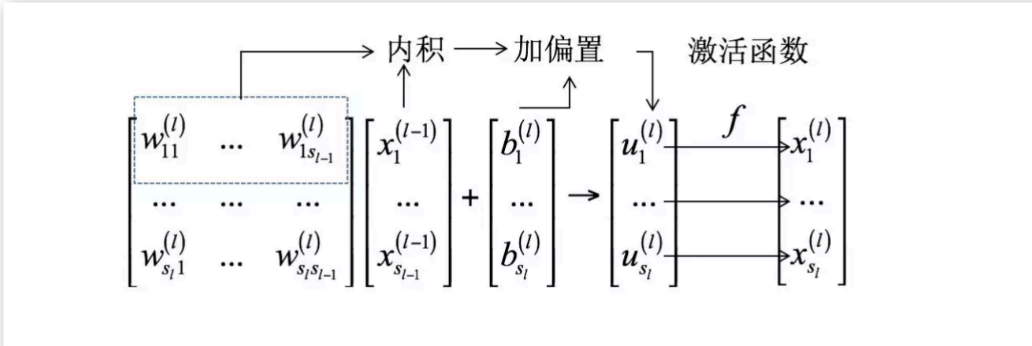
根据上面的结论可以方便的推导出神经网络的求导公式。假设神经网络有$n_l$层，第l层神经元个数为$S_l$。第1层从第l-1层接收的输入向量为$x^{(l-1)}$，本层的权重矩阵为$W^{(l)}$，偏置向量为$b^{(l)}$，输出向量为$x^{(l)}$。该层的输出可以写成如下矩阵形式：

$$u^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)}$$
$$x^{(l)} = f\left(u^{(l)}\right)$$

其中$W^{(l)}$是$s_l x s_{l-1}$的矩阵，$u^{(l)}$和$b^{(l)}$是$s_l$维的向量。神经网络一个层实现的变换如下图所示：




如果将神经网络按照各个层展开，最后得到一个深层的复合函数，将其代入欧氏距离损失函数，依然是一个关于各个层的权重矩阵和偏置向量的复合函数：

计算某一层时需要依靠它外面贴近的那一层

$$L = \frac{1}{2}\left\| f\left(W^{(L)} \underbrace{f\left(W^{(l-1)} f\left(W^{(l-2)}\left(\ldots f\left(W^{(1)}x + b^{(1)}\right)\right) + b^{(L-1)}\right) + b^{(L-1)}\right)}_{\text{先从最外层算起}} + b^{(L)}\right) - y \right\|^2$$

要计算某一层的权重矩阵和偏置向量的梯度，只能依赖于它紧贴着的外面那一层变量的梯度值，通过一次复合函数求导得到。

根据定义，$w^{(l)}$和$b^{(l)}$是目标函数的自变量，$u^{(l)}$和$x^{(l)}$可以看成是它们的函数。根据前面的结论，损失函数对权重矩阵的梯度为：

$$\nabla_{W^{(l)}} L = \left(\nabla_{u^{(l)}} L\right)\left(x^{(l-1)}\right)^T$$

对偏置向量的梯度为：

$$\nabla_{b^{(l)}} L = \nabla_{u^{(l)}} L$$

现在的问题是，梯度$\nabla_{u^{(l)}} L$怎么计算？我们分两种情况讨论，如果第l层是输出层，在这里只考虑对单个样本的损失函数，根据上一节推导的结论，这个梯度为：

$$\nabla_{u^{(l)}} L = \left(\nabla_{x^{(l)}} L\right) \odot f'\left(u^{(l)}\right) = \left(x^{(l)} - y\right) \odot f'\left(u^{(l)}\right)$$

这就是输出层的神经元输出值与期望值之间的误差。这样我们得到输出层权重的梯度为：

$$\nabla_{W^{(l)}} L = \left(x^{(l)} - y\right) \odot f'\left(u^{(l)}\right)\left(x^{(l-1)}\right)^T$$

等号右边第一个乘法是向量对应元素乘；第二个乘法是矩阵乘，在这里是列向量与行向量的乘积，结果是一个矩阵，尺寸刚好和权重矩阵相同。损失函数对偏置项的梯度为：

$$\nabla_{b^{(l)}} L = \left(x^{(l)} - y\right) \odot f'\left(u^{(l)}\right)$$

下面考虑第二种情况。如果第l层是隐含层，则有：

$$u^{(l+1)} = W^{(l+1)}x^{(l)} + b^{(l+1)} = W^{(l+1)} f\left(u^{(l)}\right) + b^{(l+1)}$$

假设梯度$\nabla_{u^{(l+1)}} L$已经求出，根据前面的结论，有：

$$\nabla_{u^{(l)}} L = \left(\nabla_{x^{(l)}} L\right) \odot f'\left(u^{(l)}\right) = \left(\left(W^{(l+1)}\right)^T \nabla_{u^{(l+1)}} L\right) \odot f'\left(u^{(l)}\right)$$

这是一个递推的关系，通过$\nabla_{u^{(l+1)}} L$可以计算出$\nabla_{u^{(l)}} L$，递推的终点是输出层，而输出层的梯度值我们之前已经算出。由于根据$\nabla_{u^{(l)}} L$可以计算出$\nabla_{W^{(l)}} L$和$\nabla_{b^{(l)}} L$，因此可以计算出任意层权重与偏置的梯度值。

为此我们定义误差项为损失函数对临时变量u的梯度：

$$\delta^{(l)} = \nabla_{\mathrm{u}^{(l)}} L = \begin{cases} \left(\mathrm{x}^{(l)} - \mathrm{y}\right) \odot f'\left(\mathrm{u}^{(l)}\right) & l = n_l \\ \left(\mathrm{W}^{(l+1)}\right)^{\mathrm{T}} \left(\delta^{(l+1)}\right) \odot f'\left(\mathrm{u}^{(l)}\right) & l \neq n_l \end{cases}$$

向量$\delta^{(l)}$的尺寸和本层神经元的个数相同。这是一个递推的定义，$\delta^{(l)}$依赖于$\delta^{(l+1)}$，递推的终点是输出层，它的误差项可以直接求出。

根据误差项可以方便的计算出对权重和偏置的偏导数。首先计算输出层的误差项，根据他得到权重和偏置项的梯度，这是起点；根据上面的递推公式，逐层向前，利用后一层的误差项计算出本层的误差项，从而得到本层权重和偏置项的梯度。

单个样本的反向传播算法在每次迭代时的流程为：

1.正向传播，利用当前权重和偏置值，计算每一层对输入样本的输出值
2.反向传播，对输出层的每一个节点计算其误差：

$$\delta^{(n_l)} = \left(\mathrm{x}^{(n_l)} - \mathrm{y}\right) \odot f'\left(\mathrm{u}^{(n_l)}\right)$$

3.对于$l = n_l - 1, \ldots, 2$的各层，计算第l层每个节点的误差：

$$\delta^{(l)} = \left(\mathrm{W}^{(l+1)}\right)^{\mathrm{T}} \delta^{(l+1)} \odot f'\left(\mathrm{u}^{(l)}\right)$$

4.根据误差计算损失函数对权重的梯度值：

$$\nabla_{\mathrm{W}^{(l)}} L = \delta^{(l)} \left(\mathrm{x}^{(l-1)}\right)^{\mathrm{T}}$$

对偏置的梯度为：

$$\nabla_{\mathrm{b}^{(l)}} L = \delta^{(l)}$$

5.用梯度下降法更新权重和偏置：

$$\mathrm{W}^{(l)} = \mathrm{W}^{(l)} - \eta \nabla_{\mathrm{W}^{(l)}} L$$
$$\mathrm{b}^{(l)} = \mathrm{b}^{(l)} - \eta \nabla_{\mathrm{b}^{(l)}} L$$

实现时需要在正向传播时记住每一层的输入向量$\mathrm{x}^{(l-1)}$，本层的激活函数导数值$f'\left(\mathrm{u}^{(l)}\right)$。

神经网络的训练算法可以总结为：

复合函数求导+梯度下降法

训练算法有两个版本：批量模式和单样本模式。批量模式每次梯度下降法迭代时对所有样本计算损失函数值，计算出对这些样本的总误差，然后用梯度下降法更新参数；单样本模式是每次对一个样本进行前向传播，计算对该样本的误差，然后更新参数，它可以天然的支持增量学习，即动态的加入新的训练样本进行训练。

在数学中，向量一般是列向量，但在编程语言中，向量一般按行存储，即是行向量，因此实现时计算公式略有不同，需要进行转置。正向传播时的计算公式为：

$$\mathrm{u}^{(l)} = \mathrm{x}^{(l-1)}\mathrm{W}^{(l)} + \mathrm{b}^{(l)}$$

感兴趣的读者可以阅读各个开源库，对比它们的计算公式。反向传播时的计算公式为：

$$\delta^{(l)} = \delta^{(l+1)}\left(\mathrm{W}^{(l+1)}\right)^{\mathrm{T}} \odot f'\left(\mathrm{u}^{(l)}\right)$$

对权重矩阵的计算公式为：

$$\nabla_{\mathrm{W}} L = \left(\mathrm{x}^{(l-1)}\right)^{\mathrm{T}} \delta^{(l)}$$

这些向量都是行向量。