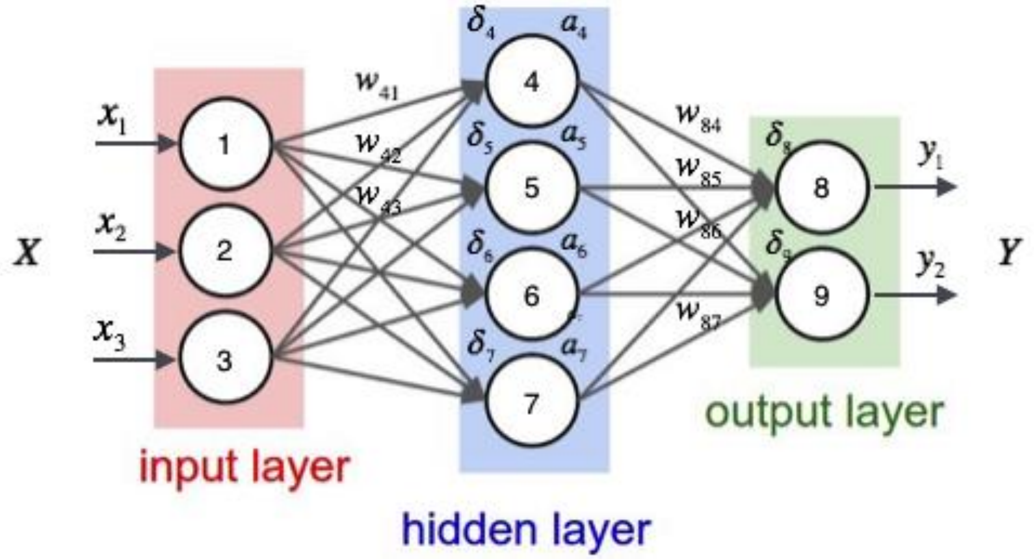
1、反向传播算法(BP)：

我们假设每个训练样本为$(\vec{x}, \vec{t})$，其中向量$\vec{x}$是训练样本的特征，而$\vec{t}$是样本的目标值。



首先，我们根据上一节介绍的算法，用样本的特征$\vec{x}$，计算出神经网络中每个隐藏层节点的输出a_i，以及输出层每个节点的输出y_i。

然后，我们按照下面的方法计算出每个节点的误差项δ_i：

- 对于输出层节点i，

$$\delta_i = y_i(1 - y_i)(t_i - y_i) \qquad (式3)$$

其中，δ_i是节点i的误差项，y_i是节点i的**输出值**，t_i是样本对应于节点i的**目标值**。举个例子，根据上图，对于输出层节点8来说，它的输出值是y_1，而样本的目标值是t_1，带入上面的公式得到节点8的误差项δ_8应该是：

$$\delta_8 = y_1(1 - y_1)(t_1 - y_1)$$

- 对于隐藏层节点，

$$\delta_i = a_i(1 - a_i) \sum_{k \in outputs} w_{ki}\delta_k \qquad (式4)$$

其中，a_i是节点i的输出值，w_{ki}是节点i到它的下一层节点k的连接的权重，δ_k是节点i的下一层节点k的误差项。例如，对于隐藏层节点4来说，计算方法如下：

$$\delta_4 = a_4(1 - a_4)(w_{84}\delta_8 + w_{94}\delta_9)$$

最后，更新每个连接上的权值：

$$w_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji} \qquad (式5)$$

其中，w_{ji}是节点i到节点j的权重，η是一个成为**学习速率**的常数，δ_j是节点j的误差项，x_{ji}是节点i传递给节点j的输入。例如，权重$w_8 4$的更新方法如下：

$$w_{84} \leftarrow w_{84} + \eta\delta_8 a_4$$

类似的，权重w_{41}的更新方法如下：

$$w_{41} \leftarrow w_{41} + \eta\delta_4 x_1$$

偏置项的输入值永远为1。例如，节点4的偏置项w_{4b}应该按照下面的方法计算：

$$w_{4b} \leftarrow w_{4b} + \eta\delta_4$$

我们已经介绍了神经网络每个节点误差项的计算和权重更新方法。显然，计算一个节点的误差项，需要先计算每个与其相连的下一层节点的误差项。这就要求误差项的计算顺序必须是从输出层开始，然后反向依次计算每个隐藏层的误差项，直到与输入层相连的那个隐藏层。这就是反向传播算法的名字的含义。当所有节点的误差项计算完毕后，我们就可以根据**式5**来更新所有的权重。

2、反向传播算法的推导：（BP）

反向传播算法其实就是链式求导法则的应用。然而，这个如此简单且显而易见的方法，却是在Roseblatt提出感知器算法将近30年之后才被发明和普及的。对此，Bengio这样回应道：

> 很多看似显而易见的想法只有在事后才变得显而易见。

接下来，我们用链式求导法则来推导反向传播算法，也就是上一小节的**式3**、**式4**、**式5**。

前方高能预警——接下来是数学公式重灾区，读者可以酌情阅读，不必强求。

按照机器学习的通用套路，我们先确定神经网络的目标函数，然后用**随机梯度下降**优化算法去求目标函数最小值时的参数值。

我们取网络所有输出层节点的误差平方和作为目标函数：

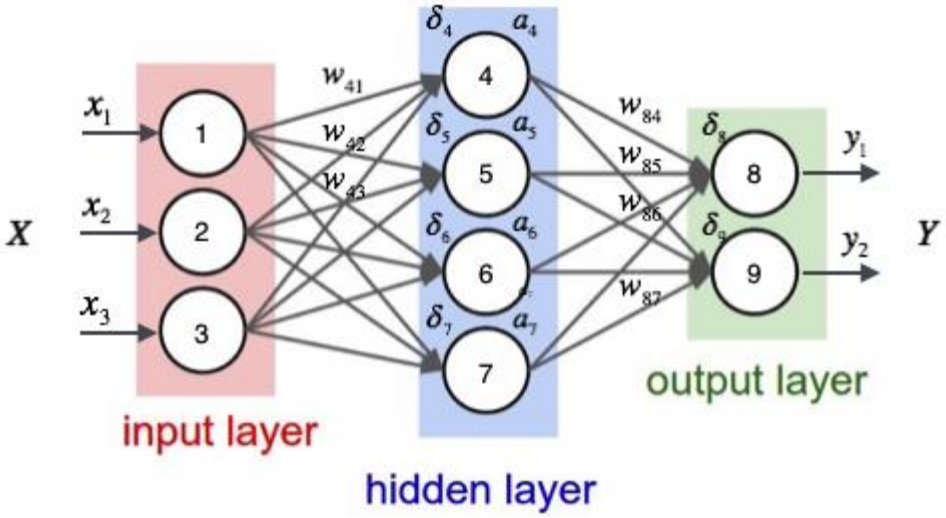
$$E_d \equiv \frac{1}{2}\sum_{i \in outputs}(t_i - y_i)^2$$

其中，E_d表示是样本d的误差。

然后，我们用文章零基础入门深度学习(2) - 线性单元和梯度下降中介绍的**随机梯度下降**算法对目标函数进行优化：

$$w_{ji} \leftarrow w_{ji} - \eta\frac{\partial E_d}{\partial w_{ji}}$$

随机梯度下降算法也就是需要求出误差E_d对于每个权重w_{ji}的偏导数（也就是梯度），怎么求呢？



观察上图，我们发现权重w_{ji}仅能通过影响节点j的输入值影响网络的其它部分，设net_j是节点j的**加权输入**，即

$$\begin{aligned} net_j &= \vec{w_j} \cdot \vec{x_j} && (21) \\ &= \sum_i w_{ji} x_{ji} && (22) \end{aligned}$$

E_d是net_j的函数，而net_j是w_{ji}的函数。根据链式求导法则，可以得到：

$$\begin{aligned} \frac{\partial E_d}{\partial w_{ji}} &= \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} && (23) \\ &= \frac{\partial E_d}{\partial net_j} \frac{\partial \sum_i w_{ji} x_{ji}}{\partial w_{ji}} && (24) \\ &= \frac{\partial E_d}{\partial net_j} x_{ji} && (25) \end{aligned}$$

上式中，x_{ji}是节点i传递给节点j的输入值，也就是节点i的输出值。

对于$\frac{\partial E_d}{\partial net_j}$的推导，需要区分**输出层**和**隐藏层**两种情况。

输出层权值训练

对于**输出层**来说，net_j仅能通过节点j的输出值y_j来影响网络其它部分，也就是说E_d是y_j的函数，而y_j是net_j的函数，其中$y_j = sigmoid(net_j)$。所以我们可以再次使用链式求导法则：

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial y_j} \frac{\partial y_j}{\partial net_j} \qquad (26)$$

考虑上式第一项：

$$\begin{aligned} \frac{\partial E_d}{\partial y_j} &= \frac{\partial}{\partial y_j} \frac{1}{2} \sum_{i \in outputs} (t_i - y_i)^2 && (27) \\ &= \frac{\partial}{\partial y_j} \frac{1}{2} (t_j - y_j)^2 && (28) \\ &= -(t_j - y_j) && (29) \end{aligned}$$

考虑上式第二项:

$$\begin{aligned}\frac{\partial y_j}{\partial net_j} &= \frac{\partial sigmoid(net_j)}{\partial net_j} &(30)\\ &= y_j(1-y_j) &(31)\end{aligned}$$

将第一项和第二项带入，得到:

$$\frac{\partial E_d}{\partial net_j} = -(t_j - y_j)y_j(1-y_j)$$

如果令$\delta_j = -\frac{\partial E_d}{\partial net_j}$，也就是一个节点的误差项$\delta$是网络误差对这个节点输入的偏导数的相反数。带入上式，得到:

$$\delta_j = (t_j - y_j)y_j(1-y_j)$$

上式就是**式3**。

将上述推导带入随机梯度下降公式，得到:

$$\begin{aligned}w_{ji} &\leftarrow w_{ji} - \eta\frac{\partial E_d}{\partial w_{ji}} &(32)\\ &= w_{ji} + \eta(t_j - y_j)y_j(1-y_j)x_{ji} &(33)\\ &= w_{ji} + \eta\delta_j x_{ji} &(34)\end{aligned}$$

上式就是**式5**。

隐藏层权值训练

现在我们要推导出隐藏层的$\frac{\partial E_d}{\partial net_j}$。

首先，我们需要定义节点j的所有直接下游节点的集合$Downstream(j)$。例如，对于节点4来说，它的直接下游节点是节点8、节点9。可以看到net_j只能通过影响$Downstream(j)$再影响E_d。设net_k是节点j的下游节点的输入，则E_d是net_k的函数，而net_k是net_j的函数。因为net_k有多个，我们应用全导数公式，可以做出如下推导:

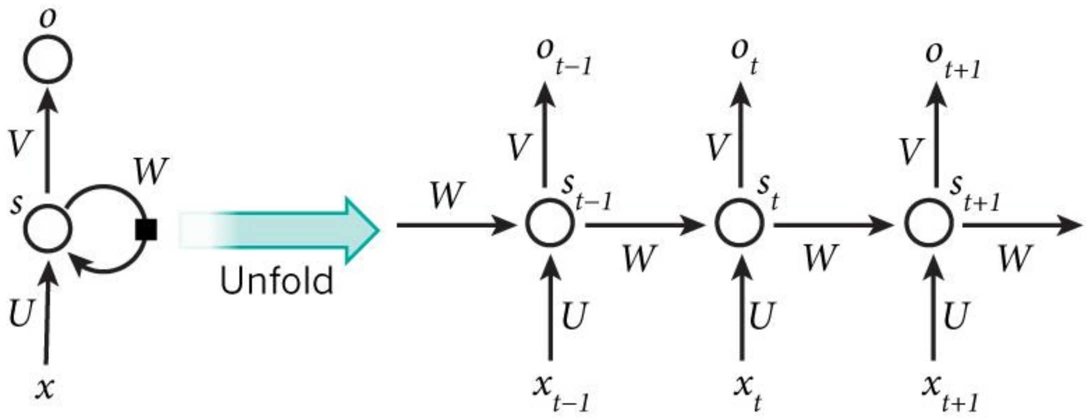
$$\begin{aligned}\frac{\partial E_d}{\partial net_j} &= \sum_{k\in Downstream(j)} \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j} &(35)\\ &= \sum_{k\in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} &(36)\\ &= \sum_{k\in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial a_j}\frac{\partial a_j}{\partial net_j} &(37)\\ &= \sum_{k\in Downstream(j)} -\delta_k w_{kj}\frac{\partial a_j}{\partial net_j} &(38)\\ &= \sum_{k\in Downstream(j)} -\delta_k w_{kj} a_j(1-a_j) &(39)\\ &= -a_j(1-a_j)\sum_{k\in Downstream(j)} \delta_k w_{kj} &(40)\end{aligned}$$

因为$\delta_j = -\frac{\partial E_d}{\partial net_j}$，带入上式得到:

$$\delta_j = a_j(1-a_j)\sum_{k\in Downstream(j)} \delta_k w_{kj}$$

上式就是**式4**。

3、循环神经网络（RNN）：



现在看上去就比较清楚了，这个网络在t时刻接收到输入x_t之后，隐藏层的值是s_t，输出值是o_t。关键一点是，s_t的值不仅仅取决于x_t，还取决于s_{t-1}。我们可以用下面的公式来表示**循环神经网络**的计算方法：

$$
\begin{aligned}
o_t &= g(Vs_t) \qquad (式1) && (1)\\
s_t &= f(Ux_t + Ws_{t-1}) \qquad (式2) && (2)
\end{aligned}
$$

式1是**输出层**的计算公式，输出层是一个**全连接层**，也就是它的每个节点都和隐藏层的每个节点相连。V是输出层的**权重矩阵**，g是**激活函数**。式2是隐藏层的计算公式，它是**循环层**。U是输入x的权重矩阵，W是上一次的值s_{t-1}作为这一次的输入的**权重矩阵**，f是**激活函数**。

从上面的公式我们可以看出，**循环层**和**全连接层**的区别就是**循环层**多了一个**权重矩阵** W。

如果反复把**式2**带入到**式1**，我们将得到：

$$
\begin{aligned}
o_t &= g(Vs_t) && (3)\\
&= Vf(Ux_t + Ws_{t-1}) && (4)\\
&= Vf(Ux_t + Wf(Ux_{t-1} + Ws_{t-2})) && (5)\\
&= Vf(Ux_t + Wf(Ux_{t-1} + Wf(Ux_{t-2} + Ws_{t-3}))) && (6)\\
&= Vf(Ux_t + Wf(Ux_{t-1} + Wf(Ux_{t-2} + Wf(Ux_{t-3} + \ldots)))) && (7)
\end{aligned}
$$

从上面可以看出，**循环神经网络**的输出值o_t，是受前面历次输入值x_t、x_{t-1}、x_{t-2}、x_{t-3}、...影响的，这就是为什么**循环神经网络**可以往前看任意多个**输入值**的原因。

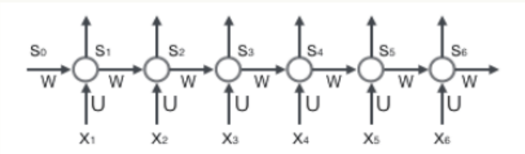
4、循环神经网络的训练算法：BPTT

BPTT算法是针对**循环层**的训练算法，它的基本原理和BP算法是一样的，也包含同样的三个步骤：

1. 前向计算每个神经元的输出值；
2. 反向计算每个神经元的**误差项**δ_j值，它是误差函数E对神经元j的**加权输入**net_j的偏导数；
3. 计算每个权重的梯度。

最后再用**随机梯度下降**算法更新权重。

循环层如下图所示：



前向计算

使用前面的**式2**对循环层进行前向计算：

$$\mathrm{s}_t = f(U\mathrm{x}_t + W\mathrm{s}_{t-1})$$

注意，上面的s_t、x_t、s_{t-1}都是向量，用**黑体字母**表示；而U、V是**矩阵**，用大写字母表示。**向量的下标**表示**时刻**，例如，s_t表示在t时刻向量s的值。

我们假设输入向量x的维度是m，输出向量s的维度是n，则矩阵U的维度是$n \times m$，矩阵W的维度是$n \times n$。下面是上式展开成矩阵的样子，看起来更直观一些：

$$\begin{bmatrix} s_1^t \\ s_2^t \\ . \\ . \\ s_n^t \end{bmatrix} = f(\begin{bmatrix} u_{11} u_{12} \ldots u_{1m} \\ u_{21} u_{22} \ldots u_{2m} \\ . \\ . \\ u_{n1} u_{n2} \ldots u_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_m \end{bmatrix} + \begin{bmatrix} w_{11} w_{12} \ldots w_{1n} \\ w_{21} w_{22} \ldots w_{2n} \\ . \\ . \\ w_{n1} w_{n2} \ldots w_{nn} \end{bmatrix} \begin{bmatrix} s_1^{t-1} \\ s_2^{t-1} \\ . \\ . \\ s_n^{t-1} \end{bmatrix}) \qquad (19)$$

在这里我们用**手写体字母**表示向量的一个**元素**，它的下标表示它是这个向量的第几个元素，它的上标表示第几个**时刻**。例如，s_j^t表示向量s的第j个元素在t时刻的值。u_{ji}表示**输入层**第i个神经元到**循环层**第j个神经元的权重。w_{ji}表示**循环层**第t-1时刻的第i个神经元到**循环层**第t个时刻的第j个神经元的权重。

误差项的计算

BTPP算法将第l层t时刻的**误差项**δ_t^l值沿两个方向传播，一个方向是其传递到上一层网络，得到δ_t^{l-1}，这部分只和权重矩阵U有关；另一个是方向是将其沿时间线传递到初始t_1时刻，得到δ_1^l，这部分只和权重矩阵W有关。

我们用向量net_t表示神经元在t时刻的**加权输入**，因为：

$$\mathrm{net}_t = U\mathrm{x}_t + W\mathrm{s}_{t-1} \qquad (20)$$

$$\mathrm{s}_{t-1} = f(\mathrm{net}_{t-1}) \qquad (21)$$

因此：

$$\frac{\partial \mathrm{net}_t}{\partial \mathrm{net}_{t-1}} = \frac{\partial \mathrm{net}_t}{\partial \mathrm{s}_{t-1}} \frac{\partial \mathrm{s}_{t-1}}{\partial \mathrm{net}_{t-1}} \qquad (22)$$

我们用a表示列向量，用a^T表示行向量。上式的第一项是向量函数对向量求导，其结果为Jacobian矩阵：

$$\begin{aligned}
\frac{\partial \mathrm{net}_t}{\partial \mathrm{s}_{t-1}} &= \begin{bmatrix} \frac{\partial net_1^t}{\partial s_1^{t-1}} & \frac{\partial net_1^t}{\partial s_2^{t-1}} & \dots & \frac{\partial net_1^t}{\partial s_n^{t-1}} \\ \frac{\partial net_2^t}{\partial s_1^{t-1}} & \frac{\partial net_2^t}{\partial s_2^{t-1}} & \dots & \frac{\partial net_2^t}{\partial s_n^{t-1}} \\ & . & & \\ & . & & \\ \frac{\partial net_n^t}{\partial s_1^{t-1}} & \frac{\partial net_n^t}{\partial s_2^{t-1}} & \dots & \frac{\partial net_n^t}{\partial s_n^{t-1}} \end{bmatrix} && (23) \\
&= \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ & . & & \\ & . & & \\ w_{n1} & w_{n2} & \dots & w_{nn} \end{bmatrix} && (24) \\
&= W && (25)
\end{aligned}$$

同理，上式第二项也是一个Jacobian矩阵：

$$\begin{aligned}
\frac{\partial \mathrm{s}_{t-1}}{\partial \mathrm{net}_{t-1}} &= \begin{bmatrix} \frac{\partial s_1^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_1^{t-1}}{\partial net_2^{t-1}} & \dots & \frac{\partial s_1^{t-1}}{\partial net_n^{t-1}} \\ \frac{\partial s_2^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_2^{t-1}}{\partial net_2^{t-1}} & \dots & \frac{\partial s_2^{t-1}}{\partial net_n^{t-1}} \\ & . & & \\ & . & & \\ \frac{\partial s_n^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_n^{t-1}}{\partial net_2^{t-1}} & \dots & \frac{\partial s_n^{t-1}}{\partial net_n^{t-1}} \end{bmatrix} && (26) \\
&= \begin{bmatrix} f'(net_1^{t-1}) & 0 & \dots & 0 \\ 0 & f'(net_2^{t-1}) & \dots & 0 \\ & . & & \\ & . & & \\ 0 & 0 & \dots & f'(net_n^{t-1}) \end{bmatrix} && (27) \\
&= diag[f'(\mathrm{net}_{t-1})] && (28)
\end{aligned}$$

其中，diag[a]表示根据向量a创建一个对角矩阵，即

$$diag(\mathrm{a}) = \begin{bmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \dots & 0 \\ & . & & \\ & . & & \\ 0 & 0 & \dots & a_n \end{bmatrix}$$

最后，将两项合在一起，可得：

$$\begin{aligned}\frac{\partial \mathrm{net}_t}{\partial \mathrm{net}_{t-1}} &= \frac{\partial \mathrm{net}_t}{\partial \mathrm{s}_{t-1}}\frac{\partial \mathrm{s}_{t-1}}{\partial \mathrm{net}_{t-1}} & (29)\\ &= W diag[f'(\mathrm{net}_{t-1})] & (30)\\ &= \begin{bmatrix} w_{11}f'(net_1^{t-1}) & w_{12}f'(net_2^{t-1}) & \dots & w_{1n}f(net_n^{t-1})\\ w_{21}f'(net_1^{t-1}) & w_{22}f'(net_2^{t-1}) & \dots & w_{2n}f(net_n^{t-1})\\ & & . & \\ & & . & \\ w_{n1}f'(net_1^{t-1}) & w_{n2}f'(net_2^{t-1}) & \dots & w_{nn}f'(net_n^{t-1}) \end{bmatrix} & (31)\end{aligned}$$

上式描述了将δ沿时间往前传递一个时刻的规律，有了这个规律，我们就可以求得任意时刻k的**误差项**δ_k：

$$\begin{aligned}\delta_k^T &= \frac{\partial E}{\partial \mathrm{net}_k} & (32)\\ &= \frac{\partial E}{\partial \mathrm{net}_t}\frac{\partial \mathrm{net}_t}{\partial \mathrm{net}_k} & (33)\\ &= \frac{\partial E}{\partial \mathrm{net}_t}\frac{\partial \mathrm{net}_t}{\partial \mathrm{net}_{t-1}}\frac{\partial \mathrm{net}_{t-1}}{\partial \mathrm{net}_{t-2}}\dots\frac{\partial \mathrm{net}_{k+1}}{\partial \mathrm{net}_k} & (34)\\ &= W diag[f'(\mathrm{net}_{t-1})]W diag[f'(\mathrm{net}_{t-2})]\dots W diag[f'(\mathrm{net}_k)]\delta_t^l & (35)\\ &= \delta_t^T\prod_{i=k}^{t-1} W diag[f'(\mathrm{net}_i)] \qquad (式3) & (36)\end{aligned}$$

式3就是将误差项沿时间反向传播的算法。

循环层将**误差项**反向传递到上一层网络，与普通的**全连接层**是完全一样的，这在前面的文章零基础入门深度学习(3) - 神经网络和反向传播算法中已经详细讲过了，在此仅简要描述一下。

循环层的**加权输入**net^l与上一层的**加权输入**net^{l-1}关系如下：

$$\begin{aligned}\mathrm{net}_t^l &= U\mathrm{a}_t^{l-1} + W\mathrm{s}_{t-1} & (37)\\ \mathrm{a}_t^{l-1} &= f^{l-1}(\mathrm{net}_t^{l-1}) & (38)\end{aligned}$$

上式中net_t^l是第l层神经元的**加权输入**(假设第l层是**循环层**)；net_t^{l-1}是第l-1层神经元的**加权输入**；a_t^{l-1}是第l-1层神经元的输出；f^{l-1}是第l-1层的**激活函数**。

$$\begin{aligned}\frac{\partial \mathrm{net}_t^l}{\partial \mathrm{net}_t^{l-1}} &= \frac{\partial \mathrm{net}^l}{\partial \mathrm{a}_t^{l-1}}\frac{\partial \mathrm{a}_t^{l-1}}{\partial \mathrm{net}_t^{l-1}} & (39)\\ &= U diag[f'^{l-1}(\mathrm{net}_t^{l-1})] & (40)\end{aligned}$$

所以，

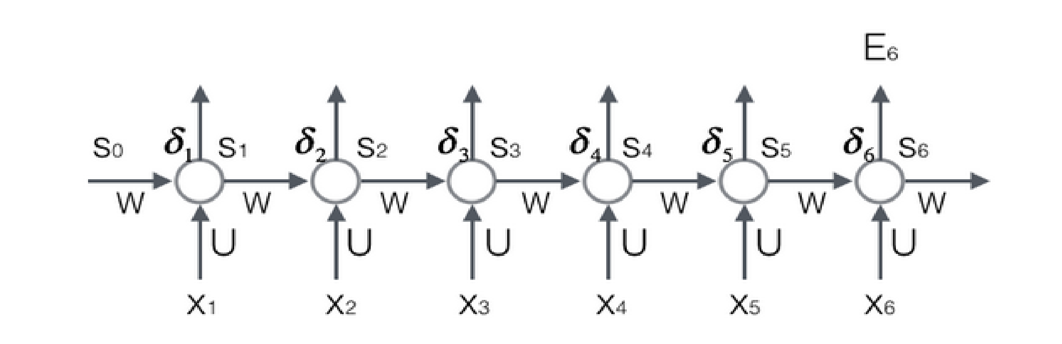
$$\begin{aligned}(\delta_t^{l-1})^T &= \frac{\partial E}{\partial \mathrm{net}_t^{l-1}} & (41)\\ &= \frac{\partial E}{\partial \mathrm{net}_t^l}\frac{\partial \mathrm{net}_t^l}{\partial \mathrm{net}_t^{l-1}} & (42)\\ &= (\delta_t^l)^T U diag[f'^{l-1}(\mathrm{net}_t^{l-1})] \qquad (式4) & (43)\end{aligned}$$

式4就是将误差项传递到上一层算法。

权重梯度的计算

现在，我们终于来到了BPTT算法的最后一步：计算每个权重的梯度。

首先，我们计算误差函数E对权重矩阵W的梯度$\frac{\partial E}{\partial W}$。



上图展示了我们到目前为止，在前两步中已经计算得到的量，包括每个时刻t **循环层**的输出值s_t，以及误差项δ_t。

回忆一下我们在文章零基础入门深度学习(3) - 神经网络和反向传播算法介绍的全连接网络的权重梯度计算算法：只要知道了任意一个时刻的**误差项**δ_t，以及上一个时刻循环层的输出值s_{t-1}，就可以按照下面的公式求出权重矩阵在t时刻的梯度$\nabla_{Wt}E$：

$$\nabla_{W_t}E=\begin{bmatrix}\delta_1^t s_1^{t-1} & \delta_1^t s_2^{t-1} & \ldots & \delta_1^t s_n^{t-1}\\ \delta_2^t s_1^{t-1} & \delta_2^t s_2^{t-1} & \ldots & \delta_2^t s_n^{t-1}\\ . & & & \\ . & & & \\ \delta_n^t s_1^{t-1} & \delta_n^t s_2^{t-1} & \ldots & \delta_n^t s_n^{t-1}\end{bmatrix} \qquad (式5)$$

在**式5**中，δ_i^t表示t时刻**误差项**向量的第i个分量；s_i^{t-1}表示t-1时刻**循环层**第i个神经元的输出值。

我们下面可以简单推导一下**式5**。

我们知道：

$$\begin{aligned}\mathrm{net}_t &= U\mathrm{x}_t+W\mathrm{s}_{t-1} && (44)\\ \begin{bmatrix}net_1^t\\ net_2^t\\ .\\ .\\ net_n^t\end{bmatrix} &= U\mathrm{x}_t+\begin{bmatrix}w_{11} & w_{12} & \ldots & w_{1n}\\ w_{21} & w_{22} & \ldots & w_{2n}\\ . & & & \\ . & & & \\ w_{n1} & w_{n2} & \ldots & w_{nn}\end{bmatrix}\begin{bmatrix}s_1^{t-1}\\ s_2^{t-1}\\ .\\ .\\ s_n^{t-1}\end{bmatrix} && (45)\\ &= U\mathrm{x}_t+\begin{bmatrix}w_{11}s_1^{t-1}+w_{12}s_2^{t-1}\ldots w_{1n}s_n^{t-1}\\ w_{21}s_1^{t-1}+w_{22}s_2^{t-1}\ldots w_{2n}s_n^{t-1}\\ .\\ .\\ w_{n1}s_1^{t-1}+w_{n2}s_2^{t-1}\ldots w_{nn}s_n^{t-1}\end{bmatrix} && (46)\end{aligned}$$

因为对W求导与$U\mathrm{x}_t$无关，我们不再考虑。现在，我们考虑对权重项w_{ji}求导。通过观察上式我们可以看到w_{ji}只与net_j^t有关，所以：

$$\begin{aligned}\frac{\partial E}{\partial w_{ji}} &= \frac{\partial E}{\partial net_j^t}\frac{\partial net_j^t}{\partial w_{ji}} && (47)\\ &= \delta_j^t s_i^{t-1} && (48)\end{aligned}$$

按照上面的规律就可以生成**式5**里面的矩阵。

我们已经求得了权重矩阵W在t时刻的梯度$\nabla_{Wt}E$，最终的梯度$\nabla_W E$是各个时刻的梯度**之和**:

$$\begin{aligned}\nabla_W E &= \sum_{i=1}^{t}\nabla_{W_i}E \qquad (49)\\ &= \begin{bmatrix} \delta_1^t s_1^{t-1} & \delta_1^t s_2^{t-1} & \dots & \delta_1^t s_n^{t-1} \\ \delta_2^t s_1^{t-1} & \delta_2^t s_2^{t-1} & \dots & \delta_2^t s_n^{t-1} \\ . & & & \\ . & & & \\ \delta_n^t s_1^{t-1} & \delta_n^t s_2^{t-1} & \dots & \delta_n^t s_n^{t-1} \end{bmatrix} + \dots + \begin{bmatrix} \delta_1^1 s_1^0 & \delta_1^1 s_2^0 & \dots & \delta_1^1 s_n^0 \\ \delta_2^1 s_1^0 & \delta_2^1 s_2^0 & \dots & \delta_2^1 s_n^0 \\ . & & & \\ . & & & \\ \delta_n^1 s_1^0 & \delta_n^1 s_2^0 & \dots & \delta_n^1 s_n^0 \end{bmatrix} \qquad (\text{式}6)\ (50)\end{aligned}$$

式6就是计算**循环层**权重矩阵W的梯度的公式。

5、BPTT 算法推导（BPTT）；

前面已经介绍了$\nabla_W E$的计算方法，看上去还是比较直观的。然而，读者也许会困惑，为什么最终的梯度是各个时刻的梯度之和呢？我们前面只是直接用了这个结论，实际上这里面是有道理的，只是这个数学推导比较绕脑子。感兴趣的同学可以仔细阅读接下来这一段，它用到了矩阵对矩阵求导、张量与向量相乘运算的一些法则。

我们还是从这个式子开始：

$$\mathrm{net}_t = U\mathrm{x}_t + Wf(\mathrm{net}_{t-1})$$

因为$U\mathrm{x}_t$与W完全无关，我们把它看做常量。现在，考虑第一个式子加号右边的部分，因为W和$f(\mathrm{net}_{t-1})$都是W的函数，因此我们要用到大学里面都学过的导数乘法运算：

$$(uv)' = u'v + uv'$$

因此，上面第一个式子写成：

$$\frac{\partial \mathrm{net}_t}{\partial W} = \frac{\partial W}{\partial W} f(\mathrm{net}_{t-1}) + W \frac{\partial f(\mathrm{net}_{t-1})}{\partial W}$$

我们最终需要计算的是$\nabla_W E$：

$$\begin{aligned}
\nabla_W E &= \frac{\partial E}{\partial W} && (51)\\
&= \frac{\partial E}{\partial \mathrm{net}_t}\frac{\partial \mathrm{net}_t}{\partial W} && (52)\\
&= \delta_t^T \frac{\partial W}{\partial W} f(\mathrm{net}_{t-1}) + \delta_t^T W \frac{\partial f(\mathrm{net}_{t-1})}{\partial W} \qquad (\text{式}7) && (53)
\end{aligned}$$

我们先计算式7加号左边的部分。$\frac{\partial W}{\partial W}$是矩阵对矩阵求导，其结果是一个四维张量(**tensor**)，如下所示：

$$\begin{aligned}
\frac{\partial W}{\partial W} &= \begin{bmatrix} \frac{\partial w_{11}}{\partial W} & \frac{\partial w_{12}}{\partial W} & \dots & \frac{\partial w_{1n}}{\partial W} \\ \frac{\partial w_{21}}{\partial W} & \frac{\partial w_{22}}{\partial W} & \dots & \frac{\partial w_{2n}}{\partial W} \\ . \\ . \\ \frac{\partial w_{n1}}{\partial W} & \frac{\partial w_{n2}}{\partial W} & \dots & \frac{\partial w_{nn}}{\partial W} \end{bmatrix} && (54)\\
&= \begin{bmatrix} \begin{bmatrix} \frac{\partial w_{11}}{\partial w_{11}} & \frac{\partial w_{11}}{\partial w_{12}} & \dots & \frac{\partial w_{11}}{\partial_{1n}} \\ \frac{\partial w_{11}}{\partial w_{21}} & \frac{\partial w_{11}}{\partial w_{22}} & \dots & \frac{\partial w_{11}}{\partial_{2n}} \\ . \\ . \\ \frac{\partial w_{11}}{\partial w_{n1}} & \frac{\partial w_{11}}{\partial w_{n2}} & \dots & \frac{\partial w_{11}}{\partial_{nn}} \end{bmatrix} & \begin{bmatrix} \frac{\partial w_{12}}{\partial w_{11}} & \frac{\partial w_{12}}{\partial w_{12}} & \dots & \frac{\partial w_{12}}{\partial_{1n}} \\ \frac{\partial w_{12}}{\partial w_{21}} & \frac{\partial w_{12}}{\partial w_{22}} & \dots & \frac{\partial w_{12}}{\partial_{2n}} \\ . \\ . \\ \frac{\partial w_{12}}{\partial w_{n1}} & \frac{\partial w_{12}}{\partial w_{n2}} & \dots & \frac{\partial w_{12}}{\partial_{nn}} \end{bmatrix} & \dots \\ . \\ . \end{bmatrix} && (55)\\
&= \begin{bmatrix} \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ . \\ . \\ 0 & 0 & \dots & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ . \\ . \\ 0 & 0 & \dots & 0 \end{bmatrix} & \dots \\ . \\ . \end{bmatrix} && (56)
\end{aligned}$$

接下来，我们知道$s_{t-1}=f(\mathrm{net}_{t-1})$，它是一个列向量。我们让上面的四维张量与这个向量相乘，得到了一个三维张量，再左乘行向量δ_t^T，最终得到一个矩阵：

$$
\begin{aligned}
\delta_t^T\frac{\partial W}{\partial W}f(\mathrm{net}_{t-1}) &= \delta_t^T\frac{\partial W}{\partial W}\mathrm{s}_{t-1} && (57)\\
&= \delta_t^T\begin{bmatrix}\begin{bmatrix}1&0&\dots&0\\0&0&\dots&0\\.\\.\\0&0&\dots&0\end{bmatrix}&\begin{bmatrix}0&1&\dots&0\\0&0&\dots&0\\.\\.\\0&0&\dots&0\end{bmatrix}&\dots\\ .\\ .\end{bmatrix}\begin{bmatrix}s_1^{t-1}\\s_2^{t-1}\\.\\.\\s_n^{t-1}\end{bmatrix} && (58)\\
&= \delta_t^T\begin{bmatrix}\begin{bmatrix}s_1^{t-1}\\0\\.\\.\\0\end{bmatrix}&\begin{bmatrix}s_2^{t-1}\\0\\.\\.\\0\end{bmatrix}&\dots\\ .\\ .\end{bmatrix} && (59)\\
&= \begin{bmatrix}\delta_1^t&\delta_2^t&\dots&\delta_n^t\end{bmatrix}\begin{bmatrix}\begin{bmatrix}s_1^{t-1}\\0\\.\\.\\0\end{bmatrix}&\begin{bmatrix}s_2^{t-1}\\0\\.\\.\\0\end{bmatrix}&\dots\\ .\\ .\end{bmatrix} && (60)\\
&= \begin{bmatrix}\delta_1^ts_1^{t-1}&\delta_1^ts_2^{t-1}&\dots&\delta_1^ts_n^{t-1}\\\delta_2^ts_1^{t-1}&\delta_2^ts_2^{t-1}&\dots&\delta_2^ts_n^{t-1}\\.\\.\\\delta_n^ts_1^{t-1}&\delta_n^ts_2^{t-1}&\dots&\delta_n^ts_n^{t-1}\end{bmatrix} && (61)\\
&= \nabla_{W_t}E && (62)
\end{aligned}
$$

接下来，我们计算式7加号右边的部分：

$$
\begin{aligned}
\delta_t^TW\frac{\partial f(\mathrm{net}_{t-1})}{\partial W} &= \delta_t^TW\frac{\partial f(\mathrm{net}_{t-1})}{\partial\mathrm{net}_{t-1}}\frac{\partial\mathrm{net}_{t-1}}{\partial W} && (63)\\
&= \delta_t^TWf'(\mathrm{net}_{t-1})\frac{\partial\mathrm{net}_{t-1}}{\partial W} && (64)\\
&= \delta_t^T\frac{\partial\mathrm{net}_t}{\partial\mathrm{net}_{t-1}}\frac{\partial\mathrm{net}_{t-1}}{\partial W} && (65)\\
&= \delta_{t-1}^T\frac{\partial\mathrm{net}_{t-1}}{\partial W} && (66)
\end{aligned}
$$

于是，我们得到了如下递推公式：

$$
\begin{aligned}
\nabla_WE &= \frac{\partial E}{\partial W} && (67)\\
&= \frac{\partial E}{\partial\mathrm{net}_t}\frac{\partial\mathrm{net}_t}{\partial W} && (68)\\
&= \nabla_{W_t}E+\delta_{t-1}^T\frac{\partial\mathrm{net}_{t-1}}{\partial W} && (69)\\
&= \nabla_{W_t}E+\nabla_{W_{t-1}}E+\delta_{t-2}^T\frac{\partial\mathrm{net}_{t-2}}{\partial W} && (70)\\
&= \nabla_{W_t}E+\nabla_{W_{t-1}}E+\dots+\nabla_{W_1}E && (71)\\
&= \sum_{k=1}^{t}\nabla_{W_k}E && (72)
\end{aligned}
$$

这样，我们就证明了：最终的梯度∇_WE是各个时刻的梯度之和。

6、RNN 的梯度爆炸和消失问题

不幸的是，实践中前面介绍的几种RNNs并不能很好的处理较长的序列。一个主要的原因是，RNN在训练中很容易发生**梯度爆炸**和**梯度消失**，这导致训练时梯度不能在较长序列中一直传递下去，从而使RNN无法捕捉到长距离的影响。

为什么RNN会产生梯度爆炸和消失问题呢？我们接下来将详细分析一下原因。我们根据**式3**可得：

$$
\begin{aligned}
\delta_k^T &= \delta_t^T \prod_{i=k}^{t-1} W diag[f'(\mathrm{net}_i)] && (73) \\
\|\delta_k^T\| &\leqslant \|\delta_t^T\| \prod_{i=k}^{t-1} \|W\| \|diag[f'(\mathrm{net}_i)]\| && (74) \\
&\leqslant \|\delta_t^T\| (\beta_W \beta_f)^{t-k} && (75)
\end{aligned}
$$

上式的β定义为矩阵的模的上界。因为上式是一个指数函数，如果t-k很大的话（也就是向前看很远的时候），会导致对应的**误差项**的值增长或缩小的非常快，这样就会导致相应的**梯度爆炸**和**梯度消失**问题（取决于β大于1还是小于1）。

通常来说，**梯度爆炸**更容易处理一些。因为梯度爆炸的时候，我们的程序会收到NaN错误。我们也可以设置一个梯度阈值，当梯度超过这个阈值的时候可以直接截取。

梯度消失更难检测，而且也更难处理一些。总的来说，我们有三种方法应对梯度消失问题：

1. 合理的初始化权重值。初始化权重，使每个神经元尽可能不要取极大或极小值，以躲开梯度消失的区域。
2. 使用relu代替sigmoid和tanh作为激活函数。原理请参考上一篇文章零基础入门深度学习(4) - 卷积神经网络的**激活函数**一节。
3. 使用其他结构的RNNs，比如长短时记忆网络（LTSM）和Gated Recurrent Unit（GRU），这是最流行的做法。我们将在以后的文章中介绍这两种网络。

附录：

本小结内容主要来源于网络：

1、零基础入门深度学习（3）、（5）