第三章：第1部分

关键词：神经网络。前向传播计算。后向传播计算。神经元。最大边界损失。梯度检查。Xavier参数初始化法。学习率。Adagrad。

该笔记介绍了单层和多层的神经网络，以及它们如何用来做分类任务。然后讨论了利用被称作反向传播法的分布梯度下降法来训练网络。我们将会看到链式法则如何用来更新参数。在严谨的数学讨论之后，我们将会探讨一些在实践中训练神经网络的小技巧，这包括：（非线形）神经元、梯度检查、Xavier参数初始化法、学习率、Adagrad等。最后，我们会引出利用循环神经网络来做语言模型。

1. 神经网络：基础

基于之前的讨论，许多数据无法线形可分，导致效果有限，因此需要非线性的分类器。神经网络是拥有非线性决策边界的分类起家族，如图1。现在我们知道了神经网络可以产生怎样的决策边界，接下来看看它是怎么实现的。

1 神经元

神经元是一个广义的计算单元，输入n个变量，输出1个结果。通过控制参数（也叫做权重）来控制输出的不同。最有名的神经元是“sigmoid”或"二元logistic回归"单元。这个单元需要n维的输入向量x，产生一个标量激活值a。这个神经元通常和n维权重向量和1维偏置标量联系在一起。其输出可以用下式表示

$$a = \frac{1}{1+\exp(-w^Tx+b)}$$

我们也可以将w和b写在一起，等价于下式。

$$a = \frac{1}{1+\exp(-[w^T \quad b]\cdot[x \quad 1])}$$

这个式子可以可视化成图2的样子。

1.2 单层神经元

接着上面的话题，我们扩展到多个神经元，将输入x作为许多个这样的神经元的输入，如图3。

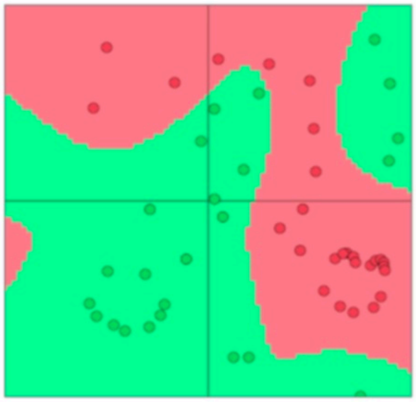

Figure 1: We see here how a non-linear decision boundary separates the data very well. This is the prowess of neural networks.

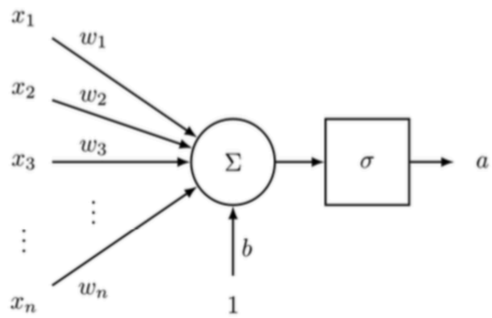

Figure 2: This image captures how in a sigmoid neuron, the input vector x is first scaled, summed, added to a bias unit, and then passed to the squashing sigmoid function.

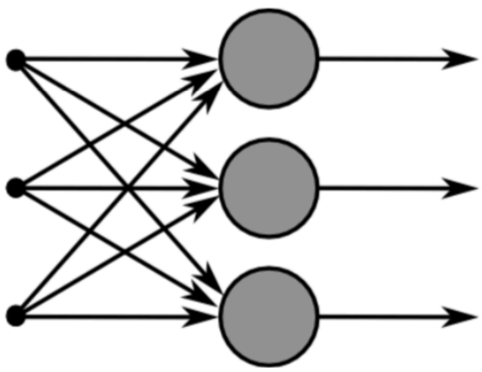

Figure 3: This image captures how multiple sigmoid units are stacked on the right, all of which receive the same input x.

如果我们把不同的神经元用权重$\{w^{(1)},\cdots,w^{(m)}\}$和偏置$\{b_1,\cdots,b_m\}$来表达，可以将每个神经元的激活值表示成

$$
\begin{aligned}
a_1 &= \frac{1}{1+\exp(w^{(1)T}x+b_1)} \\
&\vdots \\
a_m &= \frac{1}{1+\exp(w^{(m)T}x+b_m)}
\end{aligned}
$$

为了简化表达，我们用下面的缩写来替代，进而能够更加方便的表达复杂的网络。

$$\sigma(z) = \begin{bmatrix} \frac{1}{1+\exp(z_1)} \\ \vdots \\ \frac{1}{1+\exp(z_m)} \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_m \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \in \mathbb{R}^m$$

$$W = \begin{bmatrix} - & w^{(1)T} & - \\ & \cdots & \\ - & w^{(m)T} & - \end{bmatrix} \in \mathbb{R}^{m\times n}$$

我们可以将输出写成

$$z = Wx + b$$

sigmoid函数输出的激活值可以写成

$$\begin{bmatrix} a^{(1)} \\ \vdots \\ a^{(m)} \end{bmatrix} = \sigma(z) = \sigma(Wx + b)$$

那么，这些激活值想要告诉我们什么？一种说法是，这些激活值可以作为一些指标，标明是否出现了一些特定的特征的加权组合。我们可以使用这些激活值的组合来做一些分类任务。

1.3 前向传播计算

我们已经看到了输入向量$x \in \mathbb{R}^n$是怎样输入到一层sigmoid单元，并且产生激活向量$a \in \mathbb{R}^m$。但是背后的原理的什么？考虑接下来的在NLP中的命名实体识别任务作为例子。

$$"Museums\ in\ Paris\ are\ amazing"$$

我们想要将中心词$"Paris"$分类，判断它是否是一个实体。在这样的例子中，我们不仅仅想要判断在词向量窗口中是否出现了这个词，我们还要将其和窗口中其他的词语之间的关系捕捉到。例如，当$Museums$是第一个词，并且只有in是第二个词时。直接将输入送到softmax函数中，没办法得到这样的非线性决策。相反，我们需要依赖在1.2节介绍的中间层。我们使用另外一个矩阵$U \in \mathbb{R}^{m\times 1}$，利用激活值，来产生一个没有被归一化的分数。

$$s = U^T a = U^T f(Wx + b)$$

其中f是激活函数。

维度分析：如果我们将每个单词，用4维的词向量表示，并且使用5个单词形成的窗口作为输入（和上面的例子一样），这样输入就是$x \in \mathbb{R}^{20}$。如果我们在隐藏层中使用8个sigmoid单元，并且利用了激活函数产生了1个输出得分，这样，$W \in \mathbb{R}^{8\times 20}$, $b \in \mathbb{R}^8$, $U \in \mathbb{R}^{8\times 1}$, $s \in \mathbb{R}$。

如图4，可以看到矩阵和向量中的每个元素，在网络中对应的位置和含义。

1.4 最大的边界的目标函数

和大多数机器学习模型一样，神经网络也需要一个优化目标，一个衡量错误或者好坏的指标，进而我们可以去最小化或是最大化。我们会讨论一个流行的错误指标，称为最大的边界目标。使用这个目标的想法是为了保证，"真"的数据的得分要比"假"的数据的得分要高。

利用之前的例子，我们把$”Museums\ in\ Paris\ are\ amazing”$这句话标称"真"的得分记做$s$，并且把$”Not\ all\ museums\ in\ Paris”$标记成"假"的分数记做$s_c$。（下标$c$表示这个窗口中的数据是不正常的）

我们的目标函数是最大化$(s - s_c)$或者是最小化$(s_c - s)$。然而，我们修改目标，仅仅在$s_c > s \Rightarrow (s_c - s) > 0$的情况下算作出错。这个原理是因为，我们仅仅关心那些"真"的数据的得分要高于"假"的数据的得分，其余的都不考虑。因此，我们希望，当$s_c > s$时，误差为$s_c - s$，而反过来误差为0。因此，我们的优化目标是

$$\text{minimize } J = \max(s_c - s, 0)$$

然而，上面的优化目标，在某种程度上，还不是很有把握的产生一个安全的边界。我们希望"真"的数据的得分比"假"的数据的得分要高出一定的范围。换句话说，我们希望当$(s - s_c < \Delta)$时，应该算作误差，而不仅仅是当$(s - s_c < 0)$。因此，我们修改优化目标为

$$\text{minimize } J = \max(\Delta + s_c - s, 0)$$

我们把边界放缩一下，使得$\Delta = 1$，并且使的其他优化中的参数不会影响到表现。更多的信息，可以查阅函数间隔与几何间隔的话题，通常在讲支撑向量机中会提到。最后，在所有训练的窗口中，我们定义以下优化目标。

$$\text{minimize } J = \max(1 + s_c - s, 0)$$

上式中，$s_c = U^T f(W x_c + b)$， $s = U^T f(W x + b)$。

1.5 后向传播的训练方法：单个元素

在本节，我们讨论，当1.4节中的损失函数J大于0时，如何训练模型中不同的参数。当损失等于0时，不需要更新任何的参数。因此，我们使用梯度下降法（或者是SGD）来更新参数。我们需要所有参数的梯度信息作为更新参数的条件，如下式

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_{\theta^{(t)}} J$$

后向传播通过使用微分的链式法则，来计算损失函数对每个前向传播中的参数的梯度的方法。为了更加深入的理解，我们采用图5中一个基本的网络结构来运行后向传播方法。

我们使用只有一层隐藏层和一个输出单元的神经网络。为了能够更好的表达网络，我们先给出一些定义。

- x_i是输入到神经网络的一个输入向量。
- s是神经网络的输出。
- 每层（包括输入和输出层）都有神经元会接收到输入，并且产生输出。第k层中的第j个神经元会接收

到输入中的标量$z_j^{(k)}$，并且产生激活值$a_j^{(k)}$，这是标量。

- 我们将在$z_j^{(k)}$位置上计算的后向传播误差称之为$\delta_j^{(k)}$。
- 第一层表示输入层，而不是第一个隐藏层。因此，对于输入层而言，$x_j = z_j^{(1)} = a_j^{(1)}$。
- $W^{(k)}$是一个转移矩阵，将来自第k层的输出转移到第$k+1$层的输入。因此我们用以下的定义，将1.3小节中提到的矩阵表示成$W^{(1)} = W$（layer1->layer2的转移矩阵），$W^{(2)} = U$。

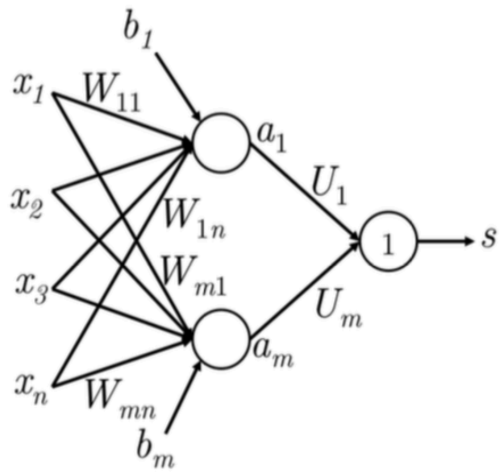

Figure 4: This image captures how a simple feed-forward network might compute its output.

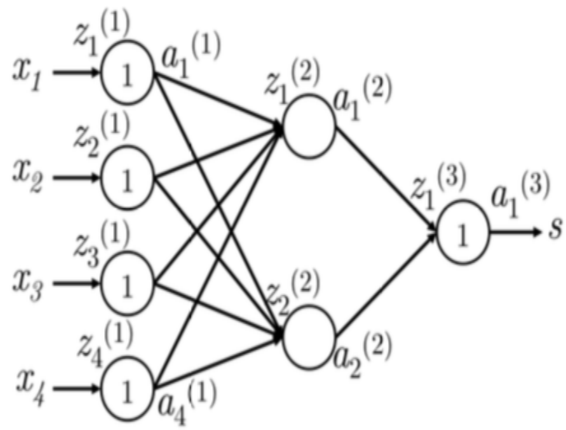

Figure 5: This is a 4-2-1 neural network where neuron j on layer k receives input $z_j^{(k)}$ and produces activation output $a_j^{(k)}$.

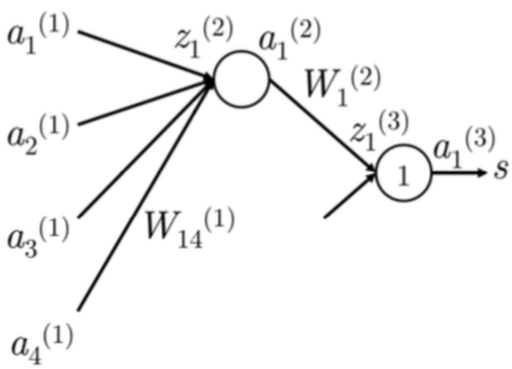

Figure 6: This subnetwork shows the relevant parts of the network required to update $W_{ij}^{(1)}$

那我们开始吧。假设损失函数是$J = (1 + s_c - s)$是一个正数（$J = \max(1 + s_c - s, 0) > 0$），并且我们希望更新参数$W_{14}^{(1)}$（如图5和图6显示，$W_{14}^{(1)}$是layer1和layer2之间的权重，从layer2的neuron1指回layer1的neuron4）。我们发现$W_{14}^{(1)}$仅仅和$z_1^{(2)}$进而是$a_1^{(2)}$有关。这是一个理解后向传播很重要的点，即参数的后向传播梯度仅仅受到该参数有贡献的值的影响。这里，$a_1^{(2)}$是在计算得分的前向传播过程中，和$W_1^{(2)}$相乘。我们可以从最大的边界损失中可以看出。

$$\frac{\partial J}{\partial s} = -\frac{\partial J}{\partial s_c} = -1$$

因此，我们可以忽略从J到s的导数，直接考虑从s到W的导数$\frac{\partial s}{\partial W_{ij}^{(1)}}$，因此

$$\begin{aligned}
\frac{\partial s}{\partial W_{ij}^{(1)}} &= \frac{\partial W^{(2)} a^{(2)}}{\partial W_{ij}^{(1)}} = \frac{\partial W_i^{(2)} a_i^{(2)}}{\partial W_{ij}^{(1)}} = W_i^{(2)} \frac{\partial a_i^{(2)}}{\partial W_{ij}^{(1)}} \\
\Rightarrow W_i^{(2)} \frac{\partial a_i^{(2)}}{\partial W_{ij}^{(1)}} &= W_i^{(2)} \frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial W_{ij}^{(1)}} \\
&= W_i^{(2)} \frac{f(z_i^{(2)})}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial W_{ij}^{(1)}} \\
&= W_i^{(2)} f'(z_i^{(2)}) \frac{\partial z_i^{(2)}}{\partial W_{ij}^{(1)}} \\
&= W_i^{(2)} f'(z_i^{(2)}) \frac{\partial}{\partial W_{ij}^{(1)}} (b_i^{(1)} + a_1^{(1)} W_{i1}^{(1)} + a_2^{(1)} W_{i2}^{(1)} + a_3^{(1)} W_{i3}^{(1)} + a_4^{(1)} W_{i4}^{(1)}) \\
&= W_i^{(2)} f'(z_i^{(2)}) \frac{\partial}{\partial W_{ij}^{(1)}} (b_i^{(1)} + \sum_k a_k^{(1)} W_{ik}^{(1)}) \\
&= W_i^{(2)} f'(z_i^{(2)}) a_j^{(1)} \\
&= \delta_i^{(2)} \cdot a_j^{(1)}
\end{aligned}$$

这一段比较复杂，我们一步一步来看。

现在的目标是求得分s对参数$W^{(1)}$的偏导数。

$$\begin{aligned}
\frac{\partial s}{\partial W^{(1)}} &= \frac{\partial}{\partial W^{(1)}} (W^{(2)})^T a^{(2)} \\
&= \frac{\partial}{\partial W^{(1)}} (W^{(2)})^T f(z^{(2)}) \\
&= \frac{\partial}{\partial W^{(1)}} (W^{(2)})^T f(W^{(1)} x + b)
\end{aligned}$$

可以看到$W^{(1)}$出现在了激活函数中，与输入x有关系。

以$W_{ij}^{(1)}$为例，$W_{ij}^{(1)}$是layer2的第i个神经元指回layer1的第j个神经元的权重，因此只与layer2的第i个神经元有关，因此只会影响$a_i^{(2)}$。所以

$$
\begin{aligned}
\frac{\partial}{\partial W_{ij}^{(1)}}(W^{(2)})^T f(W^{(1)}x+b) &= \frac{\partial}{\partial W_{ij}^{(1)}} W_i^{(2)} f(W_{i:}^{(1)}x+b_i) \\
&= W_i^{(2)} \frac{\partial}{\partial W_{ij}^{(1)}} f(W_{i:}^{(1)}x+b_i) \\
&= W_i^{(2)} f'(W_{i:}^{(1)}x+b_i) \frac{\partial}{\partial W_{ij}^{(1)}} (W_{i:}^{(1)}x+b_i) \\
&= W_i^{(2)} f'(W_{i:}^{(1)}x+b_i) \frac{\partial}{\partial W_{ij}^{(1)}} \left(\sum_{j=1}^{n} W_{ij}^{(1)} x_j + b_i \right) \\
&= W_i^{(2)} f'(W_{i:}^{(1)}x+b_i) x_j \\
&= W_i^{(2)} f'(W_{i1}^{(1)}x_1 + \cdots W_{in}^{(1)}x_n + b_i) x_j \\
&= \delta_i^{(2)} x_j
\end{aligned}
$$

至此，得分s关于第一层权重矩阵$W_{ij}^{(1)}$的偏导数的公式就写出来了。

我们定义$\delta_i^{(2)} = W_i^{(2)} f'(W_{i1}^{(1)}x_1 + \cdots W_{in}^{(1)}x_n + b_i)$为local error signal，同时$x_j$为local input signal。所以整个对$W_{ij}^{(1)}$的梯度被化简为$\delta_i^{(2)} \cdot a_j^{(1)} = \delta_i^{(2)} \cdot x_j$，其中$\delta_i^{(2)}$定义为从layer2的第$i$个神经元，朝着输入的方向传回来的误差信号，而$a_j^{(1)}$通过$W_{ij}^{(1)}$加权后，输入到layer2的第$i$个神经元。

接下来，利用图6可以更好地讨论在后向传播中的"误差分享"解释。假设我们准备更新$W_{14}^{(1)}$。

1. 从输出值$a_1^{(3)}$的位置开始，将等于1的误差信号往回传。
2. 然后用神经元的局部梯度，即把$z_1^{(3)}$映射到$a_1^{(3)}$的函数的梯度，乘以传到这的误差信号。此时局部梯度等于1，因此在layer3的第1个位置上（也是唯一一个，因为只有一个输出值）的local error signal $\delta_1^{(3)} = 1$。
3. 此时，值为1的误差信号传到了$z_1^{(3)}$。现在我们需要把这个误差信号公平的分享到layer2的输出值$a_1^{(2)}$。
4. 这个程度是（在$z_1^{(3)}$误差信号等于$\delta_1^{(3)}$）$\times W_1^{(2)} = W_1^{(2)}$。因此，传到layer2的输出值$a_1^{(2)}$接收到的误差等于$W_1^{(2)}$。
5. 像我们在第二步中的做法一样，我们需要将误差在神经元内部传递，即从$z_1^{(2)}$到$a_1^{(2)}$。通过将该神经元的局部梯度，即$f'(z_1^{(2)})$，去乘以传递到$a_1^{(2)}$的误差信号，从而将误差信号传到$z_1^{(2)}$。
6. 因此，在$z_1^{(2)}$位置上的得到的误差信号为$f'(z_1^{(2)})W_1^{(2)}$，记做$\delta_1^{(2)}$。
7. 最后，我们需要公平的把误差信号向前传。如果要传到$W_{14}^{(1)}$，则需要将在$z_1^{(2)}$的误差$\delta_1^{(2)}$，乘以$a_4^{(1)}$，这是如我们在第4步中做的很像。
8. 因此，损失函数对于的梯度可以表达为$a_4^{(1)}\delta_1^{(2)} = a_4^{(1)} f'(z_1^{(2)})W_1^{(2)}$。

注意到，利用后向传播的方法得到的结果，和我们之前显示地使用求导的方法得到的结果完全一样。因此，我们可以通过求导的链式法则，也可以使用误差分享和传递的方法，来计算误差对于每个网络中参数的梯度。两种方法可以得到完全一样的结果。

偏置的更新：偏置项（例如$b_1^{(1)}$）在数学上和其他对神经元的输入（$z_1^{(2)}$）有贡献的权重一样，在前向传播过程中，它的权重等于1。这样，对于第k层的第i个神经元而言，偏置项的梯度为$\delta_i^{(k)}$。例如，如果我们要更新b_1^1，而不是之前讨论的W_{14}^1时，梯度将会是$f'(z_1^{(2)})W_1^{(2)}$。

$$
\begin{aligned}
\frac{\partial s}{\partial b_i} &= \frac{\partial}{\partial b_i}(W^{(2)})^T f(W^{(1)}x + b) \\
&= W_i^{(2)} \frac{\partial}{\partial b_i} f(W_{i:}^{(1)} x + b_i) \\
&= W_i^{(2)} f'(W_{i:}^{(1)} x + b_i) \frac{\partial}{\partial b_i}(W_{i:}^{(1)} x + b_i) \\
&= W_i^{(2)} f'(W_{i:}^{(1)} x + b_i) \\
&= W_i^{(2)} f'(W_{i1}^{(1)} x_1 + \cdots W_{in}^{(1)} x_n + b_i) \\
&= \delta_i^{(2)}
\end{aligned}
$$

将$\delta^{(k)}$传播到$\delta^{(k-1)}$的一般性步骤

1. 如图7，从第k层的第i个神经元$z_i^{(k)}$，传播而来的误差$\delta_i^{(k)}$。
2. 我们通过路径的权重$W_{ij}^{(k-1)}$，将其后向传播至$a_j^{(k-1)}$。
3. 因此，在$a_j^{(k-1)}$接收到的误差等于$\delta_i^{(k)} W_{ij}^{(k-1)}$。
4. 然而，如图8，$a_j^{(k-1)}$可能被前向传播至下一层的多个节点。它会接收到来自第k层中的第m个节点$z_m^{(k)}$传回来的误差。

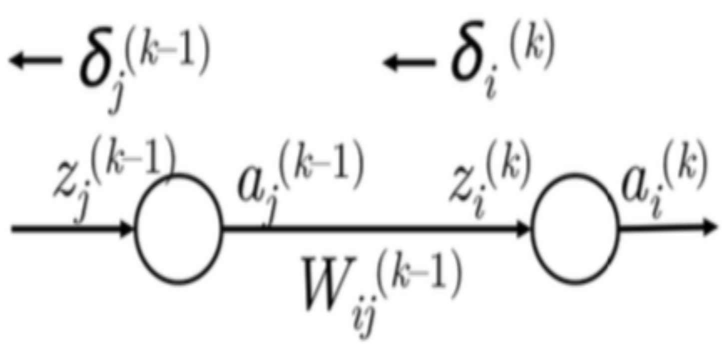

Figure 7: Propagating error from $\delta^{(k)}$ to $\delta^{(k-1)}$

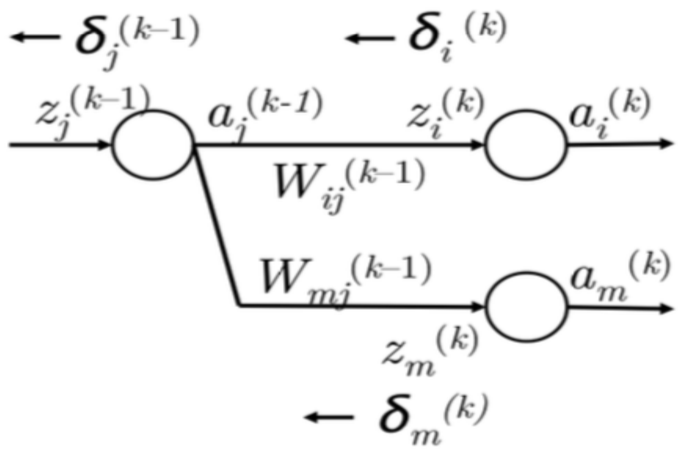

Figure 8: Propagating error from $\delta^{(k)}$ to $\delta^{(k-1)}$

5. 因此，$a_j^{(k-1)}$接收到的误差等于$\delta_i^{(k)}W_{ij}^{(k-1)}+\delta_m^{(k)}W_{mj}^{(k-1)}$。
6. 事实上，我们可以写成一般形式$\sum_i \delta_i^{(k)}W_{ij}^{(k-1)}$。
7. 现在我们有了在$a_j^{(k-1)}$的正确的误差，我们将其在$k-1$层的第j个神经元内部传递，通过乘以局部梯度$f'(z_j^{(k-1)})$。
8. 因此，传递到$z_j^{(k-1)}$的误差$\delta_j^{(k-1)}=f'(z_j^{(k-1)})\sum_i \delta_i^{(k)}W_{ij}^{(k-1)}$

1.6 后向传播的训练方法：向量化

目前，我们已经讨论了如何给每个模型中的参数计算梯度。这里，我们将上面的过程一般化，一次性的更新所有的权重矩阵和偏置向量。注意到，简单的将之前讨论的模型扩展之后，可以帮助我们建立起误差的传播。

给定一个参数$W_{ij}^{(k)}$，我们指出误差梯度等于$\delta_i^{(k+1)}\cdot a_j^{(k)}$。提醒一下，$W^{(k)}$时将$a^{(k)}$映射到$z^{(k+1)}$的矩阵。我们可以建立起整个矩阵$W^{(k)}$的误差梯度，即

$$\nabla_{W^{(k)}}=\begin{bmatrix}\delta_1^{(k+1)}a_1^{(k)} & \delta_1^{(k+1)}a_2^{(k)} & \cdots \\ \delta_2^{(k+1)}a_1^{(k)} & \delta_2^{(k+1)}a_2^{(k)} & \cdots \\ \vdots & \vdots & \ddots\end{bmatrix}=\delta^{(k+1)}a^{(k)T}$$

因此，我们可以用传递到矩阵$W^{(k)}$的误差向量$\delta^{(k+1)}$和给矩阵输入的激活向量$a^{(k)}$，通过这两个向量的外积(outer product)，我们可以写出整个矩阵的误差梯度。

现在，我们看看如何计算误差向量$\delta^{(k)}$。利用图8，我们知道了误差向量$\delta_j^{(k)}=f'(z_j^{(k)})\sum_i \delta_i^{(k+1)}W_{ij}^{(k)}$。我们可以将其写成矩阵的形式。

$$\delta^{(k)}=f'\left(z^{(k)}\right)\circ\left(W^{(k)T}\delta^{(k+1)}\right)$$

上面的形式中，$\circ$表示元素与元素相乘（element-wise product）$\left(\circ:\mathbb{R}^N\times\mathbb{R}^N\to\mathbb{R}^N\right)$。

计算效率：按元素更新，或按向量更新，我们必须意识到，在matlab或者numpy中，用向量化的方式开发，要快很多。因此，我们实践中要用向量化的方式开发。更进一步，我们在后向传播中，应该减少多余的计算。因此，我们必须保证，当利用$\delta^{(k+1)}$来更新$W^{(k)}$时，我们要保存$\delta^{(k+1)}$，以便之后计算$\delta^{(k)}$，并且要重复$(k-1)\ldots(1)$。这样一个递归的过程，使得后向传播非常快。