CSci 5521 Backpropagation v4 Fall 2018

Consider a feed-forward network with an input layer, a hidden layer, and an output layer:

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_m		$\hat{y}_n$	$\rightarrow g \rightarrow$	y_n		$\hat{z}_p$	$\rightarrow g \rightarrow$	z_p

where g is a "sigmoid" function and

$$\begin{aligned} \hat{y}_j &= v_{j0} + v_{j1}x_1 + v_{j2}x_2 + \cdots + v_{jm}x_m \quad &\text{for} \quad j = 1, \cdots, n \\ \hat{z}_i &= w_{i0} + w_{i1}y_1 + w_{i2}y_2 + \cdots + w_{in}y_n \quad &\text{for} \quad i = 1, \cdots, p \end{aligned}$$

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 m-vector, $\mathbf{y}$, $\hat{\mathbf{y}}$ are n-vectors, $\mathbf{z}$, $\hat{\mathbf{z}}$ are p-vectors, V is an $n \times (m+1)$ matrix of weights, and W is a $p \times (n+1)$ matrix of weights.

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

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

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 $i = 1, 2, \cdots, p$

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

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

The derivative of the sigmoid function $s = g(\hat{s})$ can be written in terms of the output s, so we never need the $\hat{y}$, $\hat{z}$ variables. Example: if $g(\hat{s}) = 1/(1 + e^{-\hat{s}})$ (output in range $0 < s < 1$), then $g'(\hat{s}) = s(1-s)$. If $g(\hat{s}) = \tanh \hat{s} = 2/(1 + e^{-2\hat{s}}) - 1$ (output in range $-1 < s < 1$) then $g'(\hat{s}) = 1 - s^2$. The smoothed ReLU fcn, $s = g(\hat{s}) = [\log(1 + e^{\alpha \hat{s}})]/\alpha$, has derivative $g'(\hat{s}) = 1 - e^{-\alpha s}$, where α sets the sharpness of the corner.

The formula **(c)** means, for example, that a small change Δw_{ij} to a weight w_{ij} will change E by $\Delta w_{ij} \cdot (\partial E / \partial w_{ij}) = \Delta w_{ij} \delta_i y_j = \Delta w_{ij} (z_i - t_i) g'(\hat{z}_i) y_j$. 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 $\Delta w_{ij} = -\eta \cdot \partial E / \partial w_{ij} = -\eta \cdot \delta_i y_j$ 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)**: $w_{ij} \leftarrow w_{ij} - \eta \cdot \partial E / \partial w_{ij}$ and **(d)** $v_{jk} \leftarrow v_{jk} - \eta \cdot \partial E / \partial v_{jk}$, for all i, j, k.

End.

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