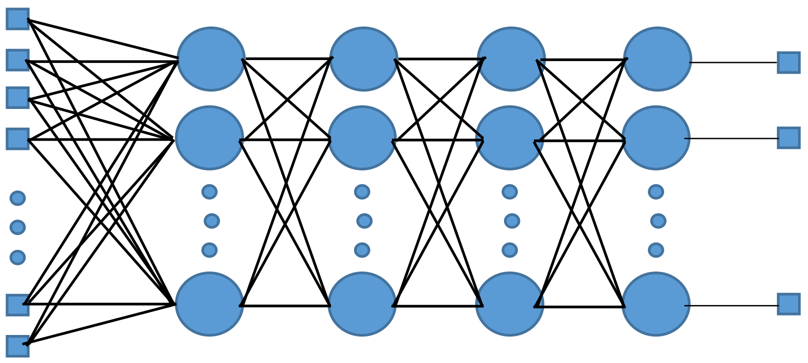
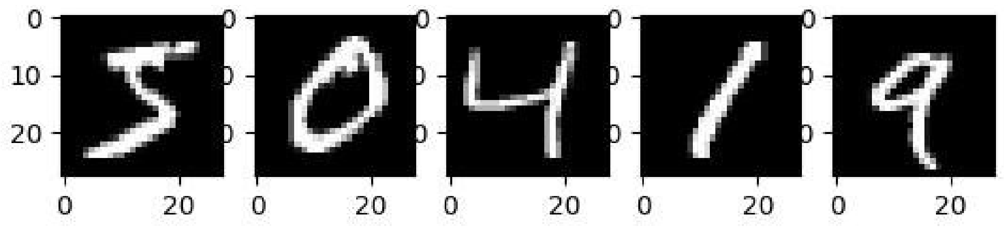
Multi-Laier Perceptron with NumPy



The project presents the realization of an MLP neural network with 4 hidden layers and aims to recognize numbers in images. The database is imported from tensorflow.keras.datasets.mnist.load_data() and contains 60000 training data and 10000 validation data



Project includes

- Data per-processing
- Feedforward
- Backpropagation
- Gradient Descent
- Mean Squared Error
- Accuracy measure
- Code implementation

Data per-processing

The structure of the data retrieved from tensorflow.keras.datasets.mnist.load_ data() has the following form

- The training data consists of 60000 images of size 28X28 bits with a cullet depth of 8 bits (gray scale), illustrating numbers and 60000 labels of size 1 digit (digits from 0 to 9).
- The validation data consists of 10000 images of 28X28 size with the same color feature, illustrating numbers and 10000 labels of 1 digit size.

The data pre-processing is performed as follows

The 28X28 image existing in the arrays 60000X28X28 and 10000X28X28, respectively, is transformed into a vector of 784 elements. Consequently, the two matrices will have the form 60000X784 and 10000X784 respectively.

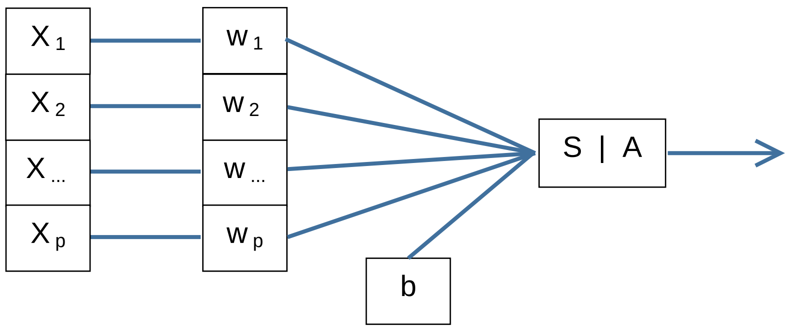
Data normalization is performed with the equation $(X_{max} - X_{min})/X_{max}$, resulting in values between 0 and 1. In this case we can divide the two matrices that have the image characteristics, to the value of 255 because the 8-bit gray scale has values between 0 and 255.

The labels corresponding to the images have the form 60000X1 and 10000X1 respectively. The values of these matrices will be transformed into categorical values. For example the value 0 = |1 0 0 0 0 0 0 0 0 0| , 1 = |0 1 0 0 0 0 0 0 0 0| , 2 = |0 0 1 0 0 0 0 0 0 0| ... Etc. This gives the matrices 60000X10 and 10000X10 respectively.

For a good management of computational resources and a better optimization of the model (neural network), it is recommended to divide the training data into smaller batches. For this project we have chosen the batch size of 100, thus the shape of the feature matrices as well as the labeling data, will be: 600X100X784X784 and 600X100X784X10 respectively

Feedforward

Feedforward, the neural network's process of passing data from the input layer to the output layer. This process is carried out by each perceptron of the network, calculating the weighted sum of inputs 1.1, and passed to the activation function 1.2. Figure 1 illustrates the shape of a perceptron, where X1..p - represents the input, w1..P - the weighting for each input, b - bias (balancing value), S - the weighted sum 1.1 and A- the activation function fig. 1



$$S = \sum (w \times X) + b \tag{1.1}$$

$$A = f(S) = \frac{1}{1+e^{-S}} \tag{1.2}$$

S - weighted sum, w - weighting, x - input values , f() - activation function , A - activation function result

In accordance with the above equations, the heat relations and their matrix form for a neural network with 4 hidden layers are given below. The data input, as shown in matrix 1.2, is a matrix 100 x 784 . The size of each line, is a 28X28 pixel image transformed into a vector of 784 . The number of lines is the number of features in the data packet that goes into the processing, 100 in this case.

$$\begin{bmatrix} X_{1,1} & \cdots & X_{1,784} \\ \vdots & \ddots & \vdots \\ X_{100,1} & \cdots & X_{100,784} \end{bmatrix} \begin{array}{l} \leftarrow input\ 1 \\ \leftarrow input\ m-1 \\ \leftarrow input\ m \end{array} \tag{1.2}$$

The initialization of the weight matrices is performed by generating a randomized matrix where the number of rows is the feature size and the number of columns is the number of perceptrons in that layer, i.e. the number of outputs. For example matrix 1.3, the matrix of weights on the first hidden layer has the number of rows equal to the size of the resulting vector from the 28X28pixel image i.e. 784. The number of columns, as can be seen in matrix 1.3, is the number of outputs to layer 2.

$$\begin{array}{cc} 784 \times 64 & \boldsymbol{nr.\ perceptrons\ in\ stratum\ L_1} \\ & \downarrow \\ \boldsymbol{nr.\ input} \rightarrow & \begin{bmatrix} w_{L1,1,1} & \cdots & w_{L1,1,64} \\ \vdots & \ddots & \vdots \\ w_{L1,784,1} & \cdots & w_{L1,784,64} \end{bmatrix} \end{array} \tag{1.3}$$

Initializing the bias vector, it is a matrix with a single row and has the number of elements equal to the number of perceptrons in the target layer. From Figure 1 we observe that for each percept there is a scalar that sums to the weighted sum.

Equation 1.4 relates the weighted sum of the input layer to which the bias vector is added. The shape of the S_{L1} matrix is 100x64, the shape resulting from multiplying the matrix X (100x784) by the matrix w_{L1} (784x64). For equations 1.6, 1.8 and 1.10, the approach is identical but the matrix X is replaced by the matrix resulting from the activation function in the previous layer, and the weighted matrix w is the matrix related to the respective layer.

$$S_{L1} = X \times w_{L1} + b_{L1} \tag{1.4}$$

$$\begin{bmatrix} S_{L1,1,1} & \cdots & S_{L1,1,64} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} = \begin{bmatrix} X_{1,1} & \cdots & X_{1,784} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} * \begin{bmatrix} w_{L1,1,1} & \cdots & w_{L1,1,64} \\ \vdots & \ddots & \vdots \\ w_{L1,784,1} & \cdots & w_{L1,784,64} \end{bmatrix} + \begin{bmatrix} b_{L1,1} \cdots b_{L1,25} \end{bmatrix}$$

The activation functions expressed in Equations 1.5, 1.7 and 1.9 are the weighted sum matrix for each layer, to which the sigmoid function (in this case, Equation 1.2) is applied for the element in the matrix.

$$A_{L1} = f(S_{L1}) \tag{1.5}$$

$$\begin{bmatrix} A_{L1,1,1} & \cdots & A_{L1,1,64} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} = f\left(\begin{bmatrix} S_{L1,1,1} & \cdots & S_{L1,1,64} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix}\right)$$

$$S_{L2} = A_{L1} \times w_{L2} + b_{L2} \tag{1.6}$$

$$\begin{bmatrix} S_{L2,1,1} & \cdots & S_{L2,1,32} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} = \begin{bmatrix} A_{L1,1,1} & \cdots & A_{L1,1,64} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} * \begin{bmatrix} w_{L2,1,1} & \cdots & w_{L2,1,32} \\ \vdots & \ddots & \vdots \\ w_{L2,64,1} & \cdots & w_{L2,64,32} \end{bmatrix} + \begin{bmatrix} b_{L2,1} \cdots b_{L2,32} \end{bmatrix}$$

$$A_{L2} = f(S_{L2}) \tag{1.7}$$

$$\begin{bmatrix} A_{L2,1,1} & \cdots & A_{L2,1,32} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} = f\left(\begin{bmatrix} S_{L2,1} & \cdots & S_{L2,1,32} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix}\right)$$

$$S_{L3} = A_{L2} \times w_{L3} + b_{L3} \tag{1.8}$$

$$\begin{bmatrix} S_{L3,1,1} & \cdots & S_{L3,1,16} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} = \begin{bmatrix} A_{L2,1,1} & \cdots & A_{L2,1,32} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} * \begin{bmatrix} w_{L3,1,1} & \cdots & w_{L3,1,16} \\ \vdots & \ddots & \vdots \\ w_{L3,32,1} & \cdots & w_{L3,32,16} \end{bmatrix} + \begin{bmatrix} b_{L3,1} \cdots b_{L3,16} \end{bmatrix}$$

$$A_{L3} = f(S_{L3}) \tag{1.9}$$

$$\begin{bmatrix} A_{L3,1,1} & \cdots & A_{L3,1,16} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} = f\left(\begin{bmatrix} S_{L3,1,1} & \cdots & S_{L3,1,16} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix}\right)$$

$$S_{L4} = A_{L3} \times w_{L4} + b_{L4} \tag{1.10}$$

$$\begin{bmatrix} S_{L4,1,1} & \cdots & S_{L4,1,10} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} A_{L3,1,1} & \cdots & A_{L3,1,16} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} * \begin{bmatrix} w_{L4,1,1} & \cdots & w_{L4,1,10} \\ \vdots & \ddots & \vdots \\ w_{L4,16,1} & \cdots & w_{L4,16,10} \end{bmatrix} + \begin{bmatrix} b_{L4,1} \cdots b_{L4,10} \end{bmatrix}$$

$$A_{L4} = f(S_{L4}) \tag{1.11}$$

$$\begin{bmatrix} A_{L4,1,1} & \cdots & A_{L4,1,10} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} = f\left(\begin{bmatrix} S_{L4,1,1} & \cdots & S_{L4,1,10} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix}\right)$$

S_{Ln} – weighted sum matrix of layer L_n ; w_{Ln} – weight matrix L_n ; X – matrix of input data ; A_{Ln} – the matrix resulting from the activation function (sigmoid in this case) for the L_n

Backpropagation

Backpropagation is the method of training the neural network in order to reduce the difference between the predicted and the actual outcome. As the name suggests it has the reverse direction to Feedforward, resulting in gradients with which the weights (w) are adjusted.

Determining gradients is done using the chain rule, which helps us find the derivative of a compound function. The case of a function $y = f(g(x))$, where g -is the function of x, and f is the function of g, the result is the derivative of y with respect to x is (2.1):

$$\frac{dy}{dx} = \frac{dy}{dg} \times \frac{dg}{dx} \tag{2.1}$$

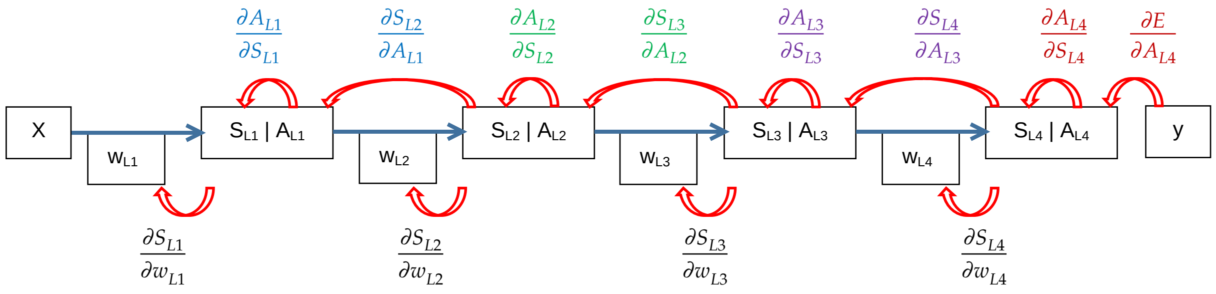
According to the principle stated above and considering equation 1.1, we can write the partial derivative equation of the correction gradient $\frac{\partial E}{\partial w}$, for a single-layer linear perceptron, as (2.2).

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \sum w \times X} \times \frac{\partial \sum w \times X}{\partial w} \tag{2.2}$$

Unde : E – loss, *w* – coefficients weights, $\sum w \times X$ – weighted sum, *X* – input values

Taking the example of equation 2.2 and the chain rule with several consecutive functions, we can determine the partial derivative equations for a model with 4 hidden layers. On the basis of Fig. 2, which represents the sketch of the model function, we derive the partial derivatives as well as their order.

fig. 2



$\frac{\partial A_{L1}}{\partial S_{L1}}$
$\frac{\partial S_{L2}}{\partial A_{L1}}$
$\frac{\partial A_{L2}}{\partial S_{L2}}$
$\frac{\partial S_{L3}}{\partial A_{L2}}$
$\frac{\partial A_{L3}}{\partial S_{L3}}$
$\frac{\partial S_{L4}}{\partial A_{L3}}$
$\frac{\partial A_{L4}}{\partial S_{L4}}$
$\frac{\partial E}{\partial A_{L4}}$
X
W_{L1}
S_{L1} | A_{L1}
W_{L2}
S_{L2} | A_{L2}
W_{L3}
S_{L3} | A_{L3}
W_{L4}
S_{L4} | A_{L4}
y
$\frac{\partial S_{L1}}{\partial w_{L1}}$
$\frac{\partial S_{L2}}{\partial w_{L2}}$
$\frac{\partial S_{L3}}{\partial w_{L3}}$
$\frac{\partial S_{L4}}{\partial w_{L4}}$

Backpropagation si Feedforward

This gives equations 2.3 which determine the gradients of the weights on the 4 hidden layers.

$$\frac{\partial E}{\partial w_{L4}} = \frac{\partial E}{\partial A_{L4}} \times \frac{\partial A_{L4}}{\partial S_{L4}} \times \frac{\partial S_{L4}}{\partial w_{L4}} \tag{2.3}$$

$$\frac{\partial E}{\partial w_{L3}} = \frac{\partial E}{\partial A_{L4}} \times \frac{\partial A_{L4}}{\partial S_{L4}} \times \frac{\partial S_{L4}}{\partial A_{L3}} \times \frac{\partial A_{L3}}{\partial S_{L3}} \times \frac{\partial S_{L3}}{\partial w_{L3}}$$

$$\frac{\partial E}{\partial w_{L2}} = \frac{\partial E}{\partial A_{L4}} \times \frac{\partial A_{L4}}{\partial S_{L4}} \times \frac{\partial S_{L4}}{\partial A_{L3}} \times \frac{\partial A_{L3}}{\partial S_{L3}} \times \frac{\partial S_{L3}}{\partial A_{L2}} \times \frac{\partial A_{L2}}{\partial S_{L2}} \times \frac{\partial S_{L2}}{\partial w_{L2}}$$

$$\frac{\partial E}{\partial w_{L1}} = \frac{\partial E}{\partial A_{L4}} \times \frac{\partial A_{L4}}{\partial S_{L4}} \times \frac{\partial S_{L4}}{\partial A_{L3}} \times \frac{\partial A_{L3}}{\partial S_{L3}} \times \frac{\partial S_{L3}}{\partial A_{L2}} \times \frac{\partial A_{L2}}{\partial S_{L2}} \times \frac{\partial S_{L2}}{\partial A_{L1}} \times \frac{\partial A_{L1}}{\partial S_{L1}} \times \frac{\partial S_{L1}}{\partial w_{L1}}$$

E –loss; A $_{Ln}$ – activation function for layer n; S $_{Ln}$ – weighted sum on layer n; w $_{Ln}$ – weight of layer n

Similar to the determination of the gradients of the weights we determine the gradients for the bias, but replacing w $_L$ by b $_L$. This is a vector that has the size of the number of perceptrons in the layer.

$$\frac{\partial E}{\partial b_{L4}} = \frac{\partial E}{\partial A_{L4}} \times \frac{\partial A_{L4}}{\partial S_{L4}} \times \frac{\partial S_{L4}}{\partial b_{L4}} \tag{2.4}$$

$$\frac{\partial E}{\partial b_{L3}} = \frac{\partial E}{\partial A_{L4}} \times \frac{\partial A_{L4}}{\partial S_{L4}} \times \frac{\partial S_{L4}}{\partial A_{L3}} \times \frac{\partial A_{L3}}{\partial S_{L3}} \times \frac{\partial S_{L3}}{\partial b_{L3}}$$

$$\frac{\partial E}{\partial b_{L2}} = \frac{\partial E}{\partial A_{L4}} \times \frac{\partial A_{L4}}{\partial S_{L4}} \times \frac{\partial S_{L4}}{\partial A_{L3}} \times \frac{\partial A_{L3}}{\partial S_{L3}} \times \frac{\partial S_{L3}}{\partial A_{L2}} \times \frac{\partial A_{L2}}{\partial S_{L2}} \times \frac{\partial S_{L2}}{\partial b_{L2}}$$

$$\frac{\partial E}{\partial b_{L1}} = \frac{\partial E}{\partial A_{L4}} \times \frac{\partial A_{L4}}{\partial S_{L4}} \times \frac{\partial S_{L4}}{\partial A_{L3}} \times \frac{\partial A_{L3}}{\partial S_{L3}} \times \frac{\partial S_{L3}}{\partial A_{L2}} \times \frac{\partial A_{L2}}{\partial S_{L2}} \times \frac{\partial S_{L2}}{\partial A_{L1}} \times \frac{\partial A_{L1}}{\partial S_{L1}} \times \frac{\partial S_{L1}}{\partial b_{L1}}$$

b $_{Ln}$ – bias for layer n

Taking into account that the neural network has a repetitive form with respect to the positioning of the perceptrons composing it, we can generalize the warm relations for the partial derivatives as follows:

- For the output layer, $\frac{\partial E}{\partial A_L}$ represents the change in loss (E) as a function of the change in the activation function result (A_L). In this situation we have chosen the simplest loss function, i.e. the difference between the model output (A_L) and the actual output (Y) used during training

$$\frac{\partial E}{\partial A_L} = A_L - Y \qquad (2.5)$$

- The influence of the result of the activation function ($A_L = f(S_L)$) on the weighted sum (S_L), is equal to the derivative of the activation function $f'(S_L)$. In this case we have chosen the sigmoid function, so the derivative of this function is $f'(S) = f(S) \times (1 - f(S))$ (taken from the literature)

$$\frac{\partial A_L}{\partial S_L} = \frac{\partial f(S_L)}{\partial S_L} = f'(S_L) = f(S_L) \times (1 - f(S_L)) \qquad (2.6)$$

- - Equation 2.7 represents the change in the weighted sum of the upper layer S_{L+1} as a function of the inputs assigned to it from the output of the lower layers A_L . This is equal to the value of the weights in the upper layer w_{L+1}. We consider $b_{L+1} = 0$, since we are interested in the modification of S_{L+1} as a function of A_L .

$$\frac{\partial S_{L+1}}{\partial A_L} = \frac{\partial (A_L \times w_{L+1} + b_{L+1})}{\partial A_L} = \frac{\partial (A_L \times w_{L+1} + 0)}{\partial A_L} = w_{L+1} \qquad (2.7)$$

- The derivative of the weighted sum S_L with respect to the weight w_L, is equal to the result of the activation function in the previous layer A_{L-1} , except for the first layer because A_{L-1} becomes the input to the model, i.e. X. We consider $b_L = 0$, since we are interested in the evolution of S_L as a function of w_L.

$$\frac{\partial S_L}{\partial w_L} = \frac{\partial (A_{L-1} \times w_L + b_L)}{\partial w_L} = \frac{\partial (A_{L-1} \times w_L + 0)}{\partial w_L} = A_{L-1} \qquad (2.8)$$

- The weighted sum S_L with respect to b_L, is equal to 1 because the inputs A_{L-1} and the weights w are 0, because we are interested in the evolution of S_L as a function of b_L.

$$\frac{\partial S_L}{\partial b_L} = \frac{\partial (A_{L-1} \times w_L + b_L)}{\partial b_L} = \frac{\partial (0 \times 0 + b_L)}{\partial b_L} = 1 \qquad (2.9)$$

Considering equations 2.5, 2.6, 2.7, 2.8 and 2.9, equations 2.3 and 2.4 can be written as 2.10 below.

Index ^{T} is the transposed matrix for realizing the junctions between layers. If for forward propagation, the matrix w_L realizes the junction between the shape of the input matrices in the L-layer and the output shape, for back propagation the reverse effect must be realized.

This is done by rotating the matrix from the bottom left corner to the top right corner.

$$\frac{\partial E}{\partial w_{L4}} = (A_{L4} - Y) \times f'(S_{L4}) \times A_{L3}^{T} \tag{2.10}$$

$$\frac{\partial E}{\partial w_{L3}} = (A_{L4} - Y) \times f'(S_{L4}) \times w_{L4}^{T} \times f'(S_{L3}) \times A_{L2}^{T}$$

$$\frac{\partial E}{\partial w_{L2}} = (A_{L4} - Y) \times f'(S_{L4}) \times w_{L4}^{T} \times f'(S_{L3}) \times w_{L3}^{T} \times f'(S_{L2}) \times A_{L1}^{T}$$

$$\frac{\partial E}{\partial w_{L1}} = (A_{L4} - Y) \times f'(S_{L4}) \times w_{L4}^{T} \times f'(S_{L3}) \times w_{L3}^{T} \times f'(S_{L2}) \times w_{L2}^{T} \times f'(S_{L2}) \times X^{T}$$

$$\frac{\partial E}{\partial b_{L4}} = (A_{L4} - Y) \times f'(S_{L4}) \times 1$$

$$\frac{\partial E}{\partial b_{L3}} = (A_{L4} - Y) \times f'(S_{L4}) \times w_{L4}^{T} \times f'(S_{L3}) \times 1$$

$$\frac{\partial E}{\partial b_{L2}} = (A_{L4} - Y) \times f'(S_{L4}) \times w_{L4}^{T} \times f'(S_{L3}) \times w_{L3}^{T} \times f'(S_{L2}) \times 1$$

$$\frac{\partial E}{\partial b_{L1}} = (A_{L4} - Y) \times f'(S_{L4}) \times w_{L4}^{T} \times f'(S_{L3}) \times w_{L3}^{T} \times f'(S_{L2}) \times w_{L1}^{T} \times f'(S_{L1}) \times 1$$

$$A_{L}^{T} \ - matrix\ A_{L} transposed, \quad w_{L}^{T} \ - matrix\ w_{L}\ transposed$$

To simplify the implementation process, we consider

– for the last layer $\delta_L = \frac{\partial E}{\partial A_L} \times \frac{\partial A_L}{\partial S_L} = (A_L - Y) \times f'(S_L)$

– for the rest of the layers $\delta_L = \delta_{L+1} \times \frac{\partial S_{L+1}}{\partial A_L} \times \frac{\partial A_L}{\partial S_L} = w_L^T \times f'(S_L)$

And we can write equations 2.10 excluding the last term as 2.11

$$\delta_4 = (A_{L4} - Y) \times f'(S_{L4}) \tag{2.11}$$

$$\begin{bmatrix} \delta_{4,1,1} & \cdots & \delta_{4,1,10} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} = \left(\begin{bmatrix} A_{L4,1,1} & \cdots & A_{L4,1,10} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} - \begin{bmatrix} Y_{1,1} & \cdots & Y_{1,10} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} \right) \times f' \left(\begin{bmatrix} S_{L4,1,1} & \cdots & S_{L4,1,10} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} \right)$$

$$\delta_3 = \delta_4 \times w_{L4}^{T} \times f'(S_{L3})$$

$$\begin{bmatrix} \delta_{3,1,1} & \cdots & \delta_{3,1,16} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} = \begin{bmatrix} \delta_{4,1,1} & \cdots & \delta_{4,1,10} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} * \begin{bmatrix} \boldsymbol{w_{L4,1,1}} & \cdots & \boldsymbol{w_{L4,16,1}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{w_{L4,1,10}} & \cdots & \boldsymbol{w_{L4,16,10}} \end{bmatrix} \times f' \left(\begin{bmatrix} S_{L3,1,1} & \cdots & S_{L3,1,16} \\ \vdots & \ddots & \vdots \\ \ldots & \ldots & \ldots \end{bmatrix} \right)$$

$$\delta_2 = \delta_3 \times w_{L3}^T \times f'(S_{L2})$$

$$\begin{bmatrix} \delta_{2,1,1} & \cdots & \delta_{2,1,32} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} \delta_{3,1,1} & \cdots & \delta_{3,16} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} * \begin{bmatrix} \boldsymbol{w_{L3,1,1}} & \cdots & \boldsymbol{w_{L3,32,1}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{w_{L3,1,16}} & \cdots & \boldsymbol{w_{L3,32,16}} \end{bmatrix} \times f'\left(\begin{bmatrix} S_{L2,1,1} & \cdots & S_{L2,1,32} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix}\right)$$

$$\delta_1 = \delta_2 \times w_{L2}^T \times f'(S_{L1})$$

$$\begin{bmatrix} \delta_{1,1,1} & \cdots & \delta_{1,1,64} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} \delta_{2,1,1} & \cdots & \delta_{2,1,32} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} * \begin{bmatrix} \boldsymbol{w_{L2,1,1}} & \cdots & \boldsymbol{w_{L2,784,1}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{w_{L2,1,64}} & \cdots & \boldsymbol{w_{L2,784,64}} \end{bmatrix} \times f'\left(\begin{bmatrix} S_{L1,1,1} & \cdots & S_{L1,1,64} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix}\right)$$

Considering equations 2.10 and 2.11 results in the following form of the gradient determinant (2.15 and 2.16). The form of the gradient matrix is identical to the form of the weighted matrix

$$\nabla E_w \leftarrow \begin{cases} \dfrac{\partial E}{\partial w_{L4}} = A_{L3}^T \times \delta_4 \\ \dfrac{\partial E}{\partial w_{L3}} = A_{L2}^T \times \delta_3 \\ \dfrac{\partial E}{\partial w_{L2}} = A_{L1}^T \times \delta_2 \\ \dfrac{\partial E}{\partial w_{L1}} = X^T \times \delta_1 \end{cases} \tag{2.12}$$

$$\frac{\partial E}{\partial w_{L4}} = \begin{bmatrix} A_{L3,1,1} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ A_{L3,1,16} & \cdots & \cdots \end{bmatrix} * \begin{bmatrix} \delta_{4,1,1} & \cdots & \delta_{4,1,10} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} \qquad \frac{\partial E}{\partial w_{L3}} = \begin{bmatrix} A_{L2,1,1} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ A_{L2,1,32} & \cdots & \cdots \end{bmatrix} * \begin{bmatrix} \delta_{3,1,1} & \cdots & \delta_{3,1,16} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$$\frac{\partial E}{\partial w_{L2}} = \begin{bmatrix} A_{L1,1,1} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ A_{L1,1,64} & \cdots & \cdots \end{bmatrix} * \begin{bmatrix} [\delta_{2,1,1} & \cdots & \delta_{2,1,32}] \\ \vdots & \ddots & \vdots \\ [\cdots & \cdots & \cdots] \end{bmatrix} \qquad \frac{\partial E}{\partial w_{L1}} = \begin{bmatrix} X_{1,1} & \cdots & X_{100,1} \\ \vdots & \ddots & \vdots \\ X_{1,784} & \cdots & X_{100,784} \end{bmatrix} * \begin{bmatrix} [\ \delta_{1,1,1} & \cdots & \delta_{1,1,64}] \\ \vdots & \ddots & \vdots \\ [\delta_{1,100,1} & \cdots & \delta_{1,100,64}] \end{bmatrix}$$

As it follows from equations 2.10, the gradients of b $_L$ are equal to δ $_L$ and the composition of the gradients resulting from each input feature (100 in this case) is realized by summing over the vertical

$$\nabla E_b \leftarrow \begin{cases} \dfrac{\partial E}{\partial b_{L4}} = \delta_4 \\ \dfrac{\partial E}{\partial b_{L3}} = \delta_3 \\ \dfrac{\partial E}{\partial b_{L2}} = \delta_2 \\ \dfrac{\partial E}{\partial b_{L1}} = \delta_1 \end{cases} \tag{2.13}$$

$$\frac{\partial E}{\partial b_{b4}} = \left[\Sigma\begin{pmatrix} [\delta_{4,1,1} \\ \vdots \\ \cdots \end{pmatrix} \ \Sigma\begin{pmatrix} \cdots \\ \ddots \\ \cdots \end{pmatrix} \ \Sigma\begin{pmatrix} \delta_{4,1,10} \\ \vdots \\ \cdots \end{pmatrix} \right] \qquad \frac{\partial E}{\partial b_{3}} = \left[\Sigma\begin{pmatrix} [\delta_{3,1,1} \\ \vdots \\ \cdots \end{pmatrix} \ \Sigma\begin{pmatrix} \cdots \\ \ddots \\ \cdots \end{pmatrix} \ \Sigma\begin{pmatrix} \delta_{3,1,16} \\ \vdots \\ \cdots \end{pmatrix} \ \right]$$

$$\frac{\partial E}{\partial b_{2}} = \left[\Sigma\begin{pmatrix} \delta_{2,1,1} \\ \vdots \\ \cdots \end{pmatrix} \ \Sigma\begin{pmatrix} \cdots \\ \ddots \\ \cdots \end{pmatrix} \ \Sigma\begin{pmatrix} \delta_{2,1,32} \\ \vdots \\ \cdots \end{pmatrix} \ \right] \qquad \frac{\partial E}{\partial b_{1}} = \left[\Sigma\begin{pmatrix} [\delta_{1,1,1} \\ \vdots \\ \cdots \end{pmatrix} \ \Sigma\begin{pmatrix} \cdots \\ \ddots \\ \cdots \end{pmatrix} \ \Sigma\begin{pmatrix} \delta_{1,1,64} \\ \vdots \\ \cdots \end{pmatrix} \ \right]$$

Gradient Descent

Gradient descent is a method of optimizing a model by finding a local minimum of a differential function. In machine learning, it has the role of correcting the weights used in the neural network. The generalized form as well as the way of working is expressed in equation 3.1

$$w_{new} = w_{old} - \alpha \times \nabla E_w \tag{3.1}$$

$$b_{new} = b_{old} - \alpha \times \nabla E_b$$

α – learning rate

Equation 2.3 presents the weight and bias optimization approach for each layer

$$w_{L4} = w_{L4} - \alpha \times \frac{\partial E}{\partial w_{L4}}\ ; \quad w_{L3} = w_{L3} - \alpha \times \frac{\partial E}{\partial w_{L3}}\ ; \quad w_{L2} = w_{L2} - \alpha \times \frac{\partial E}{\partial w_{L2}}\ ; \quad w_{L1} = w_{L1} - \alpha \times \frac{\partial E}{\partial w_{L1}} \tag{3.2}$$

$$b_{L4} = b_{L4} - \alpha \times \frac{\partial E}{\partial b_{L4}}\ ; \quad b_{L3} = b_{L3} - \alpha \times \frac{\partial E}{\partial b_{L3}}\ ; \quad b_{L2} = b_{L2} - \alpha \times \frac{\partial E}{\partial b_{L2}}\ ; \quad b_{L1} = b_{L1} - \alpha \times \frac{\partial E}{\partial b_{L1}}$$

Mean Squared Error

The mean squared error is a method of expressing the errors that a model may have. It is realized according to equation 4.1, being the average of the squares of the difference between the predicted and the actual result.

$$MSE = \frac{1}{n}\sum (Y_n - \widehat{Y_n})^2 \tag{4.1}$$

Accuracy metric

For classification tasks, this method provides a quick information of the model performance in terms of the correctness of the delivered results. The accuracy expresses the ratio of the number of correct results to the total number of results

$$\text{accuracy} = \frac{1}{n}\sum (argmax(Y_n) = argmax(\widehat{Y_n})) \tag{5.1}$$

Bibliografie

https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

https://machinelearning.tobiashill.se/2018/12/04/part-2-gradient-descent-and-backpropagation/

https://www.3blue1brown.com/lessons/backpropagation-calculus

http://neuralnetworksanddeeplearning.com/chap2.html

https://hmkcode.com/ai/backpropagation-step-by-step/

https://sefiks.com/2017/01/21/the-math-behind-backpropagation/#google_vignette

https://medium.com/@samuelsena/pengenalan-deep-learning-part-3-backpropagation-algorithm-720be9a5fbb8

https://pabloinsente.github.io/the-multilayer-perceptron

https://towardsdatascience.com/understanding-backpropagation-abcc509ca9d0/

https://www.geeksforgeeks.org/machine-learning/backpropagation-in-neural-network/