深度学习

learning17

https://learning17.github.io/archive/

https://github.com/learning17/machinelearning

一、神经网络基本理论

1.1 简介

神经网络起源于上世纪五、六十年代，由 Frank Rosenblatt 提出，当时叫感知机，由输入层、输出层和一个中间隐含层组成。输入的特征向量通过隐含层变换，在输出层得到分类结果，单层的感知机有一个很严重的问题，无法拟合复杂的函数，甚至连典型的异或都无法搞定，直到上世纪八十年代，Rumelhart、Williams 等一票大牛发明了多层感知机，Werbos 在训练算法上提出反向传播算法，从此神经网络诞生。以房价的例子具体说明神经网络，如图 1.1 所示，横坐标表示房子大小 size，纵坐标表示房价 price。直观上看，线性回归可以拟合 price 和 size 之间的关系，但是房价不可以是负数，图 1.1 中蓝色折线可以用图 1.2 表示。输入是房子的大小，输出是房子的价格，圆圈被称为 Relu 神经元，这就是最简单的单层神经网络。

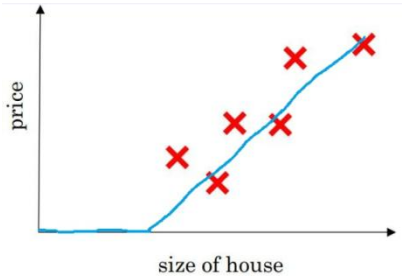


图 1.1

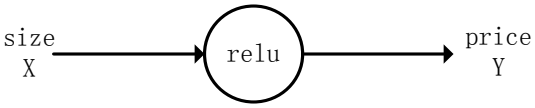


图 1.2

现在增加一些房屋特征：房间数 bedrooms、地理位置 zip code、财富 wealth。直观感觉，size 和 bedrooms 两个属性决定能住多少个人，组合起来构成一个隐含特征，记为 family size，zip code 和 wealth 两个属性决定附近学校的质量。不同特征之间的组合构成一个新的特征。可以用图 1.3 表示，多层神经网络。

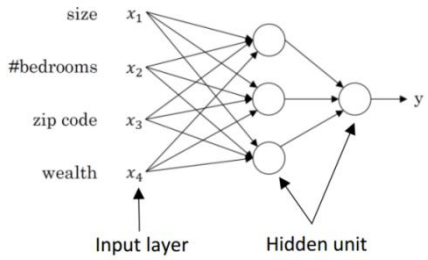


图 1.3

神经网络就是这样一层一层构建的，那么每层到底做了什么？让我们一步一步揭开神秘的面纱。

图 1.2 可以用数学公式 $y=\sigma(w\bullet x+b)$ 表示，其中 x 是输入向量，y 是输出向量，b 是偏移向量，w 是权重矩阵，σ 是激活函数。上面公式可以看作对 x 进行了下面 5 种操作，将输入空间投向另一个空间。

1）升维或降维(与 w 相乘)；

2）放大或缩小(与 w 相乘)；

3）旋转(与 w 相乘)；

4）平移(与 b 相加)；

5）非线性变换(激活函数 σ)；

http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/img/1layer.gif

下面以分类为例，将整数集合分为负数、零、正数三类，我们可以找到两个超平面（比当前低一维的子空间，该例子超平面是点）分割这三类，如图 1.4。但是对奇数和偶数进行分类，无法找到区分它们的点，如果进行简单的%2 变换，把 X 变换到另一个空间，可以很容易的进行分类。

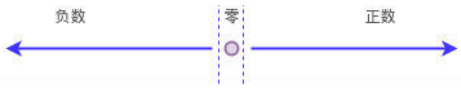


图 1.4

想要进一步感受，stanford 大学提供一个神器，同时可以参考 colah 大神的讲解。如图 1.5 所示，左边是不可分的，经过一系列的非线性变化，变为右边可分的。神经网络的一个显著的事实就是它可以计算任何的函数，网络层数越多，可以拟合(或者足够准确地近似)越复杂的函数。

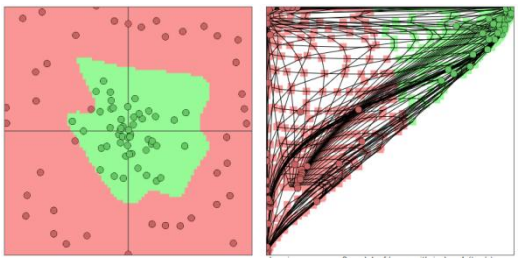


图 1.5

http://cs.stanford.edu/people/karpathy/convnetjs//demo/classify2d.html

http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

1.2 神经网络计算任何函数可视化理解

首先有两点声明：1.一个网络可以用来尽可能好的**近似地**计算任何函数，而不是**准确地**计算任何函数；2.被近似的函数不会出现突然、极陡的跳跃。

神经网络的一个显著的事实就是它可以计算任何的函数[1]，也就是说，假设某个人给你某种复杂而奇特的函数 $f(x)$，如图 1.6 所示，不管这个函数是什么样，总会确保有一个神经网络能够对任何可能的输入 x，网络的输出可以足够准确的近似 $f(x)$。

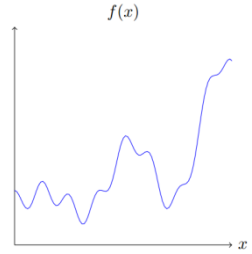


图 1.6

从简单的特例可以归纳出普遍性的定理，我们从最简单的一个输入和一个输出开始。如何构建网络近似 $f(x)$呢？从一个只包含一个隐藏层（两个隐藏神经元）和输出层（单个输出神经元）的网路开始，如图 1.7 所示。

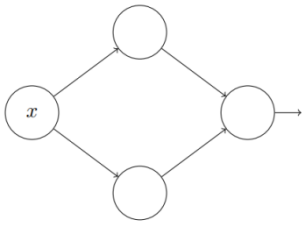


图 1.7

单个神经元是如何工作的？现在专注于隐藏层的顶层神经元，计算公式为$\sigma(wx+b)$，采用 S 型函数$\sigma(z)=1/(1+e^{-z})$。图 1.8 展示了 w、b 的变化对隐藏神经元输出的变化。

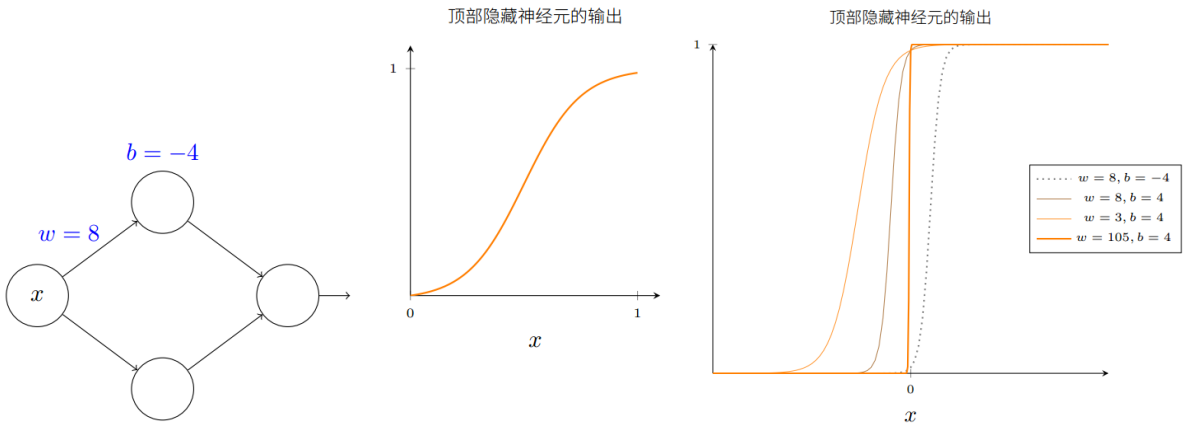


图 1.8

偏置 b 的值增大，图形左移动，形状不变；偏置 b 的值减小，图形右移，形状不变。权重 w 的值增大，图形被压缩，w 足够大，产生阶跃；权重 w 的值减小，图形被拉宽。

为了图形展示更加直观，我们直接把 S 型函数变为阶跃函数，这样隐藏神经元的输出值叠加就变得更加直观（阶跃函数只有 0、1 两个值，S 型函数有多个值，图形叠加时候，S 型函数看起来很混乱），可以把权重 w 固定在一个比较大的值，然后通过修改偏置 b 设置阶跃函数的位置。当 x 取值 s=-b/w 发生阶跃，后面的讨论我们只关心阶跃发生的位置 s，所以可以仅仅用一个参数 s 极大地简化隐藏神经元的描述。

目前为止仅仅讨论了单个隐藏神经元的输出，现在让我们看看两个隐藏神经元的输出叠加结果，两个隐藏神经元分别在 s_1、s_2 发生阶跃，作为输出层的输入权重分别为 w_1、w_2，如图 1.9 所示，右边是隐藏神经元的加权输出 $w_1a_1+w_2a_2$。

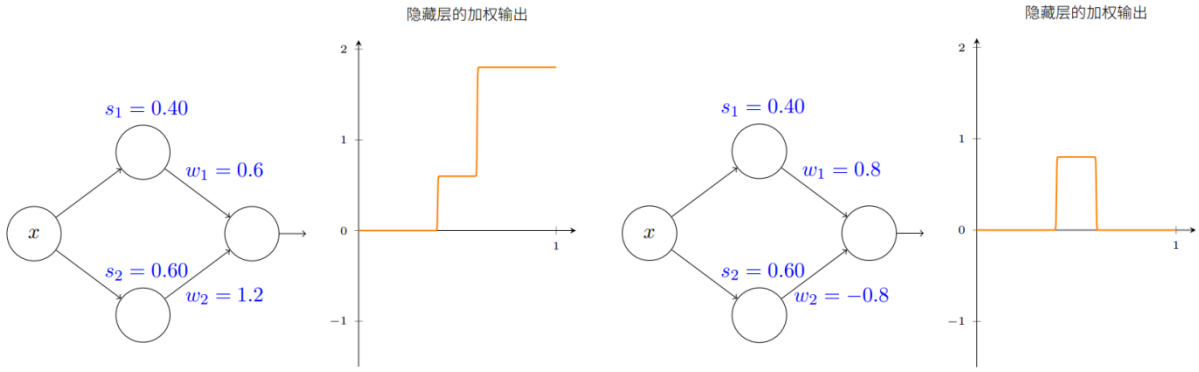


图 1.9

可以试着调整 s_1、s_2 体会下图形的变化，如果设置 $w_1=0.8$、$w_2=-0.8$，可以得到一个凸起的函数，如图 1.9 所示。当然通过调整 w_1、w_2，可以得到任意的凸起高度，简化表示，我们用一个 h 来表示高度。为了简化，移除图中的 s、w 的标记。我们可以通过增加隐藏层的神经元，得到更加复杂的图形。

回到我们最开始的问题，如何构建网络逼近图 1.6 所示的函数？上面已经分析了隐藏神经元的加权组合 $\sum_j w_j a_j$，实际网络的输出是 $\sigma(\sum_j w_j a_j + b)$。实际上隐藏层加权输出等于 $\sigma^{-1}(f(x))$（这里忽略 b 了，他是一个常量），其中 σ^{-1} 为 σ 反函数。如题 1.10 所示。我们可以增加隐藏层神经元，构建网络逼近 $\sigma^{-1}(f(x))$，如图 1.11 所示。

至此我们已经通过可视化图形说明了神经网络如何逼近一个输入一个输出的函数，对于两个输入、三个、甚至 N 个都可以构造神经网络逼近。如图 1.12 所示。

只要一个隐藏层就可以近似任何函数，为啥要研究深层网络呢？对于超级复杂函数，如果用单层隐藏层的网络，那么隐藏层的神经元数目多的超乎你想象，实际中根本无法训练，当增加网络层数，每层对上一层的输出做非线性地调整，可以有效地减少神经元数目，降低训练难度。http://neuralnetworksanddeeplearning.com/chap4.html

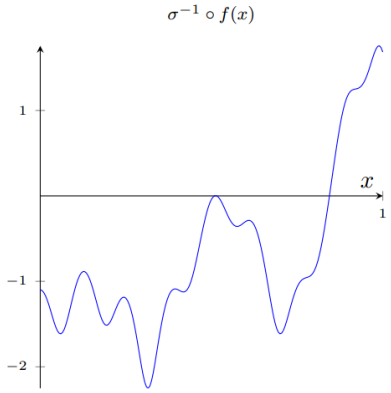


图 1.10

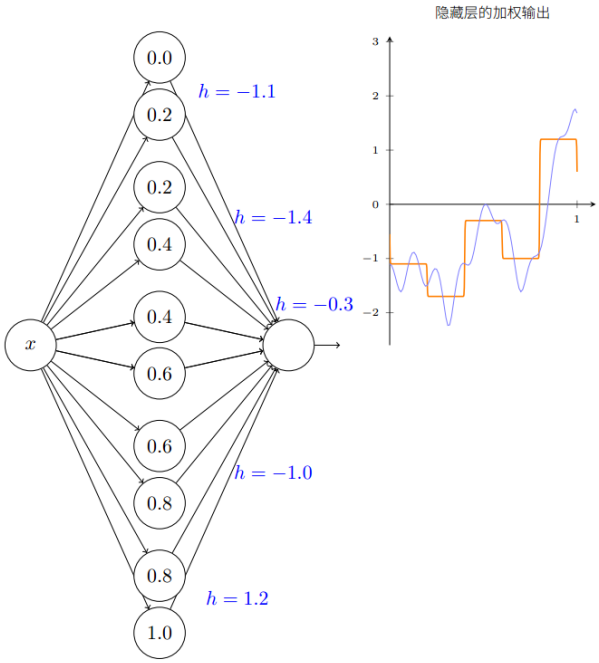


图 1.11

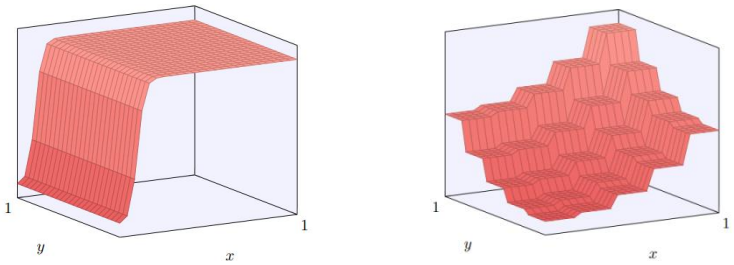


图 1.12

1.3 梯度下降算法

神经网络的输出与真实值越接近，网络预测效果越好，为了量化如何实现这个目标，定义一个代价函数

$$C(w,b)=\frac{1}{2n}\sum_{x}\| y(x)-a\|^2 \tag{1}$$

w表示网络中权重的集合，b表示所有的偏置，n表示训练数据的个数，a表示当输入为x时对应真实的 label，C称为二次代价函数，也称为均方误差或 MSE。求和公式中每一项都是非负的，如果$C(w,b)\approx 0$，表明对于所有的训练输入x，$y(x)$接近于a，相反，$C(w,b)$很大，意味着$y(x)$与a相差很大，因此我们的目标是寻找权重w和偏置b，使得代价函数$C(w,b)$最小，采用梯度下降算法实现这个目标。

我们现在不局限于神经网络来说明梯度下降算法，假设要最小化某些函数，$C(v)$，它是任意多元实值函数。为了直观表示，设C是一个只有两个变量v_1、v_2的函数，如图 1.13 所示，可以想象成一个山谷，上面有一个小球滚下来，日常经验告诉我们这个小球最终会滚到谷底，这个现象给我们一些启示。

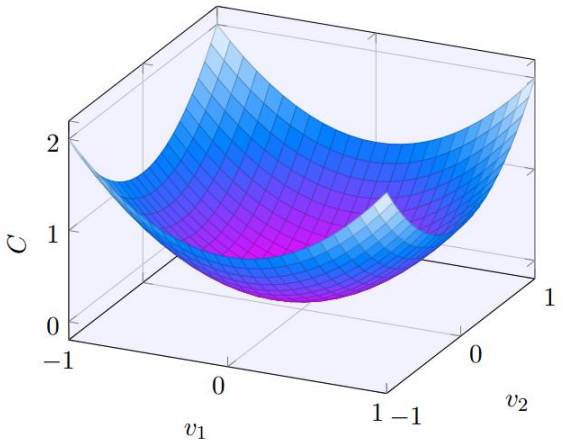


图 1.13

小球在v_1、v_2方向上分别移动一个很小的量，即Δv_1和Δv_2。根据微积分知识，C将会有如下的变化：

$$\Delta C\approx\frac{\partial C}{\partial v_1}\Delta v_1+\frac{\partial C}{\partial v_2}\Delta v_2 \tag{2}$$

寻找 Δv_1、Δv_2 使得 ΔC 为负，这样小球每次移动，C 的值就减小，即往山谷方向移动。

记 $\Delta v = (\Delta v_1, \Delta v_2)^T$，$\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T$，代入公式(2)得。

$$\Delta C = (\nabla C)^T \bullet \Delta v \tag{3}$$

从公式(3)可得，如果 Δv 按公式(4)取值，那么 $\Delta C = -\eta \| \nabla C \|^2$，保证了 $\Delta C \le 0$，即按照公式(10)改变 Δv，那么 C 会一直减少，直到 $\nabla C = 0$。

$$\Delta v = -\eta \nabla C \tag{4}$$

η 是一个很小的正数称为学习率，v 的更新规则见公式(5)。

$$v := v - \eta \nabla C \tag{5}$$

C 扩展到具有多个变量 $v_1, ..., v_m$

$$\begin{aligned} \Delta v &= (\Delta v_1, .. \Delta v_m)^T \\ \nabla C &= \left(\frac{\partial C}{\partial v_1}, .. \frac{\partial C}{\partial v_m} \right)^T \end{aligned} \tag{6}$$

$\Delta v = -\eta \nabla C$ 也可以保证 ΔC 非正。

我们再回到神经网络中，利用梯度下降算法寻找能使公式(1)取得最小值的权重 w_k 和偏置 b_l，更新规则见公式(7)。

$$\begin{aligned} w_k &:= w_k - \eta \frac{\partial C}{\partial w_k} \\ b_l &:= b_l - \eta \frac{\partial C}{\partial b_l} \end{aligned} \tag{7}$$

为了计算梯度 ∇C，需要为每个训练数据 x 单独计算梯度值 ∇C_x，然后求平均值 $\nabla C = \frac{1}{n} \sum_x \nabla C_x$，当训练数据量过大时，算法学习变得相当缓慢。

随机梯度下降算法可以加速学习，缓解上面问题。其思想就是通过随机选取少量训练数据计算 ∇C_x，进而估算梯度 ∇C。少量随机训练数据标记为 $x_1, ..., x_m$ 称为一个

mini-batch，采用 ∇C_{x_j} 的平均值代替 ∇C_x 的平均值，即：

$$\frac{\sum_{j=1}^{m}\nabla C_{x_j}}{m}\approx\frac{\sum_{x}\nabla C_x}{n}=\nabla C \tag{8}$$

代入公式(7)可得

$$\begin{aligned} w_k &:= w_k-\frac{\eta}{m}\sum_j\frac{\partial C_{x_j}}{\partial w_k}\\ b_l &:= b_l-\frac{\eta}{m}\sum_j\frac{\partial C_{x_j}}{\partial b_l}\end{aligned} \tag{9}$$

1.4 反向传播算法

1.反向传播实例

上一节，我们看到了如何用梯度下降法算法学习神经网络的权重和偏置，最重要的环节是如何计算代价函数的关于权重和偏置梯度，本节介绍一种快速求解梯度的算法**反向传播算法**。我们先从一个例子直观地感受下，然后再从数学上进行推导，反向传播是利用链式法则递归计算表达式的梯度的算法，当前梯度等于局部梯度和后一层的梯度相乘，如图 1.14 所示。1）给定一个节点的运算，其局部梯度也同时已知（$f(x,y)$、$\frac{\partial z}{\partial x}$、$\frac{\partial z}{\partial y}$）；2）由输入进入节点，首先正向传播，得到输出 z；3）然后反向传播，根据传入的后一层梯度与局部梯度相乘得到每个输入的梯度。

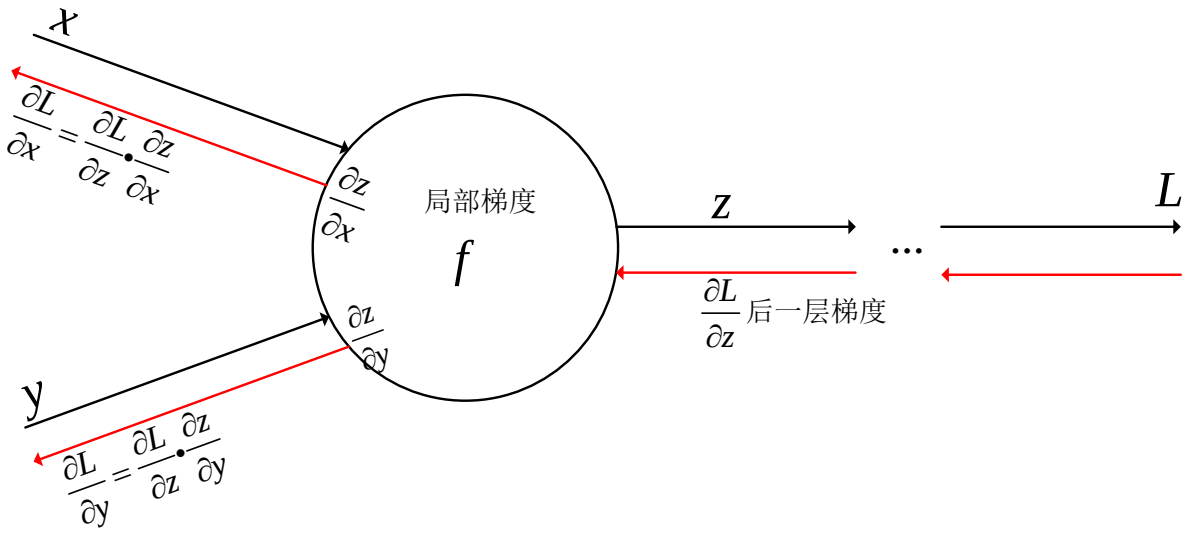


图 1.14

按照上面的法则，我们用公式(10)练练手吧，求解 L 对各个变量的偏导。将公式(10)

$$L(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}} \tag{10}$$

拆分为多个计算单元，如图 1.15 所示，假设 w_0 、 x_0 、 w_1 、 x_1 、 w_2 分别取值为 2、-1、-3、-2、-3。具体流程如下：

(1) 前向传播，计算各个节点的输出值；

(2) 反向传播， $L=\frac{1}{g}\Rightarrow\frac{\partial L}{\partial g}=-\frac{1}{g^2}=-\frac{1}{(1.37)^2}=-0.53$

(3) $g=f+1\Rightarrow\frac{\partial g}{\partial f}=1\Rightarrow\frac{\partial L}{\partial f}=\frac{\partial L}{\partial g}*\frac{\partial g}{\partial f}=-0.53*1=-0.53$

(4) $f=\exp(e)\Rightarrow\frac{\partial f}{\partial e}=\exp(e)=0.37\Rightarrow\frac{\partial L}{\partial e}=\frac{\partial L}{\partial f}*\frac{\partial f}{\partial e}=-0.53*0.37=-0.2$

(5) $e=-d\Rightarrow\frac{\partial e}{\partial d}=-1\Rightarrow\frac{\partial L}{\partial d}=\frac{\partial L}{\partial e}*\frac{\partial e}{\partial d}=-0.2*(-1)=0.2$

(6) $d=c+w_2\Rightarrow\frac{\partial d}{\partial c}=1,\frac{\partial d}{\partial w_2}=1\Rightarrow\frac{\partial L}{\partial c}=\frac{\partial L}{\partial d}*\frac{\partial d}{\partial c}=0.2,\frac{\partial L}{\partial w_2}=\frac{\partial L}{\partial d}*\frac{\partial d}{\partial w_2}=0.2$

(7) $c=a+b\Rightarrow\frac{\partial c}{\partial a}=1,\frac{\partial c}{\partial b}=1\Rightarrow\frac{\partial L}{\partial a}=\frac{\partial L}{\partial c}*\frac{\partial c}{\partial a}=0.2,\frac{\partial L}{\partial b}=\frac{\partial L}{\partial c}*\frac{\partial c}{\partial b}=0.2$

(8) $a=w_0*x_0\Rightarrow\frac{\partial a}{\partial w_0}=x_0=-1,\frac{\partial a}{\partial x_0}=w_0=2\Rightarrow\frac{\partial L}{\partial w_0}=\frac{\partial L}{\partial a}*\frac{\partial a}{\partial w_0}=-0.2,\frac{\partial L}{\partial x_0}=\frac{\partial L}{\partial a}*\frac{\partial a}{\partial x_0}=0.4$

(9) $b=w_1*x_1\Rightarrow\frac{\partial b}{\partial w_1}=x_1=-2,\frac{\partial b}{\partial x_1}=w_1=-3\Rightarrow\frac{\partial L}{\partial w_1}=\frac{\partial L}{\partial b}*\frac{\partial b}{\partial w_1}=-0.4,\frac{\partial L}{\partial x_1}=\frac{\partial L}{\partial b}*\frac{\partial b}{\partial x_1}=-0.6$

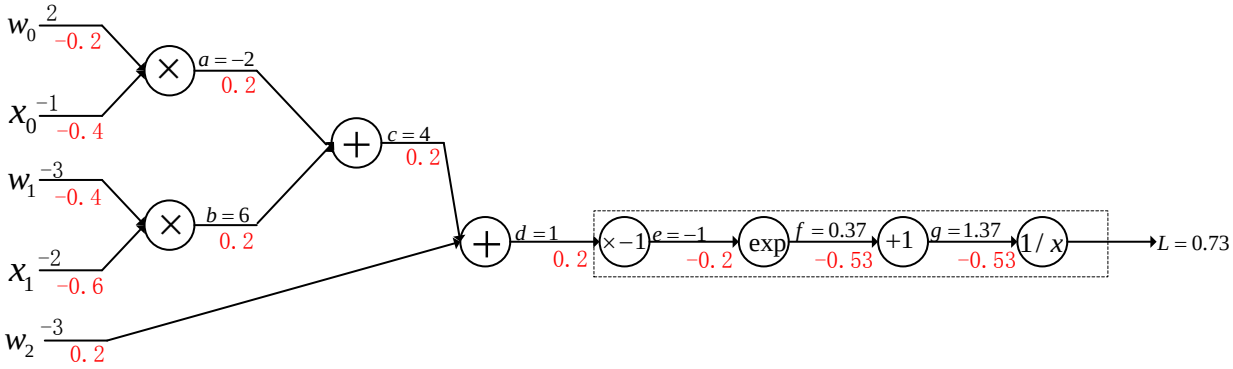


图 1.15

图 1.15 中，每个运算都单独作为一个节点存在，一般实践中会把一些常用的运算组合成一个运算门，图 1.15 中虚线框部分组合起来就是神经网络中的激活函数—Sigmoid

函数 $\sigma(z)=\frac{1}{1+e^{-z}} \Rightarrow \frac{d\sigma(z)}{z}=\sigma(z)*(1-\sigma(z))$。

2.反向传播数学推导

通过上面的例子，大家对反向传播有了深刻的认识了，下面我们从数学上推导反向传播算法。反向传播的目标是计算代价函数 C 关于 w 和 b 的偏导数 $\partial C/\partial w$ 和 $\partial C/\partial b$，我们先有两个假设：

（1）代价函数 C 是训练样本 x 的代价函数 C_x 的均值 $C=\frac{1}{m}\sum_x C_x$。反向传播实际上是对每个独立样本计算了 $\partial C_x/\partial w$ 和 $\partial C_x/\partial b$，然后在所有训练样本上进行平均化获得 $\partial C/\partial w$ 和 $\partial C/\partial b$，有了上面的假设，假设训练样本 x 已经被固定，那么可以将代价函数 C_x 看作 C。

（2）代价函数看作网络的输出函数。这里推导过程，采用二次代价函数：

$$C=\frac{1}{2}\left\|y-a^L\right\|^2$$

首先定义一些参数的含义：

（1）w^l_{jk} 表示从 $(l-1)^{th}$ 层的 k^{th} 个神经元到 l^{th} 层的 j^{th} 个神经元的连接权重，权重矩阵 w^l，其 j^{th} 行 k^{th} 列的元素是 w^l_{jk}；

（2）b^l_j 表示 l^{th} 层的 j^{th} 个神经元的偏置，偏置向量 b^l，其 j^{th} 个元素是 b^l_j；

（3）z^l_j 表示 l^{th} 层的 j^{th} 个神经元的激活函数的带权重输入，$z^l_j=\sum_k w^l_{jk}a^{l-1}_k+b^l_j$，向量 z^l，其 j^{th} 个元素是 z^l_j；

（3）a^l_j 表示 l^{th} 层的 j^{th} 个神经元的激活值，$a^l_j=\sigma\left(z^l_j\right)$，激活向量 a^l，其 j^{th} 个元素是 a^l_j；

（4）δ^l_j 表示 l^{th} 层的 j^{th} 个神经元上的误差，$\delta^l_j=\frac{\partial C}{\partial z^l_j}$；

（5）$\nabla_a C$ 表示一个向量，其 j^{th} 个元素是 $\partial C/\partial a^L_j$。

针对 δ^l_j 说明，假设 l^{th} 层的 j^{th} 个神经元发生 Δz^l_j 波动，那么使得神经元的输出由 $\sigma\left(z^l_j\right)$ 变为 $\sigma\left(z^l_j+\Delta z^l_j\right)$，这个变化会向网络后面的层进行传播，最终导致网络代价产生 $\frac{\partial C}{\partial z^l_j}\Delta z^l_j$ 的

改变。如果δ_j^l接近于 0，那么无论Δz_j^l如何变化，网络代价都不会产生变化，这时候这个神经元已经接近最优了。下面给出反向传播的四个方程：

- **输出层误差的方程：**

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right)\\ \delta^L &= \nabla_a C \odot \sigma'\left(z^L\right)\end{aligned} \tag{11}$$

证明：

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial z_j^L}\\ &= \sum_k \frac{\partial C}{\partial a_k^L}\cdot\frac{\partial a_k^L}{\partial z_j^L}\\ &= \sum_k \frac{\partial C}{\partial a_k^L}\cdot\frac{\partial \sigma\left(z_k^L\right)}{\partial z_j^L}\end{aligned} \tag{12}$$

只有$k=j,\frac{\partial \sigma\left(z_k^L\right)}{\partial z_j^L}$才不为 0，所以由公式(12)和定义(4)可得公式(11)。

- **使用下一层的误差δ^{l+1}表示当前层的误差δ^l：**

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right)\odot\sigma'\left(z^l\right) \tag{14}$$

证明：

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l}\\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l}\end{aligned} \tag{15}$$

$$\begin{aligned}z_k^{l+1} &= \sum_i w_{ki}^{l+1}a_i^l + b_k^{l+1}\\ &= \sum_i w_{ki}^{l+1}\sigma\left(z_i^l\right) + b_k^{l+1}\end{aligned} \tag{16}$$

只有$i=j,\frac{\partial \sigma\left(z_i^l\right)}{\partial z_j^L}$才不为 0，所以

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1}\sigma'(z_j^l) \tag{17}$$

根据定义(4)，得：

$$\frac{\partial C}{\partial z_k^{l+1}} = \delta_k^{l+1} \tag{18}$$

由公式(13)、(15)、(16)得：

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l) = \left(w_j^{l+1}\right)^T \delta^{l+1} \sigma'(z_j^l) \tag{19}$$

由公式(19)可得公式(14)。

- **代价函数关于任意偏置的偏导数：**

$$\begin{aligned} \frac{\partial C}{\partial b_j^l} &= \delta_j^l \\ \frac{\partial C}{\partial b^l} &= \delta^l \end{aligned} \tag{21}$$

证明：

$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_j^l} \tag{22}$$

$$z_k^l = \sum_i w_{ki}^l a_i^{l-1} + b_k^l \tag{23}$$

只有 $k = j, \frac{\partial z_k^l}{\partial b_j^l}$ 才不为 0，所以：

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \tag{24}$$

由公式(24)和定义(4)得公式(21)：

- **代价函数关于任意权重的偏导数：**

$$\begin{aligned} \frac{\partial C}{\partial w_{jk}^l} &= a_k^{l-1} \delta_j^l \\ \frac{\partial C}{\partial w^l} &= \delta^l \cdot (a^{l-1})^T \end{aligned} \tag{26}$$

证明：

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_i \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{jk}^l} \tag{27}$$

$$z_i^l = \sum_n w_{in}^l a_n^{l-1} + b_i^l \tag{28}$$

只有 $n = k, i = j, \frac{\partial w_{in}^l}{\partial w_{jk}^l}$ 才不为 0，所以：

$$\frac{\partial C}{\partial w_{jk}^{l}}=\frac{\partial C}{\partial z_{j}^{l}}a_{k}^{l-1} \tag{29}$$

由公式(29)和定义(4)可以得公式(26)。

上面已经给出反向传播四大公式的证明，总结下反向传播算法：

1. 输入训练样本的集合

2. 对于每个训练样本 x：设置对应的输入激活值为 $a^{x,1}$，并执行下面的步骤：

- **前向传播：** 对每个 $l=2,3,..,L$ 计算 $z^{x,l}=w^{l}a^{x,l-1}+b^{l}$ 和 $a^{x,l}=\sigma\left(z^{x,l}\right)$；
- **输出误差** $\delta^{x,L}$**：** 计算向量 $\delta^{x,L}=\nabla_{a}C_{x}\odot\sigma'\left(z^{x,L}\right)$；
- **反向传播误差：** 对每个 $l=L-1,L-2,...,2$ 计算 $\delta^{x,l}=\left[\left(w^{l+1}\right)^{T}\delta^{x,l+1}\right]\odot\sigma'\left(z^{x,l}\right)$；

3. 梯度下降： 对每个 $l=L-1,L-2,...,2$ 根据 $w^{l}\to w^{l}-\frac{\eta}{m}\sum_{x}\delta^{x,l}\cdot(a^{x,l-1})^{T}$ 和 $b^{l}\to b^{l}-\frac{\eta}{m}\sum_{x}\delta^{x,l}$ 更新权重和偏置。

1.5 激活函数

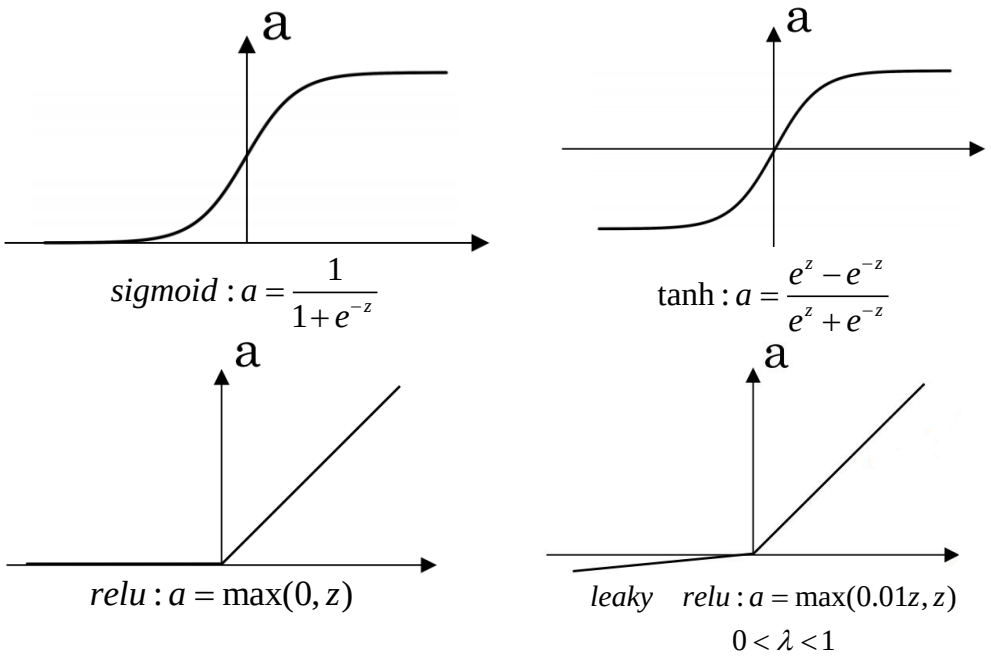


图 1.16

- **sigmoid：** 一般情况下不使用这个激活函数，除非特殊情况，如输出层是 0、1 二分类，可以在输出层采用这个激活函数。缺点：

1）当神经元的激活值在接近 0 或 1 处时会饱和，梯度几乎为 0。在反向传播时候，这个局部梯度将会与整个损失函数关于该单元输出的梯度相乘，因此如果局部梯度非常小，那么相乘结果也会接近零，出现梯度消失，最终导致权重基本没有更新；2）输出不是 0 均值，导致神经网络后面层中的神经元得到的数据将不是零中心，会对梯度下降产生影响，假设后面层中的神经元输入都为正数(比如 $f=w^Tx+b$ 中每个元素都 x_i>0)，那么关于 w 的梯度（每个维度的梯度）在反向传播过程中，要么全部是正数，要么全部是负数(关于每个 w_i 的梯度都是用 x_i 乘以 f 的梯度，所以要么全正，要么全负，完全依赖于 f 的梯度)，这将出现 z 字型的下降，使得收敛很缓慢。(如果真实梯度是第二、第四象限方向，而梯度下降只能在第一或第三象限区域，所以只能垂直或者水平下降，形成 z 字型)，如图 1.17 所示；3）exp 函数计算耗时。

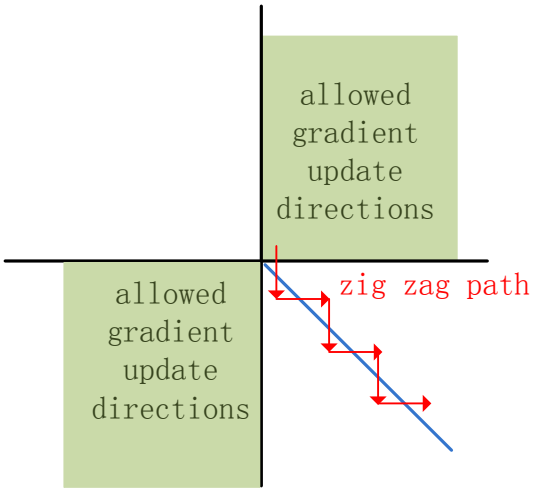


图 1.17

- **tanh：**仍然存在梯度消失问题，但是它的输出是零中心的，在实际使用过程中，尽量使用 tanh 代替 sigmoid。
- **relu：**优点 1）在输入大于 0 时候，不会出现梯度消失；2）相较于 sigmoid 和 tanh 收敛速度大大提升，不含有 exp 运算，计算速度提升。缺点 1）输出不是关于中心点对称；2）当输入小于 0，在前向传播时 relu 一直处于非激活状态，那么反向传播就会出现梯度消失(未激活的神经元局部梯度为 0)，因此非激活的神经元无法进行反向传播，它的权重也不会更新；当输入大于 0，relu 神经元直接将梯度传给前面层网络；当输入为 0，梯度未定义，我们可以简单指定为 0 或 1（实际出现输入为 0 的概率很小）。3）可能出现 relu 神经元失活问题，如 $\max(0, w^Tx+b)$，假设 w、b 都

初始化为 0，或者更新过程中，w、b 更新为一种特别状态，使得 $w^T x+b$ 永远小于 0，那么该神经元权重将永远得不到更新。

- **leaky relu：**有点 1）不会出现梯度消失；2）不会出现失活问题。PRelu $\max(\alpha z, z)$，其中 α 是一个参数，可以通过正常的反向传播算法学习它。

在实际使用过程中，可以将激活函数组合使用。神经网络中为什么存在非线性激活函数呢？以简单的两层网络说明，如公式(30)，其中 g 表示激活函数。

$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ a^{[1]} &= g^{[1]}(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= g^{[2]}(z^{[2]}) \end{aligned} \tag{30}$$

如果不采用非线性激活函数，相当于 $g^{[i]}(z^{[i]}) = z^{[i]}$，所以公式(30)变为公式(31)。

$$\begin{aligned} a^{[1]} &= z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[2]} &= z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ &= W^{[2]}\left(W^{[1]}x + b^{[1]}\right) + b^{[2]} \\ &= W^{[2]}W^{[1]}x + W^{[2]}b^{[1]} + b^{[2]} \\ &= W^{'}x + b^{'} \end{aligned} \tag{31}$$

无论多深的网络，最后的效果只是线性组合罢了。

1.6 交叉熵代价函数

前面提到二次代价函数，它存在学习缓慢的问题，二次代价函数定义如公式(32)。

$$C = \frac{\left\| y - a \right\|^2}{2} \tag{32}$$

由前面反向传播算法可知，公式(11)，输出层误差为：

$$\begin{aligned} \delta_j^L &= \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \\ &= \left(a_j^L - y_j\right)\sigma'\left(z_j^L\right) \end{aligned} \tag{33}$$

代价函数关于每层权重的偏导数 $\frac{\partial C}{\partial w^l}$ 与该层的误差 δ^l 成正比，见公式(26)，每层的误差 δ^l 与下一层的误差 δ^{l+1} 成正比，并且与该层神经元的导数 $\sigma'\left(z^l\right)$ 也成正比，见公式(14)。

在上一小节我们知道，sigmoid、tanh 激活函数当输入接近 0 或 1，导数几乎为 0，出现

梯度消失现象，并且这种梯度消失不断的往上一层扩散。所以神经网络隐藏层的神经元一般采用 relu 或 leaky relu 激活函数。但是对于输出层，如果是二分类，一般采用 sigmoid（可见吴恩达教程）激活函数，这样输出层误差就有可能由于激活函数的导数接近于 0，而近似为 0，从而影响整个网络的关于权重和偏置项的梯度，导致学习缓慢的问题。

我们希望最后一层的误差和激活函数的梯度没有关系，即：

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \\ &= \left(a_j^L - y_j\right)\end{aligned} \tag{34}$$

假设输出层激活函数为 sigmoid，$a_j^L = \sigma(z_j^L)$，$\sigma'(z_j^L) = a_j^L\left(1-a_j^L\right)$ 代入公式(33)得：

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \\ &= \frac{\partial C}{\partial a_j^L}a_j^L\left(1-a_j^L\right)\end{aligned} \tag{35}$$

由公式(34)和公式(35)可得：

$$\nabla C_x \tag{36}$$

公式(36)称为交叉熵代价函数，得到输出层的误差为：

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \\ &= \frac{\left(a_j^L - y_j\right)}{a_j^L\left(1-a_j^L\right)}a_j^L\left(1-a_j^L\right) \\ &= \left(a_j^L - y_j\right)\end{aligned} \tag{37}$$

公式(37)比较完美了，输出误差和输出层的激活函数的导数没任何关系。

1.7 正则化技术

1. 偏差、方差、欠拟合、过拟合

在开始介绍正则化技术之前，我们先介绍偏差、方差、欠拟合和过拟合的概念，以及偏差、方差和欠拟合、过拟合之间的关系。

偏差：描述的是预测值的期望与真实值之间的差距。偏差越大，越偏离真实值，如图 1.18；

方差：描述的是预测值的变化范围，离散程度，也就是离预测值期望值的距离。方差越大，预数据的分布越离散。

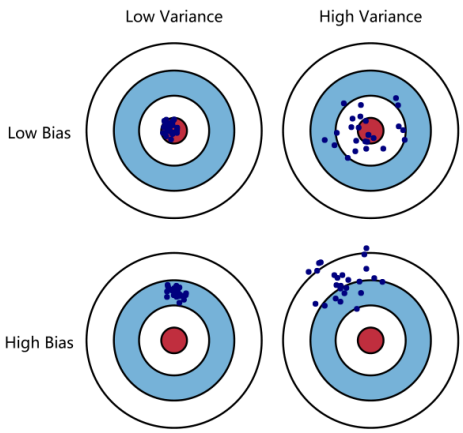


图 1.18

欠拟合：模型过于简单，不足以拟合训练数据的基本（或内在）的关系。表现模型不能在训练集上获得足够低的误差，高偏差

过拟合：由于训练数据包含抽样误差，训练时候，复杂的模型将抽样误差也考虑在内，将抽样误差也进行了很好的拟合。具体表现是模型在训练数据集上效果很好，在测试集上效果很差，模型泛化能力弱。表现训练误差和测试误差之间的差距太大，高方差。

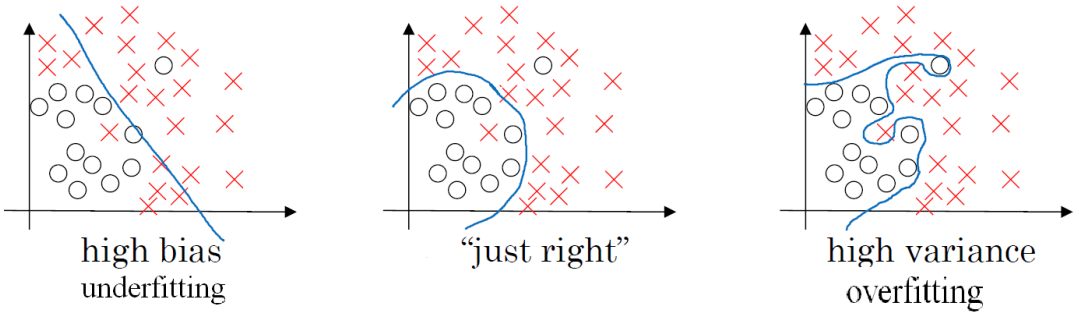
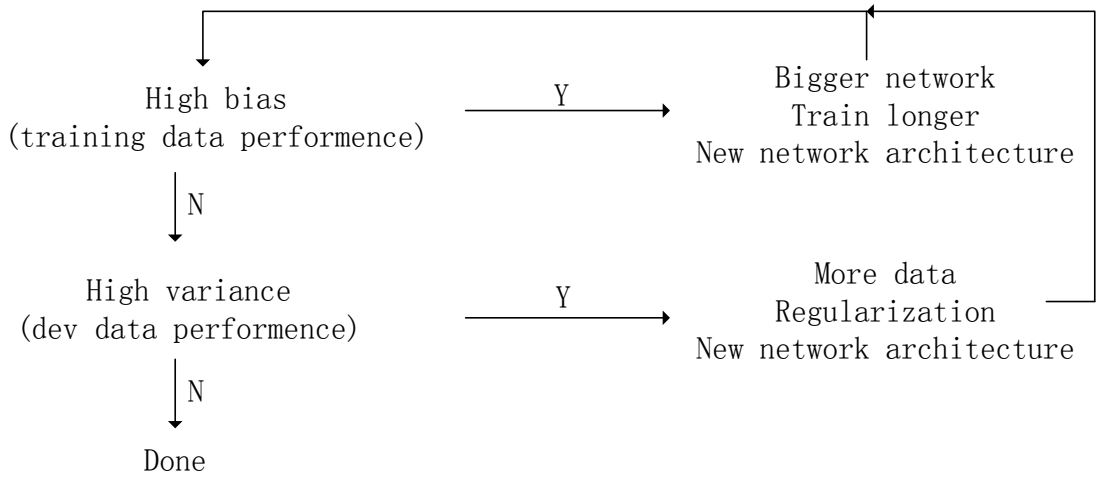


图 1.19

Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
	high variance overfitting	high bias underfitting	high bias high variance underfitting overfitting parts of data as well	low bias low variance

表 1.1



2. 正则化技术

防止过拟合，提高模型泛化能力。为什么正则化可以减少过拟合？

$$J(w^{[l]},b^{[l]})=\frac{1}{m}\sum_{i=1}^{m}l(y^{(i)},\hat{y}^{(i)})+\frac{\lambda}{2m}\left\|w^{[l]}\right\|_F^2 \tag{38}$$

公式(38)是加了正则化的损失函数，直观上理解，如果 λ 取足够的大值，那么只有当权重 $w^{[l]}$ 接近于 0，损失函数 J 才能取到最小值，也就相当于神经网络中隐藏层的权重设为 0，神经网络退化为一个很小的网络，小到如同一个逻辑回归单元，所以整个网络处于欠拟合状态。如图 1.19，在增大 λ 的过程，网络由过拟合变为欠拟合，中间会出现"just right"状态。正则化本质是减少隐藏层神经元对整个网络的影响。即网络输出不会出现过分依赖某些神经元，从而提高网络泛化能力。下面介绍常见的几种正则化技术：

1）L1 范数

L1 范数产生稀疏的解。下面给出推导过程，我们以二次代价函数为例，见公式(38)：

$$J_L(\mathrm{w})=\frac{1}{n}\left\|y-\mathrm{wx}\right\|^2+\lambda\left\|\mathrm{w}\right\|_1 \tag{38}$$

定义（次导数；次微分）对于在 p 维欧式空间中的凸开子集 U 上定义的实值函数 $f:U\to R$，p 维向量 v 称为 f 在点 $\mathrm{x}_0\in U$ 处的次导数，如果对于任意 $\mathrm{x}\in U$，满足：

$$f(\mathrm{x})-f(\mathrm{x}_0)\geq \mathrm{v}\cdot(\mathrm{x}-\mathrm{x}_0)$$

由在点 x_0 处的所有次导数所组成的集合称为 x_0 处的次微分，记为 $\partial f(\mathrm{x}_0)$。

性质 1：在 x_0 处的次导数的集合是一个非空闭区间 $[a,b]$。

$$a=\lim_{x\to x_0^-}\frac{f(x)-f(x_0)}{x-x_0}\quad b=\lim_{x\to x_0^+}\frac{f(x)-f(x_0)}{x-x_0}$$

性质 2：凸函数 f 在 x_0 处可导，当且仅当次微分只有一个点组成，这个点就是 x_0 的导数。

性质 3：点 x_0 是凸函数 f 的一个全局最小值点，当且仅当 $0\in\partial f(x_0)$，也就是次微分中包含 0。

假设特征之间相互正交，即：

$$\frac{1}{n}xx^T=I,\quad \hat{w}=\left(xx^T\right)^{-1}xy^T=\frac{1}{n}xy^T \text{ 是 } \frac{1}{n}\left\|y-wx\right\|^2 \text{ 的解析解。}$$

假设 $\bar{w}=\left(\bar{w}^1,\bar{w}^2,...,\bar{w}^p\right)^T$ 是 $J_L(w)$ 全局最优值点，考虑第 j 个变量 $\bar{w}^j$，有两种情况：

- $\bar{w}^j\neq 0$，gradient 存在

最小值点的梯度值必须等于 0，所以：

$$\begin{aligned}
&\left.\frac{\partial J_L(w)}{\partial w^j}\right|_{\bar{w}^j}=0\\
&-\frac{2}{n}x\left(y-wx\right)_j^T+\lambda sign\left(\bar{w}^j\right)=0\\
&-\frac{1}{n}\left(xy^T-xx^Tw\right)_j+\frac{\lambda}{2}sign\left(\bar{w}^j\right)=0\\
&-\hat{w}^j+\bar{w}^j+\frac{\lambda}{2}sign\left(\bar{w}^j\right)=0\\
&\bar{w}^j=\hat{w}^j-\frac{\lambda}{2}sign\left(\bar{w}^j\right)
\end{aligned}\tag{39}$$

由公式(39)可得，$\bar{w}^j$ 和 $\hat{w}^j$ 是同号的，于是 $sign\left(\bar{w}^j\right)=sign\left(\hat{w}^j\right)$，所以公式(39)变为：

$$\bar{w}^j=\hat{w}^j-\frac{\lambda}{2}sign\left(\hat{w}^j\right)\tag{40}$$

等号左边乘以 $sign(\bar{w}^j)$，右边乘以 $sign\left(\hat{w}^j\right)$，得：

$$\begin{aligned}
&sign\left(\bar{w}^j\right)\cdot\bar{w}^j=sign\left(\hat{w}^j\right)\cdot\hat{w}^j-\frac{\lambda}{2}sign\left(\hat{w}^j\right)\cdot sign\left(\hat{w}^j\right)\\
&\Rightarrow\left|\hat{w}^j\right|-\frac{\lambda}{2}=\left|\bar{w}^j\right|\geq 0\\
&\Rightarrow\bar{w}^j=sign\left(\hat{w}^j\right)\left(\left|\hat{w}^j\right|-\frac{\lambda}{2}\right)\quad,\left|\hat{w}^j\right|\geq\frac{\lambda}{2}
\end{aligned}\tag{41}$$

- $\bar{w}^j=0$，gradient 不存在

根据公式(41)，若 $\left|\hat{\mathrm{w}}^j\right| \le \frac{\lambda}{2}$，得：

$$\begin{aligned} &\left|\bar{\mathrm{w}}^j\right| = \left|\hat{\mathrm{w}}^j\right| - \frac{\lambda}{2} \le 0 \\ &\Rightarrow \left|\bar{\mathrm{w}}^j\right| = 0 \end{aligned} \tag{42}$$

导数不存在，根据次导数最小值点的性质，有：

$$\begin{aligned} 0 = \bar{\mathrm{w}}^j \in \partial J_L(\bar{w}) &= \left\{ -\frac{2}{n}\left(\mathrm{x}\mathrm{y}^T - \mathrm{x}\mathrm{x}^T\mathrm{w}\right)_j + \lambda e : e \in [-1,1] \right\} \\ &= \left\{ 2\bar{\mathrm{w}}^j - 2\hat{\mathrm{w}}^j + \lambda e : e \in [-1,1] \right\} \end{aligned} \tag{43}$$

存在 $e_0 \in [-1,1]$，使得

$$\begin{aligned} &2\bar{\mathrm{w}}^j - 2\hat{\mathrm{w}}^j + \lambda e_0 = 0 \\ &\Rightarrow -2\hat{\mathrm{w}}^j + \lambda e_0 = 0 \end{aligned} \tag{44}$$

所以

$$\left|\hat{\mathrm{w}}^j\right| = \frac{\lambda}{2}|e_0| \le \frac{\lambda}{2} \tag{45}$$

由公式(41)、(42)、(45)可得：

$$\bar{\mathrm{w}}^j = sign\left(\hat{\mathrm{w}}^j\right)\left(\left|\hat{\mathrm{w}}^j\right| - \frac{\lambda}{2}\right)_+ \tag{46}$$

其中 $(x)_+ = \max(0, x)$。

2）L2 范数

L2 范数对权重做了缩放。下面给出推导过程，我们以二次代价函数为例，见公式(47)：

$$J_L(\mathrm{w}) = \frac{1}{n}\|y - \mathrm{w}\mathrm{x}\|^2 + \lambda\|\mathrm{w}\|_2^2 \tag{47}$$

关于 w 是全局可导，推导过程见公式(48)

$$\begin{aligned} &\left.\frac{\partial J_L(\mathrm{w})}{\partial \mathrm{w}}\right|_{\mathrm{w}} = 0 \\ &-\frac{2}{n}\mathrm{x}\left(\mathrm{y} - \mathrm{w}\mathrm{x}\right)^T + 2\lambda\mathrm{w} = 0 \\ &-\frac{2}{n}\left(\mathrm{x}\mathrm{y}^T - \mathrm{x}\mathrm{x}^T\mathrm{w}\right) + 2\lambda\mathrm{w} = 0 \\ &-\hat{\mathrm{w}} + \mathrm{w} + \lambda\mathrm{w} = 0 \\ &\mathrm{w} = \frac{1}{1+\lambda}\hat{\mathrm{w}} \end{aligned} \tag{48}$$

总结：L2 对 $\hat{\mathrm{w}}$ 做了一个全局的缩放；而 L1 对 $\hat{\mathrm{w}}$ 做了一个 soft thresholding，将绝对值小

于$\frac{\lambda}{2}$的那些系数直接变为 0，产生稀疏性。

3）Dropout

对第l层进行 Dropout，保留某个神经元的概率为$keep_prob$。前向传播的时候让某个神经元的激活值以一定概率$1-keep_prob$停止工作（即把该神经元的激活值设为 0），为了不影响整个网络输出的期望值，未被 Dropout 的神经元的激活值除以$keep_prob$，具体见算法 1.1，直观如图 1.20。

算法 1.1 Dropout

Train:

$d^{[l]}$ = np.random.rand($a^{[l]}$.shape[0], $a^{[l]}$.shape[1]) < keep_prob

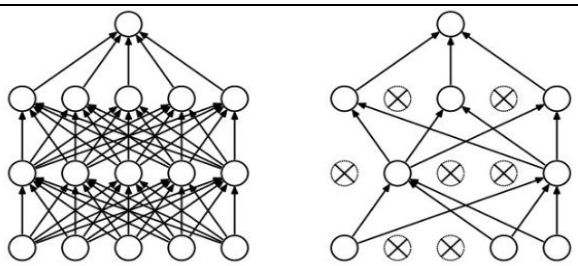
$a^{[l]}$ = np.multiple($a^{[l]}$, $d^{[l]}$)

$a^{[l]}$ /= keep_prob

$z^{[l+1]} = w^{[l+1]}a^{[l]}+b^{[l+1]}$

Test:

Do Nothing



(a) Standard Neural Net　　(b) After applying dropout.

图 1.20

为什么需要除以$keep_prob$？假设l层有 50 个 units，$keep\_prob=0.8$，大概有 10 个 units 被置 0，$z^{[l+1]}=w^{[l+1]}a^{[l]}+b^{[l+1]}$，$a^{[l]}$的期望相比 dropout 之前减少 20%，为了不影响$z^{[l+1]}$的期望，$a^{[l]}$需要除以 0.8，用来修正或弥补 dropout 掉的 20%。目的是确保即使在测试阶段不执行 Dropout 来调整数值范围，激活函数的预期结果也不会发生变化。（测试阶段不进行 Dropout，因为如果在测试阶段应用 Dropout，每次 Dropout 的神经元都是变化的，预测会受到干扰。）

为什么 Dropout 可以起到正则化作用？由前面可知正则化的本质是减少网络对某些神

经元过分依赖。直观地认识，训练一个网络，每次迭代过程中，任何一个神经元都有可能被 Dropout，所以后一层的神经元输入不会过分依赖前一层的任何一个神经元的输出，不会给前一层任何一个神经元太多权重，而是尽可能的均分到每个神经元上。和前面介绍的 L2 范数有异曲同工之妙。Cannot rely on any one feature,so have to spread out weights

4）其他正则化方法

- Data augmentation

镜像对称（Mirroring）、随机裁剪（Random Cropping）、色彩变换（Color shifting），如图 1.21。色彩变换中 PCA 颜色增强，其含义是：比如图像呈现紫色，即主要含有红色和蓝色，绿色很少，那么 PCA 颜色增强就会对红色和蓝色增减很多，绿色变化相对少一点，所以使总体的颜色保持一致。

图 1.21

- Early stopping

根据损失函数曲线图，在损失函数最小时候，终止网络训练。

1.8 梯度消失与梯度爆炸

1. 激活函数指数级增长或下降

假设训练一个极深的网络，每一层只有两个神经元，如图 1.22 所示。并且假设每层偏置项 $b^{[l]}=0$，激活函数是线性 $g(z)=z$，网络的输出如公式(49)所示。

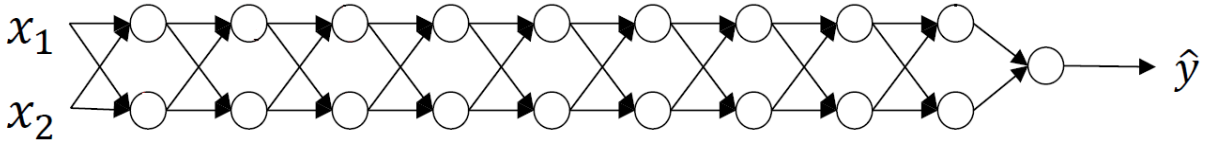


图 1.22

$$\hat{y} = w^{[L]}w^{[L-1]}w^{[L-2]}...w^{[3]}w^{[2]}w^{[1]}x \tag{49}$$

1）假设 $w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \Rightarrow \hat{y} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L} x$，对于很深的网络，$L$ 增大，$\hat{y}$ 呈指数增长，最终 $\hat{y}$ 变得很大。即 $w^{[l]} > I$，激活函数值爆炸式增长。

2）假设 $w^{[l]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \Rightarrow \hat{y} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L} x$，对于很深的网络，$L$ 增大，$\hat{y}$ 呈指数下降，最终 $\hat{y}$ 变得很小。即 $w^{[l]} < I$，激活函数值消失。

2. 梯度消失与爆炸

假设每一层只有一个神经元，$w^{[1]},...,w^{[L]}$ 是权重，激活函数是非线性，网络输出关于 $w^{[1]}$ 的导数如公式(50)。

$$\begin{aligned} a^{[l]} &= g(z^{[l]}) \\ z^{[l]} &= w^{[l]}a^{[l-1]} + b^{[l]} \\ \frac{\partial \hat{y}}{\partial w^{[l]}} &= \frac{\partial \hat{y}}{\partial a^{[L]}} \cdot \frac{\partial a^{[L]}}{\partial z^{[L]}} \cdot \frac{\partial z^{[L]}}{\partial a^{[L-1]}} \cdot ... \cdot \frac{\partial a^{[l]}}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial w^{[l]}} \\ &= \frac{\partial \hat{y}}{\partial a^{[L]}} \cdot g'(z^{[l]}) \cdot w^{[L]} \cdot ... \cdot g'(z^{[l]}) \cdot a^{[l-1]} \end{aligned} \tag{50}$$

权重采用均值为 0 标准差为 1 的高斯分布初始化，那么满足 $\left|w^{[l]}\right| < 1$，如果激活函数采用 sigmoid 函数，当激活值接近 0 或 1 时候，导数值接近于 0，所以出现反向传播时候，越往前梯度值越小，甚至接近于 0，出现梯度消失问题，网络权重无法进行更新。

如果训练过程中，权重的值变得比较大，而 $g'(z^{[l]})$ 适中，那么梯度值会出现激增现象。

1.9 优化算法

1. mini-batch 梯度下降

具体见算法 1.2，讨论 batch-size 的选择。一般 batch-size 选择 2 的指数次，如 64、128、256、512。

1）batch-size = m。变为 batch gradient descent，所有样本作为一个 batch，每迭代一次耗时比较长，需要大内存（一次迭代需要遍历所有训练数据）。

2）batch-size = 1。变为 stochastic gradient descent，每个样本是一个独立的 batch，失去向量化加速，因为每次只处理一个样本，方差大，损失函数震荡严重。

3）1< batch-size < m。向量化加速；不需要处理完所有样本，就可以梯度下降更新权值。

算法 1.2 mini-batch 梯度下降

m=1000*5000 个样本，每批次 1000 个样本。

$$X=\left[\underbrace{x^{(1)},x^{(2)},x^{(3)},...,x^{(1000)}}_{X^{\{1\}}}\mid\underbrace{x^{(1001)},...,x^{(2000)}}_{X^{\{2\}}}\mid.....\mid\underbrace{...,x^{(m)}}_{X^{\{5000\}}}\right]\in R^{n_x\times m}$$

$$Y=\left[\underbrace{y^{(1)},y^{(2)},y^{(3)},...,y^{(1000)}}_{Y^{\{1\}}}\mid\underbrace{y^{(1001)},...,y^{(2000)}}_{Y^{\{2\}}}\mid.....\mid\underbrace{...,y^{(m)}}_{Y^{\{5000\}}}\right]\in R^{1\times m}$$

For t = 1,…,5000

Forward on $X^{\{t\}}$

$Z^{[1]} = w^{[1]}X^{\{t\}}+b^{[1]}$

$A^{[1]}=g^{[1]}(Z^{[1]})$

:

compute:

$$J=\frac{1}{1000}\sum_{i=1}^{1000}l(y^{(i)},\hat{y}^{(i)})+\frac{\lambda}{2\times1000}\sum_{l}\left\|w^{[l]}\right\|_F^2$$

backprop:

$w^{[l]}:=w^{[l]}-\alpha dw^{[l]}$

$b^{[l]}:=b^{[l]}-\alpha db^{[l]}$

2. 指数加权平均

以统计每天温度为例，说明加权平均值，指数加权平均公式如公式(60)所示。

$$V_t=\beta V_{t-1}+(1-\beta)\theta_t \tag{60}$$

其中V_t表示第t天的指数加权平均温度值，近似$\frac{1}{1-\beta}$天的平均温度，θ_t表示第t天的温度值，β表示可调节的超参数值。各数值的加权随时间而指数式递减，越近期的数据加权越重，较旧的数据也给予一定的加权。如图 1.23 所示。

- $\beta=0.9$，表示近似 10 天的平均值，如图中红色曲线；
- $\beta=0.98$，表示近似 50 天的平均值，如图绿色曲线。得到的曲线波动更小，更加平坦些，因为 0.98 的权重给了V_{t-1}，仅有 0.02 权重给了θ_t，这样θ_t对V_t的影响就

小很多了，V_t 相比较 V_{t-1} 不会出现太大的图片，缺点曲线出现右移。

- $\beta = 0.5$，表示近似 2 天的平均值，如图黄色曲线。曲线波动比较大，因为平均数据太少，得到的曲线有更多噪声，更有可能出现异常值，这个曲线能更快地适用温度变化。

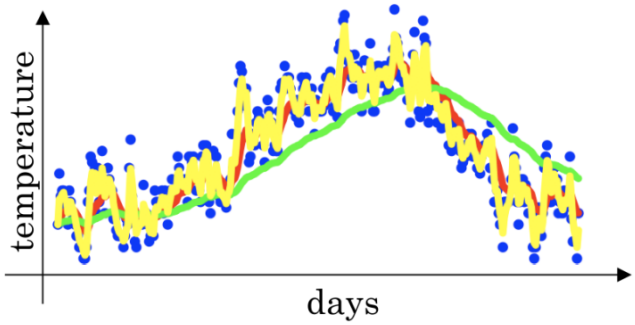


图 1.23

为什么 V_t 近似 $\frac{1}{1-\beta}$ 天的平均温度？我们以 $\beta = 0.9$ 为例说明。

$$
\begin{aligned}
&V_{100} = 0.9V_{99} + 0.1\theta_{100} \\
&V_{99} = 0.9V_{98} + 0.1\theta_{99} \\
&\qquad\vdots \\
&V_1 = 0.9V_0 + 0.1\theta_1 \qquad (61) \\
&\Rightarrow V_{100} = 0.1\theta_{100} + 0.9\left(0.9V_{98} + 0.1\theta_{99}\right) \\
&\quad = 0.1\theta_{100} + 0.1\times 0.9\theta_{99} + \left(0.9\right)^2\left(0.9V_{97} + 0.1\theta_{98}\right) \\
&\quad = 0.1\theta_{100} + 0.1\times 0.9\theta_{99} + 0.1\times\left(0.9\right)^2\theta_{98} + \ldots + 0.1\times\left(0.9\right)^{99}\theta_1
\end{aligned}
$$

通过计算发现 $\left(0.9\right)^{10} \approx 0.35$，10 天后的权重下降 $\frac{1}{3}$，所以近似只关注过去 10 天的加权。

在实际应用中如何计算加权平均？具体见算法 1.3。指数加权平均值的好处是占用极少内存。

算法 1.3 指数加权平均

```
repeat {
        get next  θ_t
        V_θ = βV_θ + (1−β)θ
}
```

3. 指数加权平均的偏差修正

前面介绍计算指数加权平均值的方法存在一个很大的问题，对前面几天温度的估计值要远远小于实际的温度值，我们以 $\beta=0.98$ 为例说明，其中 $V_1=0.98V_0+0.02\theta_1=0.02\theta_1$，$V_2=0.98V_1+0.02\theta_2=0.0196\theta_1+0.02\theta_2$。所以 V_1、V_2 远远小于前两天的温度。引入偏差修正解决预估初期预测不准确的问题，如公式(62)所示。

$$V_t'=\frac{V_t}{1-\beta^t} \tag{62}$$

修正后，$V_1'=\frac{V_1}{1-0.98}=\theta_1$，$V_2'=\frac{0.0196\theta_1+0.02\theta_2}{1-0.98^2}=0.49\theta_1+0.51\theta_2$，并且随着 t 的增加，β^t 接近于 0，当 t 很大，偏差 修正几乎没有作用。所以偏差修正，仅仅帮助更好的预测前期温度值，而不会影响后面的指数加权平均值。

4. 动量梯度下降法

假设图 1.24 是需要优化的损失函数，其中红点就是最优点，如果使用常规梯度下降方法，可能会出现在纵轴上波动比较大，如图中蓝色曲线，这种波动减缓了训练模型的速度。如果使用较大的学习率加速梯度下降，可能出出现偏离函数的范围，如图中紫色曲线。为了避免摆动过大，需要选择较小的学习率，采用 Momentum 梯度下降法，可以在纵向上减少摆动的幅度，横向上加速训练，如图中红色曲线。具体见算法 1.4。

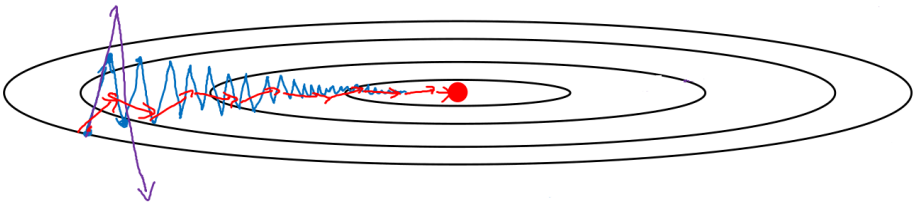


图 1.24

算法 1.4 Momentum 梯度下降法

初始化 $V_{dw}=0,V_{db}=0$

on iteration t:

compute dw,db on current mini-batch

$V_{dw}:=\beta V_{dw}+(1-\beta)dw$

$V_{db}:=\beta V_{db}+(1-\beta)db$

$w:=w-\alpha V_{dw}$

$b:=b-\alpha V_{db}$

1）在纵轴方向上，希望放慢点，如图 1.24 中，纵轴方向梯度方向正负相互变动，所以平均值接近 0，进行指数加权，纵轴梯度幅度变小；

2）横轴方向所有梯度都是指向一个方向，所以横轴进行指数加权，梯度幅度反而可能会增大，加速梯度下降；

3）超参数α控制学习率，β控制指数加权平均数，β通常取 0.9；

4）此处指数加权平均算法不一定需要使用修正偏差，因为经过几次迭代(10 次)的平均值已经超过算法的初始值，不会受算法初始值影响。

5. RMSprop(均方根)

和 Momentum 算法本质一样，都是希望在纵轴方向减少梯度幅度，横轴方向增大梯度幅度，从而达到加速梯度下降的效果。具体见算法 1.5。

算法 1.5 RMSprop 梯度下降法

初始化 $S_{dw}=0, S_{db}=0$

on iteration t:

compute dw,db on current mini-batch

$$S_{dw} := \beta S_{dw} + (1-\beta)(dw)^2$$

$$S_{db} := \beta S_{db} + (1-\beta)(db)^2$$

$$w := w - \alpha \frac{dw}{\sqrt{S_{dw}+\varepsilon}}$$

$$b := b - \alpha \frac{db}{\sqrt{S_{db}+\varepsilon}}$$

1）纵轴方向 d_w 大—> S_{dw} 也比较大—>相对减少梯度值更新；（被一个较大数相除）；

2）横轴方向 d_w 小—> S_{dw} 也比较小—>相对增大梯度值更新；（被一个较小数相除）；

3）ε 防止出现分母为 0，一般取 10^{-8}。

6. Adam 优化算法

基本原理是将 Momentum 算法和 RMSprop 算法结合起来，具体见算法 1.6。

算法 1.6 Adam 优化算法

初始化：$V_{dw}=0, S_{dw}=0, V_{db}=0, S_{db}=0$；学习率$\alpha$需要调试；$\beta_1$缺省值是 0.9；$\beta_2$ 缺省值是 0.999；ε防止分母为 0，一般取值为 10^{-8}。

On iteration t:

compute dw, db using mini-batch

Momentum 算法计算指数加权平均：

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) db$$

RMSprop 算法：

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)(dw)^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2)(db)^2$$

修正偏差：

$$V_{dw}^{correct} = \frac{V_{dw}}{1-\beta_1^t}, \quad V_{db}^{correct} = \frac{V_{db}}{1-\beta_1^t}$$

$$S_{dw}^{correct} = \frac{S_{dw}}{1-\beta_2^t}, \quad S_{db}^{correct} = \frac{S_{db}}{1-\beta_2^t}$$

更新权重：

$$W := W - \alpha \frac{V_{dw}^{correct}}{\sqrt{S_{dw}^{correct}} + \varepsilon}, \quad b := b - \alpha \frac{V_{db}^{correct}}{\sqrt{S_{db}^{correct}} + \varepsilon}$$

7. 学习率衰减

假设使用 mini-batch 梯度下降法，在迭代过程中会有噪声，在最优点附近，可能不会快速精确地收敛，会在最优点摆动，因为使用固定的学习率 α，不同 mini-batch 有不同噪声，致使其不能精确的收敛。但如果初期 α 取相对较大的值，能够学习的相对快些，当达到最优点附近时，减少 α 的值，以更精细的学习率去寻找最优点，如图 1.25 所示。

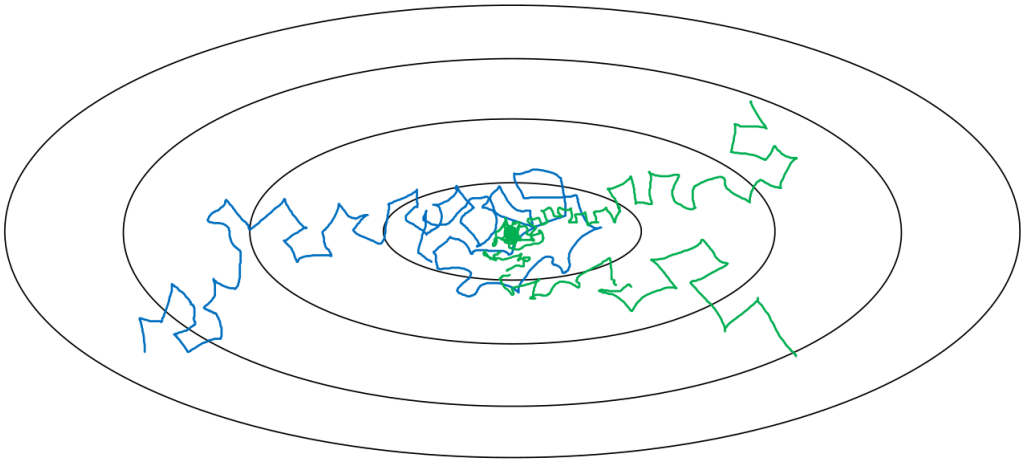


图 1.25

有以下几种衰减方法，epoch_num 表示遍历整个训练数据次数，一个 epoch 表示遍历一次所有训练数据，α_0 表示初始化学习率，decay_rate 表示衰减系数。

1） $\alpha=\dfrac{1}{1+decay\_rate\times epoch\_num}\alpha_0$

2） $\alpha=\left(decay\_rate\right)^{epoch\_num}\alpha_0$

3） $\alpha=\dfrac{k}{\sqrt{epoch\_num}}\alpha_0$，其中 k 为常数

4） $\alpha=\dfrac{k}{\sqrt{t}}\alpha_0$，其中 t 表示 mini-batch 的标记数字

5）随 t 离散下降

1.10 超参数调试

1. 调试处理

根据参数重要性，划分为 3 个等级。

Level 1： α 学习率；

Level 2：β (0.9，Momentum)、$\beta_1,\beta_2,\varepsilon$ (0.9，0.99，10^{-8},Adam)、mni-batch size、hidden units；

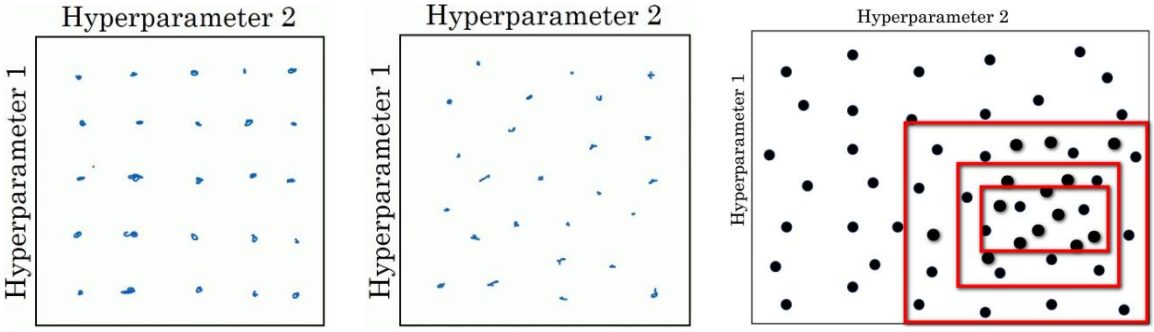
Level 3： layers、 learning rate decay。

一般选取参数的方法：

精确选择：适用参数较少的时候，对于参数较多的深度学习，不太适用，如图 1.26(a)；

随机选择：如图 1.26(b)；

粗糙到精细：比如你在二维的例子中，你进行了取值，也许你会发现效果更好的某个点，也许这个点周围的其他一些点效果也很好，那么接下来你需要放大这块小区域，然后在其中更密集的随机取值，聚集更多的资源，在这个红色的方格中进行搜索,然后逐渐缩小范围，直到到达一个满意的取值，如图 1.26(c)。



(a) (b) (c)

图 1.26

2. 超参数合适范围

1）hidden units

假设范围是 50~100，平均随机从 50~100 中选择数值即可。

2）layers

假设范围是 2~4，平均随机从 2~4 中选择数值即可。

3）学习率 α

假设范围是 0.0001~1，如果还是平均随机的取值，那么结果在 0.1~1 范围的概率是 90%，只有 10%的概率会落在 0.0001~0.1。所以这里采用对数标尺搜索超参数空间更加合理。

$$r = -4 \times np.random.rand() \leftarrow [-4, 0]$$
$$\alpha = 10^r$$
$$\begin{cases} 10^a, ..., 10^b \\ r \in [a, b] \\ \alpha = 10^r \end{cases}$$

4）加权平均值超参数 β

假设范围是 0.9~0.999，$\beta = 0.9$ 表示 10 天的平均值，$\beta = 0.999$ 表示 1000 天的平均值，$\beta = 0.9995$ 表示 2000 天的平均值，当 β 值接近 1 的时候，变化灵敏度越大，所以取值需要更精细。

$$1 - \beta \in [10^{-1}, 10^{-3}]$$
$$r \in [-1, -3]$$
$$1 - \beta = 10^r$$
$$\beta = 1 - 10^r$$

1.11 Batch Normalization

1. what is BN

顾名思义就是“批归一化”，对于训练数据一般都会进行归一化处理，即零均值化和归一化方差。如图 1.27 所示，原始数据在不同维度上的特征尺度不一致，例如 *b* 的取值范围是 1~1000，*w* 的取值范围是 0~1，等高线是一系列很狭长的椭圆（仅有两个特征维度），在这样特征分布不均匀的数据集上运行梯度下降法，必须使用相对较小的学习率，导致梯度下降非常缓慢，如果使用较大学习率，在 *w* 维度方向上梯度下降会出现较大的

摆动。如果经过归一化处理，不同维度有相同的特征的尺度，那么得到的等高线是一系列同心圆，无论从哪个位置，梯度下降法都能够更直接快速地找到最小值，并且可以使用相对较大的学习率。所以如果原始数据不同特征值取值范围差异很大，那么归一化就很重要了，可以加速网络训练的收敛速度。

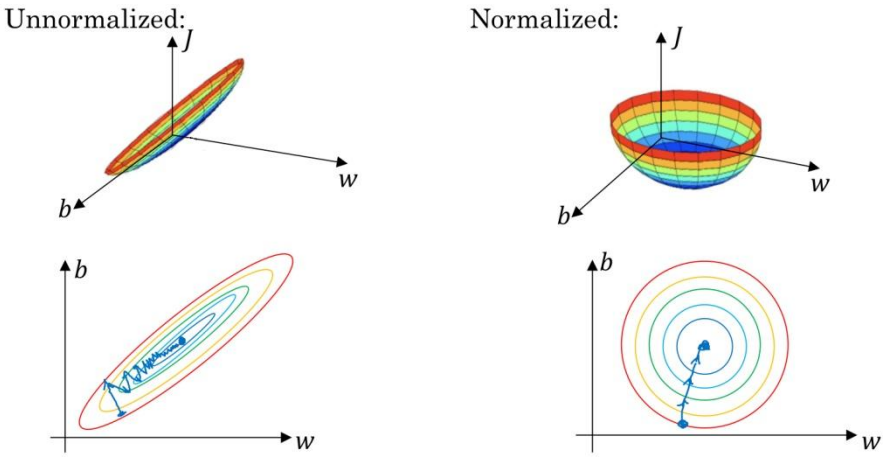


图 1.27

其实对于深度神经网络来说，每个隐层的神经元就是下一层的输入，但随着网络的加深或者在训练过程中，隐层的神经元的值发生不同的偏移或者变动，导致不同特征有不同的特征尺度，而且差异会随着网络深度增大而增大，导致深度网络很难训练。那么是不是可以对隐层的神经元也进行归一化？但是如果仅仅归一化处理，那么所有的隐层神经元都是 0 均值，方差也都为 1，严重地破坏了每层学习到的特征。举个例子，假设网络中采用 Sigmoid 激活函数，你强制把每层学到的特征进行归一化处理，也就是把 Sigmoid 激活函数的大部分输入限制在[-1,1]范围，我们知道 Sigmoid 函数在[-1,1]范围类似于一个线性函数，这就意味着，每层 Sigmoid 激活函数都类似线性函数，导致无论网络多深都类似一层线性网络。BN 引入学习参数 γ,β 进行变换重构，通过这两个参数去寻找一个平衡点，即达到缩小特征尺度的差异，又不会破坏每个隐层的神经元学到的特征。通过调整 γ,β 的值可以改变归一化后隐层神经元的特征分布。

在训练阶段，可以对 mini-batch 求均值和方差，但是测试阶段只有一个输入样本，怎样获取均值和方差呢？训练阶段每一个 mini-batch 都会计算一组 $\left(\mu,\sigma^2\right)$，可以采用指数加权平均来估算均值和方差，以供测试阶段 BN 使用。

怎样将 BN 加入神经网络？计算 l 层的 $Z^{[l]}$ 之后，进行 BN 操作，由 $\gamma^{[l]},\beta^{[l]}$ 两个参数控制 BN 学习 $\tilde{z}^{[l]}$，然后再将 $\tilde{z}^{[l]}$ 输入到激活函数得到 $a^{[l]}$。BN 详细过程见算法 1.7。

算法 1.7 Batch Normalization

Train：

对第 l 层的网络进行 BN，一个 mini-batch 有 m 个样本。

Input： $z^{(1)},...,z^{(m)}$， parameters γ,β to be learned， $\bar{\mu}_0=0,\bar{\sigma}_0{}^2=0$

$$\mu=\frac{1}{m}\sum_{i=1}^{m}z^{(i)}$$
$$\sigma^2=\frac{1}{m}\sum_{i=1}^{m}\left(z^{(i)}-\mu\right)^2$$
$$z_{norm}^{(i)}=\frac{z^{(i)}-\mu}{\sqrt{\sigma^2+\varepsilon}}$$
$$\tilde{z}^{(i)}=\gamma z_{norm}^{(i)}+\beta$$

$$\bar{\mu}:=\beta_1\bar{\mu}+\left(1-\beta_1\right)\mu$$
$$\bar{\sigma}^2:=\beta_1\bar{\sigma}^2+\left(1-\beta_1\right)\sigma^2$$

Test:

对第 l 层的网络进行 BN，只有一个测试样本

$$z_{norm}=\frac{z-\bar{\mu}}{\sqrt{\bar{\sigma}^2+\varepsilon}},\tilde{z}=rz_{norm}+\beta$$

Adding BN to network:

$$X^{\{1\}}\xrightarrow{w^{[1]},b^{[1]}}z^{[1]}\xrightarrow{\beta^{[1]},\gamma^{[1]}}\tilde{z}^{[1]}\longrightarrow g^{[1]}\left(\tilde{z}^{[1]}\right)\longrightarrow a^{[1]}\longrightarrow...$$

Parameters: $w^{[l]},b^{[l]},\beta^{[l]},\gamma^{[l]}$

$z^{[l]}=w^{[l]}a^{[l]}+b^{[l]}$，对 $z^{[l]}$ 均值化，所以 $b^{[l]}$ 被消除，不需要学习 $b^{[l]}$。

For t = 1,…,num mini-batch:

Compute forward prop on $X^{\{t\}}$

In each hidden layer use BN to replace $z^{[l]}$ with $\tilde{z}^{[l]}$

Use backprop to compute $dw^{[l]},d\beta^{[l]},d\gamma^{[l]}$

Update parameters:

$$w^{[l]}:=w^{[l]}-\alpha dw^{[l]},\beta^{[l]}:=\beta^{[l]}-\alpha d\beta^{[l]},\gamma^{[l]}:=\gamma^{[l]}-\alpha d\gamma^{[l]}$$

其中 γ,β 可以通过梯度下降算法更新， $z_{norm}^{(i)}$ 每个分量都是 0 均值和方差 1，有时候不希

望所有的隐藏神经元输出都是 0 均值和方差 1，也许有不同的分布会更有意义，所以需要去学习 γ,β 参数，有了这两个参数，可以随意设置 $\tilde{z}^{(i)}$ 的值，比如 $\gamma=\sqrt{\sigma^2+\varepsilon},\beta=\mu$，得 $\tilde{z}^{(i)}=z^{(i)}$。

2. why does BN work

1）对隐层神经元归一化，减少不同特征间的特征尺度差异，加速学习；

2）如果已经学习了 x 到 y 的映射，若 x 的分布发生改变，那么可能需要重新训练网络（例如已经学习黑猫分类器，当应用于其他颜色猫的时候，可能效果不是很好）。这种数据分布变化也称为 covariate shift。

如图 1.28 所示，从第三层隐藏层角度看，它从前层网络取得一些值，希望使输出值 $\hat{y}$ 接近真实值 y。但是当前面 $w^{[1]},b^{[1]},w^{[2]},b^{[2]}$ 的值改变，$a^{[2]}$ 也会改变，从第三层角度看，前面第二层的隐藏单元的值在不断地改变，所以就有了 covariate shift 的问题，BN 可以保证无论 $z^{[2]}$ 怎样变化，它的均值和方差保持不变，限制了前层网络参数更新影响 $z^{[2]}$ 数值的分布程度。从第三层角度看，BN 减少了输入值改变的问题，确保这些值变得更稳定，后层网络就有了更坚实的基础。即使输入分布改变了，$z^{[2]}$ 会改变更少，前层保持学习，当层改变时，迫使后层适应的程度减少。也就是削弱前层参数的作用与后层参数作用之间的联系，使得网络每层都可以自己学习，稍稍独立于其他层，有助于加速整个网络的学习。每层神经元的值不会左右移动太多，因为被均值和方差所限制，使得后层的学习工作变得更容易些。

3）正则化作用

- 在 mini-batch 上计算均值和方差，而不是在整个数据集上，均值和方差有一些小噪声；
- $z^{[l]}\rightarrow\tilde{z}^{[l]}$ 过程也会引入噪声，因为是用有噪声的均值和方差计算得出的，所以有点类似 Dropout，往每个隐藏层的激活值上增加噪声，这迫使后面的隐藏单元不会过分依赖前面任何一个隐藏单元。加的噪声很微小，所以没有巨大的正则化效果，一般仍需要将 BN 和 Dropout 一起使用；
- 应用较大的 mini-batch size 可以有效地减少计算均值和方差引入的噪声，因此可以减少正则化效果。

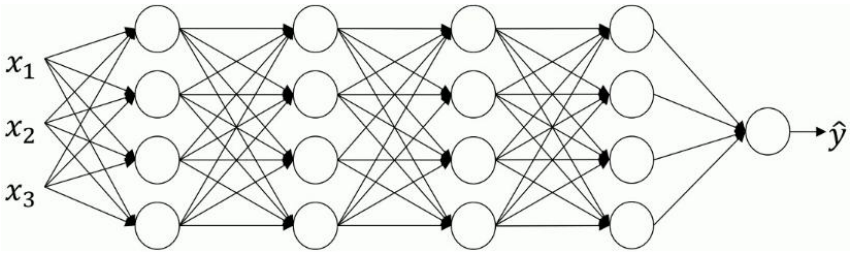


图 1.28

二、卷积神经网络

2.1 卷积

卷积层的输出神经元只和部分输入层神经元连接，相同响应图内，不同空间位置共享卷积核参数，因此卷积层可以大大降低学习参数数量。我们以具体例子介绍卷积，卷积有 valid 和 same 两种方式，valid 也称为丢弃方式，same 也称补齐方式。

1. 步长为 1 的 valid 方式

1）如图 2.1 所示，一个 5×5 的输入矩阵和一个 3×3 的卷积核进行卷积。首先：卷积核的元素和输入矩阵左上角 3×3 区域的元素对应相乘，然后相加，得到输出矩阵左上角的 4 这个元素；然后：卷积核在输入矩阵上向右移动一个方格，与输入矩阵 3×3 的区域元素对应相乘，得到输出矩阵第一行 3 元素。后面依次类推，最终得到 3×3 的输出矩阵。

2）输入矩阵大小为 $n\times n$，卷积核大小为 $f\times f$，得到输出矩阵大小为 $(n-f+1)\times(n-f+1)$

例子中 $n=5, f=3$ 得到输出矩阵为 3。

3）缺点：假设输入的是图像矩阵，每次做卷积，图像就会缩小，经过多次卷积后可能缩小为 1×1，仅仅是一个像素点，把图像的特征信息都丢弃了；边界的像素点，最多只能与卷积核卷积一次。

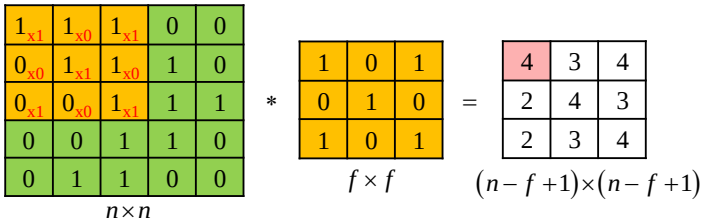


图 2.1

2. 步长为 1 的 Same 方式

在步长为 1 的情况下，输出矩阵的大小与输入的矩阵大小相等，如图如图 2.2 所示。

1）采取补 0 方式，保证输出大小与输入大小相等。输入矩阵大小为5×5，在输入矩阵四周各补一行和一列 0 元素，变为7×7的矩阵，卷积方法和上面介绍的相同，得到一个5×5的输出矩阵。

2）假设补齐 p 行和 p 列元素，输入矩阵的大小变为$(n+2p)\times(n+2p)$，得到输出矩阵的大小为$(n+2p-f+1)\times(n+2p-f+1)$，要求输出矩阵的大小与输入矩阵的大小相等，即$n+2p-f+1=n$所以可得：$p=\dfrac{f-1}{2}$。

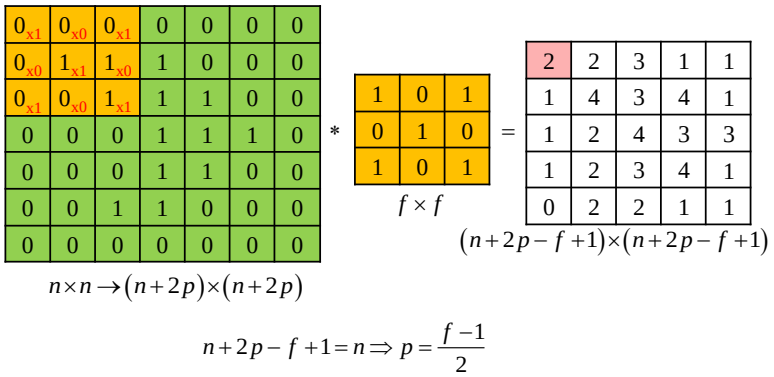


图 2.2

3. 步长大于 1 的 valid 方式

1）如图 2.3 所示，步长为 2，输入矩阵的大小为5×5，卷积核的大小为3×3，卷积方法和上面介绍相同，不同之处每次移动 2 个方格，得到2×2的输出矩阵。之前说 Valid 是丢弃方式，截止目前并没看到元素被丢掉，只有当步长大于 1，才可能会产生丢弃，如果输入矩阵的大小为5×6，即在例子中矩阵的右侧加一列，当以步长为 2，移动到最后，输入矩阵只剩 2 列元素，小于卷积核的 3 列，即卷积核已经移出图像区域，这时候结束卷积，最右边元素被丢弃。

2）输入矩阵的大小为$n\times n$，卷积核的大小为是 $f\times f$，步长为s，得到输出矩阵的大小为$\left\lfloor\dfrac{n-f}{s}+1\right\rfloor\times\left\lfloor\dfrac{n-f}{s}+1\right\rfloor$，$\lfloor\ \rfloor$表示向下取整，比如$\lfloor1.7\rfloor=1$。

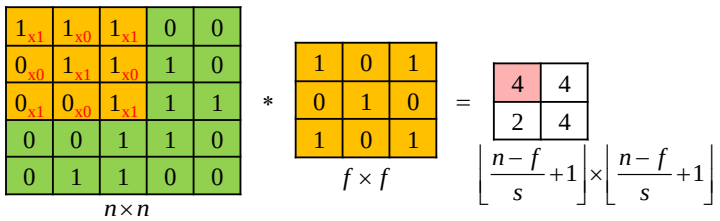


图 2.3

4. 步长大于 1 的 same 方式

1）如图 2.4 所示，输入矩阵的大小为 5×5，在输入矩阵四周各补一行和一列 0 元素，变为 7×7 的矩阵，卷积核的大小为 3×3，步长为 2，得到 3×3 的输出矩阵。

2）输入矩阵的大小为 $n\times n$，补齐 p 行和 p 列元素，卷积核是 $f\times f$，步长为 s，得到输出矩阵的大小为 $\left\lfloor\frac{n+2p-f}{s}+1\right\rfloor\times\left\lfloor\frac{n+2p-f}{s}+1\right\rfloor$，这里 $p=\frac{f-1}{2}$。

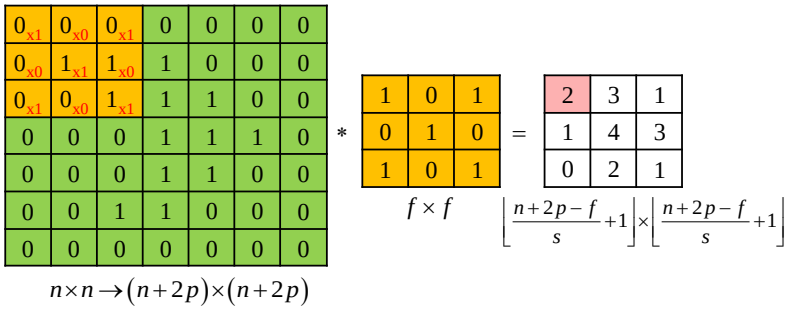


图 2.4

5. 多通道图像卷积

假设 input 是一个 mini-batch 的多通道图像，$[batch,\ in\_height,\ in\_width,\ in\_channels]$，具体含义是 [训练时一个 batch 的图片数量，图片高度，图片宽度，图像通道数]；卷积核 filter 的大小为 $[filter\_height,\ filter\_width,\ in\_channels,\ out\_channels]$，具体含义是 [卷积核的高度，卷积核的宽度，图像通道数，卷积核个数]，要求类型与参数 input 相同，filter 第三维 $in\_channels$ 的大小要等于 input 第四维 $in\_channels$ 的大小。如图 2.5。

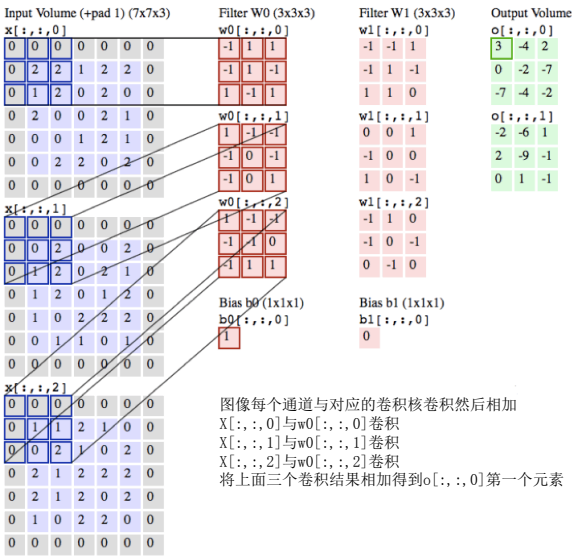


图 2.5

2.2 池化

卷积获取局部特征后，池化技术统计某个区域的特征（平均值或最大值）代表这个区域，可以减少训练参数。如图 2.6 所示，输入是一个4×4的矩阵，采用2×2的池化区域，选取最大值，步长为 2，得到输出矩阵的大小为2×2。池化有点类似卷积，区别在于池化不需要学习任何参数，只要确定池化区域 f 和步长 s，在池化区域内取最大值或者平均值。池化也具有类似卷积的 valid 和 same 两种方式，输入矩阵的大小，池化区域的大小、步长、输出矩阵的大小四者之间的关系和卷积一样。

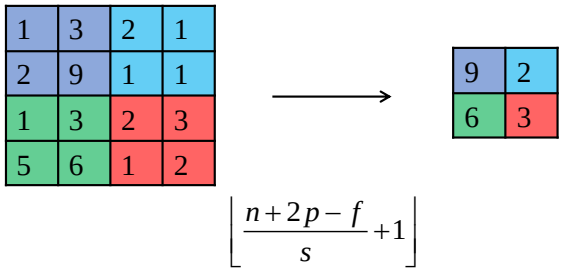


图 2.6

2.3 LeNet-5

最早的卷积神经网络之一，如图 2.7 所示，共有 7 层网络，总共包含约 6 万个参数，下面进行一层一层的进行分析：

1. 第一层 C1 卷积层

输入$32\times32\times1$，6 个5×5的卷积核，步长$strides=1$，valid 的方式，输出$28\times28\times6$。如图 2.8(a)所示，每个卷积核有5×5个权重和 1 个偏置项，总共有$6\times(5\times5+1)=156$个训练参数。连接数为$156\times(28\times28)=122304$。

2. 第二层 S2 池化层

有点区别于现在的池化技术，不是简单地取最大值，如图 2.8(b)，具体做法是：将2×2区域内的值相加，然后乘以一个训练参数，再加上一个偏置项，最后通过 sigmiod 激活函数非线性处理。

输入$28\times28\times6$，池化区域为2×2，移动步长为 2，valid 的方式，输出$14\times14\times6$。训练参数为$6\times(1+1)=12$，连接数为$6\times(4+1)\times(14\times14)=5880$。

3. 第三层 C3 卷积层

这一层很有意思，作者来一套组合拳，把 S2 中 6 个特征图进行不同组合作为卷积层的输入，这样做的好处是不对称的组合方式有利于提取组合特征，如图 2.8(c)、2.8(e)所示。C3 层中前 6 个特征图是以 S2 中 3 个相邻的特征图作为输入；C3 层中紧接着 6 个特征图以 S2 中 4 个相邻的特征图作为输入；C3 层中再紧接着 3 个特征图以 S2 中 4 个不相邻的特征图作为输入；C3 层中最后一个特征图以 S2 中所有特征图作为输入。

输入$14\times14\times6$，16 个5×5的卷积核，步长$strides=1$，valid 的方式，输出$10\times10\times16$。训练参数为$6\times(3\times5\times5+1)+6\times(4\times5\times5+1)+3\times(4\times5\times5+1)+1\times(6\times5\times5+1)=1516$，连接数为$1516\times(10\times10)=151600$。

4. 第四层 S4 池化层

输入$10\times10\times16$，池化区域为2×2，移动步长为 2，valid 的方式，输出$5\times5\times16$。训练参数为$16\times(1+1)=32$，连接数为$16\times(4+1)\times(5\times5)=2000$。

5. 第五层 C5 卷积层

输入$5\times5\times16$，120 个5×5的卷积核，步长$strides=1$，valid 的方式，输出$1\times1\times120$。训练参数为$120\times(16\times5\times5+1)=48120$，连接数$48120\times(1\times1)=48120$。

6. 第六层 F6 全连接层

输入120，输出84，然后再通过 $f(a)=A\tanh(Sa)$ 处理，训练参数$(120+1)\times84=10164$。

7. 第七层 OUTPUT 全连接层

输入 84，输出 10，对应 0~9 的 10 个数字，训练参数$84\times10=840$。最后一层采用 RBF 函数 $y_i=\sum_j\left(x_j-w_{ij}\right)^2$，$x_j$为 F6 层输出，共 84 个神经元，$y_i$对应 10 个数字分类。目前最后一层普遍采用 softmax 作为分类器，RBF 没有 softmax 那样拥有很好的概率特性。

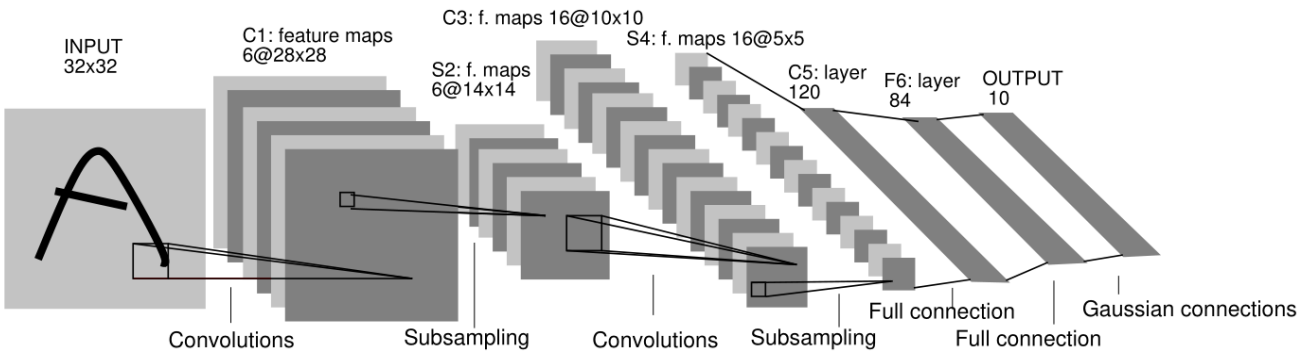
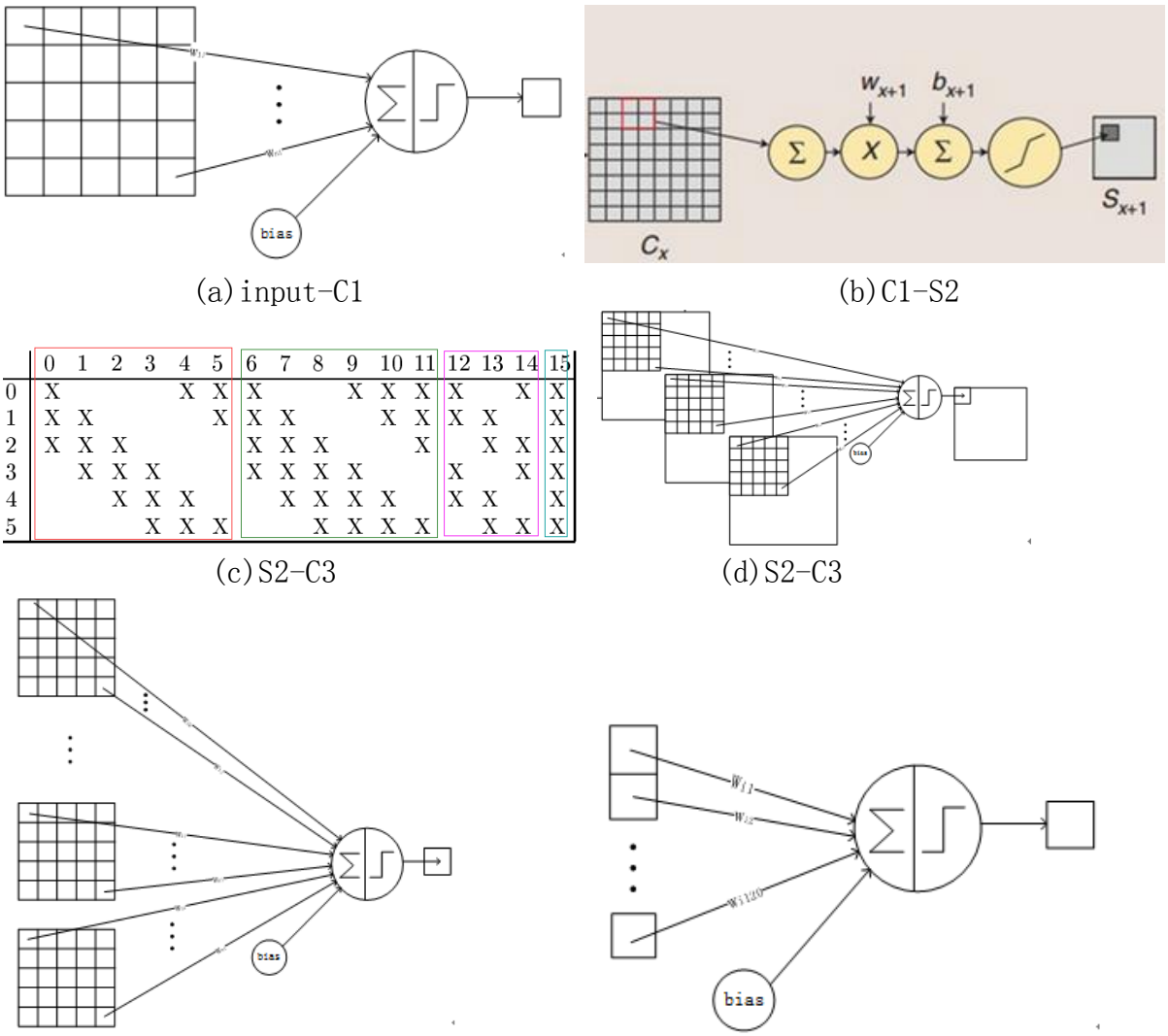


图 2.7



(a) input-C1　　(b) C1-S2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

(c) S2-C3　　(d) S2-C3

(e) S4-C5　　(f) C5-C6

图 2.8

2.4 AlexNet

AlexNet 是具有历史意义的一个网络结构，在此之前，深度学习已经沉寂了很久。2012 年，AlexNet 在当年的 ImageNet 图像分类竞赛中，top-5 错误率比上一年的冠军下降了十个百分点，而且远远超过当年的第二名。网络结构如图 2.10 所示，包含五层卷积层和三层全连接层，总共包含约 6000 万个参数。下面详细介绍 AlexNet 网络：

1. ReLu 激活函数

可参考 1.5 部分内容

2. Data Augmentation

可参考 1.7 部分内容，AlexNet 具体做法是：

1）Random Cropping：从 256×256 的图像中随机裁剪 224×224（包括 Mirroring 图像），相当于样本量增加了 $2\times(256-224)^2=2048$ 倍；

2）Color shifting：采用 PCA 方法处理整个 ImageNet 数据集合，将每个素的 RGB 值 $\left[I_{xy}^R,I_{xy}^G,I_{xy}^B\right]^T$ 加上 $[p_1,p_2,p_3][\alpha_1\lambda_1,\alpha_2\lambda_2,\alpha_3\lambda_3]^T$，其中 p_i 和 λ_i 是 3×3 的 RGB 协方差矩阵的第 i 个特征向量和特征值，α_i 是服从均值为 0 方差为 1 的标准正态分布的随机数。简单说明下怎么求解协方差矩阵的，假设训练集有 m 张图片，每张图片大小为 $w\times h$，我们将 m 张图片转化为 $\mathrm{X}\in R^{(m\times w\times h,3)}$ 的二维向量，三列分别表示 RGB 三个通道灰度值，对这个二维矩阵求协方差矩阵，可得到 3×3 的 RGB 协方差矩阵 C，见公式 63，其中 x_i 表示第 i 列的向量，x_j 表示第 j 列的向量。

$$C=\begin{bmatrix}\mathrm{cov}(\mathrm{x}_1,\mathrm{x}_1) & \mathrm{cov}(\mathrm{x}_1,\mathrm{x}_2) & \mathrm{cov}(\mathrm{x}_1,\mathrm{x}_3)\\ \mathrm{cov}(\mathrm{x}_2,\mathrm{x}_1) & \mathrm{cov}(\mathrm{x}_2,\mathrm{x}_2) & \mathrm{cov}(\mathrm{x}_2,\mathrm{x}_3)\\ \mathrm{cov}(\mathrm{x}_3,\mathrm{x}_1) & \mathrm{cov}(\mathrm{x}_3,\mathrm{x}_2) & \mathrm{cov}(\mathrm{x}_3,\mathrm{x}_3)\end{bmatrix} \tag{63}$$
$$\mathrm{cov}(\mathrm{x}_i,\mathrm{x}_j)=E\left[\left(\mathrm{x}_i-E\left[\mathrm{x}_i\right]\right)\left(\mathrm{x}_j-E\left[\mathrm{x}_j\right]\right)^T\right]$$

3. Dropout

可参考 1.7 部分内容

4. 局部归一化 LRN

$$b_{x,y}^i=\frac{a_{x,y}^i}{\left(k+\alpha\sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)}\left(a_{x,y}^j\right)^2\right)^\beta} \tag{64}$$

$a_{x,y}^i$ 表示第 i 个卷积核在 (x,y) 位置激活值，n 表示相邻的 n 个卷积核，N 表示这一层卷积核的数量，$k=2,n=5,\alpha=0.0001,\beta=0.75$。如图 2.9 所示，选取一个位置，在第三维度上穿过，可以得到 10 个相邻的元素，然后利用公式(64)进行归一化。后面研究发现 LRN 作用不大，所以后面看到的网络一般不采用 LRN。

图 2.9

5. 网络结构

第一层 卷积层

- Conv：输入的 shape 为 $227\times227\times3$，96 个 11×11 的卷积核，步长 $s=4$，*Valid* 方式，输出的 shape 为 $55\times55\times96$，其中 $\left\lfloor\frac{227-11}{4}+1\right\rfloor=55$。
- LRN：对卷积结果进行局部归一化。
- Max-Pool：输入的 shape 为 $55\times55\times96$，池化参数 $f=3,s=2$，*Valid* 方式，输出的 shape 为 $27\times27\times96$，其中 $\left\lfloor\frac{55-3}{2}+1\right\rfloor=27$。

第二层 卷积层

- Conv：输入的 shape 为 $27\times27\times96$，256 个 5×5 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $27\times27\times256$，其中 *padding*：$\frac{5-1}{2}=2$，$\left\lfloor\frac{27+2\times2-5}{1}+1\right\rfloor=27$。
- LRN：对卷积结果进行局部归一化。
- Max-Pool：输入的 shape 为 $27\times27\times256$，池化参数 $f=3,s=2$，*Valid* 方式，输出的 shape 为 $13\times13\times256$，其中 $\left\lfloor\frac{27-3}{2}+1\right\rfloor=13$。

第三层 卷积层

- Conv：输入的 shape 为 $13\times13\times256$，384 个 3×3 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $13\times13\times384$，其中 *padding*：$\frac{3-1}{2}=1$，$\left\lfloor\frac{13+2\times1-3}{1}+1\right\rfloor=13$。

第四层 卷积层

- Conv：输入的 shape 为 $13\times13\times384$，384 个 3×3 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $13\times13\times384$，其中 *padding*：$\frac{3-1}{2}=1$，$\left\lfloor\frac{13+2\times1-3}{1}+1\right\rfloor=13$。

第五层 卷积层

- Conv：输入的 shape 为 $13\times13\times384$，256 个 3×3 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $13\times13\times256$，其中 *padding*：$\frac{3-1}{2}=1$，$\left\lfloor\frac{13+2\times1-3}{1}+1\right\rfloor=13$。
- Max-Pool：输入的 shape 为 $13\times13\times256$，池化参数 $f=3,s=2$，*Valid* 方式，输出的

shape 为 $6\times6\times256$，其中 $\left\lfloor \frac{13-3}{2}+1 \right\rfloor = 6$。

第六层 全连接层

- FC：$6\times6\times256 \rightarrow 4096$
- Dropout：$keep\_prop = 0.5$

第七层 全连接层

- FC：$4096 \rightarrow 4096$
- Dropout：$keep\_prop = 0.5$

第八层 全连接层

- FC：$4096 \rightarrow 1000$

当时受限于 GPU 的计算能力，作者用两个 GPU 训练网络，第一层每个 GPU 负责 48 个卷积核的卷积计算；第二层每个 GPU 负责 128 个卷积核的卷积计算，输入仅来自单个 GPU 第一层卷积输出；第三层每个 GPU 负责 192 个卷积核的卷积计算，输入来自 2 个 GPU 第二层卷积输出；第四层每个 GPU 负责 192 个卷积核的卷积计算，输入仅来自单个 GPU 第三层卷积输出；第五层每个 GPU 负责 128 个卷积核的卷积计算，输入仅来自单个 GPU 第四层卷积输出；后面几层是全连接层。

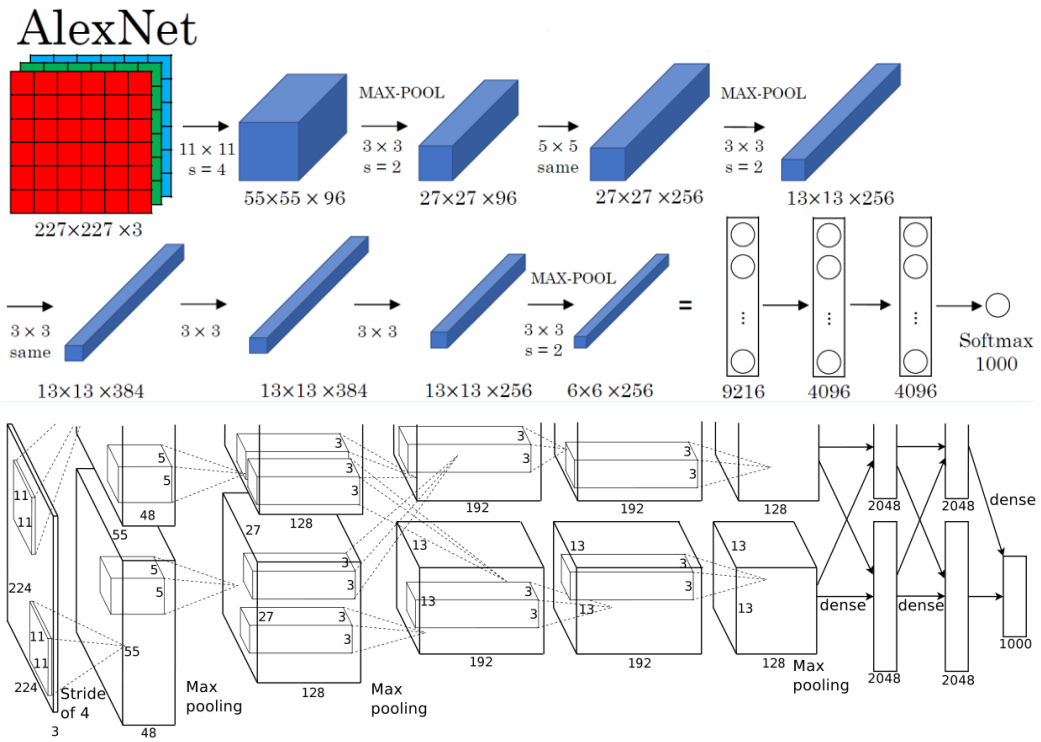


图 2.10

2.5 VGG-16

2014 年 ImageNet 分类挑战赛中取得第二名成绩，VGG-16 是指网络有 16 层，它是一种专注于构建卷积层的简单网络，总共包含约 1.38 亿个参数。整个网络中都使用比较小的卷积核，CONV： $f=3, s=1, Same$，特征图每次卷积后大小不变，只有深度改变。MAX-POOL： $f=2, s=2$，每次池化后，特征图缩小一半，深度不变。如图 2.11 所示。

1. 1-2 层卷积层

- Conv：输入的 shape 为 $224\times224\times3$，64 个 3×3 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $224\times224\times64$。
- Max-Pool：输入的 shape 为 $224\times224\times64$，$f=2, s=2$，输出的 shape 为 $112\times112\times64$。

2. 3-4 层 卷积层

- Conv：输入的 shape 为 $112\times112\times64$，128 个 3×3 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $112\times112\times128$。
- Max-Pool：输入的 shape 为 $112\times112\times128$，$f=2, s=2$，输出的 shape 为 $56\times56\times128$。

3. 5-7 层 卷积层

- Conv：输入的 shape 为 $56\times56\times128$，256 个 3×3 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $56\times56\times256$。
- Max-Pool：输入的 shape 为 $56\times56\times256$，$f=2, s=2$，输出的 shape 为 $28\times28\times256$。

4. 8-10 层 卷积层

- Conv：输入的 shape 为 $28\times28\times256$，512 个 3×3 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $28\times28\times512$。
- Max-Pool：输入的 shape 为 $28\times28\times512$，$f=2, s=2$，输出的 shape 为 $14\times14\times512$。

5. 11-13 层 卷积层

- Conv：输入的 shape 为 $14\times14\times512$，512 个 3×3 的卷积核，步长 $s=1$，*Same* 方式，输出的 shape 为 $14\times14\times512$。
- Max-Pool：输入的 shape 为 $14\times14\times512$，$f=2, s=2$，输出的 shape 为 $7\times7\times512$。

8. 14 层 全连接层

- FC：$7\times7\times512\rightarrow4096$

8. 15 层 全连接层

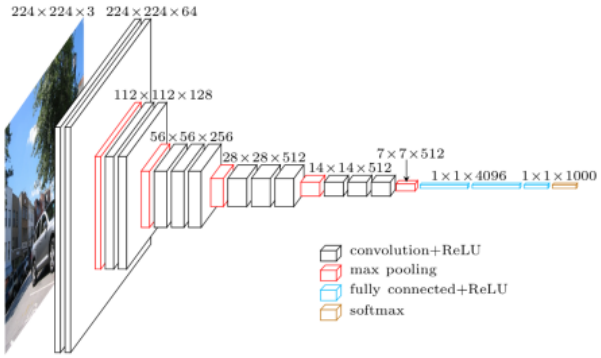
- FC：4096 → 4096

8. 16 层 全连接层

- FC：4096 → 1000

VGG 有一些列网络结构，VGG-16 的表现和 VGG-19 不相上下。在训练 VGG-16 网络时候，作者采用一个巧妙的方法初始化权重，先训练 VGG-11，然后 VGG-16 的前 4 层卷积层和后三层的全连接层权重用训练好的 VGG-11 对应权重初始化，VGG-16 其他层的网络权重随机值初始化。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



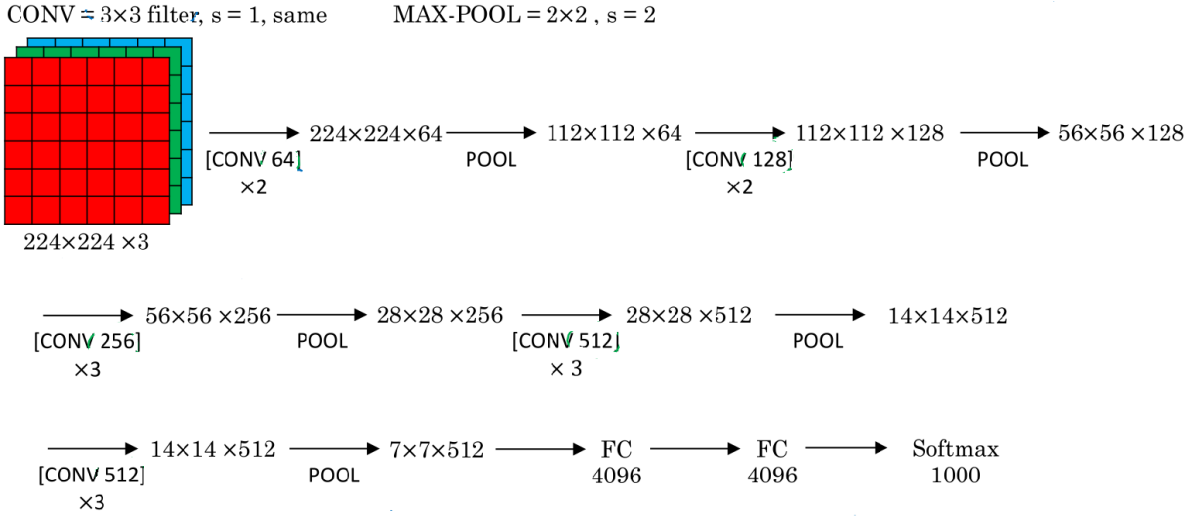


图 2.11

参考：

[1] Neural Networks and Deep Learning http://neuralnetworksanddeeplearning.com/index.html

[2] cs231n http://study.163.com/course/courseMain.htm?courseId=1003223001 http://cs231n.github.io/ https://zhuanlan.zhihu.com/p/21930884

[3] Understanding the Bias-Variance Tradeoff http://scott.fortmann-roe.com/docs/BiasVariance.html

[4] Goodfellow,Bengio,Courville https://github.com/exacity/deeplearningbook-chinese http://www.deeplearningbook.org/

[5]https://mp.weixin.qq.com/s?__biz=MzI1NTE4NTUwOQ==&mid=2650324619&idx=1&sn=ca1aed9e42d8f020d0971e62148e13be&scene=1&srcid=0503De6zpYN01gagUvn0Ht8D#wechat_redirect

[6] Andrew Ng https://mooc.study.163.com/course/2001281002#/info https://www.coursera.org/learn/machine-learning

[7] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift https://arxiv.org/abs/1502.03167

[8] A guide to convolution arithmetic for deep learning https://arxiv.org/abs/1603.07285

[9] UFLDL http://ufldl.stanford.edu/wiki/index.php/UFLDL%E6%95%99%E7%A8%8B

[10] Python Numpy Tutorial http://cs231n.github.io/python-numpy-tutorial/

[11] Andrej Karpathy blog http://karpathy.github.io/

[12] colah blog http://colah.github.io/

[13] Geoffrey Hinton https://www.coursera.org/learn/neural-networks

[14] https://github.com/mattm/simple-neural-network

[15] http://www.cnblogs.com/cloud-ken/

[16] Gradient-based learning applied to document recognition http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

[17] Alexnet ImageNet Classification with Deep Convolutional Neural Networks

[18] https://kratzert.github.io/2017/02/24/finetuning-alexnet-with-tensorflow.html

[19] http://www.cs.toronto.edu/~guerzhoy/tf_alexnet/

[20] Very Deep Convolutional Networks for Large-Scale Image Recognition

[21] http://www.cs.toronto.edu/~frossard/post/vgg16/

[22] 3D 卷积网络 http://scs.ryerson.ca/~aharley/vis/conv/