(1)

1/

# Derivation of Error Backpropogation - Regression

Start with the end in mind - Let's say that we have a fully-trained regression knowing all our weights & biases. How we compute estimates for our $\hat{y}$ values given an input vector $X$?(1)

Inputs - Each sample m of our input is a D-dimensional column vector:

(1) $$X_m = \begin{bmatrix} x_{1,m} \\ x_{2,m} \\ \vdots \\ x_{D,m} \end{bmatrix}$$

If we have M samples and stack each sample horizontally, we get a D x M matrix that looks like this:

(2) $$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,m} & \cdots & x_{1,M} \\ x_{2,1} & \cdots & x_{2,m} & \cdots & x_{2,M} \\ \vdots & & \vdots & & \vdots \\ x_{D,1} & \cdots & x_{D,m} & \cdots & x_{D,M} \end{bmatrix}$$

We can call each column vector $X_m$ to describe an arbitrary sample m

Hidden-Layer - In our fully-connected network, each input is sent to each node in the first hidden-layer (the only hidden-layer in our 2-layer example). Within each node, the inputs are multiplied by the weights and then summed with a bias term and then a non-linear transform is applied to this weighted sum:

(3) $$z_{i,m}^{[1]} = \left( \sum_{d=1}^{D} w_{d,i}^{[1]T} x_{d,m} \right) + b_i^{[1]}$$

(4) $$a_{i,m}^{[1]} = g^{[1]}\left(z_{i,m}^{[1]}\right)$$

The superscript [1] we use here just denotes the layer we are computing from. The index i refers to the node. The m refers to the sample as described above.

(1) This is called doing a "forward pass" thru the network

The $z^{[l]}_{i,m}$ values for an arbitrary layer $l$ can be collected as a vector and written as:

(5) $$Z^{[l]}_m = \begin{bmatrix} z^{[l]}_{1,m} \\ z^{[l]}_{2,m} \\ \vdots \\ z^{[l]}_{n^{[l]},m} \end{bmatrix}$$

where $n^{[l]}$ is the number of nodes in layer $l$: a hyperparameter

If we stack the vectors generated by each sample horizontally, like we did for $X$ we'd create the following $n^{[l]} \times m$ matrix:

(6) $$Z^{[l]} = \begin{bmatrix} z^{[l]}_{1,1} & z^{[l]}_{1,2} & \cdots & z^{[l]}_{1,m} & \cdots & z^{[l]}_{1,M} \\ z^{[l]}_{2,1} & z^{[l]}_{2,2} & \cdots & z^{[l]}_{2,m} & \cdots & z^{[l]}_{2,M} \\ \vdots & \vdots & & \vdots & & \vdots \\ z^{[l]}_{n^{[l]},1} & z^{[l]}_{n^{[l]},2} & \cdots & z^{[l]}_{n^{[l]},m} & \cdots & z^{[l]}_{n^{[l]},M} \end{bmatrix}$$

To get the outputs from layer $l$, we apply an activation function element-wise to (6) which we'll write as:

(7) $$A^{[l]} = g^{[l]}(Z^{[l]}) = g^{[l]}\left( \begin{bmatrix} z^{[l]}_{1,1} & \cdots & z^{[l]}_{1,m} & \cdots & z^{[l]}_{1,M} \\ \vdots & & \vdots & & \vdots \\ z^{[l]}_{n^{[l]},1} & \cdots & z^{[l]}_{n^{[l]},m} & \cdots & z^{[l]}_{n^{[l]},M} \end{bmatrix} \right)$$

$n^{[l]}$ (↕) and $M$ (↔)

In general, for L-layer, the output for each layer $l$ can be written as:

(8) $$A^{[l]} = g^{[l]}\left( W^{[l]T} A^{[l-1]} + B^{[l]} \right)$$

where $g^{[l]}$ is the activation function applied to layer $l$: another hyperparameter

So to do a complete forward pass, we use our input matrix $X$ to compute $Z^{[1]}$ by computing the output of each node in layer $l=1$ using (3), stacking them vertically into column vectors as shown in (5), then stacking the column vectors horizontally to form the matrix shown in (6).

With $Z^{[1]}$, we apply $g^{[1]}$ element-wise to get the activations $A^{[1]}$ as shown in (7). If we had an $L$-layer network, we'd cont. to compute activations for each layer using (8) until we get to our last layer $A^{[l=L]}$ where $L$ = number of layers in the network. Since this is last layer $A^{[L]} = Y$ and we're done.

Since we only have 2 layers, we only need to compute:

(9) $$X \xrightarrow{g^{[1]}} A^{[1]} \xrightarrow{g^{[2]}} Y$$

In our 2-layer regression net, $g^{[2]}$ is just the identity function, so $Z^{[2]} = Y$. When we modify our net for classification, we'll use a soft-max function for $g^{[2]}$, but we'll save these details for later.

So to summary, to compute the estimates for $Y$ with our 2-layer net we need to compute:

(10) $$Y = W^{[2]T} A^{[1]} + B^{[2]} = W^{[2]T}\left(g^{[1]}\left(W^{[1]T} X + B^{[1]}\right)\right) + B^{[2]}$$

We didn't talk about those $b_i^{[l]}$ values or those $B^{[l]}$ vectors. These are called "bias" values and are just another set of parameters we need to estimate along with the weights.

Now we know how to compute otpts from a given set of (weights & biases). Next, we need to get good values for these. We'll use gradient decent to find a set of weights which minimizes an error function. For regression, the mean square error (MSE) works well. This is defined as:

(11) $$E = \frac{1}{2}\frac{1}{N}\frac{1}{K}\sum_{m=1}^{N}\sum_{k=1}^{K}\left(y_k(x_m) - t_{k,m}\right)^2$$

Why the $\frac{1}{2}$? Well, it's just a convenience and minimizing $\frac{1}{2}E$ is the same as minimizing $E$, so we'll use it.

The way gradient decent works is by the following steps:

1) Initialize all our weights & biases to random values
2) Do a forward pass with these weights to compute outputs $Y$
3) Then update the weights using equation (12) until $E$ in (11) is sufficiently minimized

(12) $$w_{p,q}^{[l]} = w_{p,q}^{[l]} - \rho_{[l]}\frac{\partial E}{\partial w_{p,q}^{[l]}}$$

where $p$ is the $p^{th}$ weight of node $q$ in layer $l$

For layer 1 (hidden)

(13) $$w_{d,i}^{[1]} = w_{d,i}^{[1]} - \rho_1\frac{\partial E}{\partial w_{d,i}^{[1]}}$$

where $d$ is the $d^{th}$ dimension of input $X$ and $i$ is the $i^{th}$ node in layer 1

For layer 2 (output), (12) becomes

(14) $$w_{i,k}^{[2]} = w_{i,k}^{[2]} - \rho_2 \frac{\partial E}{\partial w_{i,k}^{[2]}}$$ where $i'$ is the $i^{th}$ weight for node $k$ in layer 2

The values $\rho_1$ & $\rho_2$ are referred to as the learning rate for layers 1 & 2 respectively. These 2 values are also hyperparameters.

So our next job is to figure out the gradient terms in (13) & (14). We'll start with the one in (14):

Let's define error for a single sample as:

(15) $$E_m = \frac{1}{2} \sum_{k=1}^{K} \left( y_k(x_m) - t_{k,m} \right)^2$$ which allow us to rewrite (11) as:

(16) $$E = \frac{1}{M} \frac{1}{K} \sum_{m=1}^{M} E_m$$ which allows us to rewrite (14) as:

(17) $$w_{i,k}^{[2]} = w_{i,k}^{[2]} - \rho_2 \frac{1}{M} \frac{1}{K} \sum_{m=1}^{M} \frac{\partial E_m}{\partial w_{i,k}}$$

(18) $$\frac{\partial E_m}{\partial w_{i,k}^{[2]}} = -\left( y_k(x_m) - t_{k,m} \right) \frac{\partial y_k(x_m)}{\partial w_{i,k}}$$ Notice how the $\frac{1}{2}$ goes away...

$$y_k(x_m) = w_k^{[2]T} A^{[1]} + b_k^{[2]}$$

(19) $$= w_{1,k}^{[2]} a_1^{[1]} + w_{2,k}^{[2]} a_2^{[1]} + \cdots + \underset{i,k}{w^{[2]}} \underset{i}{a^{[1]}} + \cdots + w_{n^{[1]},k}^{[2]} a_{n^{[1]}}^{[1]} + b_k^{[2]}$$

Looking closely at (19), we see that only one term is not considered a constant wrt. LHS of (18):

(20) $$\frac{\partial y_k(x_m)}{\partial w_{i,k}^{[2]}} = a_i^{[2]} \quad \therefore \quad (18) \text{ becomes:}$$

(21) $$\frac{\partial E_m}{\partial w_{i,k}^{[2]}} = \left(y_k(x_m) - t_{k,m}\right) a_i^{[2]}$$

Now for the derivative in (13):

(20) $$\frac{\partial E_m}{\partial w_{d,i}^{[1]}} = \sum_{k=1}^{K} \left(y_k(x_m) - t_{k,m}\right) \frac{\partial y_k(x_m)}{\partial w_{d,i}}$$

Chain Rule to the rescue for this term

(21) $$\frac{\partial y_k(x_m)}{\partial w_{d,i}} = \frac{\partial y_k(x_m)}{\partial a_i^{[1]}} \frac{\partial a_i^{[1]}}{\partial w_{d,i}}$$ From (19), this derivative is:

(22) $$\frac{\partial y_k(x_m)}{\partial a_i^{[1]}} = w_{i,k}^{[2]}$$ Apply the Chain Rule again to evaluate this term:

(23) $$\frac{\partial a_i^{[1]}}{\partial w_{d,i}^{[1]}} = \frac{\partial a_i^{[1]}}{\partial z_{i,m}^{[1]}} \frac{\partial z_{i,m}^{[1]}}{\partial w_{d,i}^{[1]}}$$ Using (4) to evaluate this term:

(24) $$\frac{\partial a_i^{[1]}}{\partial z_{i,m}^{[1]}} = g'^{[1]}\left(z_{i,m}^{[1]}\right)$$ To evaluate this expression, we'll need to select $g^{[1]}$. We'll leave it general for now.

Using (5) to evaluate the other term:

$$\frac{\partial z_{i,m}^{[1]}}{\partial w_{d,i}^{[1]}} = \frac{\partial}{\partial w_{d,i}^{[1]}}\left( w_{1,i}^{[1]T} x_{1,m} + w_{2,i}^{[1]T} x_{2,m} + \cdots + w_{d,i}^{[1]T} x_{d,m} + \ldots + w_{D,i}^{[1]T} x_{D,m} \right)$$

(25) $$\frac{\partial z_{i,m}^{[1]}}{\partial w_{d,i}^{[1]}} = x_{d,m}$$

Putting it all together:

(26) $$\frac{\partial E_m}{\partial w_{d,i}^{[1]}} = \sum_{k=1}^{K} \left( y_k(x_m) - t_{k,m} \right) w_{i,k}^{old\,[2]} g'\left( z_{i,m}^{[1]} \right) x_{d,m}$$

From (16), we can write:

(27) $$\boxed{\frac{\partial E}{\partial w_{d,i}^{[1]}} = \frac{1}{M}\frac{1}{K} \sum_{m=1}^{M} \sum_{k=1}^{K} \left( y_k(x_m) - t_{k,m} \right) w_{i,k}^{old\,[2]} g'^{[1]}\left( z_{i,m} \right) x_{d,m}}$$

Similarly, we can write for the layer 2 weights:

(28) $$\boxed{\frac{\partial E}{\partial w_{i,k}^{[2]}} = \frac{1}{M}\frac{1}{K} \sum_{m=1}^{M} \left( y_k(x_m) - t_{k,m} \right) a_i^{[2]}}$$

The next step is to get expressions for the bias terms. After deriving the 2 bias expressions, we'll put these into equations (13), (14) and two more for the bias terms which we still need to derive. The resulting 4 update equations are all we need to implement gradient descent, but we'll still want to do one final step: convert these update rules to matrix expressions.