Backpropagation

Calculcating the gradient for weight update

Irene Nikkarinen

April 27, 2018

1 Updating a weight in gradient descent

$$w_{ik} = w_{ik} - d_{ik} \cdot a$$

where a is the learning rate and d_{ij} the the gradient of weights between the neurons i and k.

The gradient is the effect that a specific weight has on the loss function. This can be obtained with the derivative

$$d_{ik} = \frac{\delta E}{\delta w_{ik}}$$

2 Calculating the gradient for each weight

Let out_k be the ouput of the neuron k, in_k the input of a neuron k, f the neuron's activation function and y_i the true label of the input x_i.

Calculcating the derivative with respect to a weight w_{ik} between the neurons i and k on layer l in a fully connected neural network:

According to the chain rule:

$$\frac{\delta E}{\delta w_{ik}} = \frac{\delta E}{\delta out_k} \cdot \frac{\delta out_k}{\delta in_k} \cdot \frac{\delta in_k}{\delta w_{ik}}$$

The input of a neuron consists of the sum of all the outputs of the neuron receives multiplied by the weight of each connection.

In other words, for neuron k on layer l:

$$in_k = \sum_{j \in l-1} w_{jk} \cdot out_j$$

and with f_i being the activation function of neuron i:

$$out_i = f_i(in_i)$$

Let's calculate the terms we need to define the gradient.

Starting with the simplest one. Remember, that the input of a specific neuron is the outputs of

$$\frac{\delta in_k}{\delta w_{ik}} = \frac{\delta \left(\sum_{j \in l-1}^{|l|} w_{jk} \cdot out_j + b_j \right)}{\delta w_{ik}}, \quad |l| \geq k$$

When we derive w.r.t w_{jk}, all other weights, as well as biases are treated as real numbers, and their derivative is zero. So luckily for us, the formula simplifies to the following:

$$\begin{aligned} \frac{\delta in_k}{\delta w_{ik}} &= \frac{\delta w_{ik} \cdot out_i}{\delta w_{ik}} \\ &= out_i \end{aligned}$$

This is why we want to save all the outputs of each neuron.

In the attempt of making a things more clear (and match the notation in the code), let's make the following notation:

$$e_k = \frac{\delta E}{\delta out_k} \cdot \frac{\delta out_k}{\delta in_k}$$

We shall refer to e as the error.

The error consits of two terms. The second one is nice and simple:

$$\frac{\delta out_k}{\delta in_k} = \frac{\delta f_k(in_k)}{\delta in_k} = f_k'(in_k)$$

It's also rather easy to implement: you can derive the activation function by hand, and then build a function which gives the value of the derivative in a specific point (you can go check out the DifferentiableFunction class in functions.py). This is also why we have to save all the inputs of each neuron in forward pass.

The first term, $\frac{\delta E}{\delta out_k}$, is a bit more complicated. Backpropagation is actually recursive: It calculates the gradient of a specific weight based on the output that weight affects and the error of the neuron taking that output as an input. The base case is the derivative of the loss function w.r.t the

output of the last layer. Intuitively this is done, because all the errors in the network affect the errors in future layers.

As a result. the definition is different for the output layer, and all other layers, since the output layer doesn't have a layer coming after it.

If l is in output layer and out is the output of whole the network.

$$\frac{\delta E}{\delta out_l} = \frac{\delta E}{\delta out}$$

This is a relatively simple calculation, but it depends on the choice of the error function.

For the batch-wise mean squared error, with y_i the true label of an input, and out_i the output of the network on batch i (this might be a bit confusing but out_i is actually a vector of outputs here), and out_j a specific output on a batch (not a vector):

$$\begin{aligned}\frac{\delta\left(\frac{1}{b|out|}\sum_{i=0}^{b}(y_i - out_i)^2\right)}{\delta out_j} &= \frac{\delta\frac{1}{b|out|}(y_j - out_j)^2}{\delta out_j}, \quad b \geq j \\ &= \frac{2}{b|out|}(out_j - y_j)\end{aligned}$$

If l is not in the output layer, we calculate the following using the next layer $l+1$:

$$\begin{aligned}\frac{\delta E}{\delta out_k} &= \sum_{j\in l+1}\left(\frac{\delta E}{\delta in_j}\cdot\frac{\delta in_j}{\delta out_k}\right) \\ &= \sum_{j\in l+1}\left(\frac{\delta E}{\delta out_j}\cdot\frac{\delta out_j}{\delta in_j}\cdot\frac{\delta in_j}{\delta out_k}\right) \\ &= \sum_{j\in l+1}\left(\frac{\delta E}{\delta out_j}\cdot\frac{\delta out_j}{\delta in_j}\cdot w_{kj}\right) \\ &= \sum_{j\in l+1}(e_j\cdot w_{kj}) \quad \text{Remember the error } e_k?\end{aligned}$$

So $\frac{\delta E}{\delta out_k}$ depends on the error of the next layer.
Recalling the formula of the error e_k:

$$e_k = \frac{\delta E}{\delta out_k}\cdot\frac{\delta out_k}{\delta in_k}$$

Now we can plug in $\frac{\delta E}{\delta out_k}$:

$$\begin{aligned} e_k &= \frac{\delta E}{\delta out_k} \cdot \frac{\delta out_k}{\delta in_k} \\ &= \sum_{j \in l+1} (e_j \cdot w_{kj}) \cdot f'(in_k) \end{aligned}$$

So the error of a specific layer depends recursively on the error of the next one. This is what puts the "back" to "backpropagation".

Let's take a look back at the original formula, and place in the terms we've already defined:

$$\begin{aligned} \frac{\delta E}{\delta w_{ik}} &= \frac{\delta E}{\delta out_k} \cdot \frac{\delta out_k}{\delta in_k} \cdot \frac{\delta in_k}{\delta w_{ik}} \\ &= \frac{\delta E}{\delta out_k} \cdot f'(in_k) \cdot out_i \\ &= \sum_{j \in l+1} (e_j \cdot w_{kj}) \cdot f'(in_k) \cdot out_i \\ &= e_j \cdot out_i \end{aligned}$$

Remember, that we were able to define a base case for the error e in the output layer, and the other values can be saved in forward pass. Now we have everything we need to calculate backpropagation for weights.

In conclusion, the gradient is calculated the following way for a weight on layer l:

$$\begin{aligned} d_{ik} &= \frac{\delta E}{\delta w_{ik}} \\ &= \frac{\delta E}{\delta out_k} \cdot \frac{\delta out_k}{\delta in_k} \cdot \frac{\delta in_k}{\delta w_{ik}} \\ &= e_k \cdot out_k \end{aligned}$$

Where e_k is the error of neuron k, and is defined separately for output layers (out is the output of the whole network):

$$e_k = \frac{\delta E}{\delta out} \cdot f_l'(in_k)$$

and for other layers:

$$e_k = \sum_{j \in l+1} (e_j \cdot w_{i \in j}) \cdot f_l'(in_k)$$

So by dividing the formula with the chain rule we were able to obtain terms for the equation, which can be implemented by saving specific values in forward pass.

3 Calculating the gradient for each bias

Yep, that wasn't it. But luckily this is more simple than with weights.

The biases are updated with the same formula as weights (see section 1). However, the gradient δ_{ij} turns out to be a bit different.

Like for the weights, we want to calculate the derivative of the loss function with respect to the biases.

$$d_{ij}^b = \frac{\delta E}{\delta b_{ij}}$$

Again, with the chain rule, we get

$$\frac{\delta E}{\delta b_{ij}} = \frac{\delta E}{\delta in_{ij}} \frac{\delta in_{ij}}{\delta b_{ij}}$$

We've actually defined in_{ij} before, so we get

$$\begin{aligned} \frac{\delta in_{ij}}{\delta b_{ij}} &= \frac{\delta \left(\sum_{i \in l-1}^{|l|} w_{ik} \cdot out_i + b_i \right)}{\delta b_{jk}}, \quad |l| \geq k,\ l \text{ is the current layer} \\ &= 1 \end{aligned}$$

This is, again, because when deriving, other variables than the bias are treated as fixed numbers, and their derivative is zero.

As for the first term, we can use the definition of the error:

$$\begin{aligned} \frac{\delta E}{\delta in_{ij}} &= \frac{\delta E}{\delta out_j} \cdot \frac{\delta out_j}{\delta in_{ij}} \\ &= e_j \end{aligned}$$

So we can just use the errors we've calculated to update the biases.

4 Sources

1. https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d

2. https://en.wikipedia.org/wiki/Backpropagation

3. https://www.cs.swarthmore.edu/ meeden/cs81/s10/BackPropDeriv.pdf