Tutorial 04: Fully Connected Neural Nets - Backpropagation

This tutorial aims to discuss the concept of Backpropagation for a NN in a detailed manner in order to ease the implementation of a fully trainable neural network in a given progamming language.

1 Gradient Descent for Neural Networks

The core concept in order to achieve actual learning in most of Machine Learning and especially in Deep Learning is to find the optimal model parameters for the specific data being observed.

The non-trivial question however is, how to find such parameters for a given model and an the available set of training data.

For a NN the tunable (i.e., trainable) parameters are the weights and biases, each of which can be tuned independently.

A common approach to train such a network is to use the **gradient descent algorithm** in order to minimise the loss function $L(\hat{\boldsymbol{Y}}(\boldsymbol{\Theta}, \boldsymbol{X}), \boldsymbol{Y})$ on a given training set $\boldsymbol{X}$ with labels $\boldsymbol{Y}$ with respect to the (flattened) network parameters

$$\boldsymbol{\Theta} = (\boldsymbol{W}^0_{11}, \ldots, \boldsymbol{W}^0_{N_0 1}, \ldots, \boldsymbol{W}^0_{N_0 N_1}, \boldsymbol{b}^0_1, \ldots, \boldsymbol{b}^0_{N_1}, \ldots, \boldsymbol{W}^n_{N_{n-1} N_n}, \ldots, \boldsymbol{b}^n_{N_n})^{\mathrm{T}}. \quad (1)$$

Each weight and each bias therefore represents an independent dimension, spanning a parameter space. From a given point in the parameter space, i.e., configuration of weights and biases, we use the property of the negative gradient – to point to the direction of steepest decline – to iteratively make small improvements to our loss function until we reach a sufficiently small value. A single update step for a parameter configuaration $\boldsymbol{\Theta}_i$ can be expressed as

$$\boldsymbol{\Theta}_{i+1} = \boldsymbol{\Theta}_i - \alpha \nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}, \boldsymbol{X})|_{\boldsymbol{\Theta}=\boldsymbol{\Theta}_i}, \quad (2)$$

with the so-called **learning rate** α scaling the magnitude of each step. Note that bigger gradients lead to bigger changes in $\boldsymbol{\Theta}$. The non-trivial question now is how to determine the gradient $\nabla_{\boldsymbol{\Theta}}$ for a neural network.

As a starting point we will try to calculate what the entry of the gradient, which corresponds to a weight of the last layer $\boldsymbol{W}^n_{ij} = w^n_{ij}$ for a single data point $\boldsymbol{x}_i \in \mathbb{R}^{1\times N_0}$

and according label $\boldsymbol{y}_i \in \mathbb{R}^{1\times N_n}$, i.e., we want to calculate

$$\left.\frac{\mathrm{d}L(\hat{\boldsymbol{y}}(\boldsymbol{\Theta},\boldsymbol{x}),\boldsymbol{y})}{\mathrm{d}w^n_{ij}}\right|_{\boldsymbol{\Theta}=\boldsymbol{\Theta}_i,\boldsymbol{x}=\boldsymbol{x}_i,\boldsymbol{y}=\boldsymbol{y}_i} \tag{3}$$

Note: The choice of a weight from the last layer will become more obvious later on.

As the loss function only depends implicitly on the weight w^n_{ij} via the variable $\hat{\boldsymbol{y}}$ we can use the **chain rule** from differential calculus to obtain a result. Due to the – in general – high dimensionality of the prediction, we need to consider the impact of the weight w^n_{ij} via each element of $\hat{\boldsymbol{y}}$ (and for the moment of $\boldsymbol{y}$) when applying the chain rule:

$$\frac{\mathrm{d}L(\hat{\boldsymbol{y}},\boldsymbol{y})}{\mathrm{d}w^n_{ij}}(\boldsymbol{\Theta},\boldsymbol{x}) = \sum_{k=1}^{N_n}\frac{\mathrm{d}L}{\mathrm{d}\hat{\boldsymbol{y}}_k}\frac{\mathrm{d}\hat{\boldsymbol{y}}_k}{\mathrm{d}w^n_{ij}}(\boldsymbol{\Theta},\boldsymbol{x}) + \sum_{k=1}^{N_n}\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{y}_k}\frac{\mathrm{d}\boldsymbol{y}_k}{\mathrm{d}w^n_{ij}} = \sum_{k=1}^{N_n}\frac{\mathrm{d}L}{\mathrm{d}\hat{\boldsymbol{y}}_k}\frac{\mathrm{d}\hat{\boldsymbol{y}}_k}{\mathrm{d}w^n_{ij}}(\boldsymbol{\Theta},\boldsymbol{x}). \tag{4}$$

The second step in (4) follows from the fact that the label does not depend on the weights.

As previously discussed, the prediction $\hat{\boldsymbol{y}}$ can be calculated from the input according to

$$\hat{\boldsymbol{y}}(\boldsymbol{\Theta},\boldsymbol{x}) = \mathrm{act}^n(\boldsymbol{i}^n(\boldsymbol{\Theta},\boldsymbol{x})) = \mathrm{act}^n(\boldsymbol{h}^{n-1}(\boldsymbol{\Theta}',\boldsymbol{x})\boldsymbol{W}^n + \boldsymbol{b}^n), \tag{5}$$

introducing a new notation $\boldsymbol{i}^j$ for the value of the neurons in layer j before applying the activation function. The variable $\boldsymbol{\Theta}'$ represents the network parameters without the weights and biases of the last layer $\boldsymbol{W}^n$ and $\boldsymbol{b}^n$. It turns out that we need to apply the chain rule again

$$\frac{\mathrm{d}\,\hat{\boldsymbol{y}}_k}{\mathrm{d}w^n_{ij}}(\boldsymbol{\Theta},\boldsymbol{x}) = \sum_{l=1}^{N_n}\frac{\mathrm{d}(\mathrm{act}^n(\boldsymbol{i}^n)_k)}{\mathrm{d}\boldsymbol{i}^n_l}\frac{\mathrm{d}\boldsymbol{i}^n_l}{\mathrm{d}\boldsymbol{w}^n_{ij}}(\boldsymbol{\Theta},\boldsymbol{x}). \tag{6}$$

For activation functions, which work element-wise on their input, the sum in (10) reduces to the element $l = k$. However, for activation functions like the softmax function, changes in $\boldsymbol{h}^n_i$ will influence the value of $\boldsymbol{h}^n_j$, as the layer as a whole is supposed to be normalised.

Nevertheless, we are now at the point where we can evaluate the derivative

$$\frac{\mathrm{d}\boldsymbol{i}^n_l}{\mathrm{d}w^n_{ij}}(\boldsymbol{\Theta},\boldsymbol{x}) = \frac{\mathrm{d}(\boldsymbol{h}^{n-1}(\boldsymbol{\Theta}',\boldsymbol{x})\boldsymbol{W}^n + \boldsymbol{b}^n)_l}{\mathrm{d}w^n_{ij}}. \tag{7}$$

Considering the matrix multiplication, we note that w^n_{ij} has a linear impact only on the jth component of h^n. Therefore, we obtain

$$\frac{\mathrm{d}\boldsymbol{i}^n_l}{\mathrm{d}w^n_{ij}}(\boldsymbol{\Theta},\boldsymbol{x}) = \begin{cases}\boldsymbol{h}^{n-1}_i(\boldsymbol{\Theta}',\boldsymbol{x}), & j = l,\\ 0, & \text{else.}\end{cases} \tag{8}$$

Further uses of the chain rule were not necessary, as $\boldsymbol{h}^{n_i-1}(\boldsymbol{\Theta}',\boldsymbol{x})$ does not depend on w^n_{ij} anymore. A derivative with respect to any component of the bias $\boldsymbol{b}^n_i$, follows the procedure, but from the last derivation, we obtain

$$\frac{\mathrm{d}\boldsymbol{i}_l^n}{\mathrm{d}\boldsymbol{b}_i^n}(\boldsymbol{\Theta},\boldsymbol{x}) = \begin{cases} 1, & j = l, \\ 0, & \text{else.} \end{cases} \tag{9}$$

However, in order to obtain derivatives with respect to weights of layers closer to the front the chain rule has to be applied over and over again.

Inserting our result for the derivative (7) into (10), we can get rid of the sum, as elements for $j \neq l$ are equal to zero.

$$\frac{\mathrm{d}\,\hat{\boldsymbol{y}}_k}{\mathrm{d}w_{ij}^n}(\boldsymbol{\Theta},\boldsymbol{x}) = \frac{\mathrm{d}(\mathrm{act}^n(\boldsymbol{i}^n)_k)}{\mathrm{d}\boldsymbol{i}_j^n}(\boldsymbol{\Theta},\boldsymbol{x})\boldsymbol{h}_i^{n-1}(\boldsymbol{\Theta}',\boldsymbol{x}). \tag{10}$$

In order to display the gradient for the weights in a "matrix shape", we will perform the following substitution, introducing the term **error** term $\boldsymbol{\delta}$

$$\boldsymbol{\delta}_j^{n,k}(\boldsymbol{\Theta},\boldsymbol{x}) = \frac{\mathrm{d}(\mathrm{act}^n(\boldsymbol{i}^n)_k)}{\mathrm{d}\boldsymbol{i}_j^n}(\boldsymbol{\Theta},\boldsymbol{x}). \tag{11}$$

This leads to

$$d\boldsymbol{W}_{ij}^n(\boldsymbol{\Theta},\boldsymbol{x}) = \sum_{k=1}^{N_n} \frac{\mathrm{d}L}{\mathrm{d}\hat{\boldsymbol{y}}_k}\frac{\mathrm{d}\,\hat{\boldsymbol{y}}_k}{\mathrm{d}w_{ij}^n}(\boldsymbol{\Theta},\boldsymbol{x}) = \sum_{k=1}^{N_n} \frac{\mathrm{d}L}{\mathrm{d}\hat{\boldsymbol{y}}_k}(\boldsymbol{\delta}^{n,k}\boldsymbol{h}^{n-1})_{ij}^{\mathrm{T}}(\boldsymbol{\Theta},\boldsymbol{x}), \tag{12}$$

or equivalently

$$d\boldsymbol{W}_{ij}^n(\boldsymbol{\Theta},\boldsymbol{x}) = (\boldsymbol{\delta}^n\boldsymbol{h}^{n-1})_{ij}^{\mathrm{T}}(\boldsymbol{\Theta},\boldsymbol{x}) \tag{13}$$

with

$$\boldsymbol{\delta}^n = \sum_{k=1}^{N_n} \frac{\mathrm{d}L}{\mathrm{d}\hat{\boldsymbol{y}}_k}\boldsymbol{\delta}^{n,k} \tag{14}$$

and $\mathrm{d}\boldsymbol{W}_{ij}^n$ being the component of the gradient according to the weight $w^n ij$, using a matrix notation. The product in (15) is between a column vector $\boldsymbol{\delta}$ and a row vector $\boldsymbol{h}$, i.e., a dyadic product, resulting in a matrix. Similarly, we obtain for the gradient components according to the bias

$$\mathrm{d}\boldsymbol{b}_i^n(\boldsymbol{\Theta},\boldsymbol{x}) = \sum_{k=1}^{N_n} \frac{\mathrm{d}L}{\mathrm{d}\hat{\boldsymbol{y}}_k}\frac{\mathrm{d}\,\hat{\boldsymbol{y}}_k}{\mathrm{d}b_i^n}(\boldsymbol{\Theta},\boldsymbol{x}) = \boldsymbol{\delta}_i^n(\boldsymbol{\Theta},\boldsymbol{x}). \tag{15}$$

Given the formulas (12) and (15), we are now able to compute the gradient components for all paramaters of the last layer. However, in order to compute derivatives of paramaters in previous layers, we still arrive at an equation according to (7). However we still have to apply the chain additional times to obtain a result for the derivatives. Using our new notation we can rewrite the change of the loss function with respect to the $w_i^n j$ (4) before replacing the derivative (8), using (10), (11) and (14)

$$\frac{\mathrm{d}L}{\mathrm{d}w_{ij}^n}(\boldsymbol{\Theta},\boldsymbol{x}) = \sum_{l=1}^{N_n} \boldsymbol{\delta}_l^n \frac{\mathrm{d}\boldsymbol{i}_l^n}{\mathrm{d}w_{ij}^{n-1}}(\boldsymbol{\Theta},\boldsymbol{x}). \tag{16}$$

For a weight of the previous layer $n-1$, we apply the chain rule a further two times to obtain

$$\frac{\mathrm{d}L}{\mathrm{d}w_{ij}^{n-1}}(\boldsymbol{\Theta}, \boldsymbol{x}) = \sum_{l=1}^{N_n} \boldsymbol{\delta}_l^n \sum_{k=1}^{N_{n-1}} \frac{\mathrm{d}\boldsymbol{i}_l^n}{\mathrm{d}\boldsymbol{h}_k^{n-1}} \sum_{m=1}^{N_{n-1}} \frac{\mathrm{d}\boldsymbol{h}_k^{n-1}}{\mathrm{d}\boldsymbol{i}_m^{n-1}} \frac{\mathrm{d}\boldsymbol{i}_m^{n-1}}{\mathrm{d}w_{ij}^{n-1}}(\boldsymbol{\Theta}, \boldsymbol{x}) \tag{17}$$

The term $\mathrm{d}\boldsymbol{i}_l^n / \mathrm{d}\boldsymbol{h}_k^{n-1}$ is the weight connecting the two neurons k and l from layers $n-1$ and n. We will further assume the activation function for inner neurons only depends on the input to the respective neuron itself (unlike the softmax function). We can therefore simplify the expression (17) to

$$\frac{\mathrm{d}L}{\mathrm{d}w_{ij}^{n-1}}(\boldsymbol{\Theta}, \boldsymbol{x}) = \sum_{l=1}^{N_n} \sum_{k=1}^{N_{n-1}} \boldsymbol{\delta}_l^n W_{kl}^n \frac{\mathrm{d}\boldsymbol{h}_k^{n-1}}{\mathrm{d}\boldsymbol{i}_k^{n-1}} \frac{\mathrm{d}\boldsymbol{i}_k^{n-1}}{\mathrm{d}w_{ij}^{n-1}}(\boldsymbol{\Theta}, \boldsymbol{x}). \tag{18}$$

We now introduce our error of the n-1th layer as

$$\boldsymbol{\delta}_k^{n-1}(\boldsymbol{\Theta}, \boldsymbol{x}) = (\boldsymbol{W}^n \boldsymbol{\delta}^n)_k \frac{\mathrm{dact}^{n-1}(\boldsymbol{i}^{n-1})_k}{\mathrm{d}\boldsymbol{i}_k^{n-1}}(\boldsymbol{\Theta}, \boldsymbol{x}) \tag{19}$$

in order to rewrite (17) to a very similar form as in (16)

$$\frac{\mathrm{d}L}{\mathrm{d}w_{ij}^{n-1}}(\boldsymbol{\Theta}, \boldsymbol{x}) = \sum_{k=1}^{N_{n-1}} \boldsymbol{\delta}_k^{n-1} \frac{\mathrm{d}\boldsymbol{i}_k^{n-1}}{\mathrm{d}w_{ij}^{n-1}}(\boldsymbol{\Theta}, \boldsymbol{x}), \tag{20}$$

and finally

$$d\boldsymbol{W}_{ij}^{n-1}(\boldsymbol{\Theta}, \boldsymbol{x}) = (\boldsymbol{\delta}^{n-1} \boldsymbol{h}^{n-2})_{ij}^{\mathrm{T}}(\boldsymbol{\Theta}, \boldsymbol{x}). \tag{21}$$

This same method can be applied over and over again to obtain derivatives with respect to any weight of a deeper layer.

To summarise: Now it becomes obvious, why the method is called backpropagation. We start by calculating the error $\boldsymbol{\delta}^n$, which only depends on the forward propagation result of the last layer $\boldsymbol{h}^n$. We then use the forward propagation result $\boldsymbol{h}^{n-1}$ to obtain the gradient component of any paramater in the last layer. Based on this we work our way from the back of the network to the front, layer by layer, calculating the error terms $\boldsymbol{\delta}^i$ from the previous layers $\boldsymbol{\delta}^{i+1}$ and our gradient components accordingly.

Note: An important feature of backpropagation is that we can can avoid calculating the same expressions multiple times for the errors, as well as for the neuron activations, by keeping already evaluated expressions in the memory.

In general, we want our gradient to optimise not just the loss function over a single training examples independently, as this can be a quite stochastic process with gradients for different training examples pointing into very different directinos, but rather the loss of the whole training set. We will further assume the total loss over the training set accounts for each training equally, i.e., we are considering the mean loss over all training examples

$$L(\boldsymbol{\Theta}, \boldsymbol{X}) = \frac{1}{M} \sum_{k=1}^{M} L(\boldsymbol{\Theta}, \boldsymbol{x}_k). \tag{22}$$

The derivatives of the total loss are then equal to the sum of the derivatives of the losses for each training example:

$$\nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}, \boldsymbol{X}) = \frac{1}{M} \sum_{k=1}^{M} \nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}, \boldsymbol{x}_k). \tag{23}$$

As for the previous description of forward propagation, we can again use a convenient matrix notation for an efficient implementation of the backpropagation for M training examples. The activations and inputs for any hidden layer for this purpose are of the dimension $\boldsymbol{I}^n, \boldsymbol{H}^n \in \mathbb{R}^{M \times N_n}$. Any error is of the dimension $\boldsymbol{\Delta}^n \in \mathbb{R}^{N_n \times M}$. We introduce the relation for the error of the nth level for the kth activation

$$\boldsymbol{\Delta}^{n,k}_{jm}(\boldsymbol{\Theta}, \boldsymbol{X}) = \frac{\mathrm{d}(\mathrm{act}^n(\boldsymbol{I}^n)_{mk})}{\mathrm{d}\boldsymbol{I}^n_{mj}}(\boldsymbol{\Theta}, \boldsymbol{X}), \tag{24}$$

which can be seen as the transpose of the matrix containing the element-wise derivative of the activation of that layer. $\boldsymbol{\Delta}^n$ can then be calculated according to the one-example case before in (14):

$$\boldsymbol{\Delta}^n(\boldsymbol{\Theta}, \boldsymbol{X}) = \sum_{k=1}^{N_n} \mathrm{d}\boldsymbol{L}^{k^{\mathrm{T}}} \circ \boldsymbol{\Delta}^{n,k}(\boldsymbol{\Theta}, \boldsymbol{X}), \tag{25}$$

with the broadcasted derivative of the Loss function for each training example $d\boldsymbol{L}^k_{mj} = \mathrm{d}L/\mathrm{d}\boldsymbol{y}_{mk} \forall j$ and the Hadamard (element-wise) product $\circ$.

Errors of previous layers can the be calculated as

$$\boldsymbol{\Delta}^{i-1}(\boldsymbol{\Theta}, \boldsymbol{X}) = \boldsymbol{W}^i \boldsymbol{\Delta}^i \circ (\mathrm{d}\mathbf{Act}^{i-1})^{\mathrm{T}}(\boldsymbol{\Theta}, \boldsymbol{X}), \tag{26}$$

with the element-wise derivation matrix of the activation function $\mathrm{d}\mathbf{Act}^{i-1}_{mj} = \boldsymbol{\Delta}^{i-1,j}_{jm}$. Considering the definition of the matrix product it becomes apparent that every column of $\boldsymbol{\Delta}^i$ is the column vector $\boldsymbol{\delta}^i$ of the according training example.

In order to calculate the components of gradient for any of the weights we use the matrix multiplication to sum up the contributions of each training example:

$$d\boldsymbol{W}^k_{ij}(\boldsymbol{\Theta}, \boldsymbol{X}) = \frac{1}{M} (\boldsymbol{\Delta}^k \boldsymbol{H}^{k-1})^{\mathrm{T}}_{ij}(\boldsymbol{\Theta}, \boldsymbol{X}). \tag{27}$$

For the bias however, we need to take the average over each row of the according Δ:

$$d\boldsymbol{b}^k_j(\boldsymbol{\Theta}, \boldsymbol{X}) = \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{\Delta}^k_{jm}(\boldsymbol{\Theta}, \boldsymbol{X}). \tag{28}$$

2 Backpropagation Example

We now consider a fully connected neural network for a classification problem with $n-1$ hidden layers, each with a sigmoid activation and an ouput layer with softmax activation and the cross-entropy as the loss function.

The first step is now to evaluate apply the previously obtained formulas to our problem, we are considering the cross entropy loss function for a multi-class classification problem

$$\text{crossentropy}(\boldsymbol{P}) = -log(\boldsymbol{P}_i), \qquad i : \text{correct label}, \tag{29}$$

which only depends on the prediction of the correct label. Using (25), we obtain

$$\boldsymbol{\Delta}^n_{jm}(\boldsymbol{\Theta}, \boldsymbol{X}) = -\frac{1}{\boldsymbol{H}^n_{mi}} \boldsymbol{\Delta}^{n,i}_{jm}(\boldsymbol{\Theta}, \boldsymbol{X}) \qquad i : \text{correct label}. \tag{30}$$

The second step is to evaluate the term $\Delta^{m,i}$, i.e., the derivative of the softmax function with respect to its input. Using (24) and some calculus, which will be part of the exercise sheet, we can show that

$$\boldsymbol{\Delta}^{n,i}_{jm}(\boldsymbol{\Theta}, \boldsymbol{X}) = \begin{cases} -\boldsymbol{H}^n_{mj}\boldsymbol{H}^n_{mi}(\boldsymbol{\Theta}, \boldsymbol{X}), & i \neq j \\ \boldsymbol{H}^n_{mi}(1 - \boldsymbol{H}^n_{mi})(\boldsymbol{\Theta}, \boldsymbol{X}), & i = j \end{cases} \tag{31}$$

Given that the activation function for any layer but the last one is assumed to be the sigmoid function, we can use our result from exercise sheet 02 to express $\mathrm{d}\mathbf{Act}^i$ in (26), the last unknown quantity in the general description above, as

$$\mathrm{d}\mathbf{Act}^i_{mj}(\boldsymbol{\Theta}, \boldsymbol{X}) = \boldsymbol{H}^i_{mj}(1 - \boldsymbol{H}^i_{mj})(\boldsymbol{\Theta}, \boldsymbol{X}). \tag{32}$$