

神经网络：从不懂到装懂

孟渔樵

2020 年 5 月 17 日

想要学得会，先把公式背
要想考不倒，亲自来推导

目录

1 微积分基础	8
1.1 入门	8
1.1.1 入门一只脚：圆的面积	8
1.1.2 入门另一只脚：函数的面积	8
1.2 导数	9
1.2.1 导数入门：汽车的路程与速度	9
1.2.2 常见误区	10
1.3 用几何来推导	10
1.3.1 $f(x) = x^2$ 的导数 $\frac{df}{dx}$	10
1.3.2 $f(x) = x^3$ 的导数 $\frac{df}{dx}$	10
1.3.3 $f(x) = \frac{1}{x}$ 的导数	11
1.3.4 \sqrt{x} 的导数	12
1.3.5 $\sin(x)$ 的导数	12
1.4 直观理解链式法则和乘积法则	13
1.4.1 加法法则	13
1.4.2 乘法法则	13
1.4.3 函数复合	14
1.5 指数函数求导 $M(t) = n^t$	14
1.5.1 e 的定义过程	14
1.5.2 内容	15
1.5.3 推导过程	15
1.6 隐函数求导	15
1.6.1 内容	15
1.6.2 推导过程	15
1.7 极限	17
1.7.1 导数的正式定义	17
1.7.2 极限的 (ϵ, δ) 定义	17
1.7.3 洛必达法则	17
1.8 积分与微积分基本定理	18
1.8.1 推导	18
1.8.2 内容	19
1.9 面积与斜率的关系	19

目录	3
----	---

1.9.1 求 $f(x) = \sin(x)$ 在 $(0, \pi)$ 上的平均值	20
1.9.2 意义	20
1.10 高阶导数	20
1.10.1 表示	20
1.10.2 意义	20
1.11 泰勒级数	21
1.11.1 作用	21
1.11.2 代数推导过程	21
1.11.3 几何推导二次项意义	21
2 线性代数基础	23
2.1 向量是什么	23
2.2 向量加法与数乘的理解	23
2.2.1 向量	23
2.2.2 向量加法	23
2.2.3 数乘向量	24
2.3 线性组合, 张成的向量与基	24
2.3.1 基向量	24
2.3.2 选择不同的基向量	24
2.3.3 线性组合	24
2.3.4 向量空间	24
2.4 矩阵与线性变换	25
2.4.1 线性变换	25
2.4.2 矩阵	26
2.4.3 矩阵乘法	26
2.5 矩阵乘法与线性变换组合	26
2.5.1 矩阵乘法	26
2.6 行列式	27
2.6.1 定义	28
2.7 逆矩阵、列空间和零空间	28
2.8 矩阵的用途	28
2.8.1 逆矩阵	28
2.8.2 秩	29
2.8.3 矩阵的列空间	29

目录	4
2.8.4 零空间（核）	29
2.9 点积与对偶性	29
2.9.1 点积	29
2.10 叉积的标准介绍	31
2.10.1 导入	31
2.10.2 计算方法	31
2.10.3 实际定义	31
2.10.4 几何解释	31
2.11 特征向量与特征值	32
2.11.1 几何意义	32
2.11.2 计算思想	32
2.11.3 特征基	33
2.12 抽象向量空间	33
2.12.1 用矩阵计算导数	33
3 机器学习	34
3.1 学习方式分类	34
3.2 监督学习的步骤	35
3.3 误差、偏差和方差的区别和联系	37
3.3.1 误差(error)	37
3.3.2 噪声(Noise)	37
3.3.3 偏差(Bias)	37
3.3.4 方差(Variance)	37
3.4 损失函数	38
3.4.1 常见的损失函数	38
3.5 代价函数	39
3.5.1 为什么需要代价函数	39
3.5.2 代价函数作用原理	39
3.5.3 代价函数一定要非负吗？	40
3.5.4 常见的代价函数	40
3.5.5 为什么用交叉熵代替二次代价函数	42
4 神经网络	43
4.1 什么是神经网络	43

目录	5
----	---

4.2 为什么要用神经网络	43
4.3 神经网络怎么训练	43
4.3.1 成本函数的建立	43
4.3.2 多元分类MLP的逻辑成本函数	44
4.3.3 反向传播算法	45
4.3.4 参数更新方法（以隐层到输出层的连接权 w_{hj} 为例）	46
4.3.5 算法流程	48
4.3.6 算法分类	49
4.3.7 过拟合针对	49
4.3.8 参数优化	50
4.4 超参数	51
4.4.1 定义	51
4.4.2 设置方法	51
4.4.3 超参数搜索一般过程	52
4.5 激活函数	52
4.5.1 问题	52
4.5.2 常用激活函数	52
4.5.3 作用	52
4.5.4 常见激活函数的导数计算	53
4.5.5 激活函数性质	54
4.5.6 选择激活函数的策略	54
4.5.7 ReLu激活函数的优点	55
4.5.8 Softmax函数	55
4.6 Batch Size	55
4.6.1 为什么需要Batch Size	55
4.6.2 Batch Size值的选择	56
4.6.3 增大Batch Size的好处	56
4.6.4 盲目增大Batch Size的坏处	56
4.7 归一化	57
4.7.1 归一化含义	57
4.7.2 归一化好处	57
4.7.3 归一化能提高求解最优解速度的理由	58
4.7.4 归一化类型	58

目录	6
5 深度学习	59
5.1 为什么需要深度学习	59
5.1.1 必然性	59
5.1.2 可行性	59
5.2 什么是深度学习	59
5.2.1 增加神经元数目的神经网络	59
5.2.2 增加隐含层数量的神经网络	59
5.3 深度神经网络难以训练的原因	60
5.3.1 梯度消失	60
5.3.2 梯度爆炸	60
5.3.3 权重矩阵的退化导致模型的有效自由度减少	60
5.4 预训练与微调(fine tuning)	61
5.4.1 深度网络存在问题	61
5.4.2 解决方法	62
5.4.3 模型微调fine tuning的定义	62
5.4.4 微调时网络参数的更新	62
5.4.5 fine tuning模型的三种状态	62
5.5 正则化Dropout	62
5.5.1 为什么要正则化	62
5.5.2 理解	63
5.5.3 工作流程	63
5.5.4 工作机制	64
6 卷积神经网络	66
6.1 特征选择	66
6.1.1 优化的常见问题	66
6.1.2 理由	66
6.1.3 本质	66
6.1.4 作用	66
6.1.5 具体方法	66
6.2 卷积	67
6.2.1 卷积公式	67
6.2.2 公式理解	67
6.2.3 例子：信号分析	68

6.3	卷积神经网络的结构	69
6.3.1	全连接层	69
6.3.2	采样层	70
6.3.3	卷积层	70
6.3.4	级联分类器	71
6.4	卷积层：提取特征	72
6.4.1	图示	72
6.4.2	输出	73
6.4.3	卷积顺序	73
6.4.4	补0	74
6.5	采样层：特征选择	74
6.5.1	示例：Maxpooling	74
6.5.2	好处	75
6.5.3	性质	75
6.6	全连接层：进行分类	76
6.6.1	图示	76
6.6.2	转化方法	76
6.6.3	作用	77
6.6.4	对模型的影响因素	77
6.7	局部响应归一化作用	77
6.7.1	理解局部响应归一化	77
6.7.2	实例	78
6.8	批归一化BN	79
6.8.1	优点	80
6.8.2	算法流程	80
6.9	生成对抗网络	82

1 微积分基础

1.1 入门

1.1.1 入门一只脚：圆的面积

- 微积分思想：将困难问题转化为许多小量之和
- 应用：圆的面积 $S = \pi r^2$, 公式从何而来?
 1. 将圆切割成无数个同心圆环
 2. 取出其中一个圆环，假设其内半径为 r
 3. 将圆环剪开，展开成一个矩形（类似），长度为 $2\pi r$, 宽度假设为 dr
 4. 可得该矩形（圆环）面积为： $2\pi r dr$
 5. 随着半径 r 取得越来越小，该近似越来越精确
 6. 将圆环展成的矩形按长度从小到大一头对齐的方式排列起来（宽度都是 dr ），成一个直角三角形（近似）
 7. 对于该三角形
 - (a) 矩形宽组成的直角边长度： r (所有 dr 加起来)
 - (b) 另一条直角边长度： $2\pi r$ (近似)
 8. 由于该直角三角形由所有圆环组成，所以该直角三角形的面积和圆的面积相等（近似）。由此可得圆的面积为 $S = \frac{1}{2}(2\pi r)r = \pi r^2$
- 注意：当 dr 足够小 ($dr \rightarrow 0$) 时，以上所有的“近似”都可去掉（极限思想）

1.1.2 入门另一只脚：函数的面积

- 对函数 $f(x)$, 其自变量 x 每次变动 dx , 该函数图像与 x 轴之间的面积变动 $dA = f(x)dx$
 1. 类似于 1.1, 首先将改变的一长条面积看做矩形（近似），则 dA 等于该矩形的面积
 2. $f(x)$: 矩形的长

- 3. dx : 矩形的宽
- 4. 由此可得 $dA = f(x)dx$

- 注意: 当 dx 足够小 ($dx \rightarrow 0$) 时, 以上所有的“近似”都可去掉 (极限思想)

1.2 导数

1.2.1 导数入门: 汽车的路程与速度

- 一辆汽车花费 10s 从静止状态沿直线开到 100m 外重新静止

- 计算速度: $v = \frac{ds}{dt}$
- * ds : 路程改变量
- * dt : 时间改变量
- $\frac{ds}{dt}(t) = \frac{s(t+dt)-s(t)}{dt}$ ($dt \rightarrow 0$)

- 计算某点的导数: 设 $S(t) = t^3$, 求 $\frac{ds}{dt}(2)$

- 代入式子 $\frac{ds}{dt}(2) = \frac{s(2+dt)-s(2)}{dt}$
- 将 $S(t) = t^3$ 代入可得: $\frac{ds}{dt}(2) = \frac{(2+dt)^3 - 2^3}{dt}$
- 化简可得:

$$\begin{aligned}\frac{ds}{dt}(2) &= \frac{2^3 + 3(2)^2 dt + 3(2)(dt)^2 + (dt)^3 - 2^3}{dt} \\ &= 3(2)^2 + 3(2)(dt) + (dt)^2\end{aligned}$$

- 令 $dt \rightarrow 0$, 原式化为 $3(2)^2 = 12$
- 结果为: $3(2)^2 = 12$

- 计算函数的导数: 设 $S(t) = t^3$, 求 $\frac{ds}{dt}(t)$

- 将上面推导过程中的 2 全部替换为 t 即可
- 结果为: $\frac{ds}{dt}(t) = 3(t)^2$

1.2.2 常见误区

- 导数是瞬时变化率 (X): 瞬时代表着没有变化。例如一张照片里的车没有速度。因此导数应该是附近变化的趋势。
- dt 是无穷小: 无穷小实际上意味着0。所以 dt 不是无穷小, 而是一个有限量, 只不过趋近于0。

1.3 用几何来推导

在此再次重申一下导数的意义: 导数是在一种微小的变化量下另一种变化量变化的趋势。对于函数 $f(x) = x^2$ 来说, 其导数 $\frac{df}{dx}$ 的意义是在 x 变化量很小 (仅为 dx) 的情况下, 其函数值的改变量 df 的变化趋势是多少。

1.3.1 $f(x) = x^2$ 的导数 $\frac{df}{dx}$

- 首先, 将 x^2 想象成一个正方形的面积, 其边长为 x
- 然后, 我们让 x 变成 $x + dx$, 也就是说, 正方形的边长增加了 dx
- 我们可以发现, 新增的面积, 实际上是由三部分组成, 分别是两个长度为 x , 宽度为 dx 的长方形, 和一个边长为 dx 的正方形
- 由此我们可以得出:

$$df = 2xdx + dx^2$$

- 因为 dx 足够小, 所以我们可以忽略掉其高次方的项, 即上式中的 dx^2
- 所以, $df = 2xdx$
- 将两遍同除 dx , 可得

$$\frac{df}{dx} = 2x$$

1.3.2 $f(x) = x^3$ 的导数 $\frac{df}{dx}$

过程和上面类似。

- 首先, 将 x^3 想象成一个正方体的体积, 其边长为 x
- 然后, 我们让 x 变成 $x + dx$, 也就是说, 正方体的边长增加了 dx

- 我们可以发现，新增的体积，实际上是由七部分组成，分别是三个底面积为 x^2 ，高度为 dx 的长方体，三个底面积为 dx^2 ，高度为 x 的长方体，和一个边长为 dx^3 的正方体
- 由此我们可以得出：

$$df = 3x^2 dx + 3xdx^2 + dx^3$$

- 因为 dx 足够小，所以我们可以忽略掉其高次方的项，即上式中的

$$3xdx^2 + dx^3$$

- 所以， $df = 3x^2 dx$
- 将两遍同除 dx ，可得

$$\frac{df}{dx} = 3x^2$$

1.3.3 $f(x) = \frac{1}{x}$ 的导数

- 首先，想象一个面积恒定为1的矩形，其一条边的长度为 x ，另一条边长度就为 $\frac{1}{x}$
- 现在，我们可以将问题转化为：当 x 增加了 dx ， $\frac{1}{x}$ 需要减少多少 $d(\frac{1}{x})$ 才能够维持该矩形面积不变呢？
- 注意，因为当 x 增加时， $\frac{1}{x}$ 减小，所以 dx 和 $d\frac{1}{x}$ 的改变方向不一样。
- 该矩形减少的面积为 $xd(\frac{1}{x})$
- 而矩形增加的面积为 $dx(\frac{1}{x} - d\frac{1}{x})$
- 令两者相等，化简可得

$$\begin{aligned} xd\frac{1}{x} &= dx\left(\frac{1}{x} - d\frac{1}{x}\right) \\ d\frac{1}{x}(x+dx) &= dx\frac{1}{x} \\ \frac{d\frac{1}{x}}{dx} &= \frac{1}{x(x+dx)} \\ \frac{d\frac{1}{x}}{dx} &= \frac{1}{x^2 + xdx} \end{aligned}$$

- 由于 $dx \rightarrow 0$, 所以上式中的 $x dx$ 可以忽略。
- 同时, 由于 dx 和 $d\frac{1}{x}$ 改变方向不同, 所以原式需要添上一个负号
- 综上所述可得:

$$\frac{d\frac{1}{x}}{dx} = -\frac{1}{x^2}$$

1.3.4 \sqrt{x} 的导数

- 首先, 想象一个面积恒定为 x 的矩形, 其边长为 \sqrt{x}
- 现在, 我们可以将问题转化为: 当正方形的面积 x 增加了 dx , 该矩形的边长 \sqrt{x} 如何变化?
- 我们可以发现, 新增的面积, 实际上是由三部分组成, 分别是两个长度为 \sqrt{x} , 宽度为 $d\sqrt{x}$ 的长方形, 和一个边长为 $d\sqrt{x}$ 的正方形
- 所以该正方形增加的面积为 $2\sqrt{x}d\sqrt{x} + d\sqrt{x}^2$
- 由于 $dx \rightarrow 0$, 所以 $d\sqrt{x}^2 \rightarrow 0$, 所以 $d\sqrt{x}^2$ 的高次项可以忽略。
- 令两者相等, 化简可得

$$\begin{aligned} dx &= 2\sqrt{x}d\sqrt{x} \\ \frac{1}{2\sqrt{x}} &= \frac{d\sqrt{x}}{dx} \\ \frac{d\sqrt{x}}{dx} &= \frac{1}{2\sqrt{x}} \end{aligned}$$

- 综上所述可得:

$$\frac{d\sqrt{x}}{dx} = \frac{1}{2\sqrt{x}}$$

1.3.5 $\sin(x)$ 的导数

- 首先, 创建一个单位圆 (圆心在原点, 半径为1)
- 假设目前的角度为 θ , 即弧度已经走过了 θ 的长度, 则目前其 $\sin(\theta)$ 为该点到 x 轴的距离
- 现在的问题是: 当你的弧度改变 $d\theta$, 将使你距离 x 轴的高度改变多少?

- 我们以 $d\theta$ 这条弧度为斜边创建一个直角三角形（因为 $d\theta \rightarrow 0$, 所以可以近似为一条直线）
- 则该小三角形和大三角形相似。
- 则有 $\frac{d(\sin(\theta))}{d\theta} = \frac{1}{\|x\|}$, 等式左边为小三角形推出, 右边为大三角形推出。
- 而 $\frac{1}{\|x\|} = \cos(\theta)$
- 综上所述可得:

$$\frac{d(\sin(\theta))}{d\theta} = \cos(\theta)$$

1.4 直观理解链式法则和乘积法则

1.4.1 加法法则

内容 函数的和的导数 = 函数的导数的和

$$\frac{d(g(x) + h(x))}{dx} = \frac{dg}{dx} + \frac{dh}{dx}$$

直观理解 将两个函数 $g(x) + h(x)$ 看成是一个函数 $I(x)$, 该函数的函数值为两个函数值的和。由此易得: 该函数在 x 改变 dx 的情况下函数值的该变量为两个函数各自在该情况下函数值该变量的和。

1.4.2 乘法法则

内容 左乘右导 + 右乘左导

$$\text{Left } d \text{ right} + \text{right } d \text{ left}.$$

$$\frac{d(g(x)h(x))}{dx} = g(x)\frac{dh(x)}{dx} + h(x)\frac{dg(x)}{dx}$$

直观理解: 面积法 假设 $\frac{d(g(x)h(x))}{dx} = \frac{d(\sin(x)x^2)}{dx}$

- 假设存在一个矩形, 边长分别为 $\sin(x)$ 和 x^2
- 当 x 改变了 dx , 边长分别改变了 $d(\sin(x))$ 和 dx^2 , 也就是 $\cos(x)$ 和 $2x$
- 则面积改变量由三部分组成, 分别为 $2x \sin(x)$ 的矩形, $x^2 \cos(x)$ 的矩形, $2x \cos(x)$ 的矩形

- 由于 $dx \rightarrow 0$, 所以 $d\sin(x)$ 和 dx^2 的高次项可以忽略, 即 $2x\cos(x)$ 被忽略
- 由此可得:

$$\frac{d(\sin(x)x^2)}{dx} = 2x\sin(x) + x^2\cos(x)$$

1.4.3 函数复合

内容 $\frac{d(g(h(x)))}{dx} = \frac{dg(h(x))}{dx} \frac{dh(x)}{dx} = \frac{dg}{dh} \frac{dh}{dx}$

直观理解: 数轴法 假设 $d\frac{d(g(h(x)))}{dx} = \frac{d(\sin(x^2))}{dx}$

- 假设有三根数轴, 分别代表了 x , x^2 , $\sin(x^2)$
- 令 x 改变 dx , 则 x^2 改变了 dx^2
- 令 x^2 为 h , 则 $dx^2 = dh$, $\sin(x^2) = \sin(h)$, $d\sin(x^2) = d\sin(h)$
- 由此可得:

$$\begin{aligned} d\sin(h) &= \cos(h)dh \\ d\sin(x^2) &= \cos(x^2)dx^2 \\ d\sin(x^2) &= \cos(x^2)2xdx \\ \frac{d\sin(x^2)}{dx} &= 2x\cos(x) \end{aligned}$$

1.5 指数函数求导 $M(t) = n^t$

$$\frac{n^{t+dt} - n^t}{dt} = ???$$

1.5.1 e 的定义过程

假设 $n = 2$

$$\begin{aligned} \frac{dM(t)}{dt} &= \frac{2^{t+dt} - 2^t}{dt} \\ \frac{dM(t)}{dt} &= 2^t \frac{2^{dt} - 1}{dt} \end{aligned}$$

- 问题转化为: $\frac{2^{dt}-1}{dt} = ???$
- 因为 $dt \rightarrow 0$, 所以 $\frac{2^{dt}-1}{dt}$ 趋近于一个常数, 大概为 $0.6931472\dots$, 我们令其为 C
- 由此可得: $\frac{dM(t)}{dt} = \frac{2^{dt}-1}{dt} 2^t = C 2^t$
- 常数 C 根据底数 n 的不同, 有不同的值。
- 根据尝试我们会发现, $n = 8$ 时的 C 值是 $n = 2$ 时 C 值的 3 倍。
- 有没有一种底数 n , 可以使得其常数 $C = 1$?
- 这个数为常数 e

1.5.2 内容

$$\frac{dn^{ct}}{dt} = \frac{e^{\ln(n)ct}}{dt} = c \ln(n) e^{\ln(n)ct}$$

1.5.3 推导过程

$$\frac{dM(t)}{dt} = \frac{d(e^{ct})}{dt} = ce^{ct}$$

- 利用 e 的性质 $2^t = e^{\ln(2)t}$
- 由此可得:

$$\frac{dn^{ct}}{dt} = c \ln(n) e^{\ln(n)ct}$$

1.6 隐函数求导

1.6.1 内容

1.6.2 推导过程

$x^2 + y^2 = 5^2$ 的导数 假设存在一个圆心在原点的圆 $x^2 + y^2 = 5^2$, 求点 $(3, 4)$ 的切线斜率。

- 关键: 将这个点放大到切线和圆几乎重合, 此时切线斜率 $Slope = \frac{dy}{dx}$

- 直接对圆的方程左右两边同时求导可得：

$$\begin{aligned}x^2 + y^2 &= 5^2 \\2xdx + 2ydy &= 0 \\\frac{dy}{dx} &= \frac{-x}{y}\end{aligned}$$

- 所以，切线斜率为：

$$Slope = \frac{-x}{y}$$

梯子问题 假设存在一个长度为5m的梯子靠在墙上，梯子的上端距离地面的距离是4m，现在梯子以1m/s的速度下落，问下端的移动速度是多少？

- 首先设上端的移动速度为 $y(t)$ ，下端的移动速度为 $x(t)$
- 两者可以得出公式： $y(t)^2 + x(t)^2 = 5^2$
- 对等式左右两边同时对 dt 求导
- 对左边求导的实际意义：当 t 改变了 dt 时，左边这个表达式的值会改变多少？

$$\begin{aligned}y(t)^2 + x(t)^2 &= 5^2 \\\frac{y(t)^2 + x(t)^2}{dt} &= 0 \\2x(t)\frac{dx}{dt} + dy(t)\frac{dy}{dt} &= 0\end{aligned}$$

- 结论和上一题一样。此题的意义在于展示隐函数变量之间的关系。

$\sin(x)y^2 = x$ 的导数 易得：

$$\sin(x)(2ydy) + y^2(\cos(x)dx) = dx$$

利用隐函数求导法则求函数导数

求 $y = \ln(x)p$

$$\begin{aligned}y &= \ln(x) \\e^y &= x \\e^y dy &= dx \\\frac{dy}{dx} &= \frac{1}{e^y} \\\frac{dy}{dx} &= \frac{1}{x}\end{aligned}$$

1.7 极限

1.7.1 导数的正式定义

$$\frac{df}{dx}(2) = \lim_{h \rightarrow 0} \frac{f(2+h) - f(2)}{h}$$

1.7.2 极限的 (ϵ, δ) 定义

在极限 $\lim_{h \rightarrow 0} \frac{f(2+h) - f(2)}{h}$ 的定义域附近 $(2 - \delta, 2 + \delta)$, 函数值 f 总是处于特定范围内的 $(f(2) - \epsilon, f(2) + \epsilon)$ 。

1.7.3 洛必达法则

假设存在两个函数 $f(x), g(x)$, 两者均在 $x = a$ 处函数值为0且可导 (意味着在此处的函数图像放大后可以看做是一条直线), 如何计算 $\frac{f(x)}{g(x)}$?

关键 由于两个函数在 $x = a$ 处函数值均为0, 所以上式无法直接求解。但我们可以取 x 为离 a 十分接近的值, 求 x 逼近于 a 的极限值。即求极限:

$$\lim_{x \rightarrow a} \frac{f(x)}{g(x)}$$

推导

- 将输入考虑为与 $x = a$ 距离 dx 的点
- 在此时, f 的值几乎就是导数值乘 dx , 即 $\frac{df}{dx}(a)dx$
- 同理, g 的值也为 $\frac{dg}{dx}(a)dx$

- 由此可得：

$$\begin{aligned}\frac{f(x)}{g(x)} &= \lim_{x \rightarrow a} \frac{f(x)}{g(x)} \\ &= \frac{\frac{df}{dx}(a)dx}{\frac{dg}{dx}(a)dx} \\ &= \frac{\frac{df}{dx}(a)}{\frac{dg}{dx}(a)} \\ &= \frac{f'(a)}{g'(a)}\end{aligned}$$

- 我们惊喜地发现：此时两个函数在 a 点比值的极限值就是它们导数在 a 点比值的极限值！

内容 假设两个函数 $f(x)$, $g(x)$ 在 $x = a$ 处函数值为 0 且可导，则

$$\frac{f(a)}{g(a)} = \lim_{x \rightarrow a} \frac{f'(a)}{g'(a)}$$

1.8 积分与微积分基本定理

1.8.1 推导

- 假设现在要计算 $v(t) = t(8 - t)$ 的车辆在八秒内走过的路程。
- 如果 $v(t) = 10$ 的话，我们就可以直接让两者相乘，获得 $s(t) = 10 * 8 = 80m/s$
- 而此时的路程实际上相当于速度函数与 x 轴相夹的面积。
- 推广该结论，对于任何速度函数 $v(t)$ ，其在几秒内跑过的路程都应该等于速度函数与 x 轴相夹的面积。
- 但是对于 $v(t) = t(8 - t)$ 的车辆，不能直接相乘。此时利用积分思想。
- 取 $dt \rightarrow 0$ ，对 $t + dt$ 的情况下，面积的增加部分可以看做一个矩形，则面积的改变量为 $v(t)dt$ ，由此可得：

$$ds = v(t)dt$$

- 由此可将 $v(t) = t(8 - t)$ 的速度函数分解成无数个 dt 区间，再将其加起来计算面积

- 由此可得:

$$s(t) = \sum dsdt = \sum t(8-t)dt$$

- 由于 $dt \rightarrow 0$, 所以我们不用 \sum 符号, 改用 \int 符号:

$$s(t) = \int t(8-t)dt$$

- 也就是说: 任何函数的导数的积分等于该函数自身

- 将上下限代入可得:

$$s = \int_0^8 t(8-t)dt = 4t^2 - \frac{1}{3}t^3|_0^8 = 85.33m$$

- 推广可得:

$$\int_a^b f(x)dx = F(a) - F(b)$$

$$\frac{dF}{dx}(x) = f(x)$$

$$F(x) = \int f(x)dx + C$$

1.8.2 内容

$$\int_a^b f(x)dx = F(a) - F(b)$$

$$\frac{dF}{dx}(x) = f(x)$$

$$F(x) = \int f(x)dx + C$$

1.9 面积与斜率的关系

从另一个角度诠释积分和导数的关系

1.9.1 求 $f(x) = \sin(x)$ 在 $(0, \pi)$ 上的平均值

- 根据平均值的定义: $\bar{f} = \frac{\sum f(x)}{|x|}$
- 但是由于本题的定义域是连续的, 所以先利用 $dx \rightarrow 0$, 将区域分解成 $\frac{\pi}{dx}$ 个区间
- 对每个区间的函数值进行求和(此处是求积分)再除以区间个数即可:

$$\begin{aligned}\bar{f} &= \frac{\int_0^\pi \sin(x)}{\frac{\pi}{dx}} \\ &= \frac{\int_0^\pi \sin(x) dx}{\pi} \\ &= \frac{F(\pi) - F(0)}{\pi} \\ &= \frac{-(\cos(\pi) + C) - (-(\cos(0) + C))}{\pi} \\ &= \frac{2}{\pi}\end{aligned}$$

1.9.2 意义

上面这个例子意味着: 函数的平均值=原函数在该定义域的两端的两点的连线的斜率

原因 函数在某点的函数值实际上就是其原函数在该点的切线的斜率。所以函数的平均值也可以看做其原函数所有切线斜率的平均值, 也就是其两端两点连线的斜率。

1.10 高阶导数

1.10.1 表示

$$\frac{d^2 f}{dx^2}$$

1.10.2 意义

导数的变化率

1.11 泰勒级数

1.11.1 作用

用多项式函数逼近其他函数，让函数变得更加友好

1.11.2 代数推导过程

假设现在用函数 $P(x) = c_0 + c_1x + c_2x^2$ 在 $x = 0$ 的附近表示 $\cos(x)$ ，求三个常数值。

- 首先，两者在 $x = 0$ 处的函数值应该相等，所以有：

$$P(0) = c_0 = \cos(0) = 1$$

- 其次，为了让 $P(x)$ 逼近 $\cos(x)$ ，两者在此时的函数走向应该一样，即两者在 $x = 0$ 处的导数值应该相等：

$$P'(0) = c_1 = -\sin(0) = 0$$

- 再次，两者在此时的导数的改变方向也应该一样，即两者在 $x = 0$ 处的二阶导数值应该相等：

$$P''(0) = 2c_2 = -\cos(0) = 1$$

$$c_2 = -\frac{1}{2}$$

- 可以无限套娃下去，得出公式：

$$P^{(n)}(x) = \cos^{(n)}(x)$$

- 由此可得泰勒公式：

$$f(x) = \frac{f(a)}{0!}(x-a)^0 + \frac{f'(a)}{1!}(x-a)^1 + \frac{f''(a)}{2!}(x-a)^2 + \dots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

1.11.3 几何推导二次项意义

假设还是上题。已经找到了 c_0 和 c_1 的值，求 c_2 的值。

- 我们画出 $\cos(x)$ 的导数 $-\sin(x)$ 的图像。

- 我们的题目可以转化为如何用函数 $P(x)$ 表示 $-\sin(x)$ 下的面积（在 $x = a$ 附近）
- 其中 c_0 是已知 $x = a$ 的面积， c_1 是新增的矩形面积，而因为我们不是在求积分，所以我们需要表示出矩形上面那一块三角形的面积。
- 三角形的底为 $x - a$ ，高用斜率乘底表示： $(-\sin(x))'(x - a)$
- 所以三角形的面积为： $\frac{1}{2}f''(a)(x - a)^2$

2 线性代数基础

2.1 向量是什么

物理学	数学	计算机科学
由方向和长度决定的量，可以在空间中任意移动	任何东西，只要向量相加以及与数字相乘有意义即可	有序的数字列表

2.2 向量加法与数乘的理解

2.2.1 向量

$$\vec{v} = \begin{bmatrix} m \\ n \end{bmatrix}$$

代表着在二维空间中从原点出发，横坐标(x)为m，纵坐标(y)为n的箭头。且向量与实数对一一对应。

2.2.2 向量加法

代数定义

$$\vec{v} + \vec{w} = \begin{bmatrix} m \\ n \end{bmatrix} + \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} m+p \\ n+q \end{bmatrix}$$

几何定义 两个向量的和为两个向量首尾相连组成的新向量

理解 将向量看作是一种运动，是从原点向某一方向迈出一定的距离。所以沿着两个向量运动的效果和直接向他们首尾相连的位置运动的效果一样。

或者可以将向量加法看作是数轴加法的拓展。

2.2.3 数乘向量

内容

$$c\vec{v} = \begin{bmatrix} c * m \\ c * n \end{bmatrix} = \begin{bmatrix} cp \\ cq \end{bmatrix}$$

2.3 线性组合，张成的向量与基

2.3.1 基向量

- \hat{i} : 指向正右方，长度为1，为x方向的单位向量
- \hat{j} : 指向正上方，长度为1，为y方向的单位向量

此时，每个向量可以看作是其基向量的进行缩放后的和

2.3.2 选择不同的基向量

此时，所有的向量都可以照常表示，但是和使用 \hat{i} 和 \hat{j} 的表示的数字完全不同。

2.3.3 线性组合

定义 两个数乘向量的和。

线性相关 某一个向量可以表示为该组向量中其他向量的线性组合。

线性无关 一组向量中没有任何向量可以表示为其他向量的线性组合。

2.3.4 向量空间

所有可表示为给定向量线性组合的向量的集合。

另一个角度理解线性相关 某一个向量没有对该组向量组成的向量空间做任何贡献。

2.4 矩阵与线性变换

2.4.1 线性变换

线性 可以看作是“保持网格线平行且等距分布”的变换

- 变换后的直线还是直线，没有弯曲
- 原点保持固定

变换 实际上是函数的另一种说法，接收输入，输出输出。与函数的区别是此处的输入和输出都是向量。

性质 变换后的所有向量依旧是变换后基向量的特定线性组合。

例子

$$\vec{v} = \begin{bmatrix} -1 \\ 2 \end{bmatrix} = -1\hat{i} + 2\hat{j}$$

$$\text{Transformed } \vec{v} = -1(\text{Transformed } \hat{i}) + 2(\text{Transformed } \hat{j})$$

$$\text{Transformed } \hat{i} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$\text{Transformed } \hat{j} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$\begin{aligned} \text{Transformed } \vec{v} &= -1 \begin{bmatrix} 1 \\ -2 \end{bmatrix} + 2 \begin{bmatrix} 3 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} -1(1) + 2(3) \\ -1(-2) + 2(0) \end{bmatrix} \\ &= \begin{bmatrix} 5 \\ 2 \end{bmatrix} \end{aligned}$$

推广

$$\begin{aligned}\hat{i} &= \begin{bmatrix} 1 \\ -2 \end{bmatrix} & \hat{j} &= \begin{bmatrix} 3 \\ 0 \end{bmatrix} \\ \begin{bmatrix} x \\ y \end{bmatrix} &= x \begin{bmatrix} 1 \\ -2 \end{bmatrix} + y \begin{bmatrix} 3 \\ 0 \end{bmatrix} & &= \begin{bmatrix} 1x + 3y \\ -2x + 0y \end{bmatrix}\end{aligned}$$

2.4.2 矩阵

推导 将 \hat{i} 和 \hat{j} 放在一起，就组成了矩阵。

$$\begin{bmatrix} \hat{i} & \hat{j} \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

2.4.3 矩阵乘法

推导 现在让这个线性变换作用于向量 $\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$x \begin{bmatrix} a \\ c \end{bmatrix} + y \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

定义

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = x \begin{bmatrix} a \\ c \end{bmatrix} + y \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

2.5 矩阵乘法与线性变换组合

2.5.1 矩阵乘法

推导 假设对一个向量 \vec{v} 进行两次线性变换 A 和 B ，其变换结果和直接进行线性变换到最终结果的效果是一样的。由此可得：

$$BA\vec{v} = Z\vec{v}$$

$$BA = Z$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}$$

注意，这里的阅读顺序为从右向左，先进行A矩阵的变换，再进行B矩阵的变换。

例子 现在有矩阵 $M_1 = \begin{bmatrix} 1 & -2 \\ 1 & 0 \end{bmatrix}$, $M_2 = \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix}$, 求 $M_1 M_2$

$$M_2 M_1 = ?$$

$$\begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

$$\begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 1 \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -2 \\ 0 \end{bmatrix} = -2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 0 \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \end{bmatrix}$$

$$M_2 M_1 = \begin{bmatrix} 2 & 0 \\ 1 & -2 \end{bmatrix}$$

内容 用变量代替上面的具体值可得:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e \\ g \end{bmatrix} = e \begin{bmatrix} a \\ c \end{bmatrix} + g \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} ae + bg \\ ce + dg \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} f \\ h \end{bmatrix} = f \begin{bmatrix} a \\ c \end{bmatrix} + h \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} af + bh \\ cf + dh \end{bmatrix}$$

$$M_2 M_1 = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

运算律

交换律: 不存在 很明显对一个向量进行线性变换时先拉伸后旋转和先旋转后拉伸效果不同。

结合律: 存在 结合律并没有改变向量和线性变换的作用顺序。

2.6 行列式

当我们对一个空间进行变换时, 里面的向量都会被拉伸。问题是: 我们怎么测量变换究竟对空间有多少拉伸或者挤压?

2.6.1 定义

线性变换缩放空间的比例（带有方向，所以值可以为负）。

$$\det\begin{pmatrix} 3 & 2 \\ 0 & 2 \end{pmatrix} = 6$$

2.7 逆矩阵、列空间和零空间

2.8 矩阵的用途

求解线性方程组。

$$2x + 5y + 3z = -3$$

$$4x + 0y + 8z = 0$$

$$1x + 3y + 0z = 2$$

用矩阵来表示常数系数：

$$\begin{bmatrix} 2 & 5 & 3 \\ 4 & 0 & 8 \\ 1 & 3 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \\ 2 \end{bmatrix}$$

$$A\vec{x} = \vec{v}$$

意义 A 代表一种线性变换，求解 \vec{x} 意味着让我们去找到一个向量 x 使其在进行该线性变换后与 v 重合。

2.8.1 逆矩阵

前提 矩阵的行列式不为0

作用 让我们有办法求解上面的问题：让向量 v 进行 A 的逆线性变换即可得到 x

形式

$$A^{-1} = \begin{bmatrix} 3 & 1 \\ 0 & -2 \end{bmatrix}^{-1}$$

求法 根据定义：矩阵 A 的逆矩阵代表着相反的线性变换，这意味着一个向量左乘 $A^{-1}A$ 为它自己。即：

$$A^{-1}A = E$$

2.8.2 秩

意义 变换后空间的维度数/列空间的维数

2.8.3 矩阵的列空间

所有可能的输出向量 $A\vec{v}$ 组成的集合

2.8.4 零空间（核）

经过矩阵变换后落在原点的向量的集合

2.9 点积与对偶性

2.9.1 点积

标准看法 $\vec{v} \cdot \vec{w} = \begin{bmatrix} 2 \\ 7 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 2 \\ 8 \end{bmatrix} = 2 * 8 + 7 * 2 + 1 * 8 = 38$

几何意义 $\vec{v} \cdot \vec{w} = (\text{length of projected } \vec{w})(\text{length of } \vec{v})$

对偶性

为什么向量的点积和其向量的顺序无关？

- 假设两个向量 \vec{w} 和 \vec{v} 是关于某一个直线对称相等的。在此时两个向量向彼此的投影和向量的长度都一样。
- 现在将其中一个向量（假设为 \vec{w} ）变成原来的两倍。
- 对于 \vec{w} 向 \vec{v} 做投影来说， \vec{w} 的长度变成了原来的两倍，其投影长度自然也是原来的两倍。
- 对于 \vec{v} 向 \vec{w} 做投影来说， \vec{v} 的长度变成了原来的两倍。

- 现在将其中一个向量变成原来的x倍，由以上可得其点积自然变成原来的x倍。
- 因此，点积的结果和向量的顺序无关。

为什么点积的几何意义和其计算方式是有关系的呢？ 我们已知一个二维向量做线性变换实际上就是对 \hat{i} 和 \hat{j} 做线性变换，而二维向量线性变换到一维（即数轴）上的结果是一个数。

- 假设存在向量 $\vec{v} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$ ，现在要将整个空间线性变换到数轴上。
- 对于原本的向量 \vec{v} 来说，其基向量 $\hat{i} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\hat{j} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- 现在将向量都线性变换到一个数轴上，实际上就是将两个基向量变成一维向量。假设此变换为 $\begin{bmatrix} 1 & -2 \end{bmatrix}$ ，意味着 \hat{i} 落在2上， \hat{j} 落在1上。
- 向量 \vec{v} 进行该线性变换的结果： $\begin{bmatrix} 1 & -2 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = 1(4) + (-2)(3) = -2$

此处的计算很接近向量的点积，只要将变换矩阵 $\begin{bmatrix} 1 & -2 \end{bmatrix}$ 看作是一个倾倒的二维向量。

- 假设二维空间里存在一个倾斜的数轴，其0点就在原点上。现在该数轴上有一个向量 \hat{u} ，长度为1，指向正方向。
- 现在假设该数轴就是上述线性变换的结果。问 \hat{i} 和 \hat{j} 如何计算？
- 根据对称性可知： \hat{i} , \hat{j} 分别和 \hat{u} 关于某一直线对称，所以两个基向量分别与 \hat{u} 做投影的结果和 \hat{u} 向两个基向量做投影的结果相等。而 \hat{u} 向两个基向量做投影的结果就是该向量的坐标。即 $\begin{bmatrix} u_x & u_y \end{bmatrix}$ ，即基向量变换后的位置。
- 因此，计算任意向量 \vec{v} 到该数轴的投影可以解读为：该向量经过该线性变换到数轴上的长度=投影矩阵(\hat{u})和这个向量的坐标的乘积= \vec{v} 和 \hat{u} 的点积。
- 现在将单位向量 \hat{u} 进行缩放x倍。

- 这意味着原来的基向量的线性变换也变成了原来的x倍。
- 因此，计算任意向量 \vec{v} 到该数轴的投影同样可以解读为：该向量经过该线性变换到数轴上的长度=投影矩阵（ \hat{u} ）和这个向量的坐标的乘积= \vec{v} 和 \hat{u} 的点积。

2.10 叉积的标准介绍

2.10.1 导入

在二维空间中的两个向量 \vec{v} 和 \vec{w} ，两者的叉积为以两者为边组成的平行四边形的面积。

$$\vec{v} \times \vec{v} = \text{Area of parallelogram}$$

如果 \vec{v} 在 \vec{w} 的右边，则面积为正；如果 \vec{v} 在 \vec{w} 的左边，则面积为负。

2.10.2 计算方法

根据行列式的定义，叉积的大小=两个向量组成的行列式的值

2.10.3 实际定义

叉积的结果是一个向量：长度为两个向量组成的行列式的值，方向与两个向量组成的平面垂直，由右手定则确定。

$$\vec{v} \times \vec{v} = \vec{p}$$

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \det\left(\begin{bmatrix} \hat{i} & v_1 & w_1 \\ \hat{j} & v_2 & w_2 \\ \hat{k} & v_3 & w_3 \end{bmatrix}\right) = \vec{p}$$

2.10.4 几何解释

由于最终叉积的结果是一个向量，所以可以看做是两个特殊向量的点积结果：

$$\vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \cdot \begin{bmatrix} \hat{i} \\ \hat{j} \\ \hat{k} \end{bmatrix}$$

与此同时，叉积的结果又可以表示为上述行列式的值。

- 根据行列式的几何定义可得，该行列式的值为三个向量组成的平行六面体的体积。
- 根据点积的几何定义可得，该点积的结果为一个向量朝着另一个向量做投影的长度乘另一个向量的长度。此处我们假设做投影的向量是由基向量组成的向量。
- 假设我们选取向量 \vec{v} 和 \vec{w} 所在的平面为底面，根据平行六面体的体积公式可得： \vec{v} 和 \vec{w} 组成的平行四边形的底面积*高
- 此时我们会发现，所谓的高就是基向量组成的向量向垂直于 \vec{v} 和 \vec{w} 确定的平面的向量投影的长度。
- 因此，假设要用点积表示叉积的结果，那么向量 \vec{p} 就必须是一个垂直于 \vec{v} 和 \vec{w} 确定的平面的，且长度为两者确定的平行四边形面积的向量。

2.11 特征向量与特征值

2.11.1 几何意义

特征向量 在对二维空间进行线性变换时，有一些向量并不进行旋转，只是进行缩放。这些向量就是特征向量。

特征值 特征向量的缩放因子。

2.11.2 计算思想

首先根据定义：

$$A\vec{v} = \lambda\vec{v}$$

其中 \vec{v} 为特征向量， λ 为特征值。由于等式左边为矩阵和向量的乘法，右边为数乘向量，为了统一思想，我们将 λ 变成矩阵 I' ：

$$\begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix}$$

现在，将等式右边的移到等式左边可得：

$$(A - I')\vec{v} = \vec{0}$$

根据线性变换的性质：只有当 $(A - I')$ 这个线性变换是将高维压缩到低维时，才会存在非0向量 \vec{v} 满足该等式。即：

$$\det(A - I') = 0$$

2.11.3 特征基

如果所有的特征向量都是基向量，则会发生什么？实际上，这会导致A矩阵变成一个对角矩阵，其对角线上的每一个数字都是一个特征值。

计算特征基 将特征向量组成一个矩阵T，令原矩阵A左乘 T^{-1} ，右乘T，所得的是对角矩阵。

2.12 抽象向量空间

2.12.1 用矩阵计算导数

假设原函数只存在多项式。

- 假设该向量空间共有n维，令一个向量的每一行代表x的一个次方，从第一行到第n行分别是 x^1 到 x^n 。
- 该空间存在n个基向量，每一个基向量只有一行非0，且这一行为 x^k 。
- 现在令导数矩阵的每一列为相对应基向量的导数。可得导数矩阵A：

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 2 & 0 & \dots \\ 0 & 0 & 0 & 3 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

- 当要计算导数时，将该函数的每一项多项式的系数提取出来，排列成列向量B，随后将B左乘导数矩阵A即可得到函数的导数的系数向量C

3 机器学习

3.1 学习方式分类

主要分为以下四类：

1. 监督学习

- 特点：监督学习是使用已知正确答案的示例来训练网络。已知数据和其一一对应的标签，训练一个预测模型，将输入数据映射到标签的过程。
- 常见应用场景：监督式学习的常见应用场景如分类问题和回归问题。
- 算法举例：常见的有监督机器学习算法包括支持向量机(Support Vector Machine, SVM)，朴素贝叶斯(Naive Bayes)，逻辑回归(Logistic Regression)，K近邻(K-Nearest Neighborhood, KNN)，决策树(Decision Tree)，随机森林(Random Forest)，AdaBoost以及线性判别分析(Linear Discriminant Analysis, LDA)等。深度学习(Deep Learning)也是大多数以监督学习的方式呈现。

2. 非监督学习

- 定义：在非监督式学习中，数据并不被特别标识，适用于你具有数据集但无标签的情况。学习模型是为了推断出数据的一些内在结构。
- 常见应用场景：常见的应用场景包括关联规则的学习以及聚类等。
- 算法举例：常见算法包括Apriori算法以及k-Means算法。

3. 半监督学习

- 特点：在此学习方式下，输入数据部分被标记，部分没有被标记，这种学习模型可以用来进行预测。
- 常见应用场景：应用场景包括分类和回归，算法包括一些对常用监督式学习算法的延伸，通过对已标记数据建模，在此基础上，对未标记数据进行预测。

- 算法举例：常见算法如图论推理算法（Graph Inference）或者拉普拉斯支持向量机（Laplacian SVM）等。

4. 弱监督学习

- 特点：弱监督学习可以看做是多个标记的数据集合，次集合可以是空集，单个元素，或包含多种情况（没有标记，有一个标记，和有多个标记）的多个元素。数据集的标签是不可靠的，这里的不可靠可以是标记不正确，多种标记，标记不充分，局部标记等。已知数据和其一一对应的弱标签，训练一个智能算法，将输入数据映射到一组更强的标签的过程。标签的强弱指的是标签蕴含的信息量的多少，比如相对于分割的标签来说，分类的标签就是弱标签。
- 常见应用场景：在企业数据应用的场景下，人们最常用的可能就是监督式学习和非监督式学习的模型。在图像识别等领域，由于存在大量的非标识的数据和少量的可标识数据，目前半监督式学习是一个很热的话题。
- 算法举例：举例，给出一张包含气球的图片，需要得出气球在图片中的位置及气球和背景的分割线，这就是已知弱标签学习强标签的问题。

3.2 监督学习的步骤

假设我们要训练一个网络，让其从照片库中（其中包含气球的照片）识别出气球的照片。以下就是我们在这个假设场景中所要采取的步骤。

1. 数据集的创建和分类：首先，浏览你的照片（数据集），确定所有包含气球的照片，并对其进行标注。然后，将所有照片分为训练集和验证集。目标就是在深度网络中找一函数，这个函数输入是任意一张照片，当照片中包含气球时，输出1，否则输出0。
2. 数据增强（Data Augmentation）：当原始数据搜集和标注完毕，一般搜集的数据并不一定包含目标在各种扰动下的信息。数据的好坏对于机器学习模型的预测能力至关重要，因此一般会进行数据增强。对于图像数据来说，数据增强一般包括，图像旋转，平移，颜色变换，裁剪，仿射变换等。

3. 特征工程 (Feature Engineering): 一般来讲，特征工程包含特征提取和特征选择。常见的手工特征(Hand-Crafted Feature)有尺度不变特征变换(Scale-Invariant Feature Transform, SIFT)，方向梯度直方图(Histogram of Oriented Gradient, HOG)等。由于手工特征是启发式的，其算法设计背后的出发点不同，将这些特征组合在一起的时候有可能会产生冲突，如何将组合特征的效能发挥出来，使原始数据在特征空间中的判别性最大化，就需要用到特征选择的方法。在深度学习方法大获成功之后，人们很大一部分不再关注特征工程本身。因为，最常用到的卷积神经网络(Convolutional Neural Networks, CNNs)本身就是一种特征提取和选择的引擎。研究者提出的不同的网络结构、正则化、归一化方法实际上就是深度学习背景下的特征工程。
4. 构建预测模型和损失: 将原始数据映射到特征空间之后，也就意味着我们得到了比较合理的输入。下一步就是构建合适的预测模型得到对应输入的输出。而如何保证模型的输出和输入标签的一致性，就需要构建模型预测和标签之间的损失函数，常见的损失函数(Loss Function)有交叉熵、均方差等。通过优化方法不断迭代，使模型从最初的初始化状态一步步变化为有预测能力的模型的过程，实际上就是学习的过程。
5. 训练: 选择合适的模型和超参数进行初始化，其中超参数比如支持向量机中核函数、误差项惩罚权重等。当模型初始化参数设定好后，将制作好的特征数据输入到模型，通过合适的优化方法不断缩小输出与标签之间的差距，当迭代过程到了截止条件，就可以得到训练好的模型。优化方法最常见的就是梯度下降法及其变种，使用梯度下降法的前提是优化目标函数对于模型是可导的。
6. 验证和模型选择: 训练完训练集图片后，需要进行模型测试。利用验证集来验证模型是否可以准确地挑选出含有气球在内的照片。在此过程中，通常会通过调整和模型相关的各种事物（超参数）来重复步骤2和3，诸如里面有多少个节点，有多少层，使用怎样的激活函数和损失函数，如何在反向传播阶段积极有效地训练权值等等。
7. 测试及应用: 当有了一个准确的模型，就可以将该模型部署到你的应用程序中。你可以将预测功能发布为API (Application Programming

Interface, 应用程序编程接口) 调用, 并且你可以从软件中调用该API, 从而进行推理并给出相应的结果。

3.3 误差、偏差和方差的区别和联系

3.3.1 误差(error)

- 一般地, 我们把学习器的实际预测输出与样本的真是输出之间的差异称为“误差”。
- $\text{Error} = \text{Bias} + \text{Variance} + \text{Noise}$, Error反映的是整个模型的准确度。

3.3.2 噪声(Noise)

描述了在当前任务上任何学习算法所能达到的期望泛化误差的下界, 即刻画了学习问题本身的难度。

3.3.3 偏差(Bias)

- Bias衡量模型拟合训练数据的能力 (训练数据不一定是整个 training dataset, 而是只用于训练它的那一部分数据, 例如: mini-batch), Bias反映的是模型在样本上的输出与真实值之间的误差, 即模型本身的精准度。
- Bias越小, 拟合能力越高 (可能产生overfitting); 反之, 拟合能力越低 (可能产生underfitting)。
- 偏差越大, 越偏离真实数据。

3.3.4 方差(Variance)

- 公式: $S_N^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$
- 描述的是预测值的变化范围, 离散程度, 也就是离其期望值的距离。方差越大, 数据的分布越分散, 模型的稳定程度越差。
- 反映的是模型每一次输出结果与模型输出期望之间的误差, 即模型的稳定性。

- Variance越小，模型的泛化的能力越高；反之，模型的泛化的能力越低。

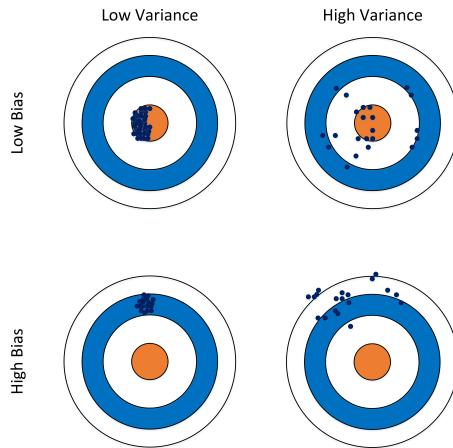


图 1: 方差和偏差对数据分布的影响

3.4 损失函数

3.4.1 常见的损失函数

损失函数可分为经验风险损失函数和结构风险损失函数。经验风险损失函数指预测结果和实际结果的差别，结构风险损失函数是在经验风险损失函数上加上正则项。

0-1损失函数

$$L(Y, f(x)) = \begin{cases} 1, & Y \neq f(x) \\ 0, & Y = f(x) \end{cases}$$

一般的在实际使用中，相等的条件过于严格，可适当放宽条件：

$$L(Y, f(x)) = \begin{cases} 1, & |Y - f(x)| \geq T \\ 0, & |Y - f(x)| < T \end{cases}$$

绝对值损失函数

$$L(Y, f(x)) = |Y - f(x)|$$

平方损失函数

$$L(Y, f(x)) = \sum_N (Y - f(x))^2$$

对数损失函数

$$L(Y, P(Y|X)) = -\log P(Y|X)$$

常见的逻辑回归使用的就是对数损失函数，有很多人认为逻辑回归的损失函数是平方损失，其实不然。逻辑回归它假设样本服从伯努利分布（0-1分布），进而求得满足该分布的似然函数，接着取对数求极值等。逻辑回归推导出的经验风险函数是最小化负的似然函数，从损失函数的角度看，就是对数损失函数。

指数损失函数

$$L(Y, f(x)) = \exp(-Y f(x))$$

Hinge损失函数

$$L(y) = \max(0, 1 - ty)$$

统一的形式：

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

其中y是预测值，范围为(-1,1)，t为目标值，其为-1或1。

3.5 代价函数

3.5.1 为什么需要代价函数

1. 为了得到训练逻辑回归模型的参数，需要一个代价函数，通过训练代价函数来得到参数。
2. 用于找到最优解的目的函数。

3.5.2 代价函数作用原理

以平方误差代价函数为例，假设函数为 $h(x) = \theta_0 x$ 。平方误差代价函数的主要思想就是将实际数据给出的值与拟合出的线的对应值做差，求出拟合出的直线与实际的差距。在实际应用中，为了避免因个别极端数据产生

的影响，采用类似方差再取二分之一的方式来减小个别数据的影响。因此，引出代价函数：

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

最优解即为代价函数的最小值 $\min J(\theta_0, \theta_1)$ 。如果是1个参数，代价函数一般通过二维曲线便可直观看出。如果是2个参数，代价函数通过三维图像可看出效果，参数越多，越复杂。当参数为2个时，代价函数是三维图像，如图2所示。

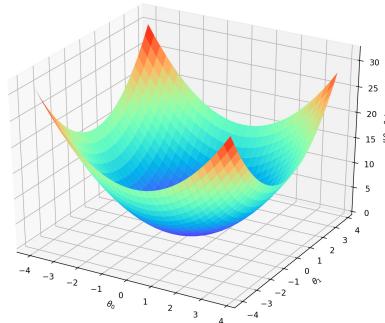


图 2：代价函数的三维图像

3.5.3 代价函数一定要非负吗？

目标函数存在一个下界，在优化过程当中，如果优化算法能够使目标函数不断减小，根据单调有界准则，这个优化算法就能证明是收敛有效的。

但是只要设计的目标函数有下界，基本上都可以，代价函数非负更为方便。

3.5.4 常见的代价函数

二次代价函数(quadratic cost)

$$J = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

其中， J 表示代价函数， x 表示样本， y 表示实际值， a 表示输出值， n 表示样本的总数。使用一个样本为例简单说明，此时二次代价函数为：

$$J = \frac{(y - a)^2}{2}$$

假如使用梯度下降法（Gradient descent）来调整权值参数的大小，权值 w 和偏置 b 的梯度推导如下：

$$\frac{\partial J}{\partial b} = (a - y)\sigma'(z)$$

其中， z_{ff} 表示神经元的输入， σ_{ff} 表示激活函数。权值 w_{ff} 和偏置 b_{ff} 的梯度跟激活函数的梯度成正比，激活函数的梯度越大，权值 w_{ff} 和偏置 b_{ff} 的大小调整得越快，训练收敛得就越快。

交叉熵代价函数(cross-entropy)

$$J = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln (1 - a)]$$

其中， J 表示代价函数， x 表示样本， y 表示实际值， a 表示输出值， n 表示样本的总数。权值 w 和偏置 b_{ff} 的梯度推导如下：

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y), \quad \frac{\partial J}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y)$$

当误差越大时，梯度就越大，权值 w 和偏置 b 调整就越快，训练的速度也就越快。

将交叉熵看做是代价函数的两点原因

- 它是 负的， $C \in 0$ 。可以看出：式子中的求和中的所有独 的项都是负数的，因为对数函数的定义域是 $(0, 1)$ ，并且求和前 有 个负号，所以结果是非负。
- 如果对于所有的训练输 x ，神经元实际的输出接近 标值，那么交 叉熵将接近0。

对数似然代价函数(log-likelihood cost) 对数似然函数常用来作为softmax回归的代价函数。深度学习中普遍的做法是将softmax作为最后一层，此时常用的代价函数是对数似然代价函数。对数似然代价函数与softmax的组合和交叉熵与sigmoid函数的组合非常相似。对数似然代价函数在二分类时可以化简为交叉熵代价函数的形式。

3.5.5 为什么用交叉熵代替二次代价函数

为什么不用二次方代价函数 由上一节可知，权值 w 和偏置 b 的偏导数为 $\frac{\partial J}{\partial w} = (a-y)\sigma'(z)x$, $\frac{\partial J}{\partial b} = (a-y)\sigma'(z)$, 偏导数受激活函数的导数影响, sigmoid函数导数在输出接近0和1时非常小, 会导致一些实例在刚开始训练时学习得非常慢。

为什么要用交叉熵 交叉熵函数权值 w 和偏置 b 的梯度推导为:

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y), \quad \frac{\partial J}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y)$$

由以上公式可知, 权重学习的速度受到 $\sigma(z) - y$ 影响, 更大的误差, 就有更快的学习速度, 避免了二次代价函数方程中因 $\sigma'(z)$ 导致的学习缓慢的情况。

4 神经网络

4.1 什么是神经网络

- 定义：具有适应性的简单单元组成的广泛并行互联的网络其组织能够模拟生物神经系统对真实世界物体所做出的交互反应
- 基础单位：神经元模型：兴奋——发送物质——引起相连神经元电位变化——达到阈值——激活——兴奋
- 组成：
 - n个其他神经元的输入信号
 - n个带权重的连接
 - 阈值
 - 激活函数及其产生的输出

4.2 为什么要用神经网络

仿生学原理：模仿人脑的结构，获得最好的表现

4.3 神经网络怎么训练

4.3.1 成本函数的建立

神经网络，乃至一切机器学习模型，究竟应该如何训练呢？

根据（愚蠢的）人类的经验，我们都是先照猫画虎，照着别人做，随后在犯错误后被家长或老师或其他长辈训斥，从而获得经验，改变提高自己，从而提升自己的能力的。

人类试图将自己的学习过程（并不怎么聪明和有效率）应用到一切机器学习模型（因此也包括神经网络）上。

其中最重要的（之一），同时也是必须最先做的工作，就是找到告诉机器它做错了的办法。就像当初我们挨打或者挨骂那样，告诉机器应该改正自己的行为。

于是，数学家们就想到了，用一个函数来表示犯错成本，这个函数值越大，说明机器做错的事情越多，错误的程度越严重，以此来对命令机器进行反思，改进，最后在这方面表现的更好（也就是函数值更低）。

所以，首先，我们需要量化一个指标来对神经网络的表现进行评价和测量，以此作为标准对神经网络的参数进行优化。这就是成本函数。

- 首先确立建立模型时想要最大化的可能性 L ，假设样本个体都相互独立，则有假设：

$$L(\mathbf{w}) = P(y|\mathbf{x}; \mathbf{w}) = \prod_{i=1}^n P(y^{(i)}|\mathbf{x}^{(i)}; \mathbf{w}) = \prod_{i=1}^n (\phi(z^{(i)}))^{y^{(i)}} ((1-\phi(z^{(i)}))^{1-y^{(i)}})$$

- 求得该函数的对数似然函数为：

$$l(\mathbf{w}) = \log(L(\mathbf{w})) = \sum_{i=1}^n [y^{(i)} \log(\phi(z^{(i)})) + (1 - y^{(i)}) \log(1 - \phi(z^{(i)}))]$$

- 用梯度下降法最小化代价函数 J 可得：

$$J(\mathbf{w}) = - \sum_{i=1}^n y^{[i]} \log(a^{[i]}) + (1 - y^{[i]}) \log(1 - a^{[i]})$$

- 其中 $y^{[i]} = \phi(z^{[i]})$: 数据集中第*i*个样本用前向传播算法计算出的Sigmoid值
- 其中上标 i 为训练集中特定样本的索引

- 添加正则化项减少过拟合机会：

$$L_2 = \lambda \|\mathbf{w}\|_2^2 = \lambda \sum_{j=1}^m w_j^2$$

- 其中 j 是某一种特征

- 综合以上两者可得：

$$J(\mathbf{w}) = - \left[\sum_{i=1}^n y^{[i]} \log(a^{[i]}) + (1 - y^{[i]}) \log(1 - a^{[i]}) \right] + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

4.3.2 多元分类MLP的逻辑成本函数

- 对于用于多元分类的MLP，返回的是一个有 t 个元素的输出向量，需要与 $t * 1$ 维独热编码表示的向量进行比较：

$$\mathbf{a}^{(out)} = \begin{bmatrix} 0.1 \\ 0.9 \\ \dots \\ 0.3 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 1 \\ \dots \\ 0 \end{bmatrix}$$

- 将逻辑成本函数推广到网络中所有的 t 激活单元，可得成本函数（不包括正则化项）为：

$$J(\mathbf{W}) = - \sum_{i=1}^n \sum_{j=1}^t y_j^{[i]} \log(a_j^{[i]}) + (1 - y_j^{[i]}) \log(1 - a_j^{[i]})$$

- 其中上标 i 为训练集中特定样本的索引
- 其中下标 j 为某一种特征

- 惩罚项为：

$$\frac{\lambda}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{u_l} \sum_{j=1}^{u_{l+1}} (w_{j,i}^{(l)})^2$$

- 其中 u_l 是一个给定的1层单位数

- 综上可得：

$$J(\mathbf{W}) = - \sum_{i=1}^n \sum_{j=1}^t y_j^{[i]} \log(a_j^{[i]}) + (1 - y_j^{[i]}) \log(1 - a_j^{[i]}) + \frac{\lambda}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{u_l} \sum_{j=1}^{u_{l+1}} (w_{j,i}^{(l)})^2$$

4.3.3 反向传播算法

为什么不用正向传播算法？ 答案其实很简单。因为正向传播算法做的是矩阵和矩阵的乘法，而反向传播算法做的是矩阵和向量的乘法，会节省很多计算资源。

适用范围 多层前馈神经网络和其他神经网络。

关键思想 从后往前，层层求导

神经网络组成 在此我们先规定一下神经网络的各个部分及其符号，利于我们之后讨论参数更新等问题。

- **输入层** d 个输入：接收输入 x_i
- **隐含层** q 个神经元：接收输入 $\alpha_h = \sum_{i=1}^d v_{ih} x_i$ ，其中 v_{ih} 为权重，输入经过激活函数以后和阈值 γ_h 进行比较，输出为 b_h
- **输出层** l 个神经元：接收输入 $\beta_j = \sum_{h=1}^q w_{hj} b_h$ ，其中 w_{hj} 为权重，输入经过激活函数以后和阈值 θ_j 进行比较，输出为 y_i

其中，一般输入个数为参数个数，输出个数为标记个数

参数个数： $d * q$ 个权重+ q 个阈值+ $q * l$ 个权重+ l 个阈值

输出 假定 $\hat{\mathbf{y}}_k = (\hat{y}_1^k, \hat{y}_2^k, \dots, \hat{y}_l^k)$ ，其中 $\hat{y}_j^k = f(\beta_j - \theta_j)$

优化目标：均方误差 对第 k 个样例求所有 1 个属性值的均方误差。 $1/2$ 求导时使用：

$$E_k = \frac{1}{2} \sum_{j=1}^l (\hat{y}_j^k - y_j^k)^2$$

激活函数 假设隐层和输出层神经元都是用 Sigmoid 函数（实际上是对数函数）

4.3.4 参数更新方法（以隐层到输出层的连接权 w_{hj} 为例）

策略：梯度下降 以目标的负梯度方向进行参数调整

一句话方法 对均方误差 E_k 和学习率 η 求第二层权重的改变量

均方误差对权重求导的过程

总目标 $\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}}$

推导过程

1. 由链式求导法则可得：

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}$$

2. 根据 β_j 的定义可得：

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h$$

3. Sigmoid 函数的性质：

$$f'(x) = f(x)(1 - f(x))$$

4. 根据定义:

$$f(\beta_j - \theta_j) = \hat{y}_j^k$$

5. 由2,3,4可得输出层神经网络的梯度项:

$$\begin{aligned} \mathbf{g}_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) \quad (a) \\ &= \hat{y}_j^k (1 - \hat{y}_j^k) (y_j^k - \hat{y}_j^k) \quad (b) \end{aligned}$$

(a) 各自求导

(b) Sigmoid函数的性质

6. 综上所述, 可得:

$$\Delta w_{hj} = \eta \mathbf{g}_j b_h$$

$$\Delta \theta_j = -\eta \mathbf{g}_j$$

7. 类似可得隐含层神经网络的梯度项:

$$\begin{aligned} e_h &= -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial a_h} \\ &= -\sum_{j=1}^l \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(a_h - \gamma_h) \quad (a) \\ &= \sum_{j=1}^l w_{hj} \mathbf{g}_j f'(a_h - \gamma_h) \quad (b) \\ &= b_h (1 - b_h) \sum_{j=1}^l w_{hj} \mathbf{g}_j \quad (c) \end{aligned}$$

(a) 利用上一层的梯度项

(b) 将 \mathbf{g}_j 代入

(c) Sigmoid函数的性质

8. 综上所述可得:

$$\Delta v_{ih} = \eta e_h x_i$$

$$\Delta \theta_j = -\eta e_h$$

结果分析

$\Delta w_{hj} = \eta \mathbf{g}_j b_h$ 和下层的权重 \mathbf{g}_j 和本层的输出 b_h 有关

$\Delta \theta_j = -\eta \mathbf{g}_j$ 本层的阈值只和梯度项有关

$\Delta v_{ih} = \eta \mathbf{e}_h x_i$ 类似

$\Delta \theta_j = -\eta \mathbf{e}_h$ 类似

4.3.5 算法流程

输入

- 将输入示例提供给输入层神经元
- 逐层将信号向前传
- 产生输出层的结果 y_j^k

迭代

- 计算输出层的误差 \mathbf{g}_j
- 将误差逆向传播至隐层神经元 \mathbf{e}_h
- 根据隐层神经元的误差来对连接权和阈值进行调整
 - 更新阈值
 - * $\Delta \theta_j = -\eta \mathbf{g}_j$
 - * $\Delta \theta_j = -\eta \mathbf{e}_h$
 - 更新权重
 - * $\Delta w_{hj} = \eta \mathbf{g}_j b_h$
 - * $\Delta v_{ih} = \eta \mathbf{e}_h x_i$

判断当前的累积误差是否已经足够小 $E = \frac{1}{m} \sum_{k=1}^m E_k$ (其中 E_k 为单个样本的均方误差)

Algorithm 1 反向传播算法

Input: $D = \{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1}^m$, 学习率 η

Output: 连接权与阈值确定的多层前馈神经网络

- 1: **while** 没达到停止条件 **do**
 - 2: **for all** $(\mathbf{x}_k, \mathbf{y}_k)$ 属于集合D **do**
 - 3: 计算输出层的误差, 根据当前参数和式计算当前样本的输出 y_k ,
计算输出层神经元的梯度项 g_j
 - 4: 将误差逆向传播至隐层神经元, 计算隐层神经元的梯度项 e_h
 - 5: 更新连接权和阈值
 - 6: **end for;**
 - 7: **end while**
 - 8: **return** E_n ;
-

4.3.6 算法分类

标准BP算法

- 优化目标: 均方误差 E_k
- 更新参数的单位: 每一个样例进行一次
- 可能出现抵消效果

累计BP算法

- 优化目标: 累积误差
- 更新参数的单位: 针对整个训练集
- 到了一定迭代次数后进一步下降很缓慢, 不如标准快

4.3.7 过拟合针对

早停

- 将数据分成训练集和验证集
- 训练集用来计算梯度、更新连接权和阈值
- 验证集用来估计误差

- 若训练误差降低但验证误差升高，则停止训练直接返回

正则化

- 基本思想：在误差目标函数中增加一个用于描述网络复杂度的部分
- 例如：连接权与阈值的平方和
- 取 ω_i 表示连接权和阈值，则误差目标函数改变为：

$$E = \lambda \frac{1}{m} \sum_{k=1}^m E_k + (1 - \lambda) \sum_i \omega_i^2$$

- λ 取值为(0, 1)
- 用于对经验误差和网络复杂度这两项进行折中，通过交叉验证法来估计

4.3.8 参数优化

目标 累计误差最小，是关于 ω 和 θ 的函数，全局最小

相关概念

- 局部最小：邻域中最小
- 全局最小：整个参数空间中最小
- 负梯度方向：函数值下降最快的方向
- 当梯度项为0时，达到局部最小，并停止更新

问题 可能有多个局部最小，避免参数最优陷入局部最小

跳出局部极小

- 以多组不同的参数值初始化，取其中误差最小的参数
- 模拟退火：每一步都以一定概率接受次优解。并且这个概率随着时间的推移而逐渐降低，以保证算法稳定
- 使用随机梯度下降：在计算梯度下降时加入随机因素
- 遗传算法

4.4 超参数

4.4.1 定义

在机器学习的上下文中，超参数是在开始学习过程之前设置值的参数，而不是通过训练得到的参数数据。通常情况下，需要对超参数进行优化，给学习机选择一组最优超参数，以提高学习的性能和效果。

常见存在于

1. 定义关于模型的更高层次的概念，如复杂性或学习能力。
2. 不能直接从标准模型培训过程中的数据中学习，需要预先定义。
3. 可以通过设置不同的值，训练不同的模型和选择更好的测试值来决定。

具体实例 算法中的学习率（learning rate）、梯度下降法迭代的数量（iterations）、隐藏层数目（hidden layers）、隐藏层单元数目、激活函数（activation function）都需要根据实际情况来设置，这些数字实际上控制了最后的参数和的值，所以它们被称作超参数。

4.4.2 设置方法

1. 猜测和检查：根据经验或直觉，选择参数，一直迭代。
2. 网格搜索：让计算机尝试在一定范围内均匀分布的一组值。
3. 随机搜索：让计算机随机挑选一组值。
4. 贝叶斯优化：使用贝叶斯优化超参数，会遇到贝叶斯优化算法本身就需要很多的参数的困难。
5. MITIE方法，好初始猜测的前提下进行局部优化。它使用BOBYQA算法，并有一个精心选择的起始点。由于BOBYQA只寻找最近的局部最优解，所以这个方法是否成功很大程度上取决于是否有一个好的起点。在MITIE的情况下，我们知道一个好的起点，但这不是一个普遍的解决方案，因为通常你不会知道好的起点在哪里。从好的方面来说，这种方法非常适合寻找局部最优解。

6. LIPO的全局优化方法。这个方法没有参数，而且经验证比随机搜索方法好。

4.4.3 超参数搜索一般过程

1. 将数据集划分成训练集、验证集及测试集。
2. 在训练集上根据模型的性能指标对模型参数进行优化。
3. 在验证集上根据模型的性能指标对模型的超参数进行搜索。
4. 步骤2和步骤3交替迭代，最终确定模型的参数和超参数，在测试集中验证评价模型的优劣。

4.5 激活函数

4.5.1 问题

1. 线性神经网络无法解决非线性问题。
2. 需要一个函数可以对得到的Y值进行分类。
3. 如果输入的绝对值特别大，或者无限大，那么输出就会特别大，直接导致error太大。更新出来的权值没有意义，或者无法更新权值。
4. 反向传播的时候线性的导数是常数，导致无法继续传播。

4.5.2 常用激活函数

如图3所示。除此以外还有SoftMax，SoftPlus等等。

4.5.3 作用

- 提高模型鲁棒性
- 提高模型非线性表达能力
- 缓解梯度消失问题
- 将特征图映射到新的特征空间从而更有利于训练
- 加速模型收敛

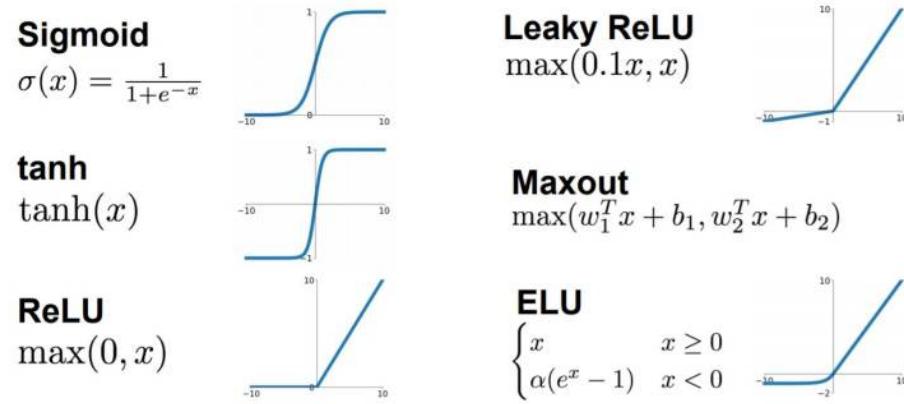


图 3: 常见激活函数

4.5.4 常见激活函数的导数计算

- Sigmoid 激活函数

- 函数表达式: $f(x) = \frac{1}{1+e^{-x}}$
- 导数: $f'(x) = \frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}}\right) = f(x)(1 - f(x))$
- 备注: 当 $x = 10$, 或 $x = -10$, $f'(x) \approx 0$, 当 $x = 0$, $f'(x) = 0.25$

- Tanh 激活函数

- 函数表达式: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- 导数: $f'(x) = -(\tanh(x))^2$
- 备注: 当 $x = 10$, 或 $x = -10$, $f'(x) \approx 0$, 当 $x = 0$, $f'(x) = 1$

- Relu 激活函数

- 函数表达式: $f(x) = \max(0, x)$

- 导数: $c(u) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \\ \text{undefined}, & x = 0 \end{cases}$

- 备注: 通常 $x = 0$ 时, 给定其导数为 1 和 0

4.5.5 激活函数性质

- 非线性：当激活函数是线性的，一个两层的神经网络就可以基本上逼近所有的函数。但如果激活函数是恒等激活函数的时候，即 $f(x) = x$ ，就不满足这个性质，而且如果 MLP 使用的是恒等激活函数，那么其实整个网络跟单层神经网络是等价的
- 可微性：当优化方法是基于梯度的时候，就体现了该性质
- 单调性：当激活函数是单调的时候，单层网络能够保证是凸函数
- $f(x) \approx x$ ：当激活函数满足这个性质的时候，如果参数的初始化是随机的较小值，那么神经网络的训练将会很高效；如果不满足这个性质，那么就需要详细地去设置初始值
- 输出值的范围：当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著；当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况下，一般需要更小的 Learning Rate。

4.5.6 选择激活函数的策略

选择一个适合的激活函数不容易，需要考虑很多因素，通常的做法是，如果不确定哪一个激活函数效果更好，可以把它们都试试，然后在验证集或者测试集上进行评价。然后看哪一种表现的更好，就去使用它。

- 如果输出是0、1值（二分类问题），则输出层选择sigmoid函数，然后其它的所有单元都选择Relu函数。
- 如果在隐藏层上不确定使用哪个激活函数，那么通常会使用Relu激活函数。有时，也会使用 tanh 激活函数，但Relu的一个优点是：当是负值的时候，导数等于0。
- sigmoid激活函数：除了输出层是一个二分类问题基本不会用它。
- tanh激活函数：tanh是非常优秀的，几乎适合所有场合。
- ReLu激活函数：最常用的默认函数，如果不确定用哪个激活函数，就使用ReLU或者Leaky ReLU，再去尝试其他的激活函数。
- 如果遇到了一些死的神经元，我们可以使用Leaky ReLU函数。

4.5.7 ReLu激活函数的优点

- 在区间变动很大的情况下，ReLu激活函数的导数或者激活函数的斜率都会远大于0，在实践中，使用ReLu激活函数神经网络通常会比使用sigmoid或者tanh激活函数学习的更快。
- sigmoid和tanh函数的导数在正负饱和区的梯度都会接近于0，这会造成梯度弥散，而Relu和Leaky ReLu函数大于0部分都为常数，不会产生梯度弥散现象。
- 需注意，Relu进入负半区的时候，梯度为0，神经元此时不会训练，产生所谓的稀疏性，而Leaky ReLu不会产生这个问题。

4.5.8 Softmax函数

定义

$$P(i) = \frac{\exp(\theta_i^T x)}{\sum_{k=1}^K \exp(\theta_k^T x)}$$

其中， θ_i 和 x 是列向量， $\theta_i^T x$ 可能被换成函数关于 x 的函数 $f_i(x)$ 。通过Softmax函数，可以使得 $P(i)$ 的范围在[0, 1]之间。在回归和分类问题中，通常 θ 是待求参数，通过寻找使得 $P(i)$ 最大的 θ_i 作为最佳参数。

e 的幂函数的意义 参考logistic函数：

$$P(i) = \frac{1}{1 + \exp(-\theta_i^T x)}$$

这个函数的作用就是使得 $P(i)$ 在负无穷到0的区间趋向于0，在0到正无穷的区间趋向1。同样Softmax函数加入了 e 的幂函数正是为了两极化：正样本的结果将趋近于1，而负样本的结果趋近于0。这样为多类别提供了方便（可以把 $P(i)$ 看做是样本属于类别的概率）。可以说，Softmax函数是logistic函数的一种泛化。

4.6 Batch Size

4.6.1 为什么需要Batch Size

Batch的选择，首先决定的是下降的方向。如果数据集比较小，可采用全数据集的形式，好处是：

- 由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。
- 由于不同权重的梯度值差别巨大，因此选取一个全局的学习率很困难。Full Batch Learning可以使用Rprop只基于梯度符号并且针对性单独更新各权值。

对于更大的数据集，假如采用全数据集的形式，坏处是：

- 随着数据集的海量增长和内存限制，一次性载入所有的数据进来变得越来越不可行。
- 以Rprop的方式迭代，会由于各个Batch之间的采样差异性，各次梯度修正值相互抵消，无法修正。这才有了后来RMSPProp的妥协方案。

4.6.2 Batch Size值的选择

可采用批梯度下降法（Mini-batches Learning）。因为如果数据集足够充分，那么用一半（甚至少得多）的数据训练算出来的梯度与用全部数据训练出来的梯度是几乎一样的。

4.6.3 增大Batch Size的好处

1. 内存利用率提高了，大矩阵乘法的并行化效率提高。
2. 跑完一次epoch（全数据集）所需的迭代次数减少，对于相同数据量的处理速度进一步加快。
3. 在一定范围内，一般来说batch size越大，其确定的下降方向越准，引起训练震荡越小。

4.6.4 盲目增大Batch Size的坏处

- 内存利用率提高了，但是内存容量可能撑不住了。
- 跑完一次epoch（全数据集）所需的迭代次数减少，要想达到相同的精度，其所花费的时间大大增加了，从而对参数的修正也就显得更加缓慢。
- batch size增大到一定程度，其确定的下降方向已经基本不再变化。

4.7 归一化

4.7.1 归一化含义

- 归纳统一样本的统计分布性。归一化在 $0 - 1$ 之间是统计的概率分布，归一化在 $-1 - +1$ 之间是统计的坐标分布。
- 无论是为了建模还是为了计算，首先基本度量单位要同一，神经网络是以样本在事件中的统计分别几率来进行训练（概率计算）和预测，且sigmoid函数的取值是0到1之间的，网络最后一个节点的输出也是如此，所以经常要对样本的输出归一化处理。
- 归一化是统一在 $0 - 1$ 之间的统计概率分布，当所有样本的输入信号都为正值时，与第一隐含层神经元相连的权值只能同时增加或减小，从而导致学习速度很慢。
- 另外在数据中常存在奇异样本数据，奇异样本数据存在所引起的网络训练时间增加，并可能引起网络无法收敛。为了避免出现这种情况及后面数据处理的方便，加快网络学习速度，可以对输入信号进行归一化，使得所有样本的输入信号其均值接近于0或与其均方差相比很小。

4.7.2 归一化好处

- 为了后面数据处理的方便，归一化的确可以避免一些不必要的数值问题。
- 为了程序运行时收敛加快。
- 同一量纲。样本数据的评价标准不一样，需要对其量纲化，统一评价标准。这算是应用层面的需求。
- 避免神经元饱和。啥意思？就是当神经元的激活在接近0或者1时会饱和，在这些区域，梯度几乎为0，这样，在反向传播过程中，局部梯度就会接近0，这会有效地“杀死”梯度。
- 保证输出数据中数值小的不被吞食。

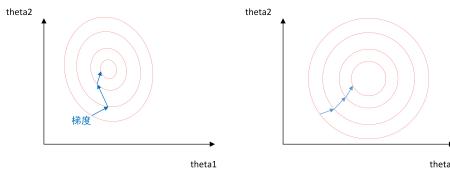


图 4: 归一化图示

4.7.3 归一化能提高求解最优解速度的理由

上图是代表数据是否均一化的最优解寻解过程（圆圈可以理解为等高线）。左图表示未经归一化操作的寻解过程，右图表示经过归一化后的寻解过程。当使用梯度下降法寻求最优解时，很有可能走“之字型”路线（垂直等高线走），从而导致需要迭代很多次才能收敛；而右图对两个原始特征进行了归一化，其对应的等高线显得很圆，在梯度下降进行求解时能较快的收敛。因此如果机器学习模型使用梯度下降法求最优解时，归一化往往非常有必要，否则很难收敛甚至不能收敛。

4.7.4 归一化类型

- 线性归一化

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- 适用范围：比较适用在数值比较集中的情况。
- 缺点：如果 \max 和 \min 不稳定，很容易使得归一化结果不稳定，使得后续使用效果也不稳定。

- 标准差标准化

$$x' = \frac{x - \mu}{\sigma}$$

- 含义：经过处理的数据符合标准正态分布，即均值为0，标准差为1 其中 μ 为所有样本数据的均值， σ 为所有样本数据的标准差。

- 非线性归一化

- 适用范围：经常用在数据分化比较大的场景，有些数值很大，有些很小。通过一些数学函数，将原始值进行映射。该方法包括 \log 、指数，正切等。

5 深度学习

5.1 为什么需要深度学习

5.1.1 必然性

梯度消失问题：在神经网络中，当前面隐藏层的学习速率低于后面隐藏层的学习速率，即随着隐藏层数目的增加，分类准确率反而下降了

5.1.2 可行性

云计算、大数据时代的到来，计算能力的大幅提高可缓解训练低效性，训练数据的大幅增加可降低过拟合风险

5.2 什么是深度学习

深度学习特指神经元数目特别多的神经网络模型。

5.2.1 增加神经元数目的神经网络

缺陷很严重，表达能力有限。例如再多的神经元也不能在两层神经网络中表达异或操作。

5.2.2 增加隐含层数量的神经网络

优势 不仅增加了隐层神经元的数量，还增加了激活函数嵌套的层数。使得模型的表达能力大大增加。理论上可以逼近任何现有的函数。

难点 误差在多隐层内逆传播时，往往会发散而不能收敛到稳定状态

解决办法

无监督逐层训练

- 基本思想：每次训练一层隐结点
- 预训练：训练时将上一层隐结点对输出作为输入，本层隐结点对输出作为下一层对输入

- 微调：预训练全部完成后，再对网络进行训练。一般使用BP算法
- 可视为将大量参数分组，对每组先找到局部较优解，再联合起来寻找全局最优

权值共享：让一组神经元使用相同的连接权 卷积神经网络

5.3 深度神经网络难以训练的原因

5.3.1 梯度消失

梯度消失是指通过隐藏层从后向前看，梯度会变的越来越小，说明前面层的学习会显著慢于后面层的学习，所以学习会卡住，除非梯度变大。

梯度消失的原因受到多种因素影响，例如学习率的大小，网络参数的初始化，的边缘效应等。在深层神经网络中，每一个神经元计算得到的梯度都会传递给前一层，较浅层的神经元接收到的梯度受到之前所有层梯度的影响。如果计算得到的梯度值非常小，随着层数增多，求出的梯度更新信息将会以指数形式衰减，就会发生梯度消失。

5.3.2 梯度爆炸

在深度网络或循环神经网络（Recurrent Neural Network, RNN）等网络结构中，梯度可在网络更新的过程中不断累积，变成非常大的梯度，导致网络权重值的大幅更新，使得网络不稳定；在极端情况下，权重值甚至会溢出，变为 Nan 值，再也无法更新。

5.3.3 权重矩阵的退化导致模型的有效自由度减少

参数空间中学习的退化速度减慢，导致减少了模型的有效维数，网络的可用自由度对学习中梯度范数的贡献不均衡，随着相乘矩阵的数量（即网络深度）的增加，矩阵的乘积变得越来越退化。在有硬饱和边界的非线性网络中（例如 ReLU 网络），随着深度增加，退化过程会变得越来越快。Duvenaud等人2014年的论文里展示了关于该退化过程的可视化如图5。

随着深度的增加，输入空间（左上角所示）会在输入空间中的每个点处被扭曲成越来越细的单丝，只有一个与细丝正交的方向影响网络的响应。沿着这个方向，网络实际上对变化变得非常敏感。

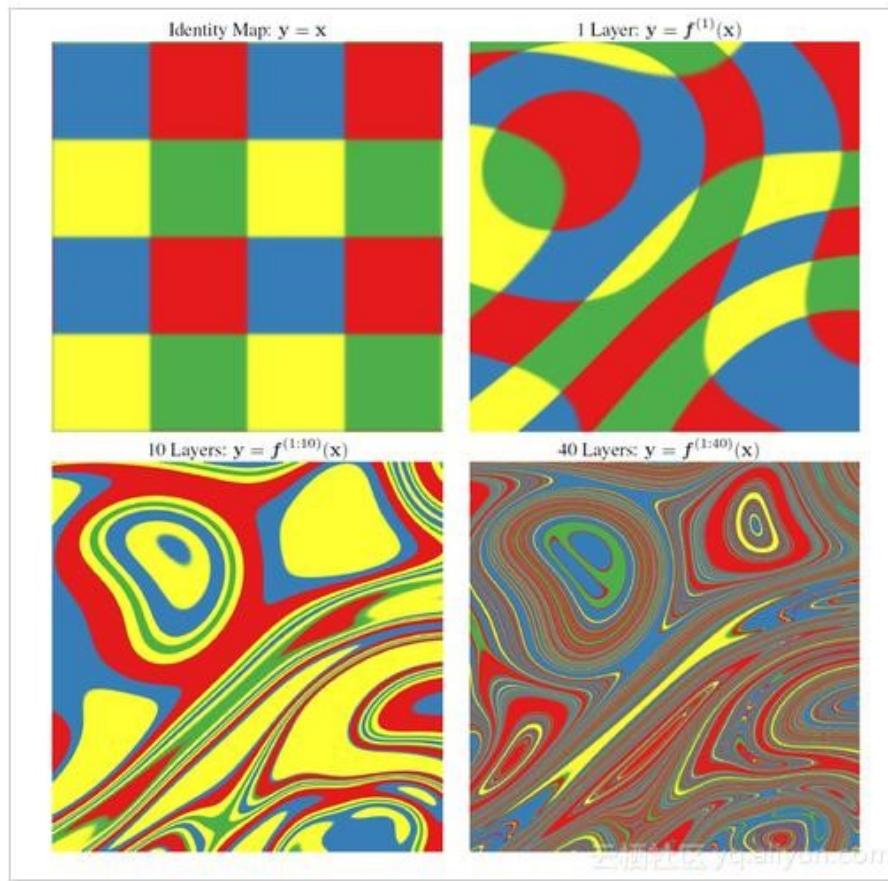


图 5: 权重矩阵的退化过程

5.4 预训练与微调(fine tuning)

5.4.1 深度网络存在问题

- 网络越深，需要的训练样本数越多。若用监督则需大量标注样本，不然小规模样本容易造成过拟合。深层网络特征比较多，会出现的多特征问题主要有多样本问题、规则化问题、特征选择问题。
- 多层神经网络参数优化是个高阶非凸优化问题，经常得到收敛较差的局部解
- 梯度扩散问题，BP算法计算出的梯度随着深度向前而显著下降，导致前面网络参数贡献很小，更新速度慢。

5.4.2 解决方法

逐层贪婪训练，无监督预训练（unsupervised pre-training）即训练网络的第一个隐藏层，再训练第二个…最后用这些训练好的网络参数值作为整体网络参数的初始值。经过预训练最终能得到比较好的局部最优解。

5.4.3 模型微调fine tuning的定义

用别人的参数、修改后的网络和自己的数据进行训练，使得参数适应自己的数据，这样一个过程，通常称之为微调(fine tuning)

5.4.4 微调时网络参数的更新

- fine tune的过程相当于继续训练，跟直接训练的区别是初始化的时候。
- 直接训练是按照网络定义指定的方式初始化。
- fine tune是用你已经有的参数文件来初始化。

5.4.5 fine tuning模型的三种状态

1. 只预测，不训练：相对快、简单，针对那些已经训练好，现在要实际对未知数据进行标注的项目，非常高效。
2. 训练，但只训练最后分类层：fine-tuning的模型最终的分类以及符合要求，现在只是在他们的基础上进行类别降维。
3. 完全训练，分类层+之前卷积层都训练：跟状态二的差异很小，当然状态三比较耗时和需要训练GPU资源，不过非常适合fine-tuning到自己想要的模型里面，预测精度相比状态二也提高不少。

5.5 正则化Dropout

5.5.1 为什么要正则化

深度学习可能存在过拟合问题——高方差，有两个解决方法，一个是正则化，另一个是准备更多的数据，这是非常可靠的方法，但你可能无法时时刻刻准备足够多的训练数据或者获取更多数据的成本很高，但正则化通常有助于避免过拟合或减少你的网络误差。

5.5.2 理解

在每个训练批次中，通过忽略一半的特征检测器（让一半的隐层节点值为0），可以明显地减少过拟合现象。这种方式可以减少特征检测器（隐层节点）间的相互作用，检测器相互作用是指某些检测器依赖其他检测器才能发挥作用。Dropout说的简单一点就是：我们在前向传播的时候，让某个神经元的激活值以一定的概率 p 停止工作，这样可以使模型泛化性更强，因为它不会太依赖某些局部的特征。

5.5.3 工作流程

假设我们要训练这样一个神经网络，如图6所示。

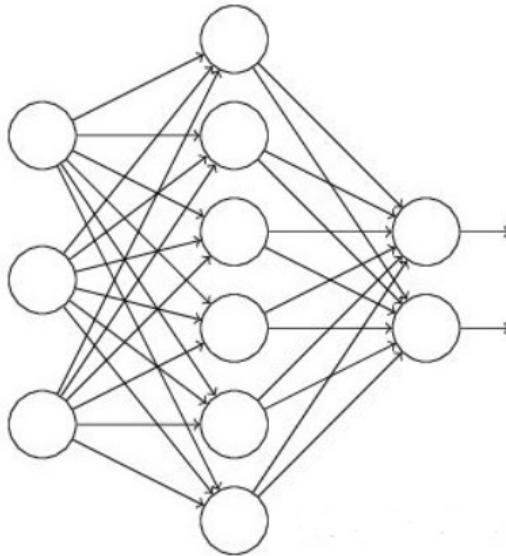


图 6: 正常的神经网络

1. 首先随机（临时）删掉网络中一半的隐藏神经元，输入输出神经元保持不变（图7中虚线为部分临时被删除的神经元）
2. 然后把输入 x 通过修改后的网络前向传播，然后把得到的损失结果通过修改的网络反向传播。一小批训练样本执行完这个过程后，在没有被删除的神经元上按照随机梯度下降法更新对应的参数(w, b)

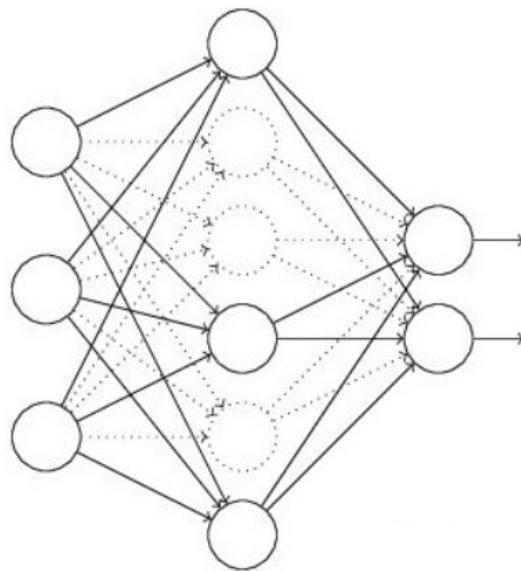


图 7: 删除部分神经元的神经网络

3. 恢复被删掉的神经元（此时被删除的神经元保持原样，而没有被删除的神经元已经有所更新）
4. 不断重复以上的过程。

5.5.4 工作机制

1. 取平均的作用：先回到标准的模型即没有dropout，我们用相同的训练数据去训练5个不同的神经网络，一般会得到5个不同的结果，此时我们可以采用“5个结果取均值”或者“多数取胜的投票策略”去决定最终结果。例如3个网络判断结果为数字9,那么很有可能真正的结果就是数字9，其它两个网络给出了错误结果。这种“综合起来取平均”的策略通常可以有效防止过拟合问题。因为不同的网络可能产生不同的过拟合，取平均则有可能让一些“相反的”拟合互相抵消。dropout掉不同的隐藏神经元就类似在训练不同的网络，随机删掉一半隐藏神经元导致网络结构已经不同，整个dropout过程就相当于对很多个不同的神经网络取平均。而不同的网络产生不同的过拟合，一些互为“反向”的拟合相互抵消就可以达到整体上减少过拟合。

2. 减少神经元之间复杂的共适应关系：因为dropout程序导致两个神经元不一定每次都在一个dropout网络中出现。这样权值的更新不再依赖于有固定关系的隐含节点的共同作用，阻止了某些特征仅仅在其它特定特征下才有效果的情况。迫使网络去学习更加鲁棒的特征，这些特征在其它的神经元的随机子集中也存在。换句话说假如我们的神经网络是在做出某种预测，它不应该对一些特定的线索片段太过敏感，即使丢失特定的线索，它也应该可以从众多其它线索中学习一些共同的特征。从这个角度看dropout就有点像L1, L2正则，减少权重使得网络对丢失特定神经元连接的鲁棒性提高。
3. Dropout类似于性别在生物进化中的角色：物种为了生存往往会倾向于适应这种环境，环境突变则会导致物种难以做出及时反应，性别的出现可以繁衍出适应新环境的变种，有效的阻止过拟合，即避免环境改变时物种可能面临的灭绝。

6 卷积神经网络

6.1 特征选择

6.1.1 优化的常见问题

在增加特征类别或者修改特征参数后：

- 准确率提升不高
- 准确率不升反降

6.1.2 理由

- 提取的特征中有冗余特征，对模型的性能几乎没有帮助。
- 提取的特征中有些可以列为噪声（或者可以称为老鼠屎），对模型的性能不仅没有帮助，还会降低模型的性能。

6.1.3 本质

是对一个给定特征子集的优良性通过一个特定的评价标准(evaluation criterion)进行衡量。通过特征选择，原始特征集合中的冗余（redundant）特征和不相关（irrelevant）特征被除去。而有用特征得以保留。

6.1.4 作用

- 降低了模型的复杂度，节省了大量计算资源以及计算时间。
- 提高了模型的泛化能力。

6.1.5 具体方法

考虑特征选择的策略的指标主要有两个：

- 特征是否发散：如果一个特征不发散，就是说这个特征大家都有或者非常相似，说明这个特征不需要。
- 特征和目标是否相关：与目标的相关性越高，越应该优先选择。

总的来说，特征选择有三种常用的思路：

1. 特征过滤 (Filter Methods): 对各个特征按照发散性或者相关性进行评分，对分数设定阈值或者选择靠前得分的特征。
 - 优点: 简单, 快。
 - 缺点: 对于排序靠前的特征, 如果他们相关性较强, 则引入了冗余特征, 浪费了计算资源。对于排序靠后的特征, 虽然独立作用不显著, 但和其他特征想组合可能会对模型有很好的帮助, 这样就损失了有价值的特征。
 - 经典方法: 皮尔逊相关系数 (Pearson's Correlation), 线性判别分析 (Linear Discriminant Analysis, LDA), 方差分析 (Analysis of Variance, ANOVA), 卡方检验 (Chi-Square)。
2. 特征筛选 (Wrapper Methods): 通过不断排除特征或者不断选择特征, 并对训练得到的模型效果进行打分, 通过预测效果评分来决定特征的去留。
 - 优点: 能较好的保留有价值的特征。
 - 缺点: 会消耗巨大的计算资源和计算时间。
 - 经典方法: 前向选择法, 后向剔除法, 迭代剔除法
3. 嵌入法 (Embedded Methods): 有不少特征筛选和特征过滤的共性, 主要的特点就是通过不同的方法去计算不同特征对于模型的贡献。
 - 经典方法: Lasso, Elastic Net, Ridge Regression等。

6.2 卷积

6.2.1 卷积公式

$$\int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau$$

6.2.2 公式理解

- 系统某一时刻的输出是由多个输入共同作用 (叠加) 的结果。
- 所谓两个函数的卷积, 本质上就是先将一个函数翻转, 然后进行滑动叠加。

6.2.3 例子：信号分析

如图8，输入信号是 $f(t)$ ，是随时间变化的。系统响应函数是 $g(t)$ ，图中的响应函数是随时间指数下降的，它的物理意义是说：如果在 $t = 0$ 的时刻有一个输入，那么随着时间的流逝，这个输入将不断衰减。换言之，到了 $t = T$ 时刻，原来在 $t = 0$ 时刻的输入 $f(0)g(T)$ 的值将衰减为 $f(0)g(T)$ 。

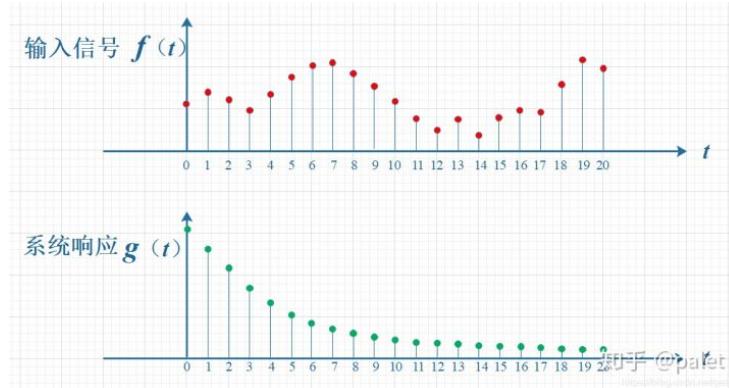


图 8: 输入信号与系统响应

考虑到信号是连续输入的，也就是说，每个时刻都有新的信号进来，所以，最终输出的是所有之前输入信号的累积效果。

如图9，在 $t = T$ 时刻的输出信号为所有输入信号和其系统响应的乘积的和，但是可以看到，此时输入信号和系统响应的对应关系是相反的。

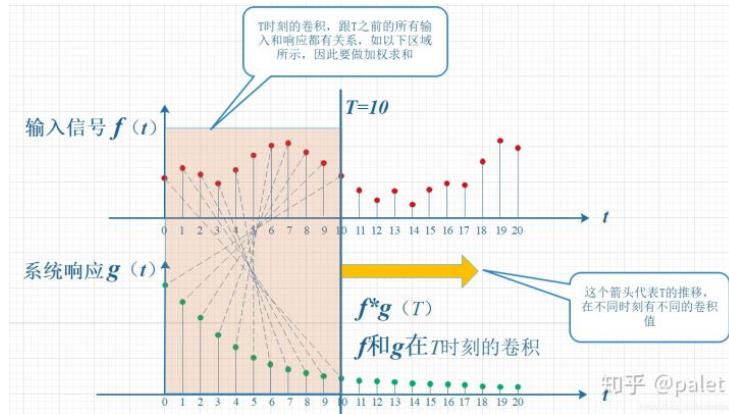


图 9: 输入信号与系统响应的对应关系

所以我们将系统响应函数 $g(t)$ 进行翻转，这样输入信号和系统响应的对应关系就是顺着的了。如图10所示。

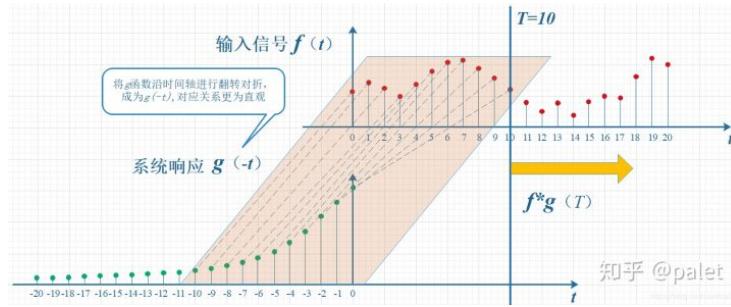


图 10: 输入信号与翻转后的系统响应

最后我们再对 $g(t)$ 加上 T ，将其和输入信号对齐。如图11所示。

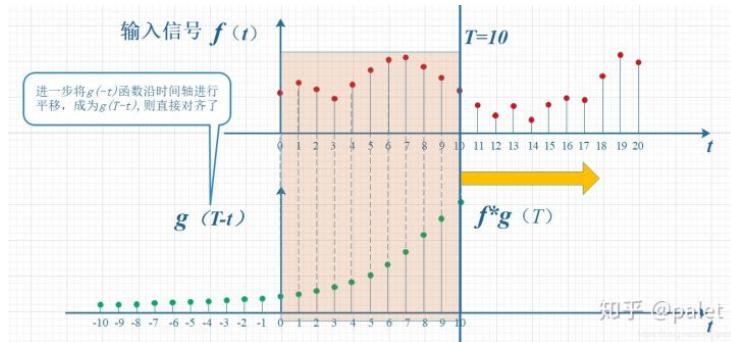


图 11: 输入信号与最终的系统响应

6.3 卷积神经网络的结构

卷积层	采样层	全连接层
提取图片特征	保留我们想要的或者重要的特征	模仿人脑进行判断

6.3.1 全连接层

假如我们想要让机器识别一个图片数字2，那么我们就需要一个类似人

脑的构造来进行识别，这就是全连接层。假设全连接层为三层的神经元个数分别为1024, 512, 10。

但是假设图片大小为 $32 * 32$ ，那么我们全连接层的参数个数为 $3 * 32 * 32 * 1024 + 1024 * 512 + 512 * 10 + 1024(bias) + 512(bias) = 3676672$ 个。参数太多会导致训练时间过长，还有可能会导致过拟合。

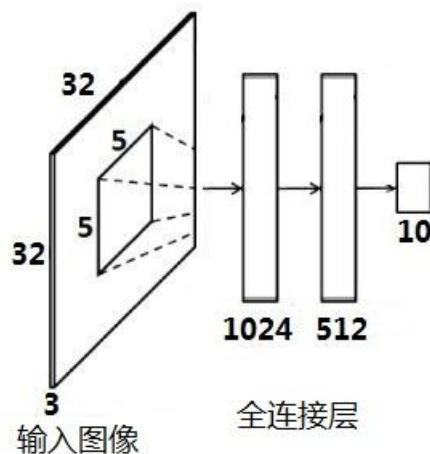


图 12: 只有全连接层的CNN

6.3.2 采样层

为了解决全连接层参数过多的问题，我们希望主要提取图像的边缘特征，对于灰色和黑色这种冗余或者不重要的区域特征，我们尽量丢弃或者少保留，那么这样可能会减少参数或者减少提参数的过程。

此时采样层应运而生。

6.3.3 卷积层

首先将图片进行分解，在顺序不变的情况下我们还是可以识别出内容是2。但是我们如果将分解后的子图的顺序进行交换，那就无法识别出数字2。由此我们可以得出：

- 如果我们只知道局部的图片，以及局部的相对位置，只要我们能将它正确组合起来，我们也可以对物体进行识别

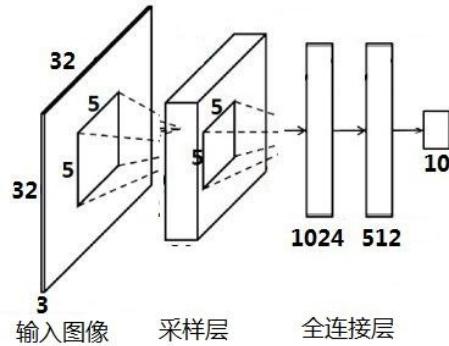


图 13: 加入采样层的CNN

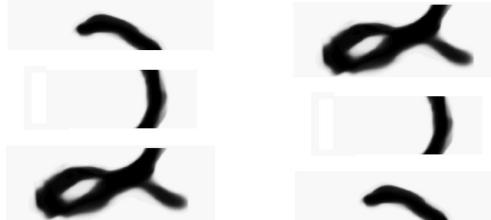
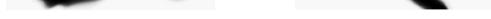
图 14: 顺序不变的子图
图

图 15: 顺序改变的子图

- 局部与局部之间关联性不大，也就是局部的变化，很少影响到另外一个局部

同时还有两个问题需要解决：

- 输入的只是原始图片，我们还没有提取图片的特征
- 我们目前要处理的参数仍然非常多，我们需要对原始输入进行降维或者减少参数

此时我们加入卷积层，一方面提取特征，另一方面减少参数个数。

6.3.4 级联分类器

除此之外，我们还会遇到两个问题：

- 卷积层提取的特征数量有限，无法提取图片全部特征

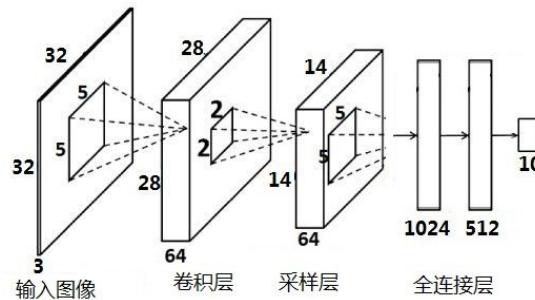


图 16: 卷积层的CNN

- 无法确定采样层选取的特征是否重要

此时我们需要一个级联分类器。

大概意思就是我从一堆弱分类器里面，挑出一个最符合要求的弱分类器，用着这个弱分类器把不想要的数据剔除，保留想要的数据然后再从剩下的弱分类器里，再挑出一个最符合要求的弱分类器，对上一级保留的数据，把不想要的数据剔除，保留想要的数据。最后，通过不断串联几个弱分类器，进过数据层层筛选，最后得到我们想要的数据。

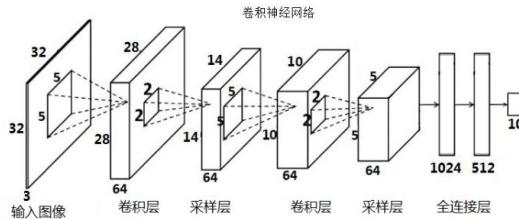


图 17: 级联分类器的CNN

6.4 卷积层：提取特征

6.4.1 图示

- 卷积核放在神经网络里，就代表对应的权重(weight)
- 卷积核和图像进行点乘(dot product)，就代表卷积核里的权重单独对相应位置的Pixel进行作用

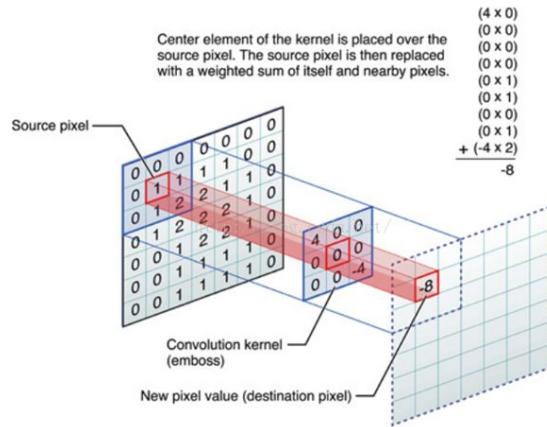


图 18: 卷积核的卷积过程

6.4.2 输出

根据神经网络公式:

$$\sum_i \omega_i x_i + b$$

在此, 我们先假设 $b = 0$, 而最终的输出 $FinalOutput = \sum_i output_{\omega_i} + b$

6.4.3 卷积顺序

除此之外, 我们还需要决定卷积的顺序: 从左到右, 每隔 x 列 Pixel, 向右移动一次卷积核进行卷积(x 可以自己定义), 其中 x 叫作 stride, 就是步长的意思, 如果我们 $x = 2$, 就是相当每隔两行或者两列进行卷积

0	2	2	0	2
1	1	1	1	1
0	0	1	2	2
0	0	1	2	2
0	2	2	2	0

图 19: 卷积顺序 ($x=1$)

0	0	0	0	0	0	0
0	0	2	2	0	2	0
0	1	1	1	1	1	0
0	0	0	1	2	2	0
0	0	0	1	2	2	0
0	0	2	2	2	0	0
0	0	0	0	0	0	0

图 20: 补0

6.4.4 补0

作用 同样是stride $x=1$ 的情况下，补0比原来没有添0的情况下进行卷积，从左到右，从上到下都多赚了2次卷积，这样第一层卷积层输出的特征图（feature map）仍然为 5×5 ，和输入图片的大小一致。

好处

- 获得的更多更细致的特征信息，可以获得更多的图像边缘信息
- 可以控制卷积层输出的特征图的size，从而可以达到控制网络结构的作用，还是以上的例子，如果没有做zero-padding以及第二层卷积层的卷积核仍然是 3×3 ，那么第二层卷积层输出的特征图就是 1×1 ，CNN的特征提取就这么结束了。

6.5 采样层：特征选择

6.5.1 示例：Maxpooling

首先假设卷积层获得的特征值矩阵如图21所示。
而Maxpooling实际上为从矩阵中选择最大的值：此处即为9。如图22所示。

9	1
3	2

图 21: 特征值矩阵

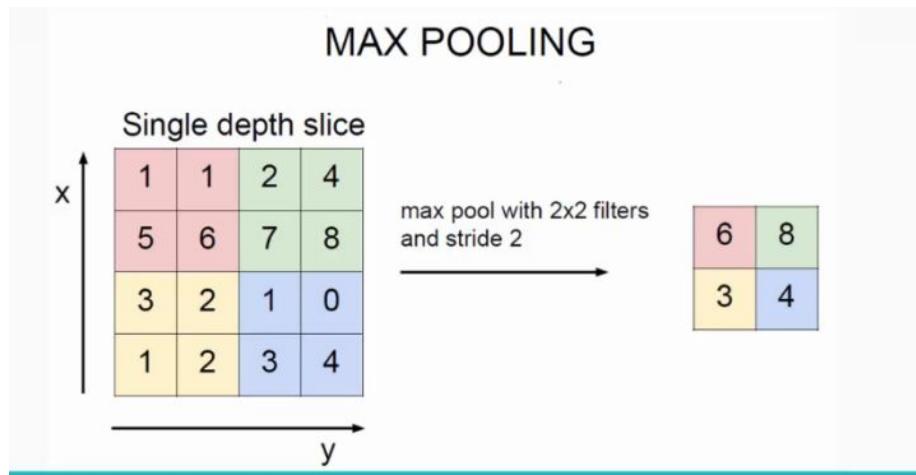


图 22: Maxpooling

6.5.2 好处

- 避免过拟合
- 避免参数过多导致的训练代价太大
- 无法满足模型的结构需求

6.5.3 性质

采样层还有一定程度的空间不变性。图像经过一个小小的平移之后，依然产生相同的池化特征。

6.6 全连接层：进行分类

6.6.1 图示

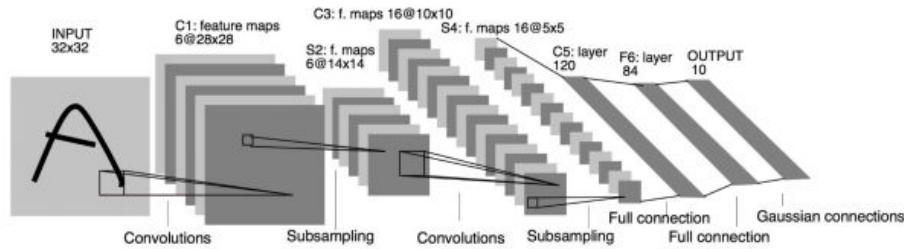


图 23: 带有全连接层的CNN结构

6.6.2 转化方法

因为采样层所提取的特征一般都是 $n * n * c$ 的，其中 c 是filter的个数。而全连接层是一个普通的神经网络，每一层是由许多神经元组成的(1*4096)的平铺结构。那么如何将 $n * n * c$ 转化成1*4096就是需要解决的问题。

此时采用的方法是进行一次“卷积”。用一个 $3 \times 3 \times 5$ 的filter去卷积激活函数的输出，得到的结果就是一个fully connected layer的一个神经元的输出，这个输出就是一个值。因为我们有4096个神经元，我们实际就是用一个 $3 \times 3 \times 5 \times 4096$ 的卷积层去卷积激活函数的输出。如图24所示。

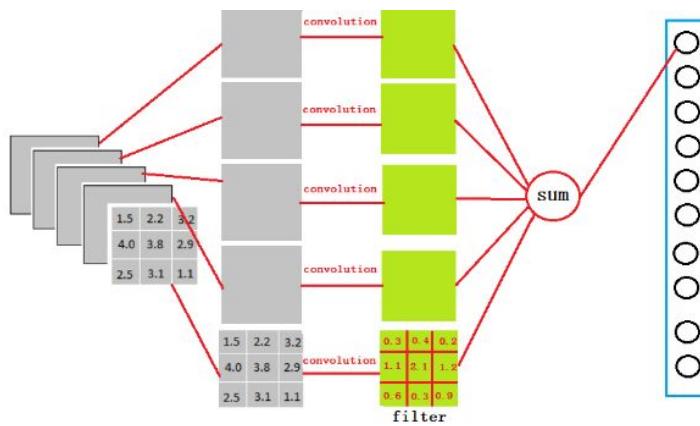


图 24: “卷积”操作

6.6.3 作用

把特征representation整合到一起，输出为一个值。可以大大减少特征位置对分类带来的影响。

6.6.4 对模型的影响因素

- 全接解层的总层数（长度）
- 单个全连接层的神经元数（宽度）
- 激活函数

6.7 局部响应归一化作用

LRN是一种提高深度学习准确度的技术方法。LRN一般是在激活、池化函数后的一种方法。

在ALexNet中，提出了LRN层，对局部神经元的活动创建竞争机制，使其中响应比较大对值变得相对更大，并抑制其他反馈较小的神经元，增强了模型的泛化能力。

6.7.1 理解局部响应归一化

局部响应归一化原理是仿造生物学上活跃的神经元对相邻神经元的抑制现象（侧抑制），其公式如下：

$$b_{x,y}^i = a_{x,y}^i / \left(k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a_{x,y}^j)^2 \right)^\beta$$

- **a:** 表示卷积层（包括卷积操作和池化操作）后的输出结果，是一个四维数组[batch,height,width,channel]。
 1. batch: 批次数(每一批为一张图片)。
 2. height: 图片高度。
 3. width: 图片宽度。
 4. channel: 通道数。可以理解成一批图片中的某一个图片经过卷积操作后输出的神经元个数，或理解为处理后的图片深度。

- $a_{x,y}^i$ 表示在这个输出结构中的一个位置 $[a, b, c, d]$, 可以理解成在某一张图中的某一个通道下的某个高度和某个宽度位置的点, 即第 a 张图的第 d 个通道下的高度为 b 宽度为 c 的点。
- N : 论文公式中的 N 表示通道数 (channel)。
- $a, n/2, k$ 分别表示函数中的 input, depth radius, bias。参数 k, n, α, β 都是超参数, 一般设置 $k = 2, n = 5, \alpha = 1 * e - 4, \beta = 0.75$
- \sum : \sum 叠加的方向是沿着通道方向的, 即每个点值的平方和是沿着 a 中的第 3 维 channel 方向的, 也就是一个点同方向的前面 $n/2$ 个通道 (最小为第 0 个通道) 和后 $n/2$ 个通道 (最大为第 $d - 1$ 个通道) 的点的平方和 (共 $n + 1$ 个点)。而函数的英文注解中也说明了把 input 当成是 d 个 3 维的矩阵, 说白了就是把 input 的通道数当作 3 维矩阵的个数, 叠加的方向也是在通道方向。

如图 25 所示。

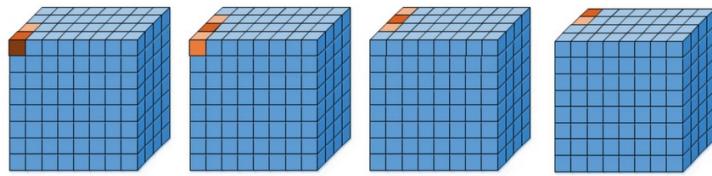


图 25: LRN

6.7.2 实例

假设存在一 LRN 如图 26 所示。

不同的颜色表示不同的通道, 因此 $N=4$ 。令超参数 $(k, \alpha, \beta, n) = (0, 1, 1, 2)$ 。
 $n=2$ 的值表示在计算位置 (i, x, y) 处的归一化值时, 我们考虑上一个过滤器和下一个过滤器相同位置的值: $(i-1, x, y)$ 和 $(i+1, x, y)$ 。例如对于 $(i, x, y) = (0, 0, 0)$, 我们有值 $(i, x, y) = 1$, 而值 $(i-1, x, y)$ 不存在, 值 $(i+1, x, y) = 1$ 。因此 $Normalized value(i, x, y) = 1/(1^2 + 1^2) = 0.5$ 。可以看到上图的下部。其余的归一化值以类似的方式计算。

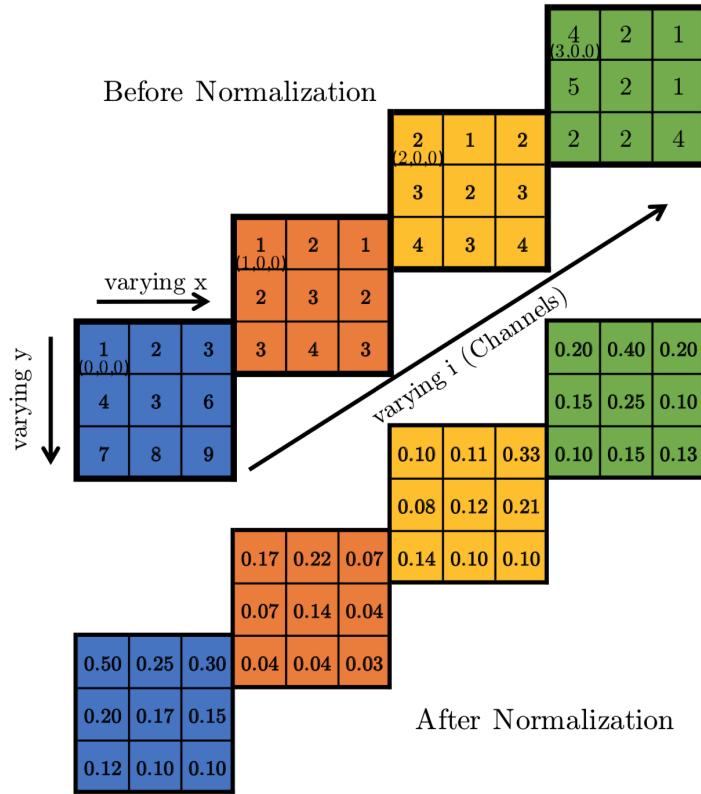


图 26: LRN

6.8 批归一化BN

以前在神经网络训练中，只是对输入层数据进行归一化处理，却没有在中间层进行归一化处理。要知道，虽然我们对输入数据进行了归一化处理，但是输入数据经过 $\sigma(WX + b)$ 这样的矩阵乘法以及非线性运算之后，其数据分布很可能被改变，而随着深度网络的多层运算之后，数据分布的变化将越来越大。如果我们能在网络的中间也进行归一化处理，是否对网络的训练起到改进作用呢？答案是肯定的。这种在神经网络中间层也进行归一化处理，使训练效果更好的方法，就是批归一化Batch Normalization (BN)。

6.8.1 优点

- 减少了人为选择参数。在某些情况下可以取消dropout和L2正则项参数,或者采取更小的L2正则项约束参数;
- 减少了对学习率的要求。现在我们可以使用初始很大的学习率或者选择了较小的学习率, 算法也能够快速训练收敛;
- 可以不再使用局部响应归一化。BN本身就是归一化网络(局部响应归一化在AlexNet网络中存在)
- 破坏原来的数据分布, 一定程度上缓解过拟合 (防止每批训练中某一个样本经常被挑选到, 文献说这个可以提高1%的精度)。
- 减少梯度消失, 加快收敛速度, 提高训练精度。

6.8.2 算法流程

输入: 上一层输出结果 $X = x_1, x_2, \dots, x_m$, 学习参数 γ, β

1. 计算上一层输出数据的均值:

$$\mu_\beta = \frac{1}{m} \sum_{i=1}^m (x_i)$$

其中, m 是此次训练样本batch的大小。

2. 计算上一层输出数据的标准差:

$$\sigma_\beta^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_\beta)^2$$

3. 归一化处理, 得到:

$$\hat{x}_i = \frac{x_i + \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

其中 ϵ 是为了避免分母为0而加进去的接近于0的很小值

4. 重构, 对经过上面归一化处理得到的数据进行重构, 得到:

$$y_i = \gamma \hat{x}_i + \beta$$

其中, γ, β 为可学习参数。

上述是BN训练时的过程，但是当在投入使用时，往往只是输入一个样本，没有所谓的均值 μ_β 和标准差 σ_β^2 。此时，均值 μ_β 是计算所有batch μ_β 值的平均值得到，标准差 σ_β^2 采用每个batch σ_β^2 的无偏估计得到。

除此以外， γ 和 β 分别是用来控制平均值和方差的参数。

6.9 生成对抗网络

Algorithm 2 Mini-batch随机梯度下降算法。其中k为超参数，代表着判别器所需的步骤数。此处为了减轻训练负担，取k=1

- 1: **for** 训练迭代次数 **do**
 - 2: **for** k步 **do**
 - 3: 从之前的噪声 $p_g(z)$ 中取出minibatch所需的m个噪声样本 $\{z^{(1)}, \dots, z^{(m)}\}$
 - 4: 从数据生成分布 $p_{data}(x)$ 中取出minibatch所需的m个数据样本 $\{x^{(1)}, \dots, x^{(m)}\}$
 - 5: 通过随机梯度下降算法更新判别器参数:
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$$
 - 6: **end for**
 - 7: 从之前的噪声 $p_g(z)$ 中取出minibatch所需的m个噪声样本 $\{z^{(1)}, \dots, z^{(m)}\}$
 - 8: 通过随机梯度下降算法更新生成器参数:
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)})))$$
 - 9: **end for**
 - 10: 基于梯度的更新可以使用任何标准的基于梯度的学习规则。此处使用的是momentum算法。
-

Algorithm 3 WGAN, 以下参数中: $\alpha = 0.00005, c = 0.01, m = 64, n_{critic} = 5$

Input: 学习率 α , 截断常数 c , 批大小 m , 生成器每次迭代的迭代数量 n_{critic}

Input: 起始critic参数值 ω_0 , 初始生成器参数值 θ_0

```

1: while  $\theta$ 没有收敛 do
2:   for  $t = 0, \dots, n_{critic}$  do
3:     从真实数据 $P_r$ 中取出所需的m个数据样例 $\{x^{(1)}, \dots, x^{(m)}\}$ 
4:     从之前的样例 $p(z)$ 中取出所需的m个样例 $\{z^{(1)}, \dots, z^{(m)}\}$ 
5:      $g_\omega \leftarrow \nabla_\omega [\frac{1}{m} \sum_{i=1}^m f_\omega(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_\omega(g_\theta(z^{(i)}))]$ 
6:      $\omega \leftarrow \omega + \alpha \cdot RMSProp(\omega, g_\omega)$ 
7:      $\omega \leftarrow clip(\omega, -c, c)$ 
8:   end for
9:   从之前的样例 $p_g(z)$ 中取出所需的m个样例 $\{z^{(1)}, \dots, z^{(m)}\}$ 
10:   $g_\theta \leftarrow \nabla_\theta \frac{1}{m} \sum_{i=1}^m f_\omega(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta + \alpha \cdot RMSProp(\theta, g_\theta)$ 
12: end while

```
