Back Propagation

August 23, 2017

1 Definition

Back propagation is a method to calculate the gradient of the loss function with respect to the weights in an artificial neural network. It is commonly used as a part of algorithms that optimize the performance of the network by adjusting the weights, for example in the gradient descent algorithm.

Intuitively, once the gradient of loss has been computed at the final layer, it is propagated all the way back to the first layer so that each node in the neural network can update its own respective gradient and adjust the weight accordingly.

2 Example

To illustrate the concept of back propagation, let's do a simple example. Let x, y, z, q and f be 5 nodes with the following properties:

- x = -2
- y = 5
- z = -4
- q(x,y) = x + y = 3
- f(q,z) = q * z = -12

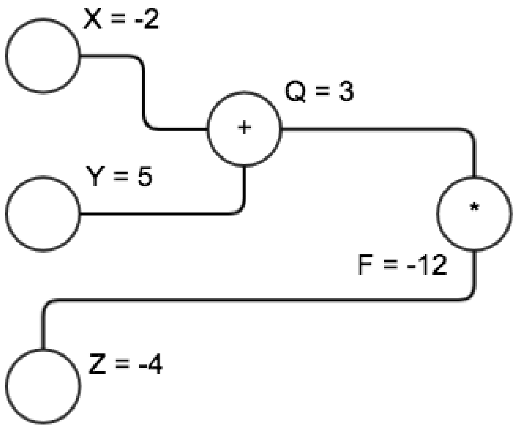

Figure 1:

In order to perform gradient descent, we need to know partial derivative of f with respect to x, y, z. In other words, we need to know how much f changes when either x, y or z changes while the other two are kept constant.

The partial derivatives are denoted as:

$$\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}, \frac{\delta f}{\delta z}$$

Since f is a direct function of z, computing $\frac{\delta f}{\delta z}$ is relatively straightforward:

$$\frac{\delta f}{\delta z} = \frac{\delta}{\delta z} q * z = q \tag{1}$$

Computing the partial derivatives for x and y is a bit more involved, as f is an indirect function of x and y through q. Luckily, we can use the chain rule for this. We must calculate the partial derivate at the last note, then propagate it back to nodes in the previous layer.

$$\frac{\delta f}{\delta x} = \frac{\delta f}{\delta q}\frac{\delta q}{\delta x} = z * 1 = -4 \tag{2}$$

$$\frac{\delta f}{\delta y} = \frac{\delta f}{\delta q}\frac{\delta q}{\delta y} = z * 1 = -4 \tag{3}$$

Increasing x or y by 1 unit while keeping the other two constant will decrease f by 4 units.

3 Forward Pass and Back Propagation

In this example, nodes in one layer (q,f) are computed using the values from the previous layers (x,y,z). This is called the forward pass where actual values of nodes in one layer are passed forward to calculate the values of nodes in the next layers.

In addition to passing forward the actual node values, we also calculate the local gradients, i.e. $\frac{\delta q}{\delta y}, \frac{\delta q}{\delta x}$ and store them in the nodes for later usage.

At the last stage, we calculate the partial derivative of the loss function with respect to the last node (f). We then propagate $\frac{\delta f}{\delta q}$ back to all the nodes in the previous layer (q) and multiply with the local gradient, namely $\frac{\delta q}{\delta y}, \frac{\delta q}{\delta x}$, which were previously calculated and stored in the node. This gives us $\frac{\delta f}{\delta x}$ and $\frac{\delta f}{\delta y}$ respectively.

This idea of computing the gradient cost at the last stage and passing them backward to to calculate the cumulative gradient at each node is called **back propagation**

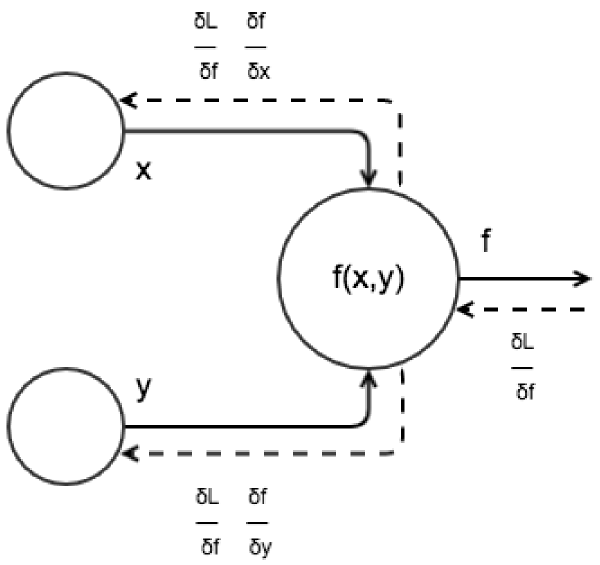

Figure 2:

4 Derivation in Neural Network

In this neural network, the first layer is simply our input data. The second and third layer are the hidden layers. The last layer is the prediction. Each connection between nodes in different layers is accompanied by a weight, which we need to tune to achieve the optimal result for the loss function. Each pre-activated value in a layer is the function of activated values in the previous nodes and the in-between weights. The activated value is simply a sigmoid (or relu) transformation of the pre-activated values. The activated value indicates whether the neuron will fire (pass word) or not.

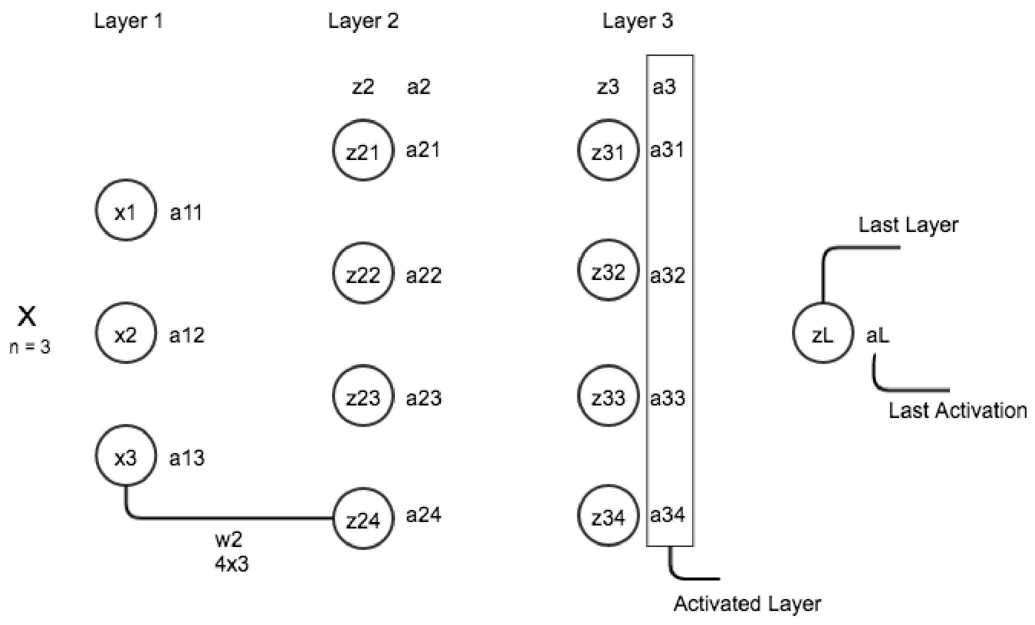

Figure 3:

4.1 Terminology

- x_n: input n
- w^n: weight matrix before layer n, i.e. between layer n-1 and n
- z^n: pre-activated values at layer n
- a^n: activated values at layer n
- w^n_{jk}: weight between node k in layer n-1 to node j in layer n. Note the backward order.
- z^n_i: pre-activated value at node j in layer n
- a^n_j: activated value at node j in layer n
- L(a^L, y): loss function between last layer of activation and y

4.2 Derivation

Our goal is to adjust the weights to reach the optimal value of the loss function. With gradient descent, we need to know $\frac{\delta L}{\delta w^n}$, which is a measure of how much a change in **w** would affect **L**.

We know that:

$$z^n = a^{n-1} \cdot w^n \tag{4}$$

Taking the partial derivative with respect to w^n and a^{n-1} give us:

$$\frac{\delta z^n}{\delta a^{n-1}} = w^n \tag{5}$$

$$\frac{\delta z^n}{\delta w^n} = a^{n-1} \tag{6}$$

The procedure to back propagate is:

1. Input X. Since this is the first layer, set the first activations a^1 to be the same as the input data.
2. Initialize the weights. A quick, easy way is sampling from the standard normal.
3. Feed forward. For each of the layer n, compute

$$\text{Pre-activated value} : z^n = a^{n-1} \cdot w^n \tag{7}$$

$$\text{Activated value} : a^n = \sigma(z^n) \tag{8}$$

4. When the forward pass reaches its last layer, calculate the error of L with regards to last pre-activated node z^L

$$\frac{\delta L}{\delta z^L} = \frac{\delta L}{\delta a^L}\frac{\delta a^L}{\delta z^L} = \frac{\delta L}{\delta a^L}\sigma'(z^L) \tag{9}$$

5. Back propagate error. For each layer n going backward, compute the product of the local gradient and the propagated gradient:

$$\frac{\delta L}{\delta z^n} = \frac{\delta L}{\delta z^{n+1}}\frac{\delta z^{n+1}}{\delta a^n}\frac{\delta a^n}{\delta z^n} = \frac{\delta L}{\delta z^{n+1}} * w^n * \sigma'(z^n) \tag{10}$$

6. Calculate gradient of loss with respect to weight:

$$\frac{\delta L}{\delta w^n} = \frac{\delta L}{\delta z^n}\frac{\delta z^n}{\delta w^n} \tag{11}$$

7. Adjust the weight:

$$w^n = w^n - \alpha\frac{\delta L}{\delta w^n} \tag{12}$$