(\mathbf{Q}^{par^r}, \text{axes} = [0, 2, 1, 3]) \Rightarrow \mathbf{Q}^{par^t} \in \mathbb{R}^{B \times N \times F \times H} \quad (10.117)$$

$$\mathbf{K}^{par^r} = \text{numpy.reshape}(\mathbf{K}^{par}, (B, F, N, H)) \quad (10.118)$$

$$\mathbf{K}^{par^t} = \text{numpy.transpose}(\mathbf{K}^{par^r}, \text{axes} = [0, 2, 1, 3]) \Rightarrow \mathbf{K}^{par^t} \in \mathbb{R}^{B \times N \times F \times H} \quad (10.119)$$

$$\mathbf{S}^{par} = \text{softmax}\left(\frac{\mathbf{Q}^{par^t} \mathbf{K}^{par^t \top}}{\sqrt{H}}\right) \Rightarrow \mathbf{S}^{par} \in \mathbb{R}^{B \times N \times F \times F} \quad (10.120)$$

在上述计算基础上，通过如下变换，即可得到输入输入的自注意力向量表示

$$\mathbf{V}^{par^r} = \text{numpy.reshape}(\mathbf{V}^{par}, (B, F, N, H)) \quad (10.121)$$

$$\mathbf{V}^{par^t} = \text{numpy.transpose}(\mathbf{V}^{par^r}, \text{axes} = [0, 2, 1, 3]) \Rightarrow \mathbf{V}^{par^t} \in \mathbb{R}^{B \times N \times F \times H} \quad (10.122)$$

$$\mathbf{X}^h = \mathbf{S}^{par} \mathbf{V}^{par^t} \Rightarrow \mathbf{X}^h \in \mathbb{R}^{B \times N \times F \times H} \quad (10.123)$$

$$\mathbf{X}^h = \text{numpy.transpose}(\mathbf{X}^h, \text{axes} = [0, 2, 1, 3]) \Rightarrow \mathbf{X}^h \in \mathbb{R}^{B \times F \times N \times H} \quad (10.124)$$

$$\mathbf{X}^h = \text{numpy.reshape}(\mathbf{X}^h, (B, F, N \times H)) \Rightarrow \mathbf{X}^h \in \mathbb{R}^{B \times F \times E} \quad (10.125)$$

10.4.1.2 解码器自注意力

解码器的自注意力和编码器的自注意力基本完全一致，需要注意的是解码过程是一个词一个词的生成过程，因此输出数据中的每个词在进行自注意力的过程时，仅可以看到当前输出位置的所有前驱词的信息，因此需要对输出数据中的词进行掩码操作，该操作即

对应图10.30中左图上的掩码操作。该掩码操作相当于执行如下操作：

$$\mathbf{A} = \mathbf{QK}^\top + \mathbf{M} \quad (10.126)$$

$$\mathbf{S}^{par} = \text{softmax}\left(\frac{\mathbf{A}}{\sqrt{H}}\right) \Rightarrow \mathbf{S}^{par} \in \mathbb{R}^{B \times N \times T \times T} \quad (10.127)$$

其中 $\mathbf{M} \in \mathbb{R}^{1 \times 1 \times T \times T}$ 为掩码，其最内部矩阵为方阵，该方阵主对角线及以下元素均为 0，主对角线上元素为 $-\infty$ 。譬如 $T = 5$ 时，最内部方阵内容如下：

$$\mathbf{M} = \begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (10.128)$$

其余操作和编码器自注意力机制一致，唯一不同的是此时需要向上面那样将输入数据换成 $\mathbf{Y} \in \mathbb{R}^{N \times T \times E}$ ，因此所有的 F 均需换成 T 。

10.4.1.3 编码解码注意力

编码解码注意力和自注意力类似，唯一不同的是计算 $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ 使用的数据有所区别，计算 \mathbf{Q} 时采用 \mathbf{Y} ，计算 \mathbf{K} 和 \mathbf{V} 时采用 \mathbf{X} ，因此有：

$$\mathbf{X}^{par} = \text{reshape}(\mathbf{X}, \text{to_shape} = [B \times F, N \times H]) \quad (10.129)$$

$$\mathbf{Y}^{par} = \text{reshape}(\mathbf{Y}, \text{to_shape} = [B \times T, N \times H]) \quad (10.130)$$

$$\mathbf{Q}^{ende-par} = \mathbf{Y}^{par} \mathbf{W}^{Q^{ende-par}} \quad (\mathbf{W}^{Q^{ende-par}} \in \mathbb{R}^{(N \times H) \times (N \times H)}) \quad (10.131)$$

$$\mathbf{K}^{ende-par} = \mathbf{X}^{par} \mathbf{W}^{K^{ende-par}} \quad (\mathbf{W}^{K^{ende-par}} \in \mathbb{R}^{(N \times H) \times (N \times H)}) \quad (10.132)$$

$$\mathbf{V}^{ende-par} = \mathbf{X}^{par} \mathbf{W}^{V^{par}} \quad (\mathbf{W}^{V^{ende-par}} \in \mathbb{R}^{(N \times H) \times (N \times H)}) \quad (10.133)$$

因此有：

$$\mathbf{S}^{ende-par} = \text{softmax}\left(\frac{\mathbf{Q}^{ende-par} \mathbf{K}^{ende-par}^t}{\sqrt{H}}\right) \Rightarrow \mathbf{S}^{ende-par} \in \mathbb{R}^{B \times N \times T \times F} \quad (10.134)$$

$$\mathbf{Y}^{ende-h} = \mathbf{S}^{ende-par} \mathbf{V}^{ende-par} \Rightarrow \mathbf{Y}^{ende-h} \in \mathbb{R}^{B \times N \times T \times H} \quad (10.135)$$

其余计算过程和编码器自注意力机制类似。

10.4.2 BERT

BERT(Bidirectional Encoder Representation from Transformers) 网络架构由 Jacob Devlin 等人²⁵提出用于预训练，学习在给定上下文下词的 Embedding 表示。BERT 采用了

²⁵BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Transformer 架构的编码器部分用于学习词在给定上下文下词的 Embedding 表示。考虑到语言模型任务为从左往右或从右往左预测下一个词的自回归任务，因此若采用该模式每个词无法依赖于当前上下文下后续词进行词向量表示。为了解决该问题，BERT 提出了两类预训练任务：

- 掩码语言模型任务
- 下一句预测任务

下面我们分别来看这两类任务。

掩码语言模型任务

若采用 self-attention 并直接用于语言模型任务，则每次预测下一个词时由于 self-attention 机制会将下一个词的信息引入到当前词的表征中，当使用该词的表征用于预测下一个词时，相当于将标注引入到了特征中，因此出现学习失效问题。为了解决该问题，BERT 提出了如下解决方案：

从待预测序列中随机选择 15% 的位置用于预测任务

- 80% 的概率下将选取出的 15% 的位置对应的词替换为 [MASK]
- 10% 的概率下将选取出的 15% 的位置对应的词替换为随机词
- 10% 的情况下不对选取出的 15% 的位置对应的词进行词替换

引入 1.5% 的随机词，相当于对数据增加部分噪音，提升模型的鲁棒性。1.5% 的情况下保留原词是因为 fine-tuning 阶段并没有 [MASK] 词。

下一句预测任务

考虑到重要的下游任务譬如问答 (Question Answering) 任务，自然语言推理 (Natural Language Inference) 任务依赖于对两个句子的关系的理解，该信息在语言模型中没有直接体现。因此 BERT 中同时设计了下一句预测任务，该任务的构建如下：

- 每个预训练序列由句子 A 和句子 B 构成
- 50% 的概率下句子 B 为句子 A 的下一个句子
- 50% 的概率下句子 B 不是句子 A 的下一个句子

实际构建预训练任务时是首选设计好“下一句预测任务”，生成该任务的标注信息，在此基础上构建“掩码语言模型任务”，生成掩码语言模型的标注信息。考虑到预训练涉及两个句子，BERT 采用如下的输入信息表征方案：

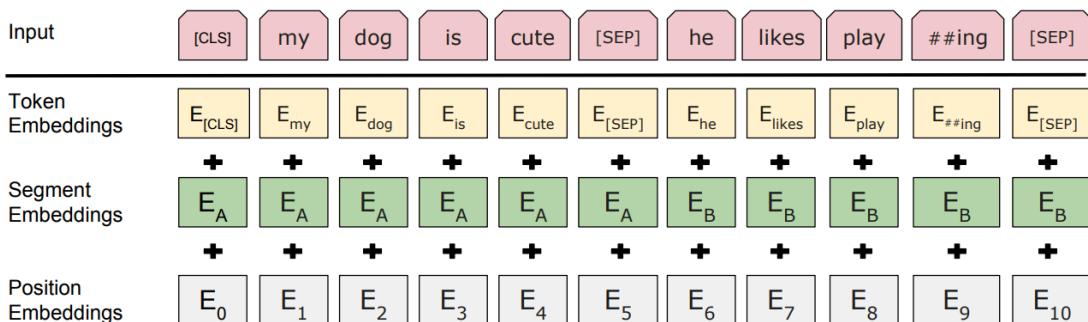


图 10.31: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

预训练阶段损失函数通过线性加权方法，同时进行上述两类任务训练。预训练阶段结束后将学习到每个词在特定上下文中 BERT 的表征信息，该表征信息即可用于下游的任务，如下是 BERT 表征用于不同类型的下游任务的 fine-tuning 方案：

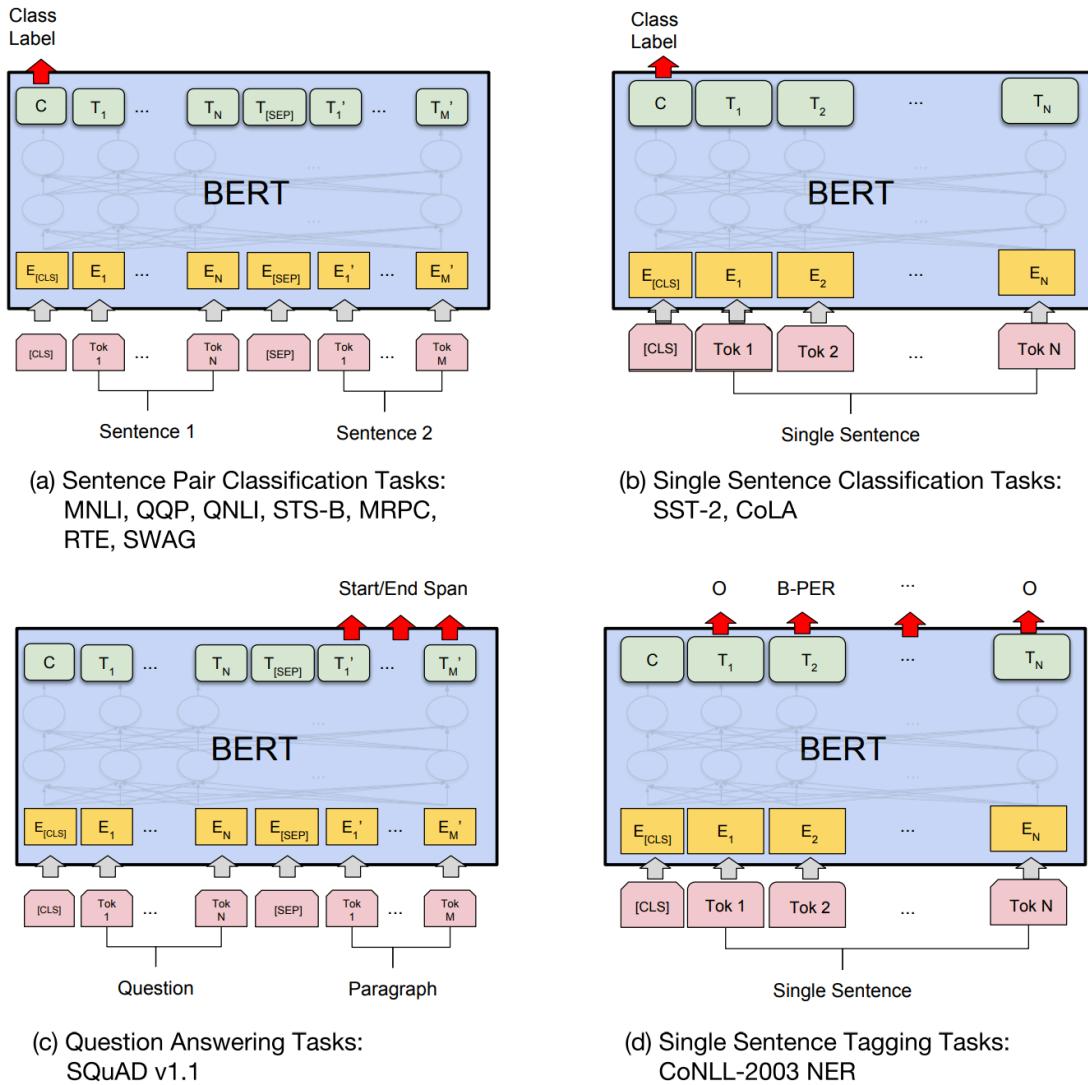


图 10.32: Illustrations of Fine-tuning BERT on Different Tasks.

10.4.3 ELECTRA

ELECTRA(Efficiently Learning an Encoder that Classifies Token Replacements Accurately) 网络架构由 Kevin Clark 等人²⁶提出用于预训练，学习在给定上下文下词的 Embedding 表示。考虑到 BERT 等 MLM(masked language modelin) 类模型训练过程中仅使用了每个样本的 15% 的词，需要大量的计算资源。ELECTRA 在 BERT 基础上提出替换词检测任务(replaced token detection)，该预训练的任务是检测样本中哪些词时输入原词，哪些词是看似合理的替换词。因此和 masking 策略不同的是，该方法将输入样本中一部分词通过掩码语音模型替换为看似合理的词，该过程解决了 BERT 这类网络在预训练过程中存在伪

²⁶ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS

造词 [MASK]，而在下游具体任务进行 fine-tuned 过程不存在这类词的问题。和 BERT 不同的是，ELECTRA 基于输入样本中的所有词进行学习，而不是仅对掩码部分词学习，因此 ELECTRA 更加计算高效。

考虑到现有的大部分预训练方法需要大量计算资源，因此基于成本和可行性考虑，ELECTRA 认为提升计算效率和下游任务的效果改进同样重要，因此 ELECTRA 同时对比了不同模型大小下在 GLUE 评测集上和已有的预训练语言模型效果对比，譬如作者提出了 ELECTRA-Small 模型，该模型可以在 1 个 V100 GPU 上使用 4 天时间完成训练，输出的模型比同样环境和模型大小下的 BERT-Small 模型 GLUE 得分高出接近 4.3 个点。

网络结构

下面让我们具体来看看 ELECTRA 模型的网络结构。下图是网络的整体架构：

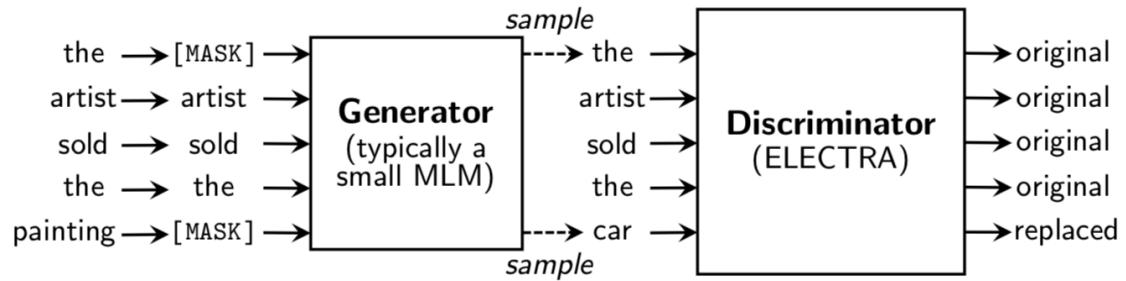


图 10.33: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, we throw out the generator and only fine-tune the discriminator on downstream tasks.

从上述架构可见，ELECTRA 包含两个神经网络，一个生成器网络 G 和一个判别器网络 D 。每个网络主要包含编码器 (Transformer 网络) 其将词序列 $\mathbf{x} = [x_1, \dots, x_n]$ 映射为上下文向量表示序列 $h(\mathbf{x}) = [\mathbf{h}_1, \dots, \mathbf{h}_n]$ 。对某个给定位置 t ，生成器通过 softmax 层输出词 x_t 的概率：

$$p_G(x_t | \mathbf{x}) = \frac{\exp(e(x_t)^\top h_G(\mathbf{x})_t)}{\sum_{x'} \exp(e(x')^\top h_G(\mathbf{x})_t)} \quad (10.136)$$

公式 10.4.3 中 e 表示词的 embedding。对于某个给定位置 t ，判别器预测词 x_t 是否是伪造词，即该词来源于生成器，而非原始样本，因此判别器的公式如下：

$$D(\mathbf{x}, t) = \text{sigmoid}(\mathbf{w}^\top h_D(\mathbf{x})_t) \quad (10.137)$$

生成器训练完成掩码语言模型任务 (MLM)，对于给定样本 $\mathbf{x} = [x_1, x_2, \dots, x_n]$ ，MLM 首先随机从 1 到 n 中随机选取 k 个位置用于掩码 $\mathbf{m} = [m_1, \dots, m_k]$ ， \mathbf{m} 中的所有位置均被替换为 [MASK] 词，替换后的样本表示为 $\mathbf{x}^{\text{masked}} = \text{REPEACE}(\mathbf{x}, \mathbf{m}, [\text{MASK}])$ 。生成器基于样本 $\mathbf{x}^{\text{masked}}$ 训练，最大化被掩码掉的原词的概率。判别器基于生成器输出的样本进行训练，区分哪些词是原始样本中的词，哪些词是判别器生成的和原始样本不一致的词，通过 $\mathbf{x}^{\text{corrupt}}$ 表示由生成器通过替换 [MASK] 词之后生成的样本，判别器判断 $\mathbf{x}^{\text{corrupt}}$

中词和 \mathbf{x} 中词是否一致。如下是训练过程中的样本生成形式化描述：

$$m_i \sim \text{unif}\{1, n\} \quad \text{for } i = 1 \text{ to } k \quad (10.138)$$

$$\mathbf{x}^{masked} = \text{REPLACE}(\mathbf{x}, \mathbf{m}, [\text{MASK}]) \quad (10.139)$$

$$\hat{\mathbf{x}}_i = p_G(x_i | \mathbf{x}^{masked}) \quad \text{for } i \in \mathbf{m} \quad (10.140)$$

$$\mathbf{x}^{corrupt} = \text{REPLACE}(\mathbf{x}, \mathbf{m}, \hat{\mathbf{x}}) \quad (10.141)$$

上述公式中 $k = \lceil 0.15n \rceil$ 。相应的生成器和判别器的损失函数分别为：

$$\mathcal{L}_{MLM}(\mathbf{x}, \theta_G) = E \left(\sum_{i \in \mathbf{m}} -\log p_G(x_i | \mathbf{x}^{masked}) \right) \quad (10.142)$$

$$\begin{aligned} \mathcal{L}_{Disc}(\mathbf{x}, \theta_D) = & E \left(\sum_{t=1}^n \mathbb{1}(x_t^{corrupt} = x_t) \log D(\mathbf{x}^{corrupt}, t) \right. \\ & \left. + \mathbb{1}(x_t^{corrupt} \neq x_t) \log (1 - D(\mathbf{x}^{corrupt}, t)) \right) \end{aligned} \quad (10.143)$$

最终训练时将上述两类损失结合在一起，生成如下损失函数：

$$\min_{\theta_G, \theta_D} \sum_{\mathbf{x} \in \mathbb{X}} \mathcal{L}_{MLM}(\mathbf{x}, \theta_G) + \lambda \mathcal{L}_{Disc}(\mathbf{x}, \theta_D) \quad (10.144)$$

预训练过程中，生成器和判别器共享词的 embedding 表示。经过预训练后，丢弃掉生成器，仅留下判别器用于下游任务的精调。考虑到 ELECTRA 模型中包含两个神经网络，若生成器也很大，则每一轮训练的成本是 BERT 的两倍，因此作者对比了不同生成器隐藏层神经元个数下 GLUE 得分，从实验可见生成器为判别器大小 $1/4 - 1/2$ 时可以取得较好的效果，如下是对比实验：

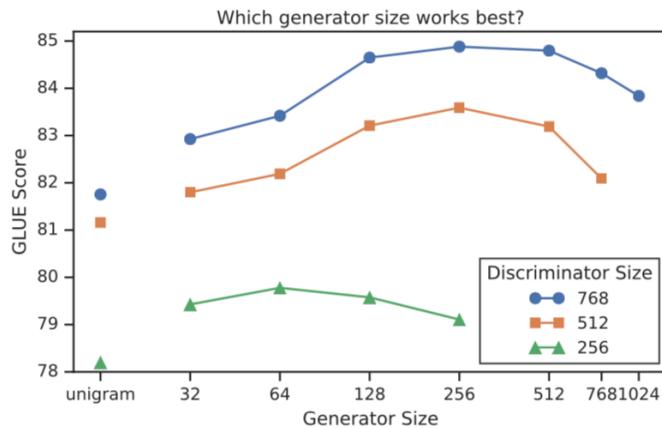


图 10.34: GLUE scores for different generator/discriminator sizes (number of hidden units). Interestingly, having a generator smaller than the discriminator improves results.

实验对比

考虑到预训练模型的成本和效果，作者对比了 ELECTRA-Small 和其他模型的效果。作者将序列长度从 512 降至 128，批大小由 256 降至 128，模型隐藏层大小由 768 降至 256，词

embedding 大小由 768 降至 128，并对比了同样参数规模的 BERT-Small，并在同样的训练计算量下对比 BERT-Small 和 ELECTRA-Small，同时对比了 ELMo 和 GPT，及 DistilBERT 模型，实验数据如下表所示：

Model	Train / Infer FLOPs	Speedup	Params	Train Time + Hardware	GLUE
ELMo	3.3e18 / 2.6e10	19x / 1.2x	96M	14d on 3 GTX 1080 GPUs	71.2
GPT	4.0e19 / 3.0e10	1.6x / 0.97x	117M	25d on 8 P6000 GPUs	78.8
BERT-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100 GPU	74.7
BERT-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	82.2
DistilBERT	- / 1.4e10	- / 2x	66M	-	77.8
ELECTRA-Small	1.4e18 / 3.7e9	45x / 8x	14M	4D on 1 V100 GPU	79.0
ELECTRA-Base	6.4e19 / 2.9e10	1x / 1x	110M	4D on 16 TPUv3s	85.1

图 10.35: Comparison of small models on the GLUE dev set. BERT-Small/Base are our implementation and use the same hyperparameters as ELECTRA-Small/Base. Infer FLOPS assumes single length-128 input. Training times should be taken with a grain of salt as they are for different hardware and with sometimes un-optimized code

从上表可见在同样训练成本下，ELECTRA-Small 模型效果远好于 BERT-Small 模型，甚至由于 GPT 模型，另外 ELECTRA-Base 模型和 BERT-Base 模型对比在 GLUE 上对比得分超过接近 3 个点。

最后作者对比了 ELECTRA 模型和 XLNET 及 RoBERT 模型的效果，总体来看，ELECTRA 模型在 GLUE 更多子任务上取得更好效果，如下是实验结果：

Model	Train FLOPs	Params	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	Avg.
BERT	1.9e20 (0.27x)	335M	60.6	93.2	88.0	90.0	91.3	86.6	92.3	70.4	84.0
XLNet	9.6e20 (1.3x)	360M	63.6	95.6	89.2	91.8	91.8	89.8	93.9	83.8	87.4
RoBERTa-100K	6.4e20 (0.90x)	356M	66.1	95.6	91.4	92.2	92.0	89.3	94.0	82.7	87.9
RoBERTa	3.2e21 (4.5x)	356M	68.0	96.4	90.9	92.4	92.2	90.2	94.7	86.6	88.9
BERT (ours)	7.1e20 (1x)	335M	67.0	95.9	89.1	91.2	91.5	89.6	93.5	79.5	87.2
ELECTRA	7.1e20 (1x)	335M	69.3	96.0	90.6	92.1	92.4	90.5	94.5	86.8	89.0
<i>Test set results for models with standard single-task finetuning (no ensembling, task-specific tricks, etc.)</i>											
BERT	1.9e20 (0.27x)	335M	60.5	94.9	89.3	86.5	89.3	86.7	92.7	70.1	83.8
SpanBERT	7.1e20 (1x)	335M	64.3	94.8	90.9	89.9	89.5	87.7	94.3	79.0	86.3
ELECTRA	7.1e20 (1x)	335M	68.2	96.9	89.6	91.0	90.1	90.1	95.4	83.6	88.1

图 10.36: Comparison of large models on the GLUE dev and test sets.