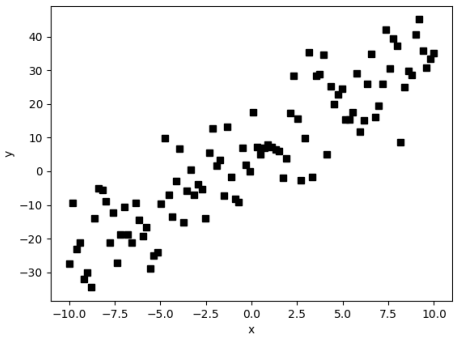
My Road To Deep Learning

Email: Duanyzhi@outlook.com
Github: https://github.com/duanyzhi
Blog: https://duanyzhi.github.io/

一、 线性回归
二、 逻辑回归
三、 CNN
四、 LSTM
五、 Faster-RCNN
6. knn
7. svm(svm 原理、里面的核)
8. 决策树,随机森林
9. k-means
10. em
11. t-sne
12 DNN

线性回归

线性回归(Linear Regression)是机器学习中简单的一种回归算法了。什么是线性回归呢，就是一堆离散的数据满足一定的线性函数关系，最简单的就是一次函数了，当然也可能是含有二次项、三次项等等，这种问题关键是已给的数据肯定是落在这个线性方程周围的。我们想通过这些数据来求出这个线性函数的表达式的方法，就叫做线性回归。因为机器学习中主要的东西其实都是一些算法运算，这里也一样。以一个一次函数为例，下图是一组满足某线性方程的离散点，假设离散点数据集是: $S=\{x^i,y^i\}_{i=1}^m$。 m 表示一共 m 个离散点个数。



线性回归数据散点图

因为线性回归就是一条直线，我们令这条直线是 $h_\theta(x)=\theta_0+\theta_1 x$。这里 θ_0,θ_1 就是我们所要求的变量。那么对于每一个输入 x^i 都会有一个对应的输出 $h_\theta(x^i)$，我们需要的就是比较输出模型 $h_\theta(x^i)$ 和 y^i 的大小，理想的模型输出就是使得这两个数尽量的相等。所以我们做一个 Cost Function，这里前面的系数 $1/2m$ 只是为了归一化用:

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

有了损失函数，我们就可以使用梯度下降法来解决这个问题。我们的目标就变成了最小化这个损失函数。

$$\theta_0,\theta_1=\underset{\theta_0,\theta_1}{\text{minimize}}\,J(\theta_0,\theta_1) \tag{2}$$

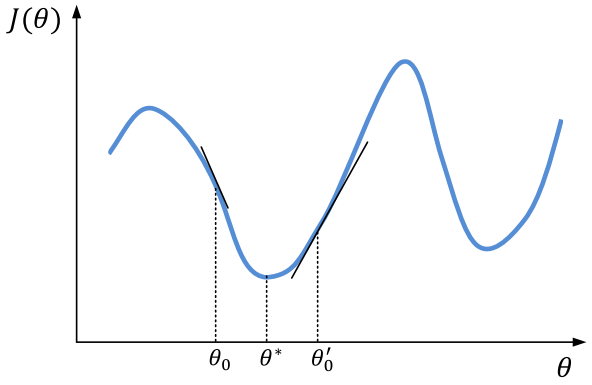
这里为什么是最小化这个损失函数呢，因为上面的损失函数是一个关于 θ 的二次函数，在解决这种多变量问题时可以假设一个变量是常量，那么就变成解决单变量问题的函数了。所以这个损失函数展开后就是一个关于 θ_0 或 θ_1 的二次函数，具有一个最有解点。这里我们就使用梯度下降来求这个点。按照一般方法，对每一个变量求解时固定其他变量，然后对损失函数求导并令导数为 0。即：

$$\frac{\partial}{\partial\theta_j}J(\theta)=0 \quad (\text{for } j=0,1) \tag{3}$$

那么，使用梯度下降求法，重复下面方程直到收敛就可以求出函数值。：

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1) \quad (\text{for } j=0 \text{ and } j=1) \tag{4}$$

其中，α表示学习速率(Learning Rate)，是决定求导时幅度的大小，学习速率过小收敛较慢，学习速率过大会跳过最优点。 后面的偏导数是这一点的二次函数上的导数，用原来的值减去这点的导数值就得到了下一迭代点的值。



梯度下降算法

$$\begin{aligned}
\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) &= \frac{\partial}{\partial\theta_0}\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)^2 \\
&= \frac{\partial}{\partial\theta_0}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0+\theta_1x^i-y^i)^2 \\
&= \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta_0}(\theta_0+\theta_1x^i-y^i)^2 \\
&= \frac{1}{2m}\sum_{i=1}^{m}2(\theta_0+\theta_1x^i-y^i)\frac{\partial(\theta_0+\theta_1x^i-y^i)}{\partial\theta_0} \\
&= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)
\end{aligned} \tag{5}$$

这样就可以得到θ_0的下一次迭代的函数关系式：

$$\theta_0^{new}=\theta_0^{old}-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i) \tag{6}$$

同理得出θ_1的下一次迭代的函数关系式：

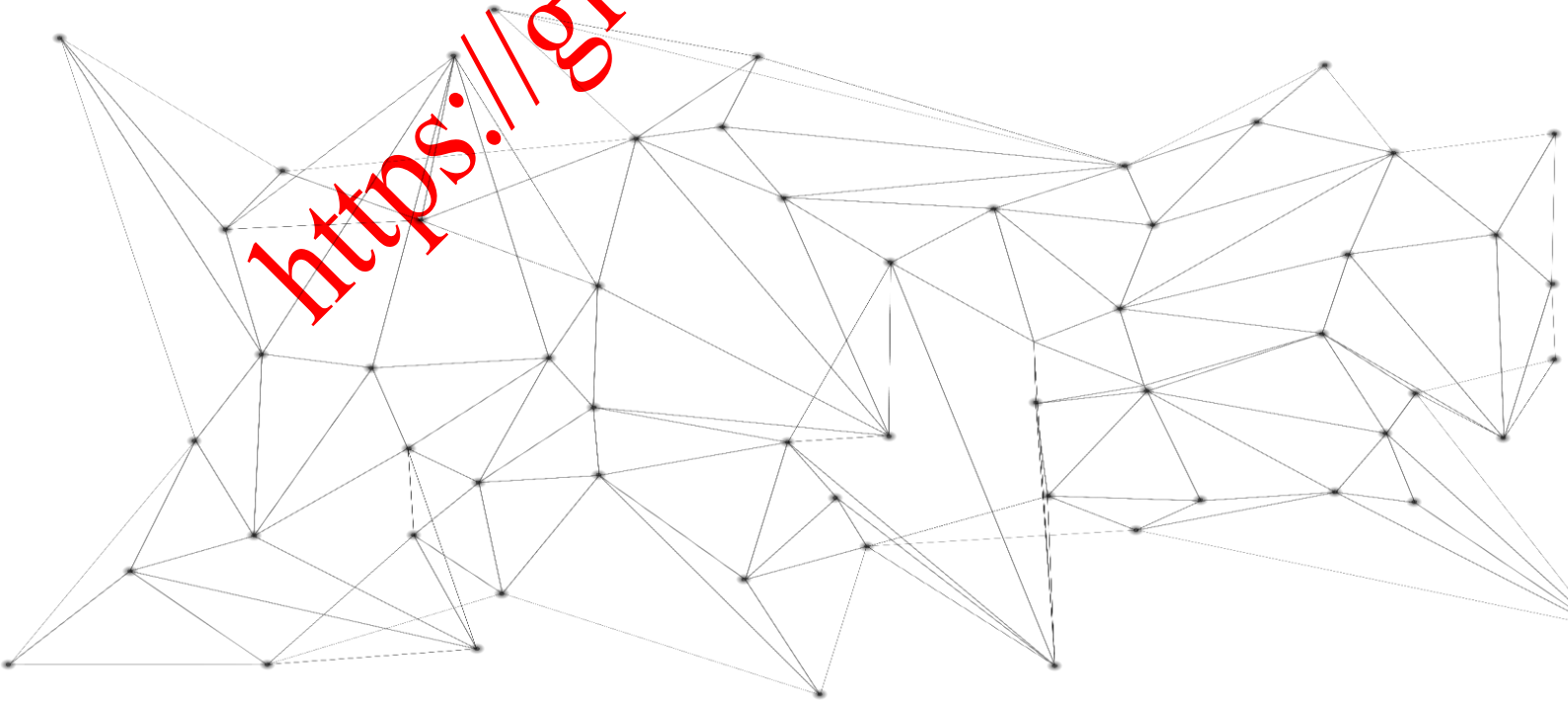
$$\theta_1^{new}=\theta_1^{old}-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)\cdot x^i \tag{7}$$

梯度下降法可写成下面的通式：

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{1}{m} \sum_{i}^{m} (h_\theta(x^i) - y^i) \cdot x_j^i \tag{8}$$

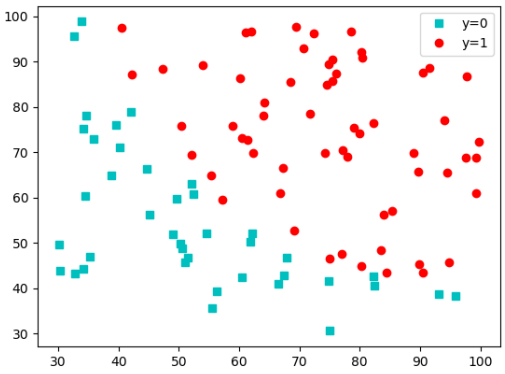
每一次迭代后新的参数用来训练下一个新的数据，这样经过一定次数迭代后，损失函数就会呈现收敛状态。最后收敛时候的参数就是我们需要的参数值。

(https://github.com/duanyzhi/My_Road_To_Deep_Learning/tree/master/linear_regression)



逻辑回归

逻辑回归(Logistic Regression)是一种非线性的回归问题。相比线性回归，逻辑回归所解决的问题中，逻辑回归一般解决分类的问题了。



逻辑回归数据分布

上图中数据是一个典型的二分类形式，我们需要寻找一个逻辑回归线来解决这个分类问题。这里假设数据集是 $S=\{x_1^i,x_2^i,y^i\}_{i=1}^m$，这里数据集中一个数据点对应两个输入 x 和一个输出 y $(\mathrm{y}=0\text{或}\mathrm{y}=1)$。一共有 m 个数据点。其实对于一个逻辑方程来说，我们并不知道有多少个变量存在，一般也无法看出数据和变量之间函数关系。所以这里设置多个变量，让系统自己学习。我们令函数关系是：

$$h_\theta(x_1,x_2)=g(\theta_1x_1+\theta_2x_2+\theta_3x_1^2+\theta_4x_2^2+\theta_5) \tag{2-1}$$

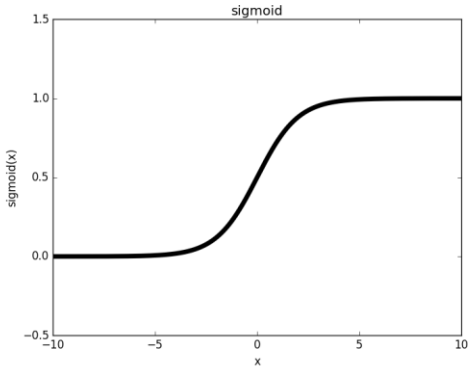
因为这里分布比较简单，输出 h 是关于输入的多项式，其中 g 表示某中函数运算。我们只去了输入的一次方和二次方。对于比较复杂的分布还可以取三次方、开方等运算。逻辑回归判断不同于线性回归。因为输出 $0\le h_\theta(x)\le 1$ 是一个连续的数，标签 y 非 0 即 1，一般满足：

$$\begin{aligned} h_\theta(x)&\ge 0.5 \quad y=1\\ h_\theta(x)&< 0.5 \quad y=0 \end{aligned} \tag{2-1}$$

因为输出分布在[0,1]之间，我们需要引入一个常用非线性函数 Sigmoid Function 来将输入值映射为[0,1]之间的数，这样才能最终和标签比较，公式 2-1 中函数 g 表示为：

$$g(\mathrm{z})=\frac{1}{1+e^{-z}} \tag{2-3}$$

Sigmoid 是最常用的非线性激励函数之一，对应到上面函数关系，$z=\theta_1x_1+\theta_2x_2+\theta_3x_1^2+\theta_4x_2^2+\theta_5$。我们将 sigmoid 图像画出来：

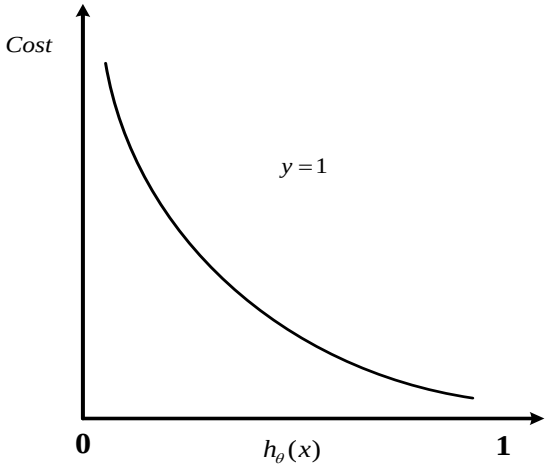


Sigmoid 函数

根据函数关系当 $y=1$ 时，$h_\theta(x)=sigmoid$。这样输入 $z=\theta_1x_1+\theta_2x_2+\theta_3x_1^2+\theta_4x_2^2+\theta_5\geq 0$ 才可以。我们依然继续搭建损失函数，，这里使用第二种常用的损失函数：交叉熵损失函数：

$$Cost(h_\theta(x),y)=\begin{cases}-\log(h_\theta(x)) & \text{if } y=1\\ -\log(1-h_\theta(x)) & \text{if } y=0\end{cases} \tag{2-4}$$

交叉熵损失函数含义是当标签是 1 时就等价于最小化 $-\log(h_\theta(x))$。因为这里 h 在[0,1]之间，此时损失函数和 h 关系如下图所示了，也是一个递减的，并且在 h=1 时最小。这和我们要的一样，即标签是 1 时，输出也是 1。所以可以用这种损失函数来代替线性回归的损失函数。但是这里要注意的时 $y=1,h=1$ 时 $Cost=1$，但 $y=1,h=0$ 是，损失函数接近无穷时不可以的，所以还要 $y=0$ 是另一个损失函数来联合计算，方法一样。



损失函数

下面联合来构造逻辑回归的损失函数，我们由标签只有 0,1 的性质可以得出以下公式：

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}Cost(h_\theta(x^i),y^i)$$
$$=-\frac{1}{m}\sum_{i=1}^{m}[y^i\log h_\theta(x^i)+(1-y^i)\log(1-h_\theta(x^i))] \quad (2\text{-}5)$$

然后求解过程任然是类似与线性回归中的求解方法，直接求导，用梯度下降法来解决。下面对于函数关系，当函数关系是我们假设的公式 2-1 的形式时，我们对其中一个变量求导。

$$\frac{\partial}{\partial\theta_j}J(\theta)=\frac{\partial}{\partial\theta_j}-\frac{1}{m}\sum_{i=1}^{m}[y^i\log h_\theta(x^i)+(1-y^i)\log(1-h_\theta(x^i))]$$
$$=-\frac{1}{m}\sum_{i=1}^{m}[y^i\frac{1}{h_\theta(x^i)}(h_\theta(x^i))'+(1-y^i)\frac{1}{1-h_\theta(x^i)}(1-h_\theta(x^i))']$$
$$=-\frac{1}{m}\sum_{i=1}^{m}[y^i\frac{1}{h_\theta(x^i)}h_\theta(x^i)(1-h_\theta(x^i))z'-(1-y^i)\frac{1}{1-h_\theta(x^i)}(1-h_\theta(x^i))h_\theta(x^i)z'] \quad (2\text{-}6)$$
$$=-\frac{1}{m}\sum_{i=1}^{m}[y^i(1-h_\theta(x^i))-(1-y^i)h_\theta(x^i)]z'$$
$$=-\frac{1}{m}\sum_{i=1}^{m}\{[y^i-h_\theta(x^i)]\frac{\partial}{\partial\theta_j}[\theta_1x_1+\theta_2x_2+\theta_3x_1^2+\theta_4x_2^2+\theta_5]\}$$
$$=\frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^i)-y^i]x_j^i$$

其中系数 m 可以省略，不影响公式收敛性。公式里面的 log 一般时以指数为底的。那么对每个函数的梯度求导公式为：$\theta_j^{new}=\theta_j^{old}-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)\cdot x_j^i$，比如我们对 $\theta_1,\theta_3,\theta_5$ 求导就会得到：

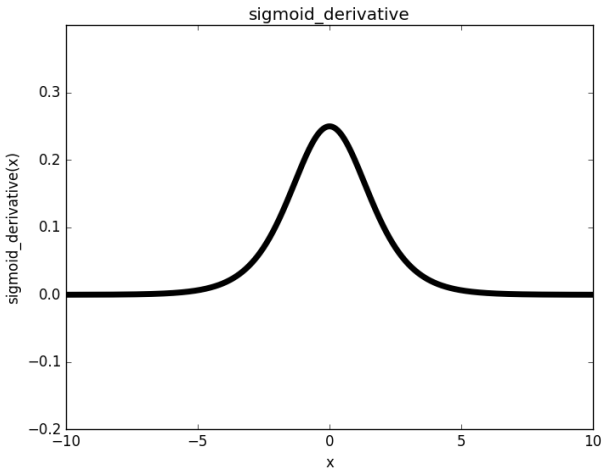
$$\theta_1^{new}=\theta_1^{old}-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)\cdot x_1^i$$
$$\theta_3^{new}=\theta_3^{old}-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)\cdot(x_1^2)^i \quad (2\text{-}7)$$
$$\theta_5^{new}=\theta_5^{old}-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)$$

通过上面迭代我们就可以求出每个变量的最优值大小。这里计算的时候有另一个问题就是 sigmoid 的求导：

$$sigmoid(z)'=(\frac{1}{1+e^{-z}})'=(\frac{1}{1+e^{-z}})(1-(\frac{1}{1+e^{-z}}))=sigmoid(z)(1-sigmoid(z)) \quad (2\text{-}8)$$

这里熟记基本求导运算：$(\frac{u}{v})'=\frac{u'v-uv'}{v^2}$ 及 $(u+v)'=u'+v'$ 。我们可以将

sigmoid 的导数形式画出来如下图:



Sigmoid 导数

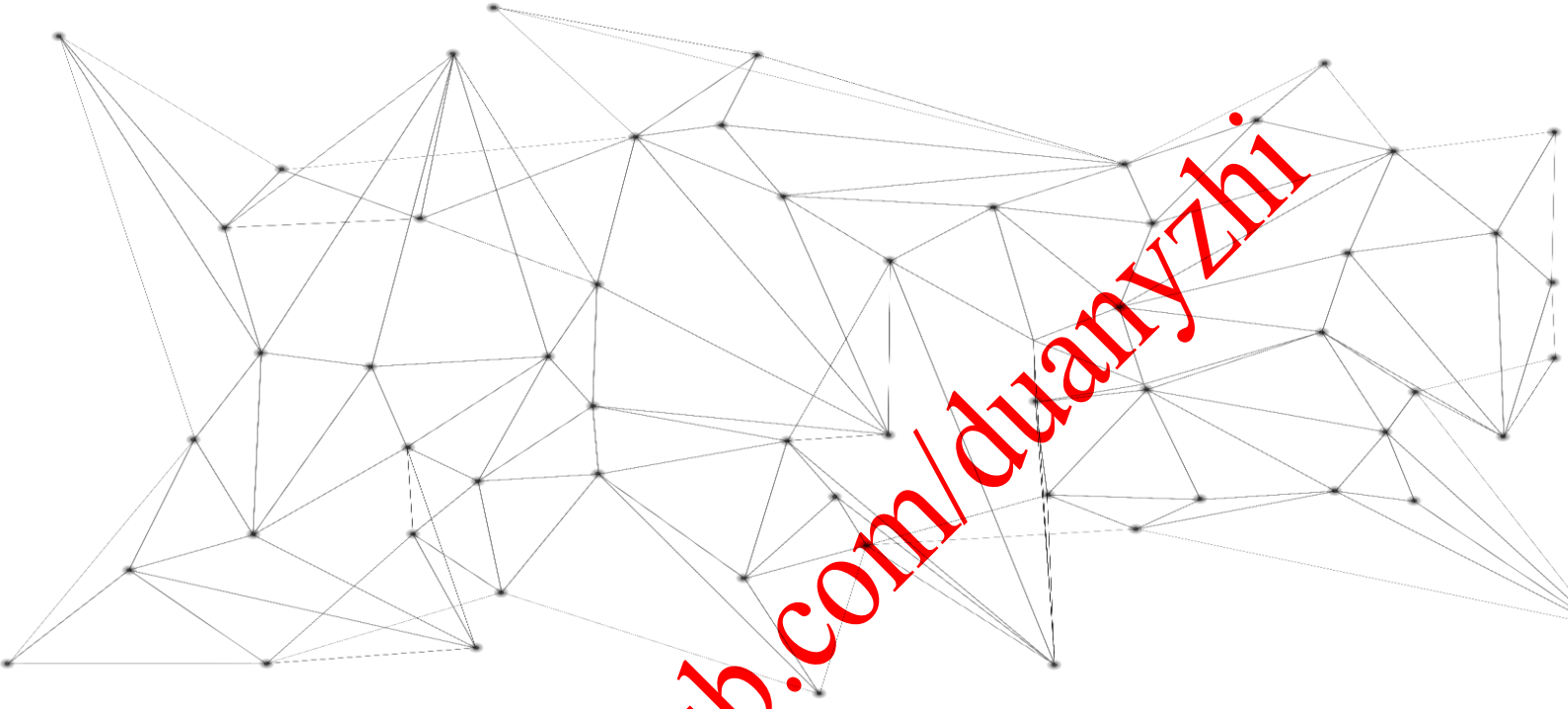
逻辑回归可以处理非线性问题，但是需要引入核函数，将线性问题映射到非线性上解决。

代码：

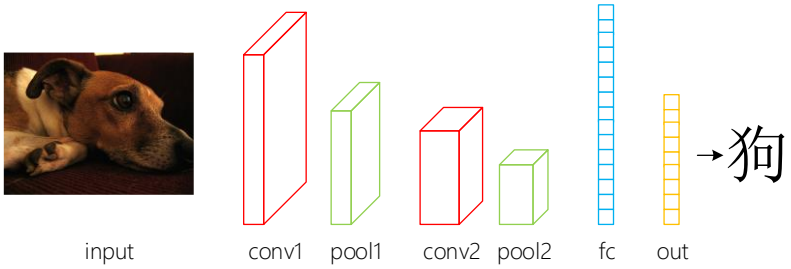
(https://github.com/duanyzhi/My_Road_To_Deep_Learning/tree/master/logistic_regression)

卷积神经网络

深度学习或者机器学习的基础个人认为应该就是卷积神经网络



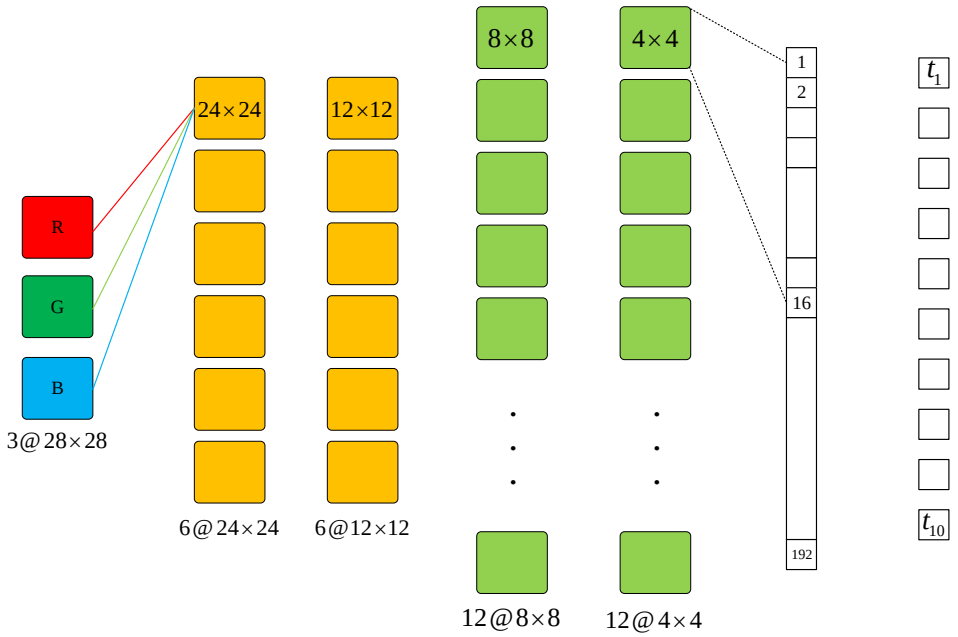
(Convolutional Neural Networks, CNN)了。卷积相当于滤波，通过卷积核选择要留下的特征。CNN 其实并不是很容易理解，尤其是里面的一些细节问题。这里以一个比较简单的 3 层 CNN 结构来说明以下。CNN 的层数是怎么确定的呢？一般具有一层卷积操作称为一层，全连接也是单独的一层。一个简单的 CNN 模型图如下所示：



卷积神经网络

如果提前没了解直接看这个图其实会很迷糊的，比如这些 3D 的框框是啥子都会不知道。可能是习惯了大家都这样画，每一层的框就是代表了卷积运算后的中间层的特征图。假设上图中输入图片是 28×28×3 大小的图像，一般输入都是这样 RGB 的三个通道的图像。一个通道图像大小其实就是一个 28×28 矩阵。3 就

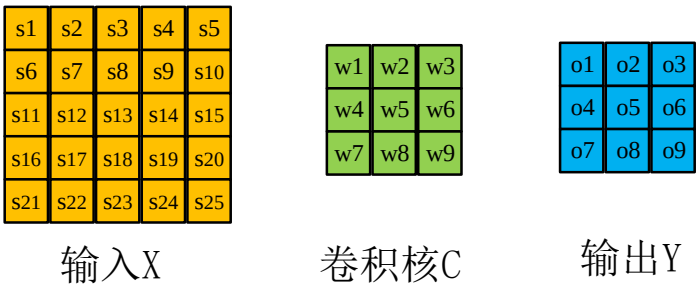
有三个通道，后面说到通道这个词也都是这个意思。我们用另一种方式来表示上面卷积（卷积核大小都是5×5）：



CNN 架构

以

操[illegible]



卷积操作单元

上面的输入是一个5×5大小的矩阵，卷积核是一个3×3大小的矩阵，输出是一个3×3大小的矩阵。那么输入和输出之间每个数满足如下计算关系：

$o1=w1\times s1+w2\times s2+w3\times s3+w4\times s6+w5\times s7+w6\times s8+w7\times s11+w8\times s12+w9\times s13$

$o2=w1\times s2+w2\times s3+w3\times s4+w4\times s7+w5\times s8+w6\times s9+w7\times s12+w8\times s13+w9\times s14$

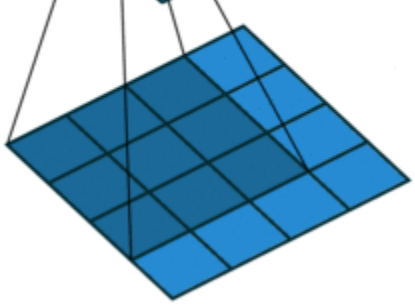
$o3=w1\times s3+w2\times s4+w3\times s5+w4\times s8+w5\times s9+w6\times s10+w7\times s13+w8\times s14+w9\times s15$

$o4=w1\times s6+w2\times s7+w3\times s8+w4\times s11+w5\times s12+w6\times s13+w7\times s16+w8\times s17+w9\times s18$

…

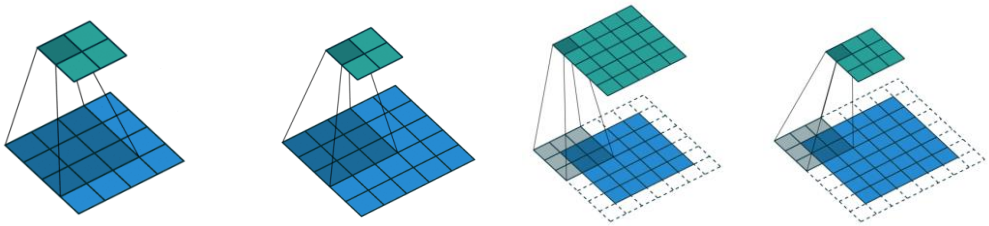
$o9=w1\times s13+w2\times s14+w3\times s15+w4\times s18+w5\times s19+w6\times s20+w7\times s23+w8\times s24+w9\times s25$

从上面公式很容易看到怎么得到卷积输出的，其实就是将卷积核从第一个输入位

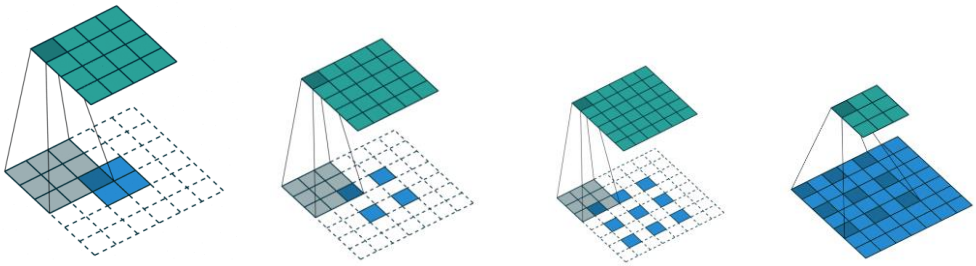


积
（也

叫去卷积或者转置卷积，deconv）过程：



四种卷积操作



四种反卷积操作

详细信息在[https://github.com/vdumoulin/conv_arithmetic]上。首先四种卷积分别表示 VALID 卷积、带步长的 VALID 卷积、SAME 卷积、带步长的 SAME 卷积。下面会具体介绍 VALID 卷积核 SAME 卷积。反卷积这里提一下，反卷积是当输入特征矩阵比较小时我们想得到较大的卷积输出就需要反卷积了。反卷积其实核补零后卷积类似。四种反卷积操作分别是没有 padding 没有 strides 反卷积（注意这里的 padding 核卷积不太一样，这里没有 padding 不是周围不补零，而是在两个输入元素之间补零）、没有 padding 有 strides 的反卷积、有 padding 有 strides 反卷积核空洞卷积(dilated conv)。反卷积具体可以参考论文[1]。空洞卷积比较特殊，一般卷积核是不会分开的，但是空洞卷积卷积核元素之间会插入零，使得卷积核的感知野(即一次卷积的范围)变大。

其实将3×3卷积核写成一个9×25的稀疏矩阵的 C 的形式，把输入 X 写成25×1的向量。可以用数学表达式$Y=CX$来表示卷积操作。那么反卷积的意思一般就是知道了 Y 和 C 的话怎么求 X 呢，即用$X=YC^T$。输出乘以卷积的转置的形式就得到了输入。这既是反卷积的意思也是卷积操作反向求导时的残差传播方法。

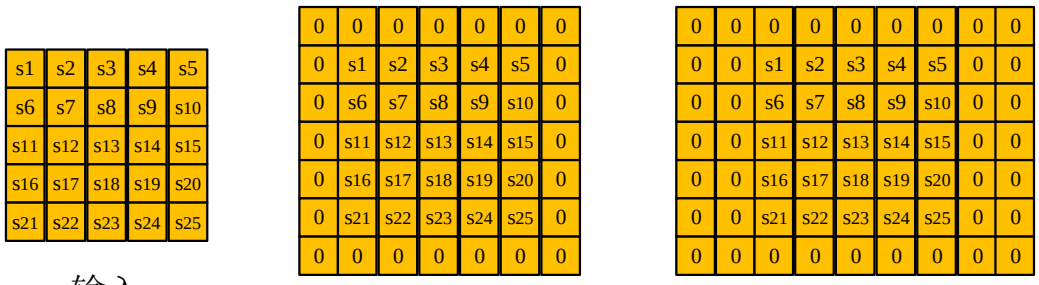
$$C=\begin{bmatrix} w1 & w2 & w3 & 0 & 0 & w4 & w5 & w6 & 0 & 0 & w7 & w8 & w9 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & w1 & w2 & w3 & 0 & 0 & w4 & w5 & w6 & 0 & 0 & w7 & w8 & w9 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & & & & & & & & & & & & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & w1 & w2 & w3 & 0 & 0 & w4 & w5 & w6 & 0 & 0 & w7 & w8 & w9 \end{bmatrix}$$

因为这里卷积从第一个元素$s1$开始到第一行就停止了，因为在往后面的话输入元素最后一列就没有了，这种卷积操作被称为 VALID 卷积。所以输出只有一个

3×3矩阵。我们每次卷积核移动一个格，第一次$s1$开始，第二次$s2$开始，这里其实有一个步长是 1。假设输入大小不是5×5，而是$W\times H$，权重大小是$F\times F$，步长的话设置为S。卷积输出大小是W',H'。那么对于 VALID 卷积有如下关系：

$$W'=\lceil\frac{W-F+1}{S}\rceil \qquad H'=\lceil\frac{H-F+1}{S}\rceil \tag{3-1}$$

其中$\lceil\rceil$ 是向上取整。如果最后实在元素不够的话就补零填充来计算。这种卷积操作会使得输入矩阵(也叫输入特征)变小。如果不希望输入大小改变或者按照一定尺寸变小的话就要用另一种卷积操作了。另一中卷积称为 SAME 卷积，和 VALID 不同的就是 SAME 在卷积前对输入矩阵进行补零操作。可以在上下左右补不同长度零，然后用补零过后的带零的矩阵来继续卷积。同样，左上角的第一个元素（肯定是零了）开始做第一次卷积，部分补零方法如下图所示：



输入　　补零后输入A　　补零后输入B

补零操作

这里可以在输入周围补一圈零(A)，也可以一遍补两列零，一边补一列零(B)。那么到底补多少零呢，这就和我们所需要的输出大小有关系了。输入输出大小假设任然是上面所说的大小，那么对于 SAME 卷积，我们知道输入和输出其实是有关系的：

$$W'=W/S \qquad H'=H/S \tag{3-2}$$

输出和步长关系很大，我们以宽度方向为例，宽度上需要总的补零列数为：

$$N\_pad=(W'-1)\times S+F-W \tag{3-3}$$

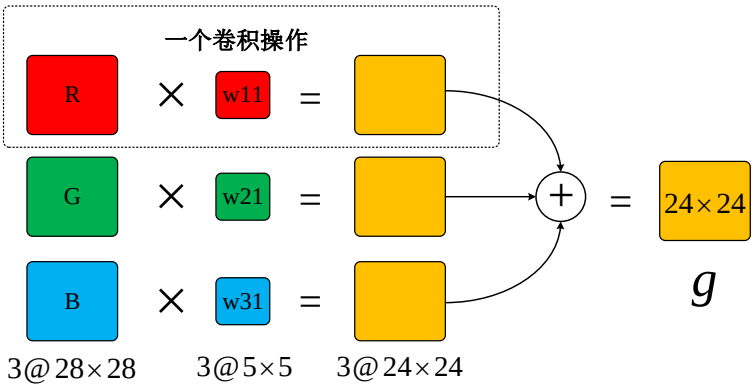
输入左边和右边分别补零的列为：

$$\begin{aligned} N\_left&=int(N\_pad)/2\\ N\_right&=N\_pad-N\_left\end{aligned} \tag{3-4}$$

这是补零的一些操作，除了补零(常数)之外还可以补映射的值等等（具体可以参考 Tensorflow 的 pad 函数）。

继续回到我们的 CNN 基本架构上，这里的我们的卷积就好理解了，我们在这都使用 VALID 卷积方法。首先是从 L=1 到 L=2 层。输入是 RGB 三通道图像，我们使用 3×6 个卷积核计算第一次卷积操作。这里就有另一个常识问题，就是卷积核个数问题。这里是 18 个卷积核进行运算，而不是 6 个。卷积核个数等于输入维度乘以输出维度。因为每一个输入维度和每一个输出维度之间都有一个卷积运算的。每个卷积运算的卷积核也都是不一样的。所以有 18 个连接，就 18 个卷积核了。这里每个卷积核大小我们设为 5×5。那么 L=2 层一共就会有 6 个 24×24 大小的输出(输入是 28，用上面公式以下就出来了)。那么 L=2 层的第一个输出矩阵：24×24 的特征值是怎么得到的呢？如下图所示：

RGB 三个通道输入分别和三个不同的卷积核分别卷积得到三个不同的输出，再将三个不同的输出直接相加得到卷积的值 g，但是这个 g 还不是第一个 24×24 的值，我们还要加上另一偏置项（一般用 b 表示），加上偏执后在经过 sigmoid 激活函数，激活函数的输出才是 L=2 层的第一个 24×24 矩阵的值。

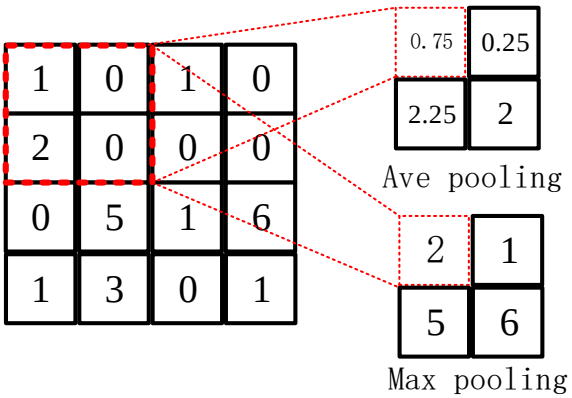


$$output = \textbf{Sigmoid}(g + b)$$

一个标准卷积单元输出运算

这里有几点需要理解，偏执 b 就是一个数，所以 $g+b$ 是一个矩阵加上一个数。那么 L=2 层有几个偏执呢：一个通道输出只有一个偏执，所以 L=2 层有 6 个输出也就是有 6 个偏执项（6 个变量数）。Sigmoid 之后的输出就得到了 L=2 层的第一个特征矩阵输出。同理对 L=2 层第二个 24×24 的特征矩阵：输入 RGB 三通道在和另外三个不同的卷积核进行运算得到第二个输出。

后面就好理解了，L=2 到 L=3 时一个简单的池化(pooling)过程。池化也叫下采样，有两种方式，平均池化(Ave Pooling)或者最大池化(Max Pooling)两种，平均池化求平均值，最大池化将最大值输出。



池化操作

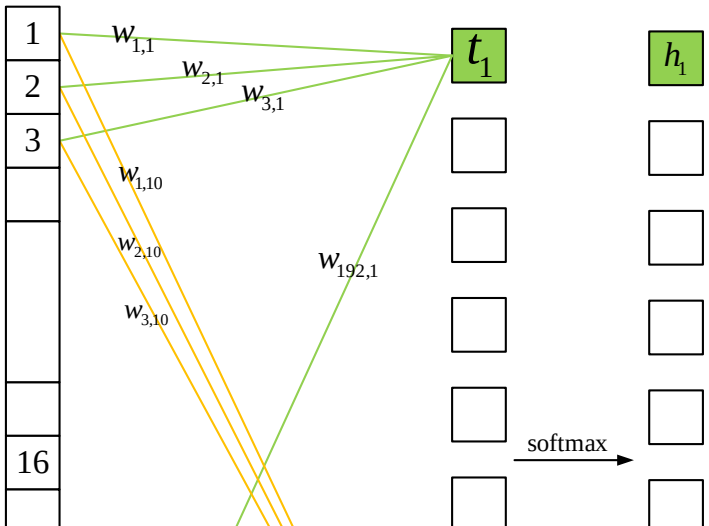
首先说一下池化的优点:

1. 大大减少了运算发杂度(比如2×2池化减少了75%的数据)
2. 从相邻区域中提取出了低级的特征(重要特征)

然后这里讨论一下 Max pooling 和 average pooling:

一般我们是使用最大池化的，因为最大池化提取的是池化单元里最大的值，即像边界这样的特征值点(值比较大或者比较明显)，而舍去不重要的小值。但是平均池化确定取个平均值，无法很好将特征提取出来，因为平均后的值可能不是重要的特征值了。但是在某些场景，比如画风转移模型上平均池化效果反而会好于最大池化。

这里池化大小是一个2×2大小，所以输出特征矩阵大小就变成了一半。后面 L=3 到 L=4 又是一个卷积操作层，一共6×12个5×5大小的卷积核，得到 12 个 8×8特征矩阵输出。然后 L=4 到 L=5 又是一个池化操作。L=5 到 L=6 没有什么计算，只是把 L=5 层的 12 个矩阵按顺序排成一个向量，大小是[1,192]。这称为 reshape,在是在全连接层之前必要的操作。从 L=6 到 L=7 时一层全连接(Full Connection)。这里全连接的输入时 192 个数，输出是 10 个数，假设 192 个输入分别是 $x_1, x_2, \cdots x_{192}$。全连接时一个输入的数和一个输出数之间就有一个权重值。



那么一个输出

$$t_1 = (w_{1,1}x_1 + w_{2,1}x_2 + \cdots w_{192,1}x_{192}) + b_1 \tag{3-5}$$

这里的输出t_1不是我们要的最后输出，还要经过 softmax 将输入映射到[0,1]之间。Softmax 函数如下：

$$h_1 = softmax(t_1) = \frac{e^{t_1}}{\sum_{i=1}^{10} e^{t_i}} \tag{3-6}$$

这里的输出h_1才是 CNN 模型的输出。我们这里采用 sigmoid 代替 softmax 来实现分类。因为 sigmoid 也是将输入映射到[0,1]上。即：

$$h_1 = sigmoid(t_1) \tag{3-7}$$

最后的输出就是我们的结果了，为什么要将输入变成[0,1]之间呢，和回归问题一样，比如我们可以用[1,0,0,0,0,0,0,0,0,0]表示狗这个类别。最后我们将 CNN 模型的输出和标准标签来求损失函数。

$$J(w,b) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(h_k^n - y_k^n)^2 \tag{3-8}$$

上式中N表示 Batch Size 的大小，即一次训练多少张图片，这里推理我们取一张图片，即$N=1$。当 Batch Size 不是 1 时，不同的就是在最后一步将 N 个 batch 的损失函数相加作为总的损失函数。c是分类数，这里是 10 分类。从损失函数我们来对前面多有变量求导，我们用l表示第几层，上面介绍了一共 7 层操作。上式改写以下：

$$J(w,b)=\frac{1}{2}\sum_{k=1}^{c}(h_k^l-y_k^l)^2=\frac{1}{2}\sum_{k=1}^{c}(sigmoid(u^l)-y_k^l)^2 \tag{3-9}$$

$$u^l=W^l x^{l-1}+b^l \tag{3-10}$$

W^l,b^l 是全连接层的权重和偏执。求各个参数的过程用的是反向传播算法(BackPropagation, BP)。上面有权重和偏执两个，我们先对偏执求导：

$$\frac{\partial}{\partial b}J(w,b)=\frac{\partial J}{\partial u}\frac{\partial u}{\partial b}=\frac{\partial J}{\partial u}=\delta \tag{3-11}$$

这里出现了一个新的概念残差 δ，表示损失函数对 u 的偏导数。后面的反向传播都是根据残差来求的。根据上式我们先求最后一层的残差值，先令 $h^l=f(u^l)=sigmoid(u^l)$：

$$\begin{aligned}
\delta_k^l&=\frac{\partial J}{\partial u_k^l}\\
&=\frac{\partial}{\partial u_k^l}\left\{\frac{1}{2}[f(u_1^l)-y_1^l]^2+\frac{1}{2}[f(u_2^l)-y_2^l]^2+\cdots+\frac{1}{2}[f(u_k^l)-y_k^l]^2+\cdots+\frac{1}{2}[f(u_{10}^l)-y_{10}^l]^2\right\}\\
&=(f(u_k^l)-y_k^l)(f(u_k^l)-y_k^l)'\\
&=(f(u_k^l)-y_k^l)f'(u_k^l)\\
&=(f(u_k^l)-y_k^l)f(u_k^l)(1-f(u_k^l))
\end{aligned} \tag{3-12}$$

这样上式所有的值都是已知的了。知道了最后一层的残差值，我们就可以任然使用梯度下降算法来求权重和偏执的更新了：

$$\begin{aligned}
W^{new}&=W^{old}-\alpha\frac{\partial}{\partial W}J(W,b)\\
&=W^{old}-\alpha\frac{\partial J}{\partial u}\frac{\partial u}{\partial W}\\
&=W^{old}-\alpha\delta^l\frac{\partial u}{\partial W}\\
&=W^{old}-\alpha\delta^l f(u^{l-1})
\end{aligned} \tag{3-13}$$

对于某一个权重就可以有：

$$\frac{\partial J}{\partial W_{ik}}=\frac{\partial J}{\partial u_k^l}\frac{\partial u_k^l}{\partial W_{ik}}=\delta_k^l f(u_i^{l-1}) \tag{3-14}$$

对于最后一层的偏执，因为一个输出只有一个偏执所以相对容易得出：

$$b^{new}=b^{old}-\alpha\delta^l \tag{3-15}$$

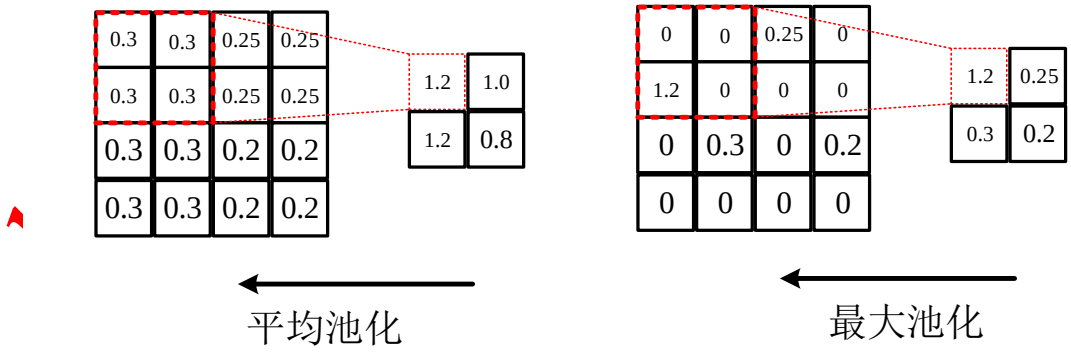
下面从最后一层推导 $l-1$ 层残差，其中 c_l 表示 l 层有多少个输出：

$$
\begin{aligned}
\delta_i^{l-1} &= \frac{\partial J}{\partial u_i^{l-1}} = \frac{\partial}{\partial u_i^{l-1}} \frac{1}{2} \sum_{k=1}^{c_l} \left(f\left(u_k^l\right) - y_k^l \right)^2 \\
&= \sum_{k=1}^{c_l} \left(f\left(u_k^l\right) - y_k^l \right) \frac{\partial}{\partial u_i^{l-1}} \left(f\left(u_k^l\right) - y_k^l \right) \\
&= \sum_{k=1}^{c_l} \left(f\left(u_k^l\right) - y_k^l \right) \frac{\partial}{\partial u_i^{l-1}} f\left(u_k^l\right) \frac{\partial u_k^l}{\partial u_k^l} \\
&= \sum_{k=1}^{c_l} \left(f\left(u_k^l\right) - y_k^l \right) f'\left(u_k^l\right) \frac{\partial u_k^l}{\partial u_i^{l-1}} \\
&= \sum_{k=1}^{c_l} \delta_k^l \frac{\partial u_k^l}{\partial u_i^{l-1}} \\
&= \sum_{k=1}^{c_l} \delta_k^l \frac{\partial}{\partial u_i^{l-1}} [\sum_{j=1}^{c_{l-1}} W_{jk}^l f\left(u_j^{l-1}\right) + b_k^l] \\
&\overset{j=i}{=} \sum_{k=1}^{c_l} \delta_k^l W_{jk}^l f'\left(u_i^{l-1}\right)
\end{aligned}
\tag{3-16}
$$

所以得出残差:

$$
\delta^{l-1} = \left(W^l\right)^T \delta^l .* f'\left(u^{l-1}\right) \tag{3-17}
$$

在往前，从 L=6 到 L=5 时 reshape 比较简单，将 L=6 层多有残差按照对应关系变成矩阵就好。从 L=5 到 L=4 时池化的反过程，池化时是取平均值，所以反向时，我们将 L=5 的一个残差值变成 L=4 层的对应四个残差值。这里可以是将这四个残差值设为一样的。如图所示:



反向池化传播

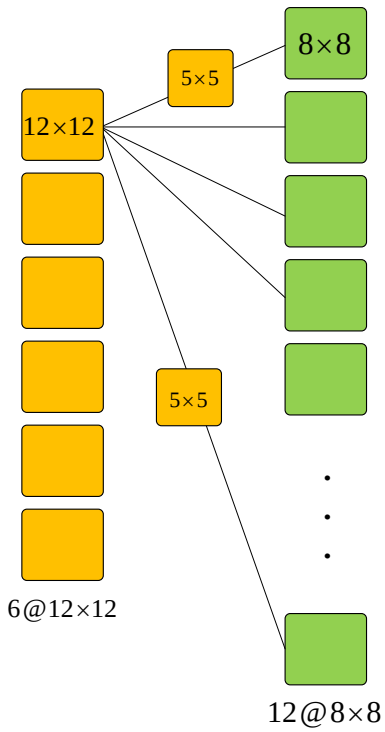
对于平均池化，我们将后项残差除以对应池化核大小(这里是 4 个元素一个池化)，然后将对应四个值都变为 0.3 残差即可。对于最大池化，只保留最大项的残差，将最大项位置残差设置为后面的残差值。其余位置残差设为 0。这里其实可以推导一下，假设池化前的输入是 x，池化输出是 $g(x)$，那么对于另外两种池化：

$$g(x)=\begin{cases}\dfrac{\sum_{k=1}^{m}x_k}{m},\ \dfrac{\partial g}{\partial x}=\dfrac{1}{m} & mean\ pooling\\ \max(x),\ \dfrac{\partial g}{\partial x_i}=\begin{cases}1 & if\quad x=\max(x)\\ 0 & otherwise\end{cases} & \max\ \ pooling\\ \|x\|_p=\left(\sum_{k=1}^{m}|x_k|^p\right),\ \dfrac{\partial g}{\partial x}=\left(\sum_{k=1}^{m}|x_k|^p\right)^{1/p-1}|x_i|^{p-1} & L^p\ \ pooling\end{cases}\tag{3-17}$$

上面关系列出了三种池化过程的反向求导，第三种 P 范数池化用的少。对平均池化，求导后是 $\frac{1}{m}$ ，所以反向的时候将后面的残差乘以 $\frac{1}{m}$ 。损失函数对池化前每一个值求残差得到：

$$\frac{\partial J}{\partial x}=\frac{\partial J}{\partial g}\frac{\partial g}{\partial x}=\delta_g\frac{\partial g}{\partial x}\tag{3-18}$$

这里 δ_g 就是 L=5 层传过来的残差。然后在乘以对 x 的求导的值就可以了。下面就是卷积的反向传播算法过程了，这个比较麻烦其实。从 L=4 到 L=3 是一个卷积的过程。首先前面提到了正向卷积过程是 6 个12×12输入和6×12个5×5的卷积核生成 12 个8×8的特征矩阵输出。通过上面计算我们已经得到了 12 个8×8的残差值，6×12个5×5个权重 w^{old} 的值我们也是知道的。我们就是要求 6 个12×12输入的残差值。



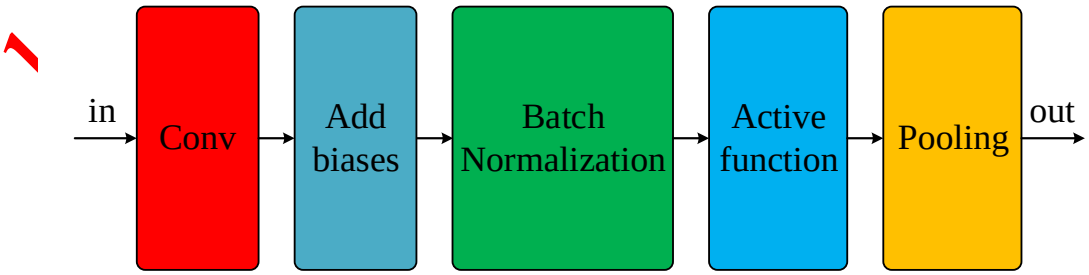
$L=3$ $L=4$

卷积反向传播

这里用的一种求解 L=3 层的残差的方法是也用卷积方法来计算，我们先将8×8的输出进行补零操作，我们将8×8输出上下左右都补四圈零，这样就将8×8输出变为了16×16的输出(为什么是 16，上面 padding 的时候介绍了，是根据输出，卷积核大小算出来的)，然后将这个值和对应的转置后的5×5卷积核进行卷积操作。这里的传播就是上面讨论的反卷积的操作过程。这样就得到了12×12的残差输出。但是我们还要考虑一个问题就是一个12×12的输入要和 12 个8×8输出都有卷积运算的，因此反向的时候也是。12 个8×8的残差分别和对应卷积核运算得到 12 个12×12的残差。我们将这 12 个12×12的残差相加得到一个12×12的残差才是 L=3 层的第一个12×12的残差的值。这样就可以顺利得到 L=3 层所有的残差了。

下面我们计算 L4 层的权重和偏执更新。偏执很简单，12 个8×8的残差都知道了，将8×8个数相加得到 12 个数就是 L=4 层的对偏执的残差值，然后用梯度下降算法即可。对于权重而言，我们这里是用上一次的 L=3 输入值，即 6 个12×12的输入和 L=4 层的 12 个8×8的残差进行卷积来求。假设需要求 L=3 层第一个输入和 L=4 层第一个输出之间的卷积核。我们就用 L=3 层的第一个12×12的特征矩阵和 L=4 层的第一个8×8残差进行 VALID 卷积，得到的5×5输出就是对应的权重的残差值。知道残差值就可以根据上面的梯度求导来计算了。这样就得到了所有的权重更新。L=1,L=2 层同理可得，这里不再叙述。

到这里就差不多介绍完了完整的 CNN 的正向传播及反向传播算法的分析和推理。还有一个是上面没有使用 Batch Normalization(BN)，一般一个卷积层包括如下几个部分：



一个卷积层组成部分

很明显，输入分别经过卷积、加上偏置、BN、激活函数、池化才得到一个卷积层的输出。输入可以直接送入下一个卷积层。上面介绍了大部分的模块，下

面介绍一下 BN,BN 的目的是使数据归一化，对输入数据 x_i 进行如下运算：

1. $u=\frac{1}{m}\sum_{i=1}^{m}x_i$

2. $\text{var}=\sigma^2=\frac{1}{m}\sum_{i=1}^{m}(x_i-u)^2$

3. $\hat{x}_i=\frac{x_i-u}{\sqrt{\text{var}+\varepsilon}}$

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

这里输出 y_i 就是 BN 的输出，其中 $\varepsilon=10^{-6}$ 是一个很小的数，γ、β 是一个变量参数，和权重一样是需要学习的。默认 $\gamma=1,\beta=0$。下面对 BN 这一块进行 BP 运算，假设损失函数还是 J，先对两个变量求导：

$$\frac{\partial J}{\partial\gamma}=\sum_{i=1}^{m}\frac{\partial J}{\partial y_i}\hat{x}_i \qquad \frac{\partial J}{\partial\beta}=\sum_{i=1}^{m}\frac{\partial J}{\partial y_i}$$

然后还是使用梯度下降算法迭代更新：

$$\gamma=\gamma-lr*\frac{\partial l}{\partial\gamma} \qquad \beta=\beta-lr*\frac{\partial l}{\partial\beta}$$

之后对于输出残差我们要得到损失函数对于输入 x_i 的残差才能继续往前传播：

$$\frac{\partial J}{\partial\hat{x}_i}=\frac{\partial J}{\partial y_i}\gamma$$

$$\frac{\partial J}{\partial var}=\sum_{i=1}^{m}\frac{\partial J}{\partial\hat{x}_i}\frac{\partial\hat{x}_i}{\partial var}=\sum_{i=1}^{m}\frac{\partial J}{\partial y_i}\gamma(x_i-u)(-\frac{1}{2})(var+\varepsilon)^{-\frac{3}{2}}$$

$$\frac{\partial J}{\partial u}=(\sum_{i=1}^{m}\frac{\partial J}{\partial\hat{x}_i}\frac{-1}{\sqrt{var+\varepsilon}})+\frac{\partial J}{\partial var}\frac{\sum_{i=1}^{m}-2(x_i-u)}{m}$$

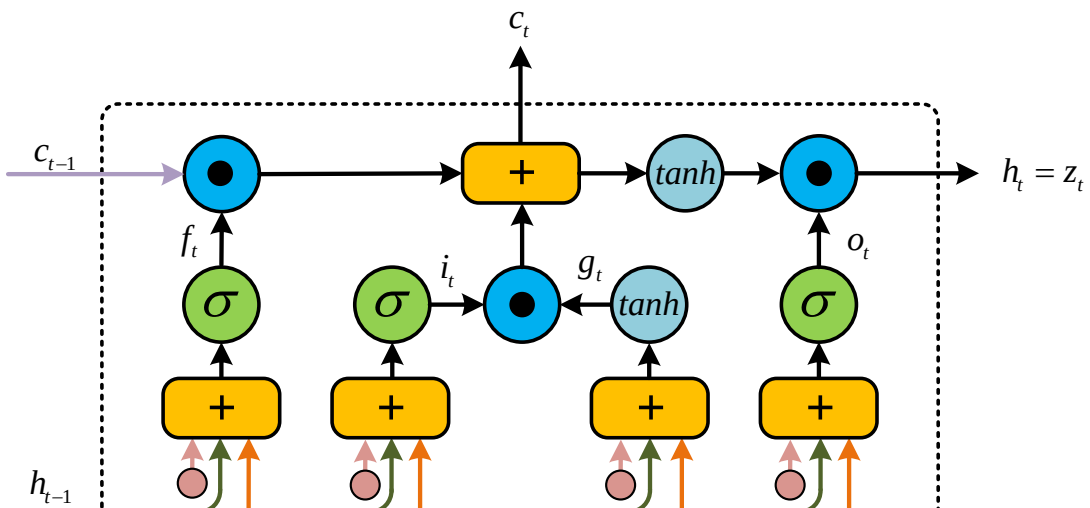
$$\frac{\partial J}{\partial x_i}=\frac{\partial J}{\partial\hat{x}_i}\frac{1}{\sqrt{var+\varepsilon}}+\frac{\partial J}{\partial var}\frac{2(x_i-u)}{m}+\frac{\partial J}{\partial u}\frac{1}{m}$$

这就得出了 BN 时的梯度传播，BN 可以直接加在上面卷积里面，梯度也是直接加在里面。

本节代码用 python-numpy 形式（包括反向求导）已经放在我的 github 上：(https://github.com/duanyzhi/cnn_numpy)

LSTM

Lstm(Long Short-Term Memory)是机器学习中另一种相对比较重要的算法结构。Lstm 一般比较适合处理具有一定关系的一个序列问题，比如通过视频识别一个动作等。下面是简单画的一个 LSTM 单元:



LSTM 单元

LSTM 各个门的方程分别为:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) = \sigma\left(\hat{i}_t\right) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) = \sigma\left(\hat{f}_t\right) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) = \sigma\left(\hat{o}_t\right) \\
g_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) = \tanh\left(\hat{g}_t\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh\left(c_t\right)
\end{aligned}
\tag{4-1}
$$

上面公式比较多，但其实也只是普通的计算。首先明确输入输出，每一个 LSTM 单元输入有 t 时刻的系统输入向量 x_t（一般是文本信息或者图像信息提取的特征向量）、上一个时刻的隐藏单元(hidden unit) h_{t-1}、上一时刻的记忆单元(memory cell unit) c_{t-1}。输出是这一个时刻的隐藏单元 h_t、这一时刻系统的输出 z_t 、记忆单元 c_t。

一般来说 lstm 是处理一段时间的信息，假设时刻总长度是 T，那么输入序列 $\langle x_1, x_2, \cdots x_T \rangle$，其中 $x_i \in R^M$ 是一个长度为 M 的向量。其他数学含义：隐藏单元 $h_t \in R^N$，遗忘门限(forget gate) $f_t \in R^N$，输出门限(output gate) $o_t \in R^N$，输入调制门(input modulation gate) $g_t \in R^N$，记忆单元 $c_t \in R^N$。另外 $\sigma(x) = \left(1+e^{-x}\right)^{-1}$ 是 sigmoid 函数，$\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$ 是双曲正切分线性激活函数(hyperbolic tangent non-linearity)。$x \odot y$ 表示向量 x：$[x_1, x_2, \cdots, x_n]$ 和向量 y：$[y_1, y_2, \cdots, y_n]$ 逐元素相乘(element-wise product)：$x \odot y = [x_1 y_1, x_2 y_2, \cdots, x_n y_n]$。

通过上面计算就可以输出下一时刻的记忆单元、隐藏单元输出。一般第一个时刻的隐藏单元输入 $h_0=0$。第一个时刻输入第一个 x_1，h_0,c_0，送入上面 LSTM 单元，输出第一个时刻 h_1,c_1。然后再将这个输出和第二个时刻的输入 x_2 再次送入上面同一个 LSTM 再次计算得到第二个时刻输出 h_2,c_2，就这样一直计算到最后一个时刻输出。可以看到 LSTM 的计算也是复用的，所有的 LSTM 是一个，变量个数也就是这一个 LSTM 的变量数。最后一层的输出 h_t 就是我们 LSTM 模型的输出，比如使用 LSTM 来分类就可以将输出在经过一层 softmax 然后和便准标签比较，我们可以通过这种方法来得到损失函数，假设是 $J(W,b)$。我们可以找出最后输的残差:

$$\delta h_t=\frac{\partial J}{\partial h_t} \tag{4-2}$$

我们通过 $h_t=o_t\odot\tanh(c_t)$ 这个残差值来求 $\delta o_t,\delta c_t$。

$$\frac{\partial J}{\partial o_t^i}=\frac{\partial J}{\partial h_t^i}\cdot\frac{\partial h_t^i}{\partial o_t^i}=\delta h_t^i\cdot\tanh\left(c_t^i\right) \tag{4-3}$$

$$\delta o_t=\delta h_t\odot\tanh(c_t) \tag{4-4}$$

$$\frac{\partial J}{\partial c_t^i}=\frac{\partial J}{\partial h_t^i}\frac{\partial h_t^i}{\partial c_t^i}=\delta h_t^i\cdot o_t^i\cdot\left(1-\tanh^2\left(c_t^i\right)\right) \tag{4-5}$$

$$\delta c_t+=\delta h_t\odot o_t\odot\left(1-\tanh^2\left(c_t\right)\right) \tag{4-6}$$

其中 i 表示每一个数，因为记忆单元也要传到下一个时刻，所以 δc_t 不仅要算出这一个时刻的残差还要加上 $t+1$ 时刻的残差值，加在一起才是 t 时刻的残差。

我们通过 $c_t=f_t\odot c_{t-1}+i_t\odot g_t$ 和残差 δc_t 来计算 $\delta i_t,\delta g_t,\delta f_t,\delta c_{t-1}$。

$$\frac{\partial J}{\partial i_t^i}=\frac{\partial J}{\partial c_t^i}\frac{\partial c_t^i}{\partial i_t^i}=\delta c_t^i\cdot g_t^i \qquad \delta i_t=\delta c_t\odot g_t \tag{4-7}$$

$$\frac{\partial J}{\partial f_t^i}=\frac{\partial J}{\partial c_t^i}\frac{\partial c_t^i}{\partial f_t^i}=\delta c_t^i\cdot c_{t-1}^i \qquad \delta f_t=\delta c_t\odot c_{t-1} \tag{4-8}$$

$$\frac{\partial J}{\partial g_t^i}=\frac{\partial J}{\partial c_t^i}\frac{\partial c_t^i}{\partial g_t^i}=\delta c_t^i\cdot i_t^i \qquad \delta g_t=\delta c_t\odot i_t \tag{4-9}$$

$$\frac{\partial J}{\partial c_{t-1}^i}=\frac{\partial J}{\partial c_t^i}\frac{\partial c_t^i}{\partial c_{t-1}^i}=\delta c_t^i\cdot f_t^i \qquad \delta c_{t-1}=\delta c_t\odot f_t \tag{4-10}$$

如果忽略非线性函数，公式(4-1)也可以写成如下形式：

$$z^t=\begin{bmatrix}g_t\\ \hat{i}_t\\ \hat{f}_t\\ \hat{o}_t\end{bmatrix}=\begin{bmatrix}W_{xc} & W_{hc}\\ W_{xi} & W_{hi}\\ W_{xf} & W_{hf}\\ W_{xo} & W_{ho}\end{bmatrix}\times\begin{bmatrix}x_t\\ h_{t-1}\end{bmatrix}=W\times I_t \tag{4-11}$$

我们来求δz_t：

$$\begin{aligned}\delta\hat{g}_t&=\delta g_t\odot\left(1-\tanh^2\left(\hat{g}_t\right)\right)\\ \delta\hat{i}_t&=\delta i_t\odot i_t\odot\left(1-i_t\right)\\ \delta\hat{f}_t&=\delta f_t\odot f_t\odot\left(1-f_t\right)\\ \delta\hat{o}_t&=\delta o_t\odot o_t\odot\left(1-o_t\right)\\ \delta z_t&=\left[\delta\hat{g}_t,\delta\hat{i}_t,\delta\hat{f},\delta\hat{o}_t\right]^T\end{aligned} \tag{4-12}$$

又因为$z_t=W\times I_t$，我们可以根据δz_t求出$\delta W_t,\delta h_{t-1}$：

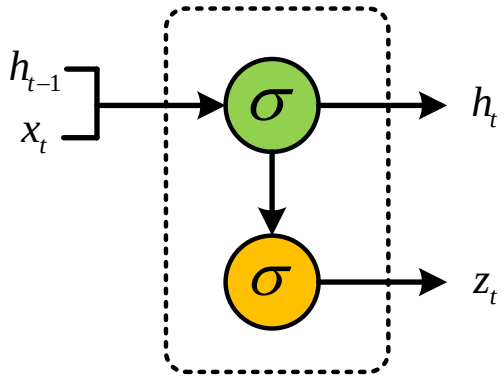
$$\delta I_t=W^T\times\delta z_t$$

又因为：$I_t=\begin{bmatrix}x_t\\ h_{t-1}\end{bmatrix}$，那么可以$\delta W_t,\delta h_{t-1}$可以从$\delta I_t$中检索出来。

$$\delta W_t=\delta z_t\times\left(I_t\right)^T$$

又因为输入又 T 个时刻，那么$\delta W=\sum_{t=1}^{T}\delta W_t$，将所有时刻的残差相加就得等到了权重的残差值。然后通过之前介绍的梯度下降算法就很容易算出来下一时刻的迭代值。对于偏置比较简单就补叙述了。

lstm 的提出其实是为了解决 rnn 的梯度消失问题，那么 rnn 为什么会梯度消失呢？RNN 单元如下：



一个 RNN 单元

$$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$
$$z_t = \sigma(W_{zh}h_t + b_{zh})$$

这里激活函数可以取 sigmoid 也可以是 tanh 等。RNN 面临梯度消失（梯度爆炸）现象原因。

激活函数是 sigmoid 或者 tanh，当时间跨度较大，残差会很小，时间越长梯度越小知道消失。另一个问题是对于每一个时刻 RNN 中的W是同一个值，也就是复用的。所以残差中会出现W_{hh}累乘结果（出现矩阵高次幂），当W_{hh}为对角阵：

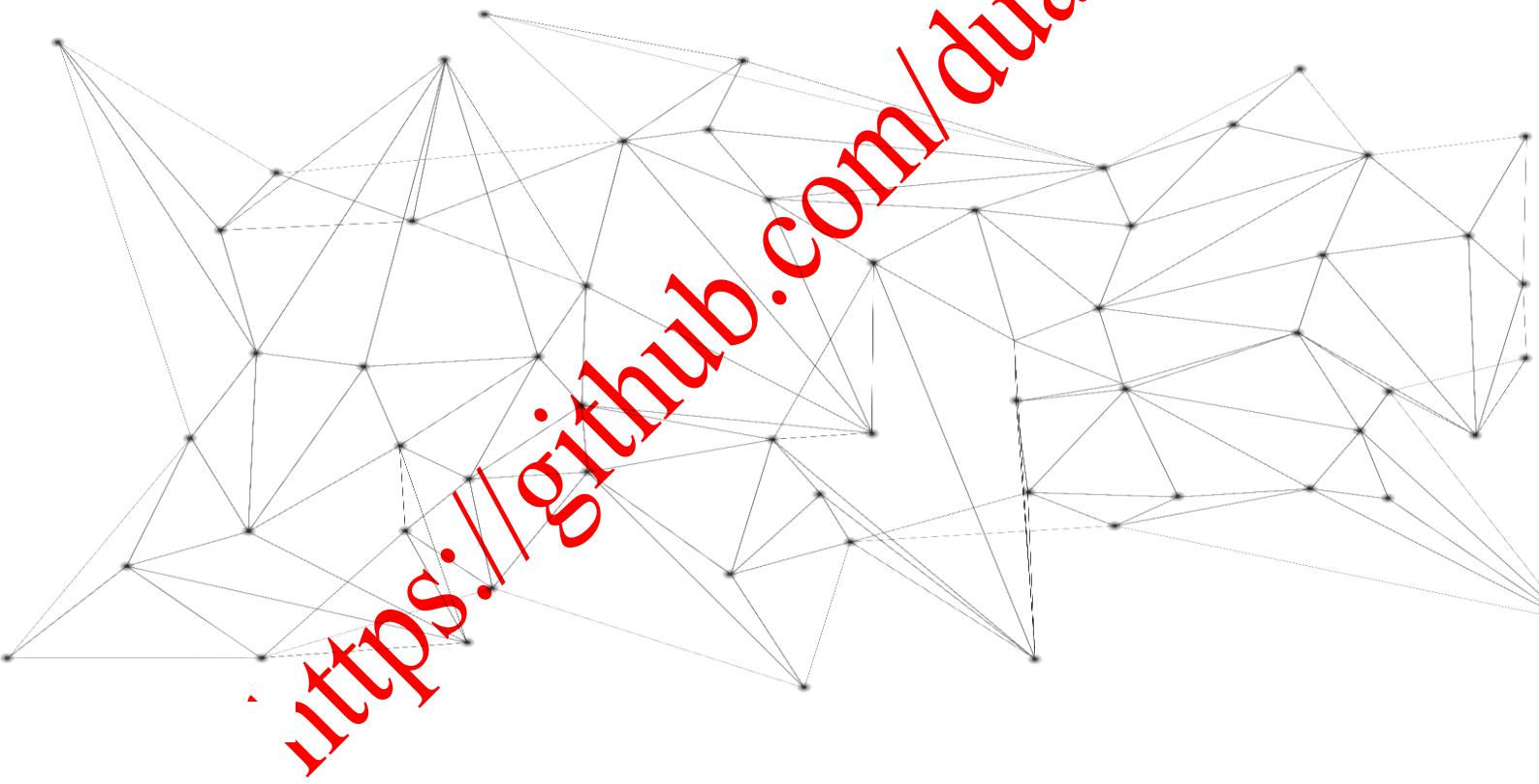
若对角线元素小于 1，则其幂次会趋近于 0，从而导致梯度消失(Gradient Vanish)。若对角线元素大于 1，则其幂次会趋近于无穷大，从而导致梯度爆炸(Gradient Explode)。

针对这些问题解决问题可以是将激活函数换成 relu 或者通过 lstm 中的门限限制。上面求过了 lstm 的各个导数：

$$\delta c_{t-1} = \delta c_t \odot f_t$$

在记忆单元从后一个时刻往前传播过程中，遗忘门限控制了其值大小，通过学习不会出现彻底消失或者爆炸现象。lstm 的门函数有选择的只让一部分信息通过，门函数 sigmoid 输出 0,之间的数用来判断过滤掉信息还是保留信息。

Faster-RCNN



ResNet

1×1卷积核作用：

1. 降维（当然也可以用来升维），在3×3卷积之前先起到降低维度的作用。这样后续计算复杂度降低。
2. 加入非线性。卷积层经过激励层，1×1卷积在前一层学习表示上添加了非线性激励，提升了网络表达能力。1×1卷积相当于是多个 feature channels 的线性叠加过程。

深度学习基础知识

一、 召回率、准确率、精准率

数据经过模型之后可分为:

正样本: 正样本被判断成正样本(True Positives, TP)
正样本被判断成负样本(False Negatives, FN)

负样本: 负样本被判断成负样本(True Negatives, TN)
负样本被判断成正样本(False Positives, FP)

召回率(recall):

$$Recall = \frac{TP}{TP+FN}$$

准确率(accuracy):

$$Accuracy = \frac{TP+TN}{TP+FN+FP+TN} = \frac{\text{所有预测对的数据}}{\text{所有数据}}$$

精准率(precision):

$$Precision = \frac{TP}{TP+FP}$$

二、 PCA

主成分分析(Principal Component Analysis, PCA)是将高维的 N 维数据映射到较低维度的 D 维上。我们取特征值最大的 D 个作为输出。假设样本矩阵为 $A_{m\times n}$ ，每一行为一个样本共 m 个样本，每一列为一个特征，共 n 个特征。对于两个不相关随机变量 X、Y 他们的协方差是 0 的:

$$\operatorname{cov}(X,Y) = \mathrm{E}\left([\mathrm{X}-\mathrm{E}[\mathrm{X}]][Y-E[\mathrm{Y}]]\right) = 0$$

为了表示矩阵每一列不相关，如果上面均值为 0,可简化:

$$\operatorname{cov}(X,Y) = \mathrm{E}\left([\mathrm{X}-0][Y-0]\right) = 0$$

这里点乘为 0，这样 A 通过某线性变化出的新矩阵 B 每一列正交。

$$B^T B = D$$

D 是对角矩阵。假设变化是 $AM = B$：

$$(AM)^T(AM) = \mathrm{D}$$
$$M^T A^T AM = D$$
$$A^T A = (M^T)^{-1} D M^{-1}$$

$A^T A$ 是一个对角的，那么特征值分解: $A^T A = VDV^{-1}$ 中 V 是正交单位阵，即 $V^T = V^{-1}$。V 就是 M。

PCA 流程如下:

1. 去中心化，将输入数据 A 每一列按减去这列平均值，使得 A 每列均值为 0：

$$\bar{x}=\frac{1}{N}\sum_{i=1}^{N}x_i \qquad x_i=x_i-\bar{x}$$

2. 求出 AA^{T} 的特征值 $D=diag\{\lambda_1,\ \lambda_2,\cdots,\ \lambda_n\}$ 特征矩阵 V

3. $B=AV$

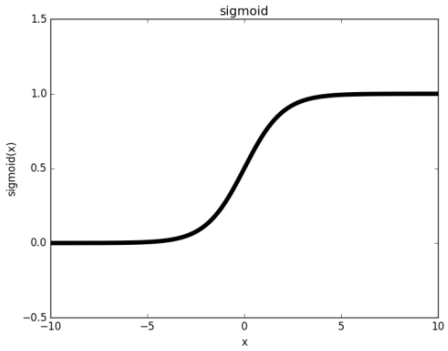
B 是我们求的每一列均值为 0 且每列正交的矩阵。$A=BV^T$，V^T 的每一行就是我们要的特征值。

三、 常用激活函数及对比

激活函数作用:增加神经网络模型的非线性。

A. Sigmoid:

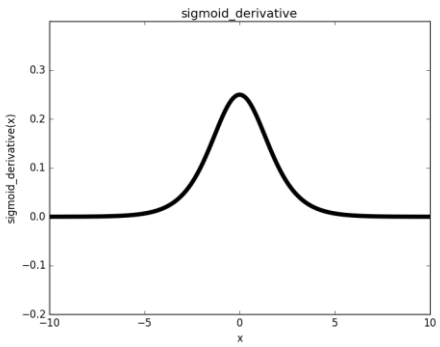
$$f(x)=\frac{1}{1+e^{-x}}$$



Sigmoid 函数

Sigmoid 导数:

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



Sigmoid 导数

指数家族有一个很大的优点是具有最大熵的性质。sigmoid 在指数函数族（Exponential Family）里面是标准的 bernoulli marginal，而 exponential family 是一个给定数据最大熵的函数族，直观理解是熵大的模型比熵小的更优化，受数据噪声影响小。Sigmoid 可以使得熵最大，在李航老师《统计学习方法》中有介绍为什么 logistic 回归要用 sigmoid:

对于一个事件几率(odds)是指该事件发生的概率与该事件不发生的概率的比值，假设发生时间概率是 p，那么该事件的几率是 $\frac{p}{1-p}$，该事件的对数几率(log odds, logit)函数是:

$$logit(p)=\log\frac{p}{1-p}$$

对输入 x 来言，输出 Y=0/1。那么输出 Y=1 的对数几率是 x 的线性函数：

$$\log\frac{P(Y=1|x)}{1-P(Y=1|x)}=wx$$

则这个概率：

$$P(Y=1|x)=\frac{e^{wx}}{1+e^{wx}}\overset{\Delta}{=}sigmoid(wx)$$

Sigmoid 优点：

1. 将输入规划到[0,1]之间，输出预测的条件概率。
2. Sigmoid 引入了非线性关系。
3. 输出有限在传递过程中不至于使数据发散。
4. Sigmoid 的求导也比较容易。
5. Sigmoid 还可以直接用于输出层概率预测替代 softmax。

Sigmoid 缺点:

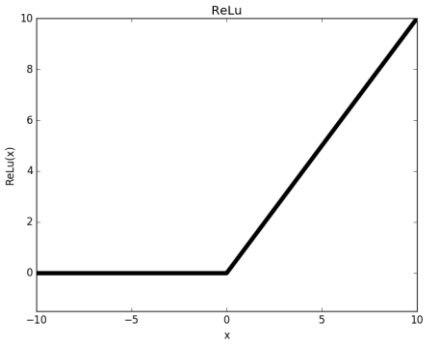
1. 饱和的时候梯度太小。由 sigmoid 导数图形看到只有 x 在 0 附近导数才会有一定的值，当 x 值不在 0 附近时就会出现导数为 0 的情况。而梯度消失的原因:在反向传播过程中，使得残差逐渐接近 0，使得梯度无法继续往前传播，特别是对于饱和神经元而言。对于 sigmoid 一旦 x 比较大或者比较小就会使得 sigmoid 的值出现饱和状态（一直是 0 或者一直是 1）。这样饱和的 sigmoid 反向会使得梯度消失。
2. sigmoid 计算含有指数项，计算量大。
3. sigmoid 的输出不是以 0 为中心的，这样传递之后后面的网络反向传递会出现抖动的情况。我们一般希望数据是以 0 为中心的。这里解决的方法就是权重初始化的时候可以选择零均值分布的高斯分布，使得 sigmoid 输入尽量在零附近。如果设置不合理就会出现 sigmoid 输出

全部偏向 0 或者 1。

非线性函数好处：如果只是用线性函数，那么无论网络多少层输出都是输入的线性关系，相当于只有一层。非线性是神经网络中的一个重要组成部分，非线性函数可以将输入映射为更广泛的值从而大大增加了运算的可能性，非线性函数使得网络近似逼近任意的函数，起到提取特征的作用。

B. ReLu 函数:

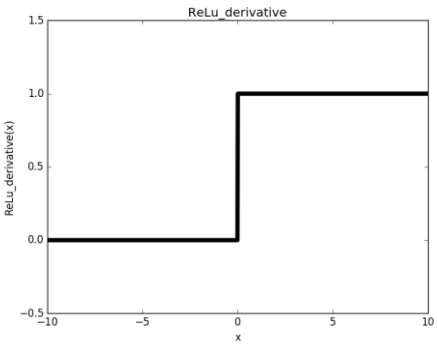
$$g(x)=\begin{cases}0 & for\ x<0\\ x & for\ x\geq 0\end{cases}$$



ReLu 函数

ReLu 求导：

$$g'(x)=\begin{cases}0 & for\ x<0\\ 1 & for\ x\geq 0\end{cases}$$



ReLu 导数

现在很多深度学习模型里面的非线性函数都会使用 relu 函数。

ReLu 函数优点:

1. 由 relu 导数可以看到不会出现残差一直是 0 的情况，即不会出现梯度消失的情况。
2. relu 求导十分简单，收敛较快。
3. relu 一部分神经元输出是 0，这就照成了网络的稀疏性，稀疏的网络运算效果是较高的。

4. relu 输出有些是 0 也较少了参数的相互依赖关系，有效缓解了过拟合现象。

ReLu 缺点：

1. 由于当输入小于 0 时，导数为 0，这样反向传播时梯度也是 0，这样对应的权重不会更新，那么起不到梯度传播的作用。relu 过于脆弱很可能有些数据节点会直接死去并不会再次被激活，即下次传递时权重不变，相当于 relu 没有被激活，失去了作用。这样的单元梯度将永远是零，比如调试程序时一般学习速率过大会出现将输入的值变得过小，从而导致输出全是 0 了（这种一般先设置一个较小学习速率学习一段时间，在从大的学习速率往下降）。

解决这中现象方法是想办法让输入小于零的时候输出不是零，而是一个很小的数。即可以使用 Leaky-ReLu 等：

$$g(x)=\begin{cases} x & if\ x\geq 0 \\ \alpha x & if\ x<0 \end{cases}$$

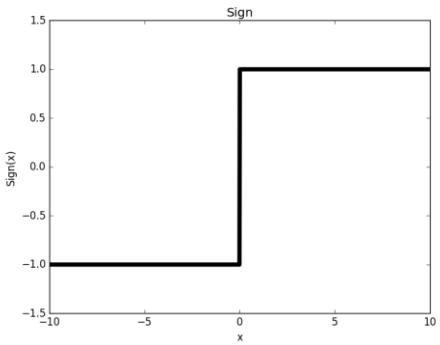
这里 α 是一个很小的正数，比如 0.01。这样就可以避免梯度都是零的情况。

2. relu 在输入大于零之后输出等于输入，但是这样输入过大的时候输出就会特别大，使得网络学习较慢。
   解决方法，使用 ReLu6 函数：

$$g(x)=\begin{cases} 0 & for\ x<0 \\ x & for\ 0\leq x\leq 6 \\ 6 & for\ x>6 \end{cases}$$

C. Sign(符号)函数：

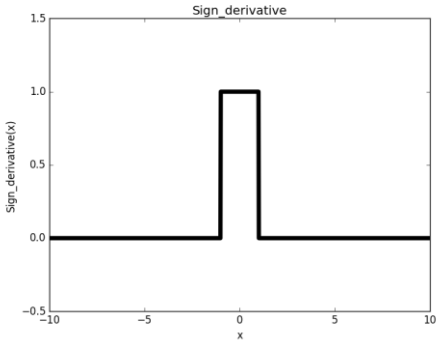
$$k(x)=\begin{cases} 1 & for\ x>0 \\ 0 & for\ x=0 \\ -1 & for\ x<0 \end{cases}$$



Sign 函数

Sign 导数：

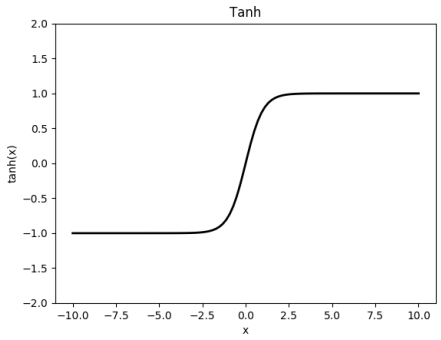
$$k'(x)=\begin{cases}1 & -1\le x\le 1\\ 0 & others\end{cases}$$



Sign 导数

D. Tanh 函数：

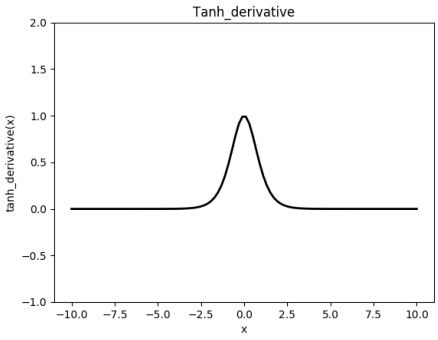
$$tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$



Tanh 函数

Tanh 导数：

$$tanh'(x)=1-tanh^{2}(x)$$

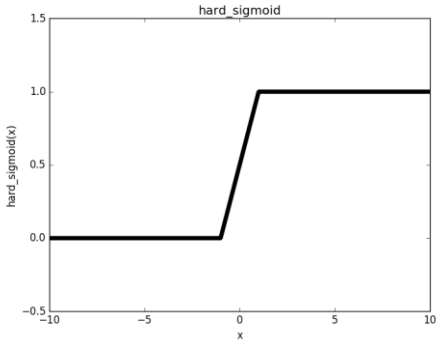


Tanh 导数

tanh 相比 sigmoid 优点是 tanh 输出是以零为中心的，也是全程可导。但是 tanh 在包和区域非常平缓，容易出现梯度消失。

E. Hard sigmoid 函数(Binary/XNOR Net 网络中用到)：

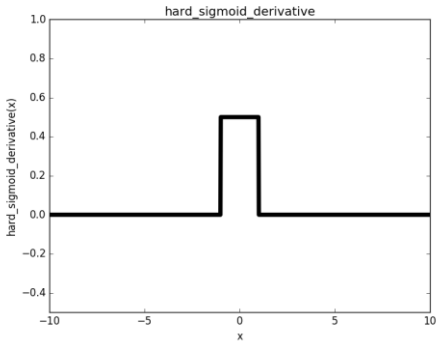
$$hs(x)=\max\left(0,\min\left(1,\frac{x+1}{2}\right)\right)=\begin{cases}1 & x>1\\ \frac{x+1}{2} & -1\le x\le 1\\ 0 & x<-1\end{cases}$$



Hard sigmoid 函数

Hard sigmoid 导数:

$$hs'(x)=\begin{cases}0.5 & -1\le x\le 1\\ 0 & others\end{cases}$$

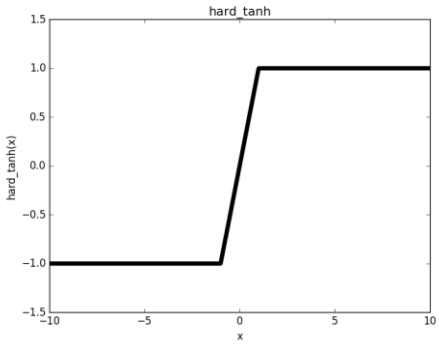


Hard sigmoid 导数

F. Hard tanh 函

$$ht$$

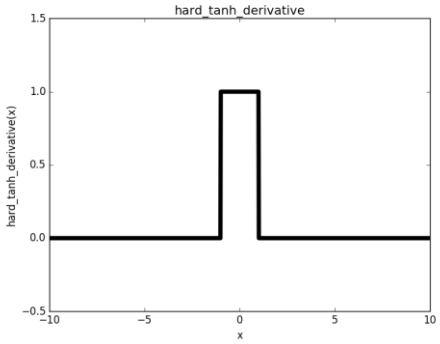
$$\begin{cases}-1 & x<-1\end{cases}$$



Hard tanh 函数

Hard tanh 导数:

$$ht'(x)=\begin{cases}1 & -1\le x\le 1\\ 0 & others\end{cases}$$



Hard tanh 导数

四、 Weights Initialization 不同的方式

初始化重要性：

如果权重初始太小，那么信号在穿过每一层时都会收缩，直到太小没用，那么网络较深就会使得前面数据无法传到后面。

如果权重太大，那么信号在穿过每一层时都会增大，直到太大而无法使用，而且权重太大求导之后也会很大，这样会出现数据爆炸现象(尤其是 sigmoid 这样，输出全是 1 了)。

1. 首先介绍下最初学习的时候最常用的就是高斯初始化，一般会将权重偏置初始话都设置为一个零均值、固定方差(比如 0.01)的高斯分布。$W \sim N(0,\sigma)$，这里偏置也可以直接都令为零。
2. 更简单的权重都是 1，偏置都是 0.
3. 目前最常用的应该是 Xavier 方式初始化。

假设输入是 $x=(x_1,x_2,\cdots,x_n)$，权重是 W，那么输出是：

$$y=W^T x=w_1x_1+w_2x_2+\cdots+w_nx_n$$

我们一般希望方差通过卷积层之后保持不变，以防止信号过大或者消失到零。我们可以通过改变权重来达到这个要求。方差不变：

$$Var(y)=Var(w_1x_1+w_2x_2+\cdots+w_nx_n)=Var(w_1x_1)+\cdots+Var(w_nx_n)$$

假设权重和输入数据是相互独立且都是零均值的：

$$Var(w_ix_i)=E[x_i]^2Var(w_i)+E[w_i]^2Var(x_i)+Var(w_i)Var(x_i)=Var(w_i)Var(x_i)$$

在假设如果所有的 x_i,w_i 都是独立的且独立同分布的，那么：

$$Var(y)=n\cdot Var(w_i)(x_i)$$

这就的得出了输入输出的方差关系。所以我们要项确保 y 的方差和输入 x 的方差一样，那么就需要令 $n\cdot Var(w_i)=1$：

$$Var(w_i)=\frac{1}{n}=\frac{1}{n_{in}}$$

为了保证输入梯度和输出梯度的方差相同，如果 $n_{in}=n_{out}$，那么就可以满足，

但是一般输入输出不会一样，因此采用一个折中的方法：

$$Var(w_i)=\frac{2}{n_{in}+n_{out}}$$

所以 Xavier 权重初始化一般是将权重令为零均值，方差为$\frac{2}{n_{in}+n_{out}}$的高斯分布。或者作者给出权重也可以选用如下均匀分布：

$$W\sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}},\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$$

Xavier 好处:Xavier 旨在处理梯度消失或者梯度爆炸现象。

但是 relu 函数一般不会有梯度消失或者爆炸现象，一般使用 relu 函数初始化为零均值，方差为$\sqrt{\frac{2}{n_l}}$的高斯分布。

五、 数据预处理过程

1. 首先常用的预处理就是零均值(zero-center)、归一化(normalize)：

去均值： $u=\frac{1}{m}\sum_{i=1}^{m}x_i$ $\quad x_i=x_i-u$

除以方差： $\text{var}=\sigma^2=\frac{1}{m}\sum_{i=1}^{m}(x_i-u)^2$ $\quad \hat{x}_i=\frac{x_i-u}{\sqrt{\text{var}}}$

这里的输出 $\hat{x}_i$ 就是真正输入卷积网络的输入。

六、 常用损失函数

上面介绍的模型中使用损失函数是平方差损失函数，也叫二次函数(Quadratic cost)，假设输入是 x 模型输出是 $h(x)$，正确的标签是 y ， 分类类别数用 c 表示，损失函数用 J 表示：

$$J=\frac{1}{2}\sum_{i=1}^{c}\left(h(x_i)-y_i\right)^2$$

另一种是在逻辑回归里用的交叉熵损失函数：

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}Cost(h_\theta(x^i),y^i)$$

$$=-\frac{1}{m}\sum_{i=1}^{m}[y^i\log h_\theta(x^i)+(1-y^i)\log(1-h_\theta(x^i))]$$

七、 Bath Size 大小选取以及影响

首先，先说一下什么是 batch size，一个网络可以一次训练一张图片，也可

以并行的计算多张图片，这个多个图片大小就是 batch size 的值。我们通过多张图片会并行的得到多个输出及多个损失函数。然后在反向求导的时候将这多个损失函数相加得到一个平均的损失函数。然后利用这个平均的损失函数来反向的训练网络，而不是一个单一的图像的损失函数。那么 batch size 有什么好处呢：

1. 对于数据如果不归一化，那么各个特征之间的维度跨越(差异)会很大，如果每次只用一张图片迭代学习，可能无法学习到整体数据信息而出现损失函数在一个局部情况，导数也可能偏向局部值。所以 batch size 可以很好的决定梯度下降的方向，由较多数据集确定的梯度方向能够较好的代表样本总体信息，从而更精准的朝向梯度最小的方向迭代。Batch size 越大，其确定的下降方向越准确，引起的训练震荡越小。
2. 除此之外，不同权重的梯度值差别是很大的，因此选取一个全局的学习速率很难。Batch size 可以取一个平均值，使得数据迭代更加合理化。
3. 跑完一个 epoch(全部训练集)时间会大大减少。
4. 内存利用率增加。
5. Batch size 越大，收敛越快。

Batch size 缺点：

1. 内存利用率提高了，但是内存容量会占用比较多。
2. Batch size 过大(如整个数据集)，那么其确定的下降方向就不会改变了。
3. Batch size 一次训练只训练一次，那么训练完整个数据集反向迭代次数就会减小。需要大大增加总体迭代次数才能有较好收敛情况。即要达到同一个精度，所需要的 epoch 数量要大大增加。

八、梯度下降算法等迭代算法

1. 梯度下降算法(Gradient Descent，GD)

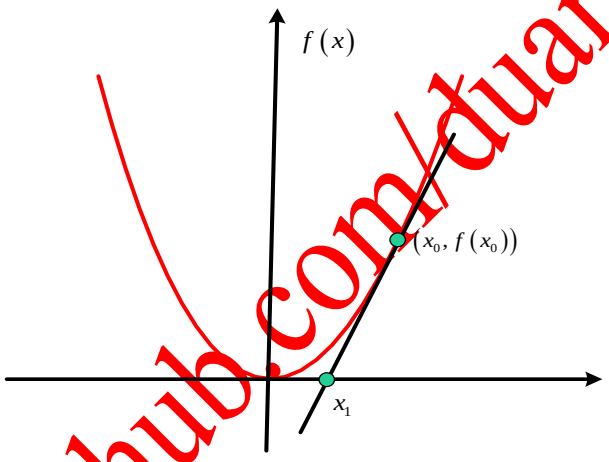
梯度下降算法是最常用的迭代球最优值算法之一。假设一个函数是 $J(\theta)$，要求函数最小值点是的变量 θ 的值。首先理解梯度的意义：在微积分里面，对多元函数参数求偏导数 $\frac{\partial J(\theta)}{\partial \theta}$，把求的各参数的偏导数以向量的形式写出来，就是梯度。梯度向量从几何意义上讲，就是函数变化增加最快的地方，沿着梯度向量的方向更容易找到函数的最大值，沿着向量相反的方向，梯度减小最快，更容易找到函数最小值。对于一元变量梯度是一个数，对于多元是一个向量用来表示变化方向。我们沿着负梯度方向迭代就可以使得函数值越来越小了。

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_j)$$

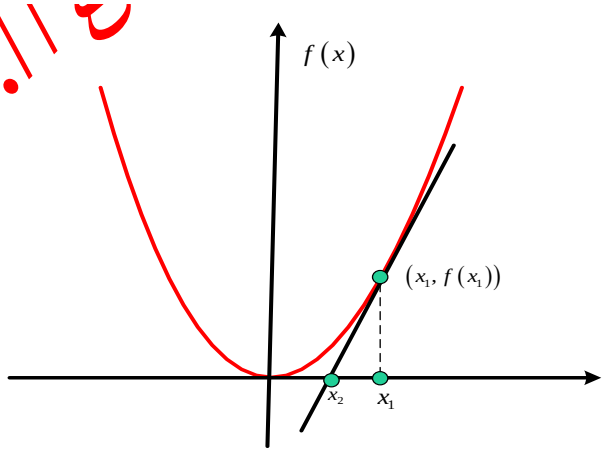
可以看到，梯度下降算法每次迭代要用到所有样本进行更新，这在数据量很大时是相当费时的。

2. 牛顿法

牛顿法是利用切线逼近与曲线特性来做的。一般在某个曲线的小范围内其切线是可以近似代替曲线的。假设要求的是函数 $f(x)$ 的最优解。那么首先先初始化一个点 x_0，在 x_0 处对曲线求切线，并且求出切线和坐标轴交点 x_1。



然后找到 x_1 在曲线上的点 $(x_1, f(x_1))$，然后再次求出切点，及切线和坐标轴对应点 x_2。这样一直迭代下去，迭代的点就会越来越靠近目标点。



那么对于 x_n 点的切线方程：$y = f(x_n) + f'(x_n)(x - x_n)$，那么 x_{n+1} 即 $y = 0$ 的解。即 $f(x_n) + f'(x_n)(x_{n+1} - x_n) = 0$：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

上面是一元函数的求法，对于多个变量求导之后就不是一个导数而是对每个变量的偏导数矩阵(海塞矩阵, Hesse matrix)。牛顿法是二阶收敛，梯度下降是一阶收敛。所以牛顿法更快。梯度下降只考虑下降最快方向，牛顿法在选择方向时还考了梯度的梯度是否最快。

3. 最速下降法

逻辑回归是凸的(convex)，而神经网络是非凸的(non-convex)。最速下降算法(Stochastic Gradient Descent, SGD)在两种情况下都能保证一定收敛性。对于梯度下降落入鞍点或者局部最优点就无法跳出，但 SGD 可以解决这个问题。SGD 和 GD 一样的更新方式，但是 SGD 每次更新只用到一个样本来进行运算迭代，任然假设模型输出函数为: $h(x) = \sum_{i=0}^{n} \theta_i x_i$，代价函数：

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{c}\left(h_\theta(x^i) - y^i\right)^2$$

求导：

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j}\frac{1}{2}\left(h_\theta(x) - y\right)^2 \\ &= \left(h_\theta(x) - y\right)\frac{\partial}{\partial \theta_j}\left(h_\theta(x) - y\right) \\ &= \left(h_\theta(x) - y\right)x_j\end{aligned}$$

这种方法每次求得的偏导其实很大，数据大了偏导就会很大迭代也会很大，数据小了则会很小。因此这就造成了 SGD 的随机性。但是 SGD 的数据都是正确的，即偏导也会一直往最优值点出迭代。但是由于每次差异比较大，因此不会出现落入局部点出不来的情况（即使落入局部最优点等到数据大的梯度来的时候自然会跳出去）。SGD 需要更多步才能收敛，由于其对导数要求很低，可以包含大量噪声，只要数据期望大致正确，就可以往最优点迭代，因此 SGD 算的也特别快。

算法及数学基础

1. 排列组合问题：

$$C_n^m = \frac{n!}{m!(\mathrm{n}-\mathrm{m}!)}$$

2. 插板法：N 个相同球放入 M 个不同箱子，每个箱子至少一个：
   相当于 N 个球中间有 N-1 个空，在 N-1 个空里选 M-1 个空插板分开即可。
   一共 C_{N-1}^{M-1} 种方法。如果每个箱子可以为空，相当于有 N+M 个球再插板。

3. 矩阵 A 特征值 λ 和特征向量 x：

$$Ax = \lambda x$$