Andriy Sheptunov

1. Forward Propagation and General Definitions

Take a multi-layer neural net, and consider its forward propagation procedure between any two arbitrary adjacent layers. Let m be the dimension of any output layer, n be the dimension of its preceding layer, and w_{nm} be the weight from neuron of index n from the input layer onto neuron indexed m in the output layer. Let $x_{1\dots n}$ represent neurons in the input layer, and $y_{1\dots m}$ represent neurons in the output layer. Finally, define a conventional sigmoid function ϕ as follows:

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

The sigmoid function is chosen as it is continuous and easily differentiable:

$$\frac{\partial \phi}{\partial z} = \phi(z)\big(1 - \phi(z)\big)$$

We will use this differential later in back propagation.
Using these definitions, the input layer will be of the form

$$\text{Input layer} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Then, the subsequent computed output layer, representing a single step of forward propagation, is as follows:

$$\text{Output layer} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} \phi(\sum w_{i1}x_i) \\ \phi(\sum w_{i2}x_i) \\ \vdots \\ \phi(\sum w_{im}x_i) \end{bmatrix} = \begin{bmatrix} \phi(w_{11}x_1 + w_{21}x_2 + \dots + w_{n1}x_n) \\ \phi(w_{12}x_1 + w_{22}x_2 + \dots + w_{n2}x_n) \\ \vdots \\ \phi(w_{1m}x_1 + w_{2m}x_2 + \dots + w_{nm}x_n) \end{bmatrix}$$

We can introduce $\mathbf{w_{1\dots m}}$ as the matrix of n weights onto $y_{1\dots m}$, and $\mathbf{x}$ as the matrix of inputs $x_{1\dots n}$, in order to rewrite the expression:

$$= \begin{bmatrix} \phi(\mathbf{w_1} \cdot \mathbf{x}) \\ \phi(\mathbf{w_2} \cdot \mathbf{x}) \\ \vdots \\ \phi(\mathbf{w_m} \cdot \mathbf{x}) \end{bmatrix}$$

2. Back Propagation

Consider the process of supervised training for a neural net. Define a training vector with dimension m as the set of desired outputs:

$$\text{Training set} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

and the actual data as the set of calculated outputs:

$$\text{Calculated output} = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \vdots \\ \hat{y_m} \end{bmatrix}$$

Then, we can use a weighted squared difference loss function to calculate the loss (error) from our output:

$$\text{Loss} = L = \frac{1}{2}(y_1 - \hat{y_1})^2 + \frac{1}{2}(y_2 - \hat{y_2})^2 + \ldots + \frac{1}{2}(y_m - \hat{y_m})^2$$
$$= \sum_{i=1}^{m} \frac{1}{2}(y_i - \hat{y_i})^2$$

Note that $\frac{1}{2}$ is present for ease of differentiation (as is customary) and have no substantial effect.

We can unwrap this using our forward propagation definition on $\hat{y_i}$ to get a better understanding of the motivation of back propagation.

$$L = \sum_{i=1}^{m} \frac{1}{2}\left(y_i - \phi(\mathbf{w_i} \cdot \mathbf{x})\right)^2$$

This shows that the loss, L, is dependent on each of the weights w in the net. Since $\mathbf{x}$ is also a set of values in a layer calculated by applying a sigmoid function to a sum, this loss function is recursive, with a base case when $\mathbf{x}$ represents the input values to the neural net.

The goal of back propagation is to find $\frac{\partial L}{\partial w}$, or the rate of change of the loss with respect to any given weight in the neural net. This differential effectively tells us how much (denoted by magnitude), and in which direction (denoted by sign) to change any given weight to change the loss in some specific way. The ultimate goal is to minimize loss by changing weights in the direction opposite to $\frac{\partial L}{\partial w}$ for each respective weight.

2.1.Gradient for last layer

Andriy Sheptunov

The first task is to compute $\frac{\partial L}{\partial w}$ for an arbitrary edge weight w pointing to the output layer of the neural net. Start by writing out the general expression for $\frac{\partial L}{\partial w}$:

$$\frac{\partial L}{\partial w} = \frac{\partial}{\partial w} \sum_{i=1}^{m} \frac{1}{2}(y_i - \hat{y}_i)^2$$

We apply the sum rule for differentiation in order to move the differential inside the sum:

$$= \sum_{i=1}^{m} \frac{\partial}{\partial w} \frac{1}{2}(y_i - \hat{y}_i)^2$$

We can now apply product rule and chain rule:

$$= \sum_{i=1}^{m} (y_i - \hat{y}_i) \frac{\partial}{\partial w}(y_i - \hat{y}_i)$$

We can apply the sum rule for differentiation again to move the differential inside the difference between the training and calculated data, using the fact that y_i is constant to eliminate it.

$$= \sum_{i=1}^{m} (y_i - \hat{y}_i)(-\frac{\partial}{\partial w} \hat{y}_i)$$

$$= \sum_{i=1}^{m} -(y_i - \hat{y}_i) \cdot \frac{\partial}{\partial w} \hat{y}_i$$

We can expand $\hat{y}_i$ using its definition:

$$= \sum_{i=1}^{m} -(y_i - \hat{y}_i) \cdot \frac{\partial}{\partial w} \phi(\mathbf{w_i} \cdot \mathbf{x})$$

To continue, we can perform chain rule on ϕ (recall that $\phi'(z) = \phi(z)\big(1 - \phi(z)\big)$):

$$= \sum_{i=1}^{m} -(y_i - \hat{y}_i) \cdot \phi(\mathbf{w_i} \cdot \mathbf{x})\big(1 - \phi(\mathbf{w_i} \cdot \mathbf{x})\big) \cdot \frac{\partial}{\partial w}(\mathbf{w_i} \cdot \mathbf{x})$$

This collapses simply to

$$= \sum_{i=1}^{m} -(y_i - \hat{y}_i) \cdot \hat{y}_i \cdot (1 - \hat{y}_i) \cdot \frac{\partial(\mathbf{w_i} \cdot \mathbf{x})}{\partial w}$$

Since $\frac{\partial}{\partial w}$ is defined for a single weight, $\mathbf{w_i} \cdot \mathbf{x}$ simply reduces to 0 for terms where the weights don't correspond to the weight being differentiated against, and the x which corresponds with

the weight we are differentiating with respect to by linearity of differentiation. We get the simplified form

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{m} -(y_i - \hat{y}_i) \cdot \hat{y}_i \cdot (1 - \hat{y}_i) \cdot x$$

This is $\frac{\partial L}{\partial w}$ with respect to any weight on an edge directed into the output layer of the net.

2.2.Gradient for second-to-last layer

If $\frac{\partial L}{\partial w}$ is being calculated for a weight pointing to the second-to-last layer in the network, then the first and second applications of chain rule don't change, and we can jump directly the the application of the third step:

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{m} -(y_i - \hat{y}_i) \cdot \hat{y}_i \cdot (1 - \hat{y}_i) \cdot \frac{\partial(\mathbf{w_i} \cdot \mathbf{x})}{\partial w}$$

Since we will now be referencing a layer prior to **x**, we will shift our definitions, using z_i for y_i, $\hat{z}_i$ for $\hat{y}_i$, and **y** for **x**. We will redefine **x** to be the values in the third-to-last layer, if such exists, and weights $\mathbf{v_i}$ to be the weights on edges pointing to a value in **y**. (Our layer values are labeled **x**, **y**, and **z** now, with **x** being the layer preceding **y**, and **z** being the output layer, **v** being the weights in between **x** and **y**). We can rewrite our expression with new definitions:

$$\frac{\partial L}{\partial v} = \sum_{i=1}^{m} -(z_i - \hat{z}_i) \cdot \hat{z}_i \cdot (1 - \hat{z}_i) \cdot \frac{\partial(\mathbf{w_i} \cdot \mathbf{y})}{\partial v}$$

For simplicity, focus specifically on

$$\frac{\partial(\mathbf{w_i} \cdot \mathbf{y})}{\partial v}$$

In the context of designing a recursive function to perform these operations in computer science, omitting everything but this portion can be understood as passing the omitted portion down as an argument in the recursive function. We can expand the remainder using our vector definitions:

$$= \frac{\partial}{\partial v}(w_{1i}y_1 + w_{2i}y_2 + \ldots w_{ni}y_n)$$

Recall that $y_{1\ldots n}$ are themselves sigmoid neurons, and expand them accordingly:

$$= \frac{\partial}{\partial v}\left(w_{1i}\phi(\mathbf{v_1} \cdot \mathbf{x}) + w_{2i}\phi(\mathbf{v_2} \cdot \mathbf{x}) + \ldots + w_{ni}\phi(\mathbf{v_n} \cdot \mathbf{x})\right)$$

In practice, if we are calculating $\frac{\partial}{\partial v}$ for a single weight, then with respect to all weights in $\mathbf{v_1}$, terms $w_{2i}\phi(\mathbf{v_2} \cdot \mathbf{x}) + \ldots + w_{ni}\phi(\mathbf{v_n} \cdot \mathbf{x})$ will be constant; with respect to all weights in $\mathbf{v_2}$, terms $w_{2i}\phi(\mathbf{v_1} \cdot \mathbf{x}) + \ldots + w_{ni}\phi(\mathbf{v_n} \cdot \mathbf{x})$ will be constant, etc. Thus, we can reduce under the constraint that $v \in \mathbf{v_a}$:

$$= \frac{\partial}{\partial v}\big(w_{ai}\phi(\mathbf{v_a} \cdot \mathbf{x})\big)$$

$$= w_{ai}\frac{\partial}{\partial v}\big(\phi(\mathbf{v_a} \cdot \mathbf{x})\big)$$

$$= w_{ai} \cdot \phi(\mathbf{v_a} \cdot \mathbf{x})\big(1 - \phi(\mathbf{v_a} \cdot \mathbf{x})\big) \cdot \frac{\partial}{\partial v}(\mathbf{v_a} \cdot \mathbf{x})$$

Using our notation from sec 2.1, this simplifies to

$$= w_{ai} \cdot \phi(\mathbf{v_a} \cdot \mathbf{x})\big(1 - \phi(\mathbf{v_a} \cdot \mathbf{x})\big) \cdot \mathbf{x}$$

This is the value of $\frac{\partial(\mathbf{w_i} \cdot \mathbf{y})}{\partial v}$, so we can compose our original function back to get $\frac{\partial L}{\partial v}$:

$$\frac{\partial L}{\partial v} = \sum_{i=1}^{m} -(z_i - \hat{z}_i) \cdot \hat{z}_i \cdot (1 - \hat{z}_i) \cdot w_{ai} \cdot \phi(\mathbf{v_a} \cdot \mathbf{x})\big(1 - \phi(\mathbf{v_a} \cdot \mathbf{x})\big) \cdot \mathbf{x} \qquad v \in \mathbf{v_a}$$

This shit is terrible

Go look at the wikipedia page

This is the shit I desperately tried to find the notation to say:

Putting it all together:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

with

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} (o_j - t_j)o_j(1 - o_j) & \text{if } j \text{ is an output neuron,} \\ (\sum_{\ell \in L} w_{j\ell}\delta_\ell)o_j(1 - o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

Andriy Sheptunov