CSci 8363 Backpropagation Fall 2017

Consider a network of the form

input			hidden layer				outer layer	
$\mathbf{x}$	$\rightarrow V \rightarrow$	$\hat{\mathbf{y}}$	$\rightarrow g \rightarrow$	$\mathbf{y}$	$\rightarrow W \rightarrow$	$\hat{\mathbf{z}}$	$\rightarrow g \rightarrow$	$\mathbf{z}$
$x_0 \equiv 1$		$\cdots$		$y_0 \equiv 1$	$\cdots$ *artificial variables for biases*			
x_1		$\hat{y}_1$	$\rightarrow g \rightarrow$	y_1		$\hat{z}_1$	$\rightarrow g \rightarrow$	z_1
x_2	$\rightarrow \quad \rightarrow$	$\hat{y}_2$	$\rightarrow g \rightarrow$	y_2	$\rightarrow \quad \rightarrow$	$\hat{z}_2$	$\rightarrow g \rightarrow$	z_2
.	$\rightarrow V \rightarrow$	.		.	$\rightarrow W \rightarrow$	.		.
.	$\rightarrow \quad \rightarrow$	.		.	$\rightarrow \quad \rightarrow$	.		.
x_p		$\hat{y}_n$	$\rightarrow g \rightarrow$	y_n		$\hat{z}_m$	$\rightarrow g \rightarrow$	z_m

where g is a "sigmoid" function and

$$\hat{y}_j = v_{j0} + v_{j1}x_1 + v_{j2}x_2 + \cdots + v_{jp}x_p \quad \text{for} \quad j = 1, \cdots, n$$
$$\hat{z}_i = w_{i0} + w_{i1}y_1 + w_{i2}y_2 + \cdots + w_{in}y_n \quad \text{for} \quad i = 1, \cdots, m$$

In matrix notation, this can be written $\hat{\mathbf{y}} = V \cdot \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$ and $\hat{\mathbf{z}} = W \cdot \begin{pmatrix} 1 \\ \mathbf{y} \end{pmatrix}$, where $\mathbf{x}$ is a p-vector, $\mathbf{y}$, $\hat{\mathbf{y}}$ are n-vectors, $\mathbf{z}$, $\hat{\mathbf{z}}$ are m-vectors, V is an $n \times (p+1)$ matrix of weights, and W is an $m \times (n+1)$ matrix of weights.

We apply an input $\mathbf{x}$ to the network, yielding an output $\mathbf{z}$. Then the error is

$$E = \tfrac{1}{2}\left((z_1 - t_1)^2 + (z_2 - t_2)^2 + \cdots + (z_m - t_m)^2 \right)$$

where t_i is the desired output for the given input $\mathbf{x}$. The goal is to minimize the error E, by gradient descent. We compute the following partial derivatives, by repeated use of the chain rule:

(a) $\delta_i \equiv \dfrac{\partial E}{\partial \hat{z}_i} = \dfrac{\partial E}{\partial z_i} \cdot \dfrac{\partial z_i}{\partial \hat{z}_i} = (z_i - t_i) \cdot g'(\hat{z}_i)$ for $i = 1, 2, \cdots, m$

(b) $\gamma_j \equiv \dfrac{\partial E}{\partial \hat{y}_j} = \dfrac{\partial E}{\partial y_j} \cdot \dfrac{\partial y_j}{\partial \hat{y}_j} = (\delta_1 w_{1j} + \delta_2 w_{2j} + \cdots + \delta_m w_{mj}) \cdot g'(\hat{y}_j)$ for $j = 1, 2, \cdots, n$

(c) $\dfrac{\partial E}{\partial w_{ij}} = \delta_i y_j$ for $\begin{cases} i = 1, 2, 3, \cdots, m \\ j = 0, 1, 2, \cdots, n \end{cases}$ **(d)** $\dfrac{\partial E}{\partial v_{jk}} = \gamma_j x_k$ for $\begin{cases} j = 1, 2, 3, \cdots, n \\ k = 0, 1, 2, \cdots, p \end{cases}$

If we use the sigmoid function which ranges between 0 and 1: $s = g(\hat{s}) = 1/(1 + e^{-\hat{s}})$ then $g'(\hat{s}) = s(1-s)$. If we use the sigmoid function which ranges from -1 to $+1$: $s = g(\hat{s}) = 2/(1 + e^{-2\hat{s}}) - 1$ then $g'(\hat{s}) = 1 - s^2$. In either case, the $\hat{y}$, $\hat{z}$ variables are not needed.

The formula **(c)** means, for example, that a small change Δw_{ij} to a weight w_{ij} will change E by $\Delta w_{ij}\delta_i y_j = \Delta w_{ij}(z_i - t_i)g'(\hat{z}_i)y_j = \Delta w_{ij}[\cdot](z_i - t_i)y_j$. If these small changes were applied at once, then E would change by $\sum_{ij} \Delta w_{ij}\delta_i y_j$, as long as the sum of squares of the Δw's are small enough. For a fixed sum of squares, the biggest reduction to E can be had by setting $\Delta w_{ij} = -\eta(z_i - t_i)[\cdot]y_j$ for a suitable scalar η (called "learning rate"). Similar updates to V are induced by formula **(d)**.

For a single layer network (e.g. Perceptrons), pretend that the y's are the inputs, and consider only the $W = (w_{ij})$ weights and their corresponding updates induced by **(a)** and **(c)**.

We then use the following overall method: Given samples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ each with a desired output $\mathbf{t}^{(1)}, \ldots, \mathbf{t}^{(N)}$, we go through the following loop (η is called the "learning rate"):

For $l = 1, 2, \ldots, N$ do
- **Let** $\mathbf{x}^{(l)}$ be applied as the input $\mathbf{x}$ to the network with $\mathbf{t}$ as the corresponding desired output.
- **Compute** the outputs from all the nodes, $\mathbf{y}$, $\mathbf{z}$, and all the partial derivatives above.
- **Apply** the corrections **(c)**: $w_{ij} \leftarrow w_{ij} - \eta\delta_i y_j$ and **(d)** $v_{jk} \leftarrow v_{jk} - \eta\gamma_j x_k$, for all i, j, k.

End.

One round through the entire loop for all l constitutes one "Epoch."

ADDENDUM

The output of the sigmoid function which ranges between 0 and 1, $s = g(\hat{s}) = 1/(1 + e^{-\hat{s}})$ can be considered as special case of the following "softmax" function:

$$\mathbf{g}(\hat{z}_k) = \exp\ \hat{z}_k \Big/ \left(\sum_i \exp\ \hat{z}_i \right) = 1 \Big/ \left(1 + \exp(-\hat{z}_k) \cdot \sum_{i \neq k} \exp\ \hat{z}_i \right),$$

where the denominator serves to normalize all the outputs so that they add up to 1. (Here we use the notation: $\exp x = e^x$.) The derivative is

$$\mathbf{g}'(\hat{z}_k) = \mathbf{g}(\hat{z}_k)(1 - \mathbf{g}(\hat{z}_k))$$

Treating the outputs $\mathbf{g}(\hat{z}_k)$ as probabilities, we can consider using the KL Divergence as the error function:

$$\mathbf{E} = -\left(t_1 \log z_1 + t_2 \log z_2 + \cdots + t_m \log z_m \right) + \text{a function of just } t$$

where $z_k = \mathbf{g}(\hat{z}_k)$. We have the derivatives

$$\frac{\partial \log(z_k)}{\partial \hat{z}_k} = \frac{\partial \log(z_k)}{\partial z_k} \cdot \frac{\partial z_k}{\partial \hat{z}_k} = \frac{1}{z_k} \cdot z_k(1 - z_k) = 1 - z_k$$

The derivative of this particular $\mathbf{E}$ with respect to any individual $\hat{z}_k$ is

$$(\mathbf{a}')\ \delta_i \equiv \frac{\partial \mathbf{E}}{\partial \hat{z}_i} = \frac{\partial \mathbf{E}}{\partial z_i} \cdot \frac{\partial z_i}{\partial \hat{z}_i} = \left(\frac{t_i}{z_i} \right) \cdot z_i(1 - z_i) = t_i \cdot (1 - z_i)$$

The remaining formulas **(b)**, **(c)**, **(d)** above still apply unchanged. For this case formula **(c)** means, for example, that a small change Δw_{ij} to a weight w_{ij} will change $\mathbf{E}$ by $\Delta w_{ij} \delta_i y_j = \Delta w_{ij} t_i (1 - z_i) y_j$. We can reduce $\mathbf{E}$ by setting $\Delta w_{ij} = -\eta t_i (1 - z_i) y_j$, for a suitable learning rate η. This amounts to a form of simple gradient descent. Faster gradient descent algorithms are also available.