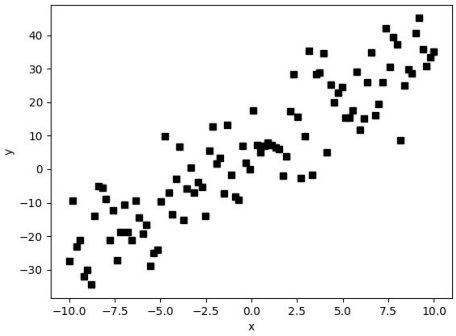
My Road To Deep Learning

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

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。

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

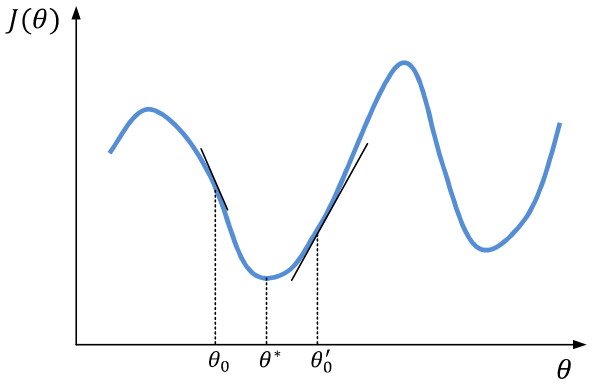
这里为什么是最小化这个损失函数呢，因为上面的损失函数是一个关于 θ 的二次函数，在解决这种多变量问题时可以假设一个变量是常量，那么就变成解决单变量问题的函数了。所以这个损失函数展开后就是一个关于 θ_0 或 θ_1 的二次函数，具有一个最有解点。这里我们就使用梯度下降来求这个点。按照一般方法，对每一个变量求解时固定其他变量，然后对损失函数求导并令导数为 0。即：

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

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：

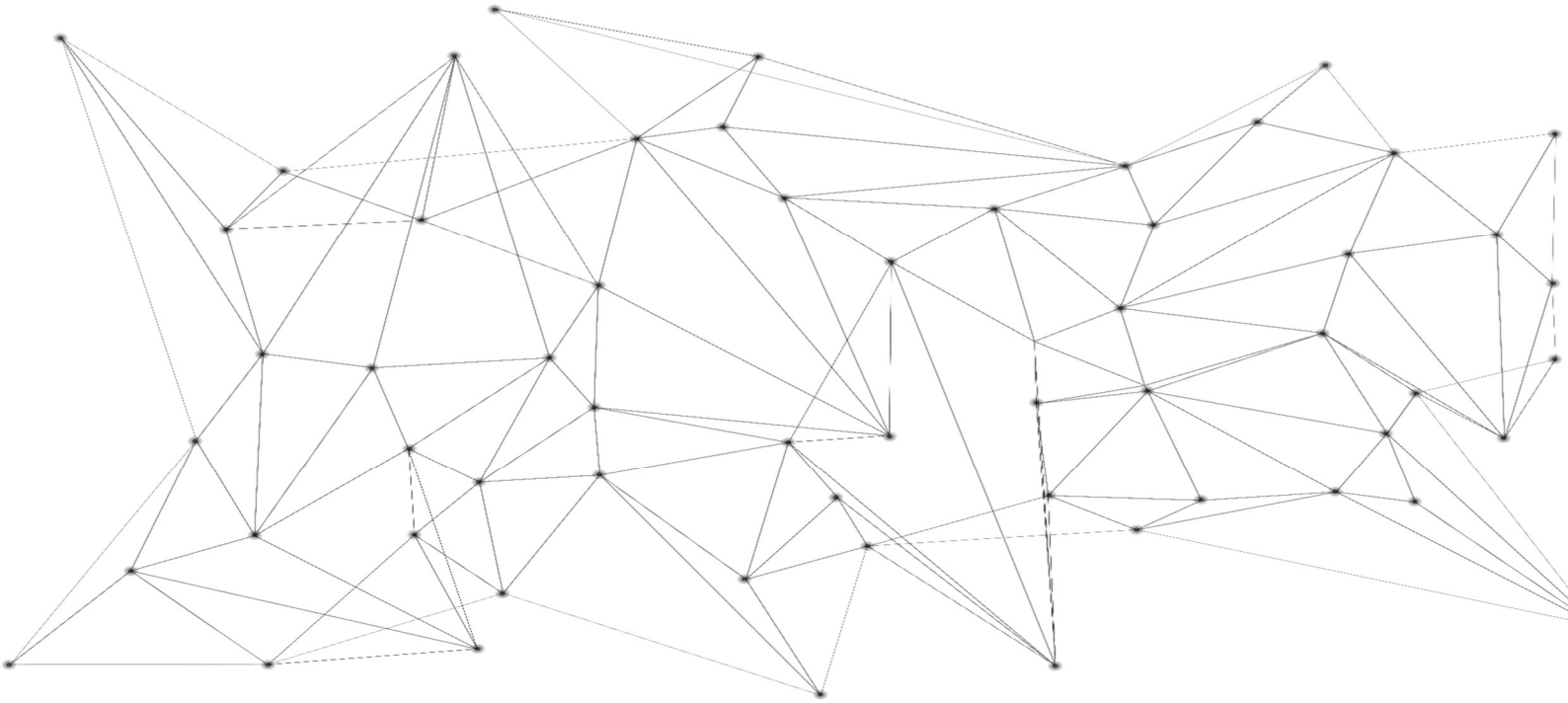
$$\theta_1^{new}=\theta_1^{old}-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)\cdot x^i \tag{7}$$

梯度下降法可写成下面的通式：

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

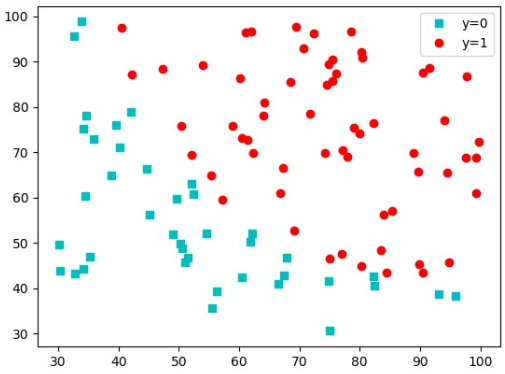
每一次迭代后新的参数用来训练下一个新的数据，这样经过一定次数迭代后，损失函数就会呈现收敛状态。最后收敛时候的参数就是我们需要的参数值。

Code:(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：

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

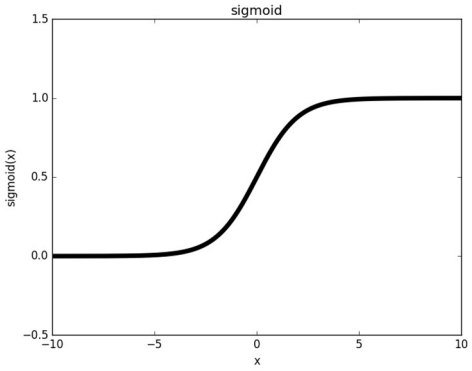
因为这里分布比较简单，输出 h 是关于输入的多项式，其中 g 表示某中函数运算。我们只去了输入的一次方和二次方。对于比较复杂的分布还可以取三次方、开方等运算。逻辑回归判断不同于线性回归。因为输出 $0\le h_\theta(x)\le 1$ 是一个连续的数，标签 y 非 0 即 1，一般满足：

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

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，$h_\theta(x)=\text{sigmoid}(z)\geq 0.5$。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。

0 **1**

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：

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^i), y^i) \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^i \log h_\theta(x^i) + (1-y^i)\log(1-h_\theta(x^i))] \end{aligned} \tag{2-5}$$

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。

$$\begin{aligned} \frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j} -\frac{1}{m}\sum_{i=1}^{m}[y^i \log h_\theta(x^i) + (1-y^i)\log(1-h_\theta(x^i))] \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^i \frac{1}{h_\theta(x^i)}(h_\theta(x^i))' + (1-y^i)\frac{1}{1-h_\theta(x^i)}(1-h_\theta(x^i))'] \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^i \frac{1}{h_\theta(x^i)}h_\theta(x^i)(1-h_\theta(x^i))z' - (1-y^i)\frac{1}{1-h_\theta(x^i)}(1-h_\theta(x^i))h_\theta(x^i)z'] \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^i(1-h_\theta(x^i)) - (1-y^i)h_\theta(x^i)]z' \\ &= -\frac{1}{m}\sum_{i=1}^{m}\{[y^i - h_\theta(x^i)]\frac{\partial}{\partial\theta_j}[\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5]\} \\ &= \frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^i) - y^i]x_j^i \end{aligned} \tag{2-6}$$

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：

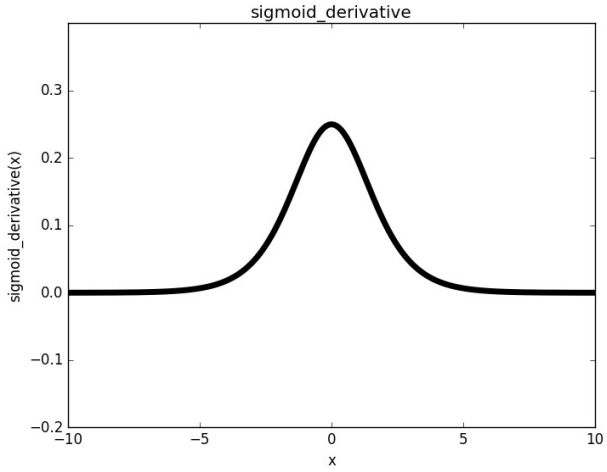
$$\begin{aligned} \theta_1^{new} &= \theta_1^{old} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)\cdot x_1^i \\ \theta_3^{new} &= \theta_3^{old} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)\cdot (x_1^2)^i \\ \theta_5^{new} &= \theta_5^{old} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)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

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：

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

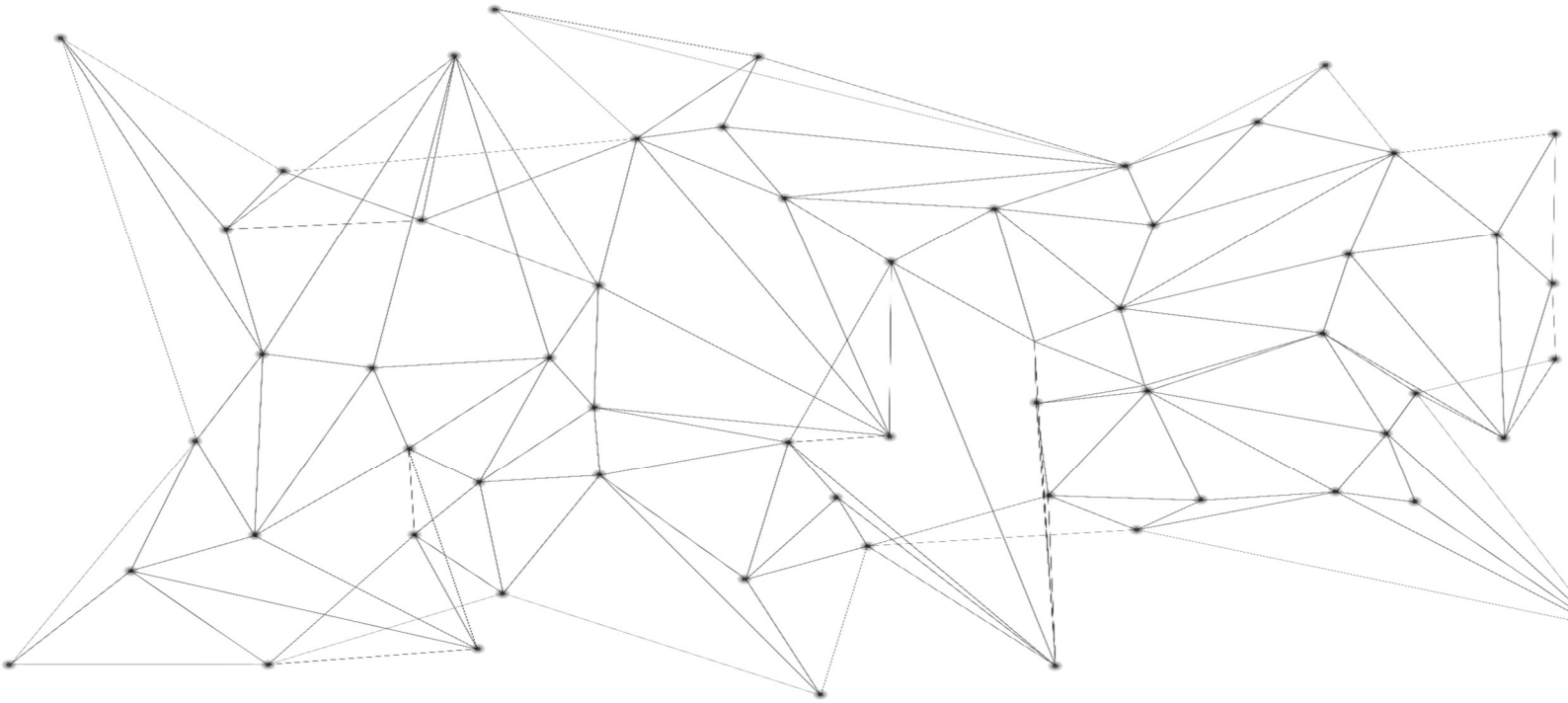
这里熟记基本求导运算：$(\frac{u}{v})' = \frac{u'v - uv'}{v^2}$ 及 $(u+v)' = u' + v'$ 。我们可以将

sigmoid 的导数形式画出来如下图:



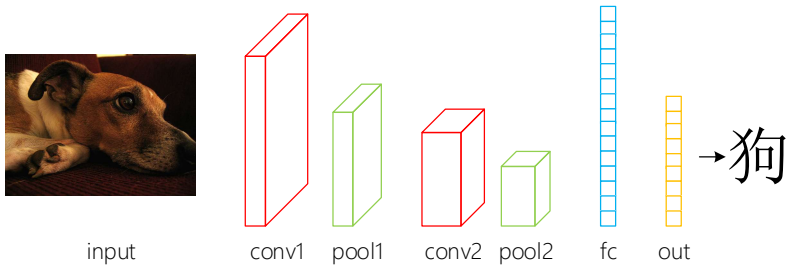
Sigmoid 导数

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



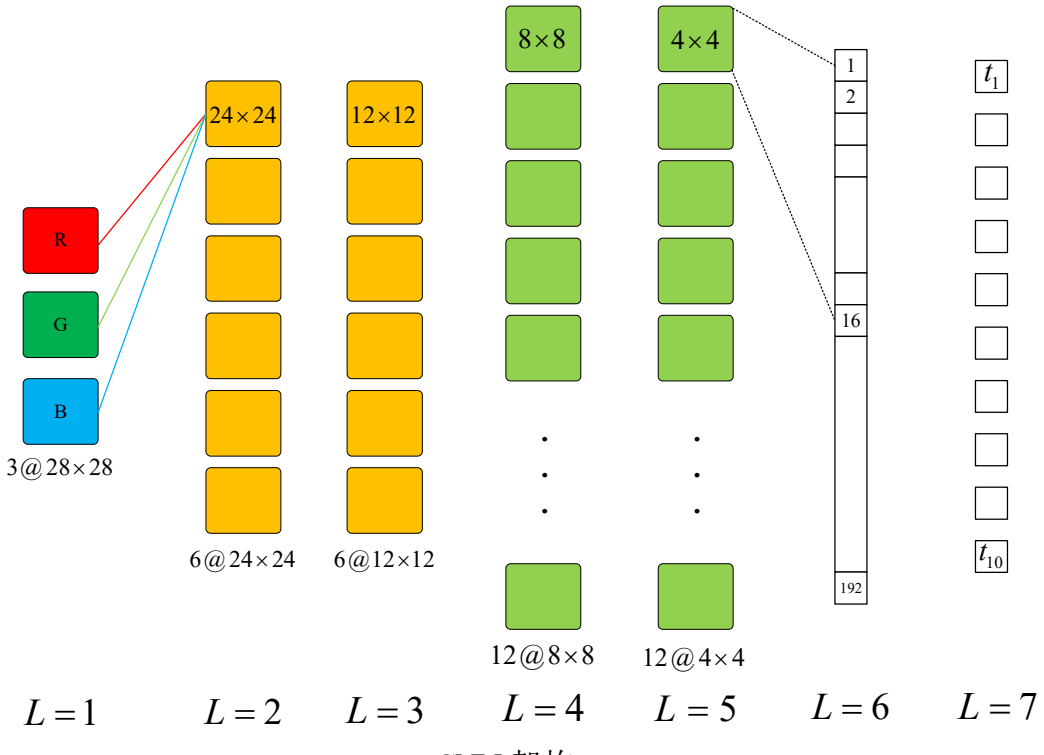
卷积神经网络

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。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：



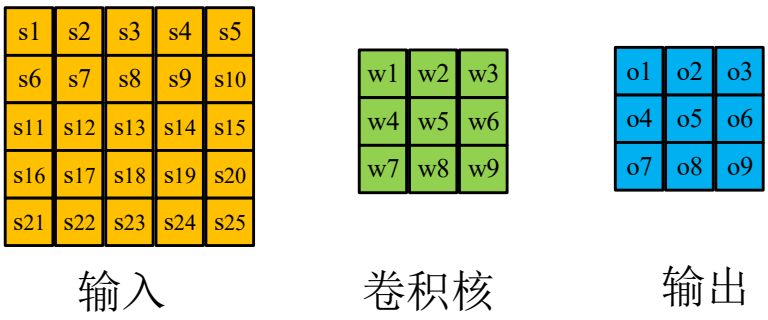
卷积神经网络

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

上面两个图表示的意义是一样的。我们将三维的图变成二维的。下面先讨论以下什么是卷积操作。对于两个矩阵，进行卷积的含义就是对应元素相乘相加的操作，如下图所示：



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:

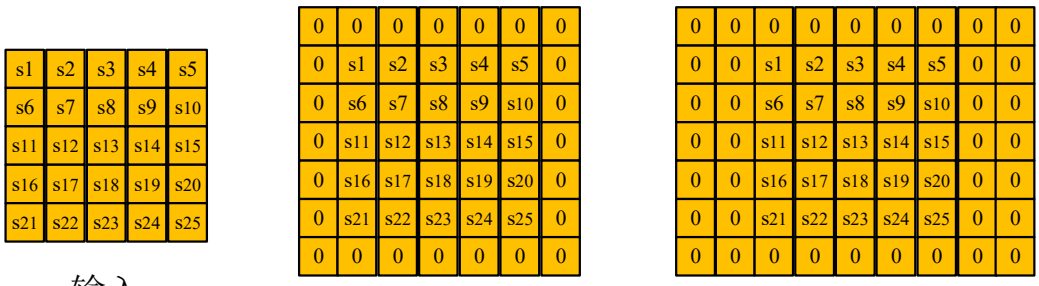
$$o1 = w1\times s1 + w2\times s2 + w3\times s3 + w4\times s6 + w5\times s7 + w6\times s8 + w7\times s11 + w8\times s12 + w9\times s13$$
$$o2 = w1\times s2 + w2\times s3 + w3\times s4 + w4\times s7 + w5\times s8 + w6\times s9 + w7\times s12 + w8\times s13 + w9\times s14$$
$$o3 = w1\times s3 + w2\times s4 + w3\times s5 + w4\times s8 + w5\times s9 + w6\times s10 + w7\times s13 + w8\times s14 + w9\times s15$$
$$o4 = w1\times s6 + w2\times s7 + w3\times s8 + w4\times s11 + w5\times s12 + w6\times s13 + w7\times s16 + w8\times s17 + w9\times s18$$
$$\cdots$$
$$o9 = w1\times s13 + w2\times s14 + w3\times s15 + w4\times s18 + w5\times s19 + w6\times s20 + w7\times s23 + w8\times s24 + w9\times s25$$

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置开始将所有输入扫描一遍。然后将对应元素相乘在相加。这就得到了一个卷积操作的输出。后面所提到的卷积操作就是这个意思了。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：

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

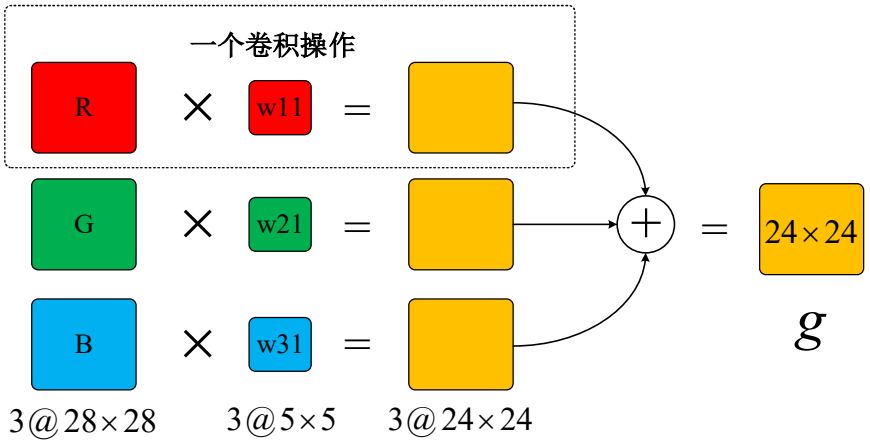
输入左边和右边分别补零的列为：

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

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。

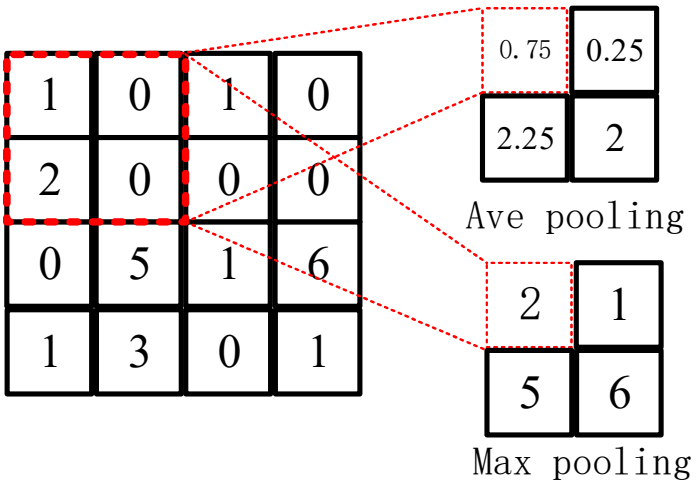


$$output = \textbf{Sigmoid}(g + \mathbf{b})$$

一个标准卷积单元输出运算

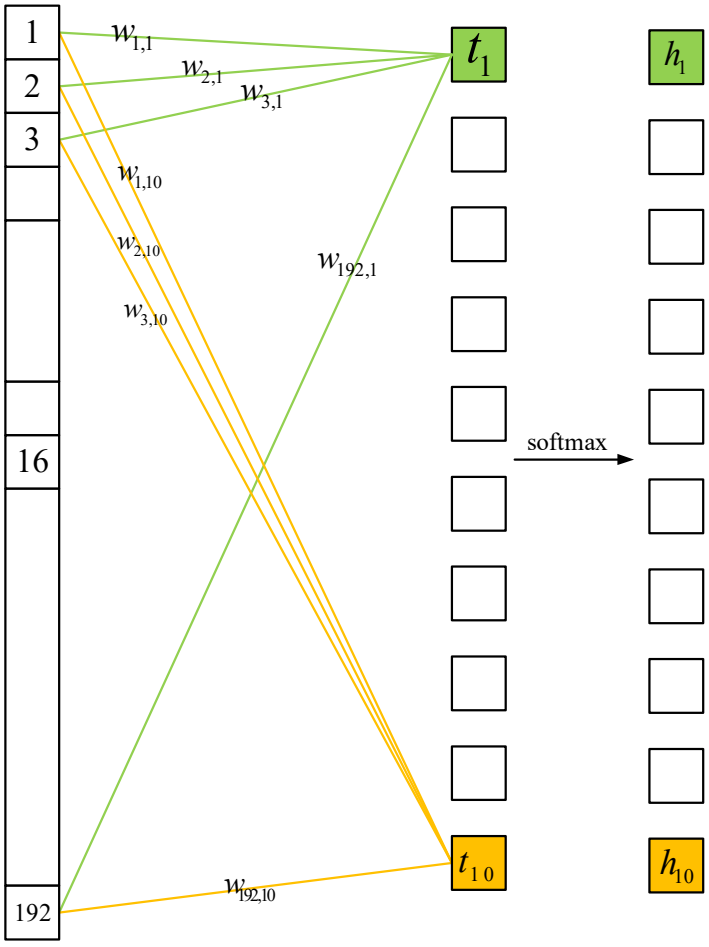
这里有几点需要理解，偏执 b 就是一个数，所以 $g+b$ 是一个矩阵加上一个数。那么 L=2 层有几个偏执呢：一个通道输出只有一个偏执，所以 L=2 层有 6 个输出也就是有 6 个偏执项（6 个变量数）。Sigmoid 之后的输出就得到了 L=2 层的第一个特征矩阵输出。同理对 L=2 层第二个 24×24 的特征矩阵：输入 RGB 三通道在和另外三个不同的卷积核进行运算得到第二个输出。

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

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。



全连接层

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:

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

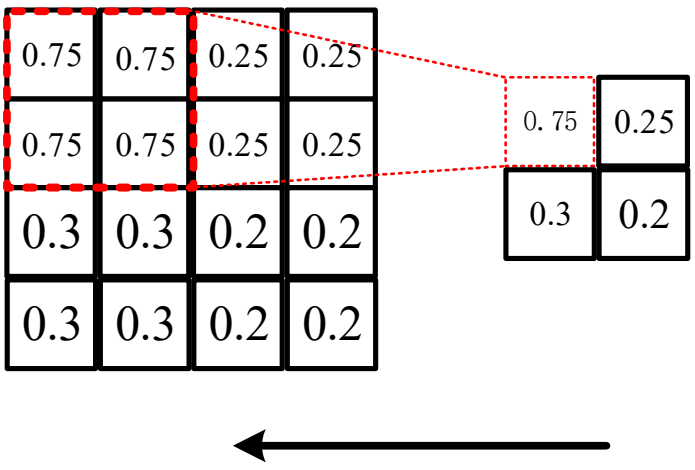
下面从最后一层推导$l-1$层残差，其中c_l表示l层有多少个输出:

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

所以得出残差:

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

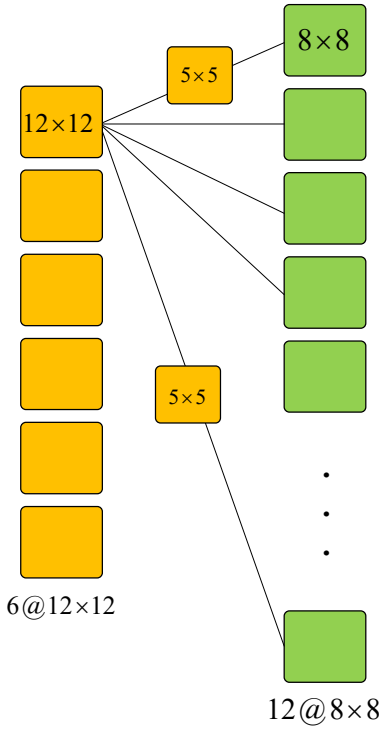
在往前，从 L=6 到 L=5 时 reshape 比较简单，将 L=6 层多有残差按照对应关系变成矩阵就好。从 L=5 到 L=4 时池化的反过程，池化时是取平均值，所以反向时，我们将 L=5 的一个残差值变成 L=4 层的对应四个残差值。一般是将这四个残差值设为一样的。如图所示:



反向池化传播

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

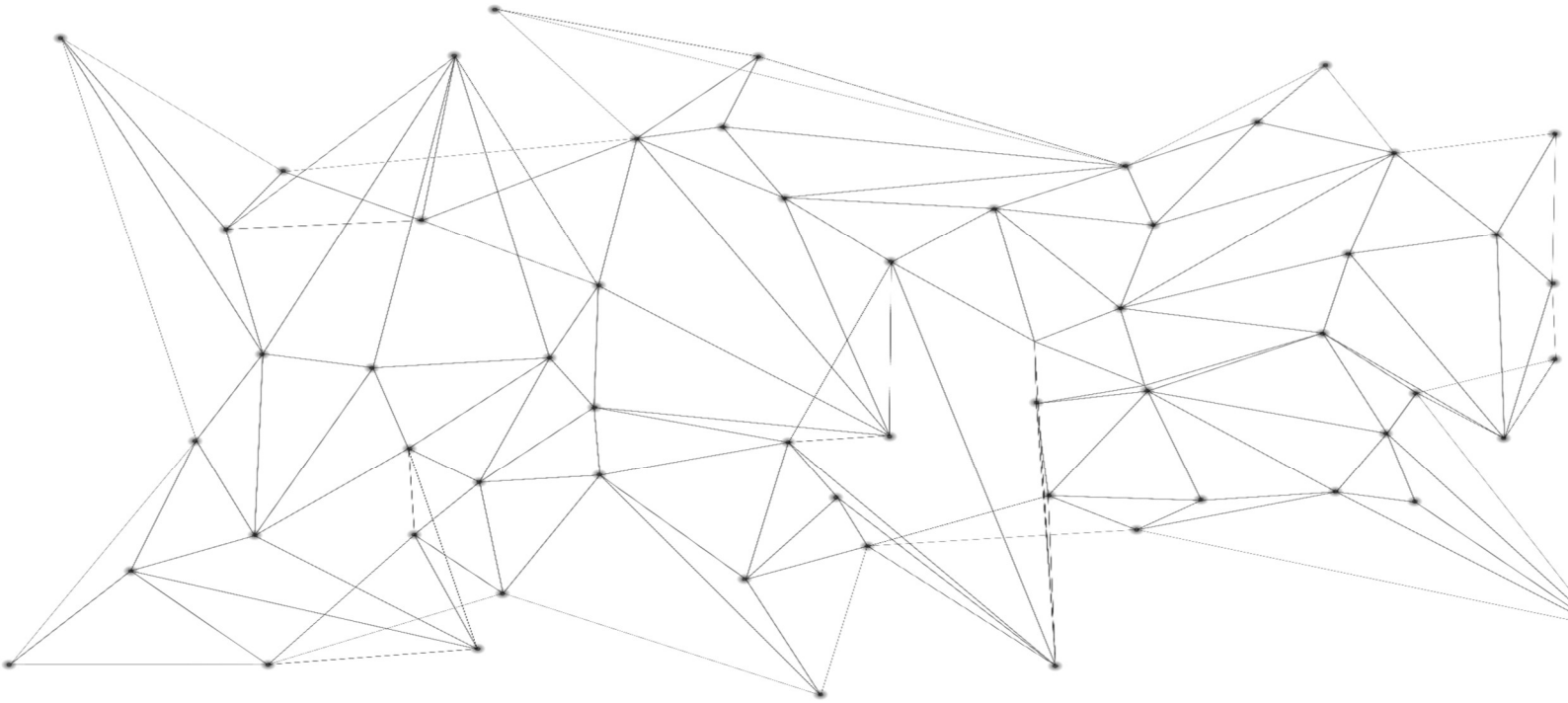
这里用的一种求解 L=3 层的残差的方法是也用卷积方法来计算，我们先将8×8 的输出进行补零操作，我们将8×8 输出上下左右都补四圈零，这样就将8×8 输出变为了16×16 的输出(为什么是 16，上面 padding 的时候介绍了，是根据输出，卷积核大小算出来的)，然后将这个值和对应的5×5 卷积核进行卷积操作（这里卷积核有些介绍好像说需要进行一个 180 度旋转之后在进行卷积）。这样就得到了12×12 的残差输出。但是我们还要考虑一个问题就是一个12×12 的输入要和 12 个8×8 输出都有卷积运算的，因此反向的时候也是。12 个8×8 的残差分别和对应卷积核运算得到 12 个12×12 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。

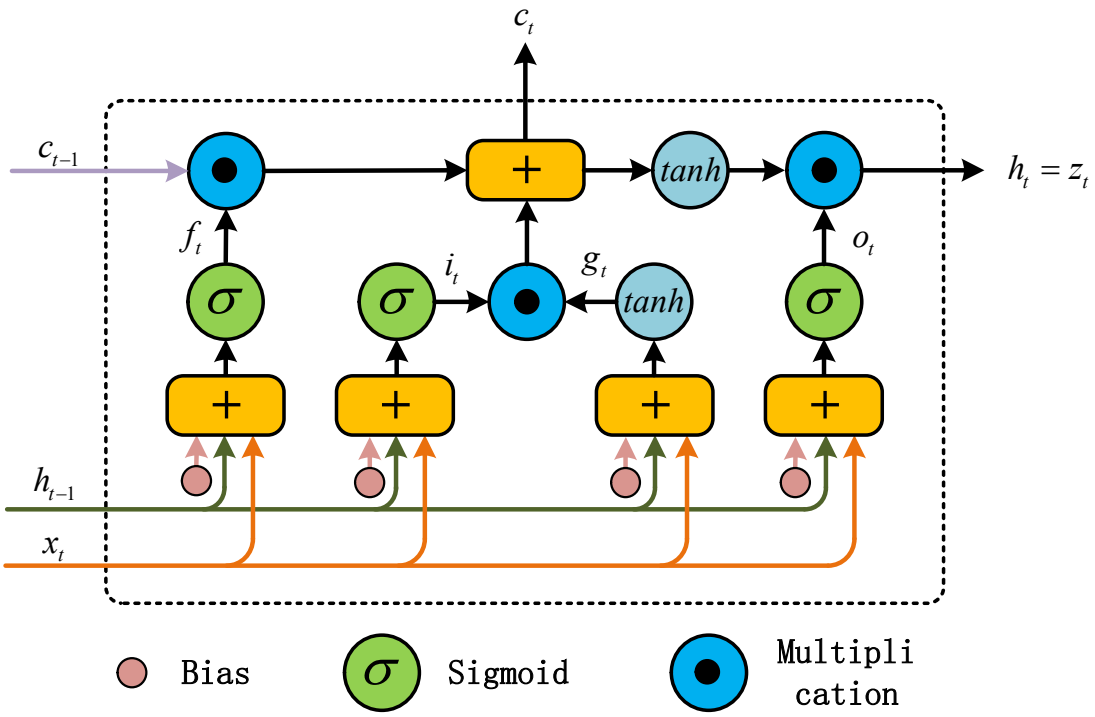
到这里就介绍完了完整的 CNN 的正向传播及反向传播算法的分析和推理。

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$\sigma(x) = (1+e^{-x})^{-1}$是 sigmoid 函数，$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$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$\delta o_t, \delta c_t$。

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

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

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

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

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$\delta c_t$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$\delta W_t, \delta h_{t-1}$：

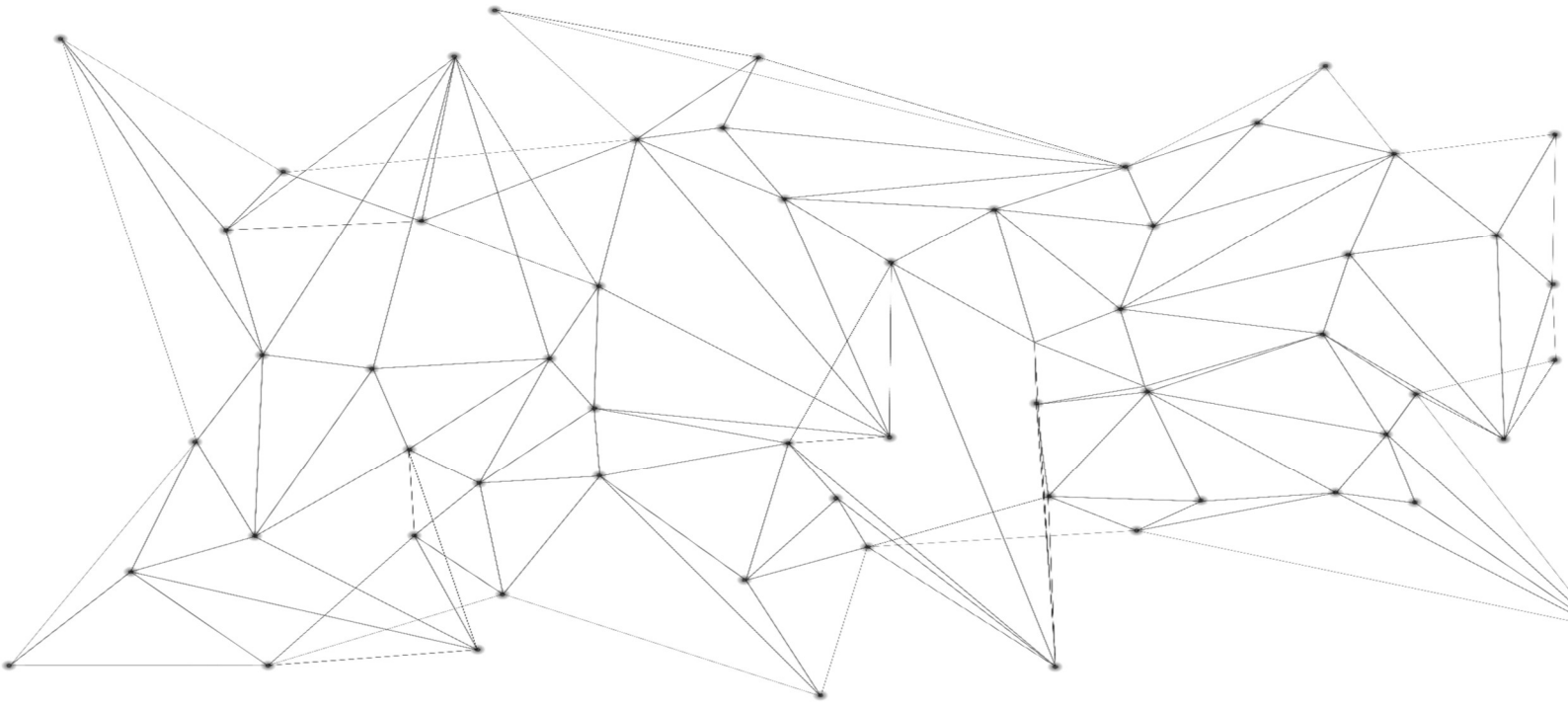
$$\delta I_t = W^T \times \delta z_t$$

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。



Faster-RCNN