Backpropagation

Roman Soletskyi

February 17, 2018

1 Introduction

Let us have a feedforward neural network, which consists of 3 layers: input, hidden, and output layers. Let us denote vector (or Python list) constructed from all values of input layer neurons as $\mathbf{x}$, hidden one as $\mathbf{y}$, and output one as $\mathbf{z}$. Letters i,j,k are going to be used to denote indices of neuron values of input, hidden, and output layers, accordingly. For example, x_i denotes output value of i neuron from input layer. Let us denote a weight of connection between i input layer neuron and j hidden layer neuron as w^a_{ij}, which is equivalent to wa[i][j] in Python and connection between j hidden layer neuron and k output layer neuron as w^b_{jk}.

Finally, further formulas are applicable to sigmoid function and its derivative:

$$\sigma(x) = \frac{1}{1+e^{-x}},\ \sigma'(x) = \sigma(x)(1-\sigma(x)) \tag{1}$$

However, most arguments work for any activation function.

2 Loss function

Training of neural network consists of, basically, reducing loss $\mathcal{L}$. Loss $\mathcal{L}$ is usually calculated as squared error measure:

$$\mathcal{L} = \frac{1}{2}\sum_k (z_k - \hat{z_k})^2 \tag{2}$$

where $\hat{z_k}$ denotes expected output value. Training can be done only by changing values of weights in tables w^a and w^b, which means that $\mathcal{L}$ is a function of several variables and weights are its variables. Thus, if we can calculate the partial derivative of $\mathcal{L}$ with respect to each weight, we can reduce loss by applying gradient descent, which was discussed in the previous lecture. In the simplest case of vanilla gradient descent weight w can be iteratively updated as:

$$w^{t+1} = w^t - \eta\frac{\partial \mathcal{L}}{\partial w^t} \tag{3}$$

3 Partial derivatives

3.1 w^b table

Firstly, we are going to calculate the partial derivative of loss $\mathcal{L}$ with respect to weights of table w^b, using the chain rule. Obviously, weight w^b_{jk} influences only on value z_k since it influences connection between y_j and z_k only. Considering this fact and using the chain rule, it can be written:

$$\frac{\partial \mathcal{L}}{\partial w^b_{jk}} = \frac{\partial \mathcal{L}}{\partial z_k}\cdot\frac{\partial z_k}{\partial s_k}\cdot\frac{\partial s_k}{\partial w^b_{jk}}, \text{where } s_k = \sum_j y_j w^b_{jk}, \text{and } z_k = \sigma(s_k) \tag{4}$$

$\frac{\partial \mathcal{L}}{\partial z_k}$ can be found from the definition of the loss (2) as $\frac{\partial \mathcal{L}}{\partial z_k} = z_k - \hat{z_k}$. $\frac{\partial z_k}{\partial s_k}$ can be found from the derivative of the sigmoid function (1) as $\frac{\partial z_k}{\partial s_k} = z_k(1-z_k)$. $\frac{\partial s_k}{\partial w^b_{jk}}$ can be found from the definition of the sum as $\frac{\partial s_k}{\partial w^b_{jk}} = y_j$. Eventually, we get:

$$\frac{\partial \mathcal{L}}{\partial w^b_{jk}} = (z_k - \hat{z_k})z_k(1-z_k)y_j \tag{5}$$

3.2 w^a table

Weight w^a_{ij} influences on value y_j only from the same logic that was used in 3.1. However, value y_j influences **all** output values $\mathbf{z}$, which makes partial derivative a lit bit more sophisticated. Again, from chain rule:

$$\frac{\partial \mathcal{L}}{\partial w^a_{ij}} = \frac{\partial \mathcal{L}}{\partial y_j} \cdot \frac{\partial y_j}{\partial s_j} \cdot \frac{\partial s_j}{\partial w^a_{ij}}, \text{where } s_j = \sum_i x_i w^a ij \tag{6}$$

Two last partial derivatives can be found by analogy with previous section:

$$\frac{\partial y_j}{\partial s_j} \cdot \frac{\partial s_j}{\partial w^a_{ij}} = y_j(1-y_j)x_i$$

Let us use the definition of the loss $\mathcal{L}$ to simplify the first partial derivative:

$$\frac{\partial \mathcal{L}}{\partial y_j} = \frac{1}{2}\frac{\partial}{\partial y_j}\Big(\sum_k (z_k - \hat{z_k})^2\Big) = \frac{1}{2}\sum_k \frac{\partial}{\partial y_j}\Big((z_k - \hat{z_k})^2\Big) = \sum_k \frac{\partial z_k}{\partial y_j}(z_k - \hat{z_k})$$

The partial derivative $\frac{\partial z_k}{\partial y_j}$ can be found from the definition of the sigmoid function (1) and the definition of the s_k from (4):

$$\frac{\partial z_k}{\partial y_j} = \frac{\partial z_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial y_j} = z_k(1-z_k)w^b_{jk}$$

Finally, we can substitute all three partial derivatives into (6):

$$\frac{\partial \mathcal{L}}{\partial w^a_{ij}} = \Big(\sum_k (z_k - \hat{z_k}) z_k(1-z_k) w^b_{jk}\Big) y_j(1-y_j)x_i \tag{7}$$

4 Backpropagation

Formulas (5) and (7) can be simplified and converted into more strict expression, which is going to work for feedforward neural network with an arbitrary number of hidden layers, by denoting new values δ_j and δ_k, which are equal to:

$$\delta_k = (z_k - \hat{z_k})z_k(1-z_k)$$
$$\delta_j = \Big(\sum_k \delta_k w^b_{jk}\Big) y_j(1-y_j)$$

Then wanted partial derivatives of the loss $\mathcal{L}$ with respect to weights are equal to:

$$\frac{\partial \mathcal{L}}{\partial w^b_{jk}} = \delta_k y_j$$
$$\frac{\partial \mathcal{L}}{\partial w^a_{ij}} = \delta_j x_i$$

Let us change the definition of δ_j for combining and δ_j and δ_k. Moreover let us reconsider i,j,k denoting j as a current layer, i as a previous layer, and k as a next layer (if exists); denote $\mathbf{x}$ as a vector constructed from all outputs of previous layer neurons, $\mathbf{y}$ as a vector from outputs of current layer neurons, and $\mathbf{z}$ as a vector from outputs of next layer neurons. A direction of previous, current, and next layers coincides with the direction input-output layers.

$$\delta_j = \begin{cases} (y_j - \hat{y_j})y_j(1-y_j), & \text{if j is an output neuron} \\ \Big(\sum_k \delta_k w_{jk}\Big) y_j(1-y_j), & \text{if j is a hidden neuron} \end{cases} \tag{8}$$

Then partial derivative of a loss with respect to weight w_{ij} is equal to:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \delta_j x_i \tag{9}$$

It can be seen that in order to calculate δ_j, δ_k or δ_j viewing from the next layer must be calculated. It can be implied that calculation of an output value is performed upwards (from input to output layer), but the calculation of an error is executed downwards (starting from output layer and ending at the last hidden layer), which gives this algorithm its name - backpropagation.