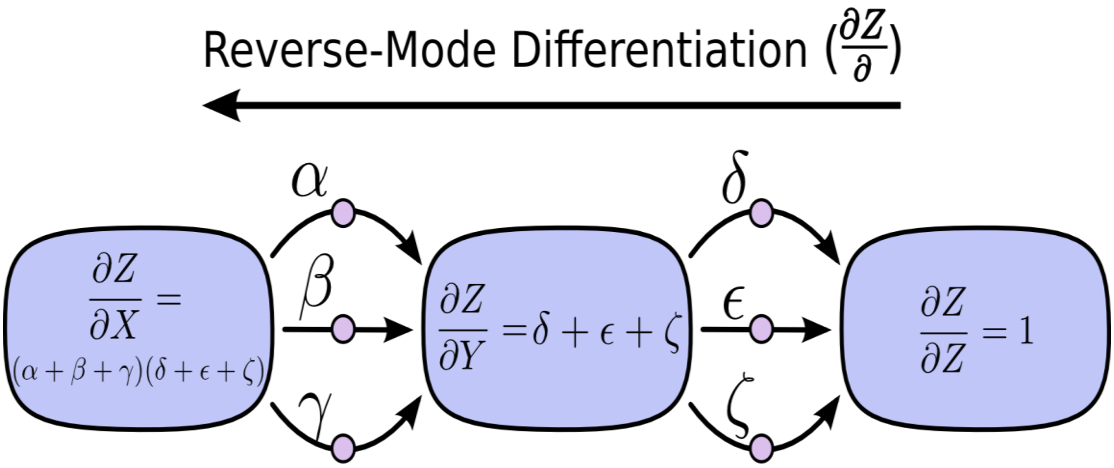
神经网络

倒序求偏导



还有个顺序计算的，那个是每个节点关于最初的节点的偏导数也就是 x 对每个节点的影响，这个倒序的计算出来的是每个节点对最终结果的影响。现实中我们关注的是各个变量如何影响 output（一般是损失函数）。倒着算导数一边就算出了，正着算要重复走节点个数次数。bp 算法其实也是利用了这个思想。（这个图其实画的并不是很严谨，注意思想就好）

神经网络模型结构可以用[]表示，就是每个 level 包括几个节点。第一个就是 input 变量个数，最后是 output 变量个数。目标函数就是，说白了其实就是 w 和 b 的函数。这里采用的是平方误差。

$$
\begin{aligned}
E &= \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{o}\| \\
&= \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{a}^{(3)}\| \\
&= \frac{1}{2}\left((y_1-a_1^{(3)})^2+(y_2-a_2^{(3)})^2\right) \\
&= \frac{1}{2}\left((y_1-f(z_1^{(3)}))^2+(y_2-f(z_2^{(3)}))^2\right) \\
&= \frac{1}{2}\left((y_1-f(w_{11}^{(3)}a_1^{(2)}+w_{12}^{(3)}a_2^{(2)}+w_{13}^{(3)}a_3^{(2)}+b_1^{(3)}))^2+(y_2-f(w_{21}^{(3)}a_1^{(2)}+w_{22}^{(3)}a_2^{(2)}+w_{23}^{(3)}a_3^{(2)}+b_2^{(3)}))^2\right)
\end{aligned}
$$

估计参数

主要思想 1，梯度下降。2，梯度的计算用到 bp 算法。
这个公式并没有展开，把第二层的 a 当成定值，然后损失函数就是关于第三层的 w 和 d 的函数。把 a 展开还会把前面几层的 w，d 引入。这是最直接但是繁琐的方式，可能算不出来。
对于第三层的 w，d 来说求他们的偏导数，不需要把 a 展开，把 a 看作个整体简洁。按照这个思想不要一次性计算，按照倒序求导的思想，先把最后一层计算出来，前面的导数到可以用后面的导数算出来。
这里求 w，d 需要各个节点的过渡，先求各个节点对 output 的影响（output 关于节点的偏

导数)，然后再求 output 关于 w，d 的偏导数。

首先是输出层的参数更新

重要思想就是 output 关于 w 的偏导总能拆分为 output 关于节点，节点关于 w。

$$\frac{\partial E}{\partial w_{11}^{(3)}} = \frac{\partial E}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial w_{11}^{(3)}}$$
$$= \delta_1^{(3)} a_1^{(2)} \qquad \delta_1^{(3)} = \frac{\partial E}{\partial z_1^{(3)}} = \frac{\partial E}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = -(y_1 - a_1^{(3)}) f'(z_1^{(3)})$$

假设神经网络共 L 层，则：

$$\delta_i^{(L)} = -(y_i - a_i^{(L)}) f'(z_i^{(L)}) \quad (1 \le i \le n_L)$$
$$\frac{\partial E}{\partial w_{ij}^{(L)}} = \delta_i^{(L)} a_j^{(L-1)} \quad (1 \le i \le n_L, 1 \le j \le n_{L-1})$$

向量形式：

$$\boldsymbol{\delta}^{(L)} = -(\boldsymbol{y} - \boldsymbol{a}^{(L)}) \odot f'(\boldsymbol{z}^{(L)})$$
$$\nabla_{W^{(L)}} E = \boldsymbol{\delta}^{(L)} (\boldsymbol{a}^{(L-1)})^{\mathsf{T}}$$

然后是隐藏层的参数更新

像之前引入关于节点的偏导（注意 w 的下标前面是后一层，后面是前一层，上标表示第几层）

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}$$
$$= \delta_i^{(l)} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}$$
$$= \delta_i^{(l)} a_j^{(l-1)}$$

然后关键就是 output 关于节点的偏导数，这里就用到了前面介绍的倒序传递求偏导数，利用前面一级的偏导数计算当前的偏导数。

$$\delta_i^{(l)} \equiv \frac{\partial E}{\partial z_i^{(l)}}$$
$$= \sum_{j=1}^{n_{l+1}} \frac{\partial E}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}}$$
$$= \sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} \frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}}$$

Output 关于当前结点的偏导数拆分为 output 关于后面节点的偏导（注意影响路径有多个）和后面节点关于当前节点的偏导数。第一个我们已经知道了，然后各级节点之间是有关系的。

由于 $z_j^{(l+1)} = \sum_{i=1}^{n_l} w_{ji}^{(l+1)} a_i^{(l)} + b_j^{(l+1)} = \sum_{i=1}^{n_l} w_{ji}^{(l+1)} f(z_i^{(l)}) + b_j^{(l+1)}$ ，所以有 $\frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}} = \frac{\partial z_j^{(l+1)}}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} = w_{ji}^{(l+1)} f'(z_i^{(l)})$ ，代入到前面计算的 $\delta_i^{(l)}$ 式中，从而有：

这样就可以逐级准确的计算各个梯度。
Bp 有四个关键公式：

$$\delta_i^{(L)} = -(y_i - a_i^{(L)}) f'(z_i^{(L)})$$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} w_{ji}^{(l+1)} \right) f'(z_i^{(l)})$$

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$$

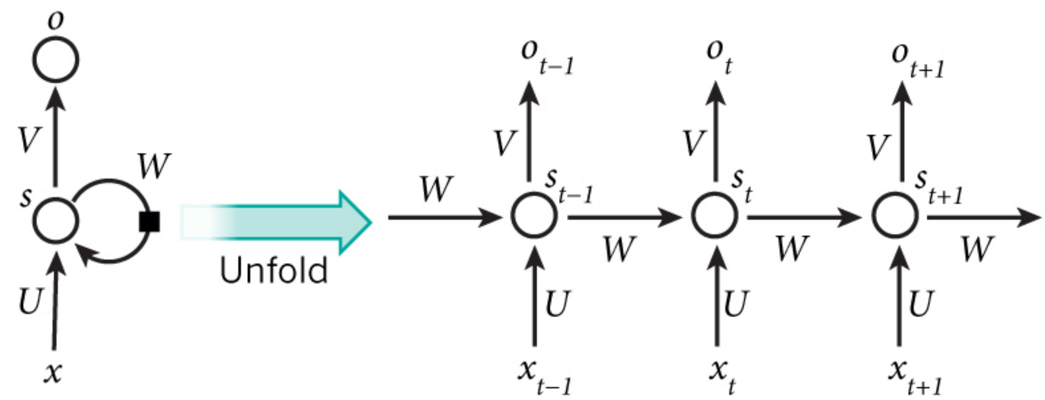
$$\frac{\partial E}{\partial b_i^{(l)}} = \delta_i^{(l)}$$

算法实现

就是求各级 w，b。
1, 首先 SGD 方法, 迭代整个数据集 epoch 次, 每次把整个数据集划分为若干个 mini_batch, 以 batch 为单位刷新参数（不要每输入一个数据就更新一次这样太不稳定了）。
2，每个 minibatch 就是平均各个样本的梯度并刷新。
3，每个样本的梯度计算。分为最终层的梯度计算和隐含层的梯度计算。

Logistic 回归其实挺像两层的神经网络, 把所有 feature 综合成为一个指标, 然后使用 ex/(1+ex) 归一。

Recurrent Neural Networks (vanilla RNNs)



模型

是一个神经网络 module 组成的 chain，纵向来看每一列都是一个神经网络，但因为想要传递信息，所以横向有条线索连接各个三层神经网络。

W，U，V 这三个参数展开的各阶段是公用的。本来 T 时间的 s，也就是第二层（隐含层）的节点的计算按照标准的神经网络只依赖 x 和 U，但是想要传入前阶段的信息，所以还依赖 w 和 s，因此可以说包括了现在及当前阶段以前的的信息（it's the memory of the network），然后利用第二层的信息计算输出概率。

$s_t = f(Ux_t + Ws_{t-1})$，f 一般是非线性函数（tanh，relu）

$o_t = \mathrm{softmax}(Vs_t)$.

求参

Bptt

首先定义损失函数：

$$
\begin{aligned}
E_t(y_t, \hat{y}_t) &= -y_t \log \hat{y}_t \\
E(y, \hat{y}) &= \sum_t E_t(y_t, \hat{y}_t) \\
&= -\sum_t y_t \log \hat{y}_t
\end{aligned}
$$

之后每次算法训练一个句子，然后计算相应梯度，对参数梯度下降后继续训练下一个句子。$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$.偏分可以根据每个阶段拆开。

$$
\begin{aligned}
\frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} \\
&= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} \\
&= (\hat{y}_3 - y_3) \otimes s_3
\end{aligned}
$$

关于 V 的偏导数只依赖当前阶段几个量： $\hat{y}_3, y_3, s_3$

关于 W 的偏导较繁琐

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

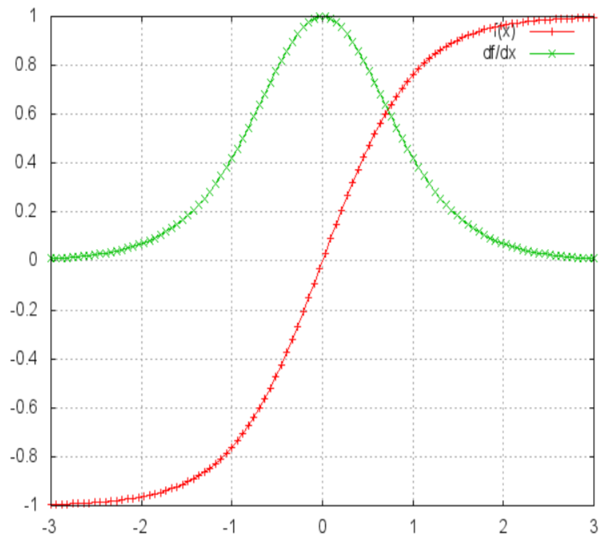
但是 $s_3 = \tanh(Ux_t + Ws_2)$ 依赖 s2，s2 和 W 有关系，依次类推所以有：

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

也就是

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}}\right)\frac{\partial s_k}{\partial W}$$

这个公式里中间那个括号很容易出现趋于 0 的，因为 tanh 导数决定的，更别提几个连乘。往回越多（t-k），趋于零概率越大（有一个为 0 就完了），这就出现了前面的梯度贡献一般没有。也就是所谓的 Vanishing gradient problem



Vanishing gradient problem 引出了 LSTM，Gated Recurrent Unit (GRU)

算法实现

其实从算法实现更容易理解梯度下降的那些公式。看你的代码注释。