Back Propagation Rule

Consider a model M with l layers. Let H^i represent the i^{th} layer of the architecture, such that, H^0 is the input layer and H^l is the output layer. Let M and N be the size of input feature vector and output vector of the network respectively. Therefore $\mathbb{R}^M$ and $\mathbb{R}^N$ is the cartesian set of all possible input vectors and output vectors respectively.

$$input \in \mathbb{R}^M \qquad output \in \mathbb{R}^N$$

Without the loss of generality let us consider the activation function at each layer to be ϕ, where ϕ is a real valued function that obeys universal approximation theorem. Let weight vector of the synapses between H^i and $H^{(i+1)}$ ($i \in \mathbb{W}$) be $w^{(i)}$, $h^{(i)}$ be the vector representing the perceptrons of the i^{th} layer and $b^{(i)}$ be the vector representing the bias applied to the perceptrons of the i^{th} layer. Thus during the feed forward process the value for the next layer is computed as:

$$\begin{aligned} h^{(k+1)} &= \phi\left(\sum_{i=1}^{N} {w_i}^{(k)} {h_i}^{(k)}\right) \\ &= \phi((w^{(k)})^T h^{(k)} + b^{(k)}) \\ &= \phi(w^{(k)}, h^{(k)}) \end{aligned} \tag{1}$$

$h^{(0)} = x = input\ feature\ vector$

$h^{(1)} = \phi(w^{(0)}, h^{(0)})$

$h^{(2)} = \phi(w^{(1)}, h^{(1)}) = \phi(w^{(1)}, \phi(w^{(0)}, h^{(0)}))$

$h^{(3)} = \phi(w^{(2)}, h^{(2)}) = \phi(w^{(2)}, \phi(w^{(1)}, h^{(1)})) = \phi(w^{(2)}, \phi(w^{(1)}, \phi(w^{(0)}, h^{(0)})))$

$\hat{y} = h^{(l)} = \phi(w^{(l-1)}, \phi(w^{(l-2)}, \ . \ . \ ., \phi(w^{(2)}, \phi(w^{(1)}, \phi(w^{(0)}, h^{(0)})))))$

Where $\hat{y}$ is the predicted value of the network for the input feature vector x. Let y is the ground truth label for x. The amount by which the predicted value ($\hat{y}$) deviates from the ground truth value (y) is called the *cost* of the model. There are several functions to calculate the cost function for the model. These are few that are more frequently used

$$Mean\ Squared\ Error = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$$Mean\ Absolute\ Error = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

$$Binary\ Cross\ Entropy\ = -\frac{1}{N}\sum_{i=1}^{N}[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)]$$

$$Categorical\ Cross\ Entropy\ = -\sum_{i=0}^{M}\sum_{j=0}^{N} y_{ij}\ln \hat{y}_{ij}$$

Note: Categorical Cross Entry is used for multi-class and multi-label classification problems, whereas binary cross entropy is used for uni-class classification problems.

Calculating Gradient:

During the training phase, the model which is initialised with random weight will adjust its weights $(w^{(0)}, w^{(1)}, w^{(2)}, \ .\ ., w^{(l-1)})$. Once the model predicts an output it compares the predicted value with the ground truth and finds the cost using a cost function J. The error is then back propagated from the output layer to the input layer until all the weights are updated. In general the weights are updated by the rule:

$$w^{(i)} \leftarrow w^{(i)} + \eta(-\nabla_{w^{(i)}}); \quad \nabla_{w^{(i)}} = \frac{\partial J}{\partial w^{(i)}} \tag{2}$$

Where η is the learning rate and ∇ is the gradient.

The gradient (∇) is opposite to the direction of the minima of the cost function J, thus when the weights are updated the sign of the gradient is flipped so that the model reaches the minima. In order to calculate the new weights we need only the gradient. During the back propagation this gradient is calculated and back propagated using chain rule of differentiation. Consider the output layer and the layer that is directly connected with the output layer (the penultimate layer). According to our description of the model in (1) the output $\hat{y}$ can be expressed as:

$$\hat{y} = \phi\big(w^{(l-1)}, h^{(l-1)}\big) = \phi\left(\sum_{i=1}^{N} {w_i}^{(l-1)} {h_i}^{(l-1)}\right)$$

For the same of simplicity let us consider that the output layer is activated using Rectified Liner Unit function, which is given by :

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

If $\hat{y} < 0$ then the output is zero then the gradient will be also zero, in which case there will be no back propagation. Now let us consider some non-zero value of $\hat{y}$. Therefore

$$\hat{y} = \sum_{i=1}^{N} {w_i}^{(l-1)} {h_i}^{(l-1)}$$

For this particular case let us consider Mean Squared Error function for calculation the cost. You can use any suitable function, but the for the sake of simplicity of math involved we stick to MSE for now. The cost can be written as:

$$J(w^{(l-1)}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{4}$$

$$\Rightarrow \quad J(w^{(l-1)}) = \frac{1}{N}\sum_{i=1}^{N}((w^{(l-1)})^T h^{(l-1)} - y_i)^2$$

$$\Rightarrow \quad J(w^{(p)}) = \frac{1}{N}\sum_{i=1}^{N}((w^{(p)})^T h^{(p)} - y_i)^2 \qquad [\because Let\ p = l-1\ ]$$

To calculate the gradient of the to update an element $w_k^{(p)}$ we partially differentiate (4) with respect to $w_k^{(p)}$.

$$\begin{aligned}\frac{\partial J}{\partial w_k^{(p)}} &= \frac{\partial}{\partial w_k^{(p)}}\frac{1}{N}\sum_{i=1}^{N}((w^{(p)})^T h^{(p)} - y_i)^2 = \frac{1}{N}\frac{\partial}{\partial w_k^{(p)}}\sum_{i=1}^{N}((w^{(p)})^T h^{(p)} - y_i)^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}\frac{\partial}{\partial w_k^{(p)}}((w^{(p)})^T h^{(p)} - y_i)^2 \\ &= \frac{2}{N}\sum_{i=1}^{N}((w^{(p)})^T h^{(p)} - y_i)\frac{\partial}{\partial w_k^{(p)}}((w^{(p)})^T h^{(p)} - y_i) \quad [\ Chain\ Rule\ ]\end{aligned} \tag{5}$$

In the above equation the parameter y_i is the ground truth and is independent of any weights. Thus differentiating that will yield 0. The term $(w^{(p)})^T h^{(p)}$ can be expanded as,

$$(w^{(p)})^T h^{(p)} = w_0^{(p)}h_0^{(p)} + w_1^{(p)}h_1^{(p)} + w_2^{(p)}h_2^{(p)} + \ \ldots + w_k^{(p)}h_k^{(p)} + \ \ldots + w_N^{(p)}h_N^{(p)}$$

Thus while partially differentiating the term $(w^{(p)})^T h^{(p)}$ with respect to $w_i^{(p)}$, all terms except $w_i^{(p)}h_i^{(p)}$ will become zero. Thus (5) can be written as.

$$\begin{aligned}\frac{\partial J}{\partial w_k^{(p)}} &= \frac{2}{N}\sum_{i=1}^{N}((w^{(p)})^T h^{(p)} - y_i)\frac{\partial}{\partial w_k^{(p)}}(w_k^{(p)}h_k^{(p)}) \\ &= \frac{2}{N}\sum_{i=1}^{N}((w^{(p)})^T h^{(p)} - y_i)h_k^{(p)} \\ &= \nabla_{w_k^{(p)}}\end{aligned} \tag{6}$$

□

We calculate the gradient using equation (6) and substitute the gradient in (2) to calculate the new value of $w_i^{(p)}$. We repeat the same process for N number of times to update all values of $w^{(p)}$. This completes the correction of weights at the penultimate layer.

Back Propagating Gradient:

We need to propagate the gradient from last layer to the input layer and update all weights that we encounter in the path. From (2) we know that the weight correction is only dependent on the gradient(∇), this we need to calculate the partial derivative of the cost function with respect to the weight value that we want to update. Since no weights ,except the weights that connects to the output layer, is directly related to the cost function in the network, we follow chain rule to find the gradient of weights all subsequent preceding layers.

Let us consider a i^{th} weight $w_i^{(q)}$, connecting to the layer $(q+1)$ that we want to update. To do this we need the gradient of the cost function J with respect to the weight: $\frac{\partial J}{\partial w_i^{(q)}}$.

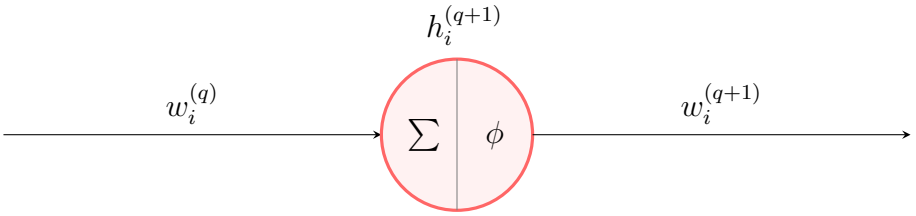

Figure 1: Perceptron of (q+1) layer in the network

The gradient of the cost function J with respect to $w_i^{(q+1)}$ can be written as:

$$
\begin{aligned}
\nabla_{w_i^{(q)}} &= \frac{\partial J}{\partial w_i^{(q)}} \\
&= \frac{\partial h_i^{(q+1)}}{\partial w_i^{(q)}} \frac{\partial J}{\partial h_i^{(q+1)}} \\
&= \frac{\partial h_i^{(q+1)}}{\partial w_i^{(q)}} \frac{\partial w_i^{(q+1)}}{\partial h_i^{(q+1)}} \frac{\partial J}{\partial w_i^{(q+1)}} \\
&= \frac{\partial h_i^{(q+1)}}{\partial w_i^{(q)}} \frac{\partial w_i^{(q+1)}}{\partial h_i^{(q+1)}} \frac{\partial h_i^{(q+2)}}{\partial w_i^{(q+1)}} \frac{\partial J}{\partial h_i^{(q+2)}} \\
&= \frac{\partial h_i^{(q+1)}}{\partial w_i^{(q)}} \frac{\partial w_i^{(q+1)}}{\partial h_i^{(q+1)}} \frac{\partial h_i^{(q+2)}}{\partial w_i^{(q+1)}} \cdots \frac{\partial w_i^{(p)}}{\partial h_i^{(p-1)}} \frac{\partial J}{\partial w_i^{(p)}} \\
\nabla_{w_i^{(q)}} &= \left(\prod_{j=q}^{p-1} \frac{\partial h_i^{(j+1)}}{\partial w_i^{(j)}} \frac{\partial w_i^{(j+1)}}{\partial h_i^{(j+1)}} \right) \frac{\partial J}{\partial w_i^{(p)}} \qquad \square
\end{aligned}
\tag{7}
$$

Using Result of (7) we can find the gradient of the cost function with respect to any weight and using (2) we can update the weights.

Activation Functions(ϕ)

$$Rectified\ Linear\ Unit\ =\ \phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$Sigmoid\ =\ \phi(x)\ =\ \frac{1}{1+e^{-x}}$$

$$Softmax\ =\ \phi(x_1, x_2,\ \ldots, x_N)\ =\ \frac{e^{x_p}}{\sum_{i=1}^{N} e^{x_i}}$$

$$Hyperbolic\ Tangent\ =\ \phi(x)\ =\ \frac{e^{2x}-1}{e^{2x}+1}$$

Note: These are few of the activation that will be frequently used in the perceptron. Out of these, Softmax function is a function that takes N inputs as parameters and computes the value. This is often used in the output layer if problem demands for a probability distribution instead of deterministic results.