



Deep Learning

Summer Semester 2024

Monday, May 6, 2024

Prof. Dr.-Ing. Christian Bergler | OTH Amberg-Weiden

Topics From Last Time: Training Parametric Models – Softmax Regression

- Approaches to multi-class classification
- Properties of the softmax function
- Model approach and error functional for softmax regression
- Softmax regression as a linear neural network

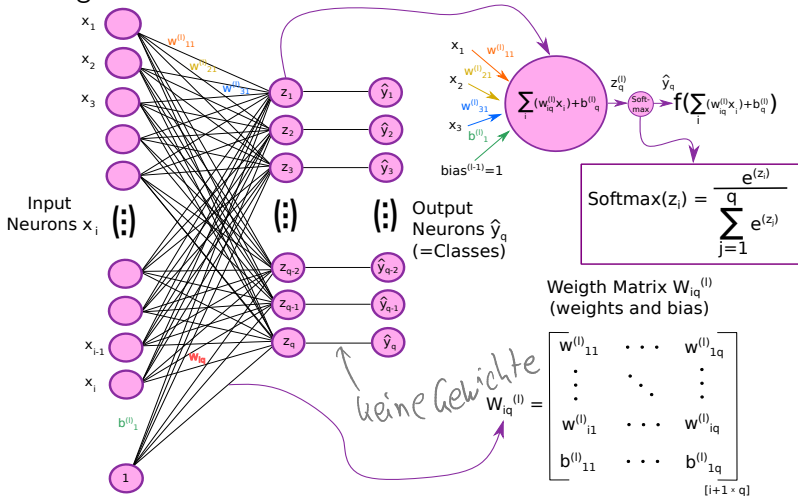
Topics of Today: Multi-Layer Perceptron

- Multilayer architectures
- Activation functions
- Universal approximation theorem
- Backpropagation

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Multi-Layer Perceptron

Recap: Softmax Regression as Neural Network



- Affine-linear transformation $z = W^T x + b \rightarrow \hat{y} = \text{softmax}(z)$

Limits Softmax Regression

- In softmax regression, the relationship between input x and output z is affine-linear
- Many real relationships cannot be mapped with affine-linear functions

Idea

- Add further (hidden) layers to the network
- Without non-linear functions a composition of affine-linear transformations, due to multiple layer, results still in just an affine-linear mapping
- Thereby: Increase the parametric complexity (layer dimensionality)

Questions

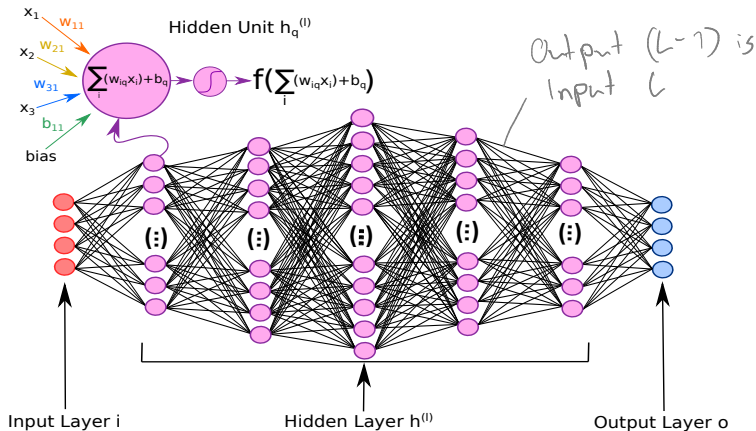
HW2, Data2, Task2

- What shape does the model function then have?
- Can more complex functions be modelled as a result?

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Multi-Layer Perceptron

Multi-Layer Perceptron or (Deep) Feedforward Neural Network



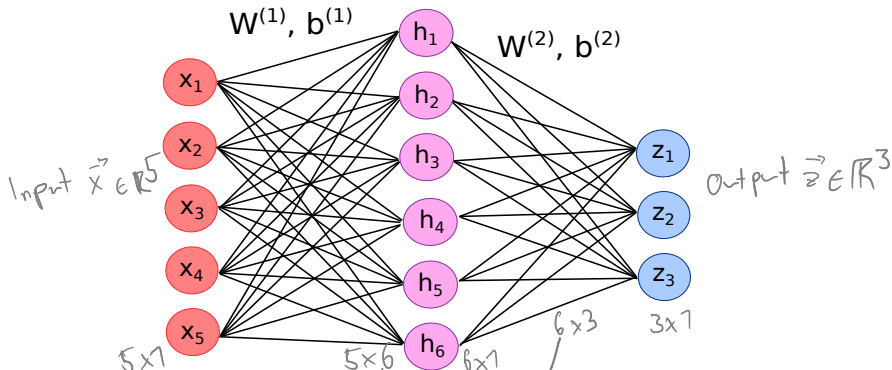
- Forward Pass: $h^{(l)} = W^{T(l)}x + b^{(l)}$, with layer index $l = 1, \dots, L$
- Output Layer $z = W^{T(L)}h + b^{(L)}$

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Multi-Layer Perceptron

Multi-Layer Perceptron or (Deep) Feedforward Neural Network

- Which dimensionality does $W^{(1)}$, $b^{(1)}$, $W^{(2)}$, and $b^{(2)}$ have?
- Provide the forward propagation path of the given model function (network)?
- Interpret the network result in terms of linearity?



Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Anzahl Parameter:

$$\begin{aligned} & \cdot w^{(1)} \in \mathbb{R}^{5 \times 6}, b^{(1)} \in \mathbb{R}^6, w^{(2)} \in \mathbb{R}^{6 \times 3}, b^{(2)} \in \mathbb{R}^3 \\ & \quad 30 + 6 + 18 + 3 = 57 \end{aligned}$$

Observations

- By simply connecting several layers in series, the space of the approximable functions remains linear
- The space is only enlarged by adding non-linear (activation) functions
- This is done by applying non-linear activation functions
- Typically, the same activation function is used for all nodes within a layer
- An application-specific activation function is often used at the output layer

Questions of understanding

- What does the choice of activation function at the output layer depend on?
- What properties would be desirable for activation functions?

leicht ableitbar fast überall differenzierbar

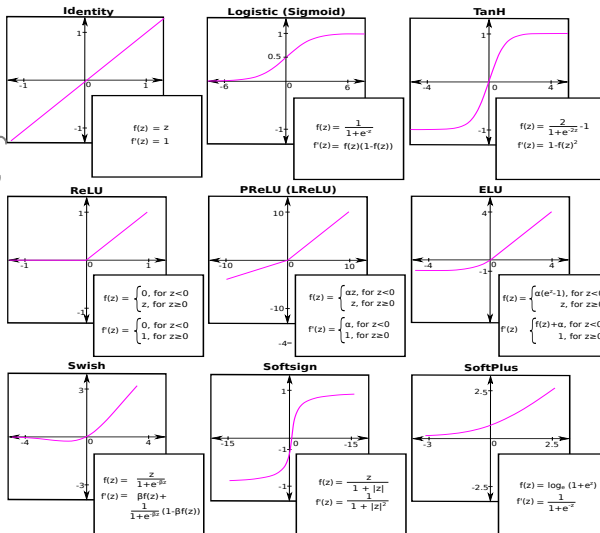
Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Multi-Layer Perceptron

Frequently Used Non-Linear Activation Function

Constanten in Activation
function immer bloß,
im Gradienten 0
→ nicht gelernt

in forward pass
aber ok



Multi-Layer Perceptron

Recap: Biological Neuron vs. Mathematical Neuron

- Mathematical model of a biological neuron to represent binary functions
- Perceptron (Rosenblatt 1958): Model with learnable weights

Biological Neuron

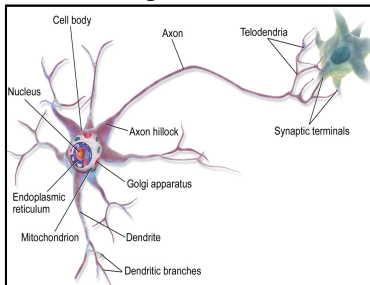
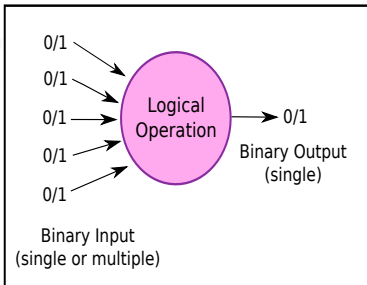
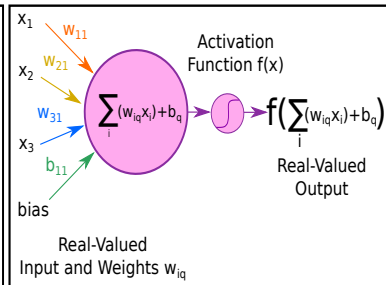


Image by Bruce Blaus (Creative Commons 3.0 - CC BY 3.0), no additional changes, taken from: https://commons.wikimedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png

Artificial Neuron



(Rosenblatt's) Perceptron

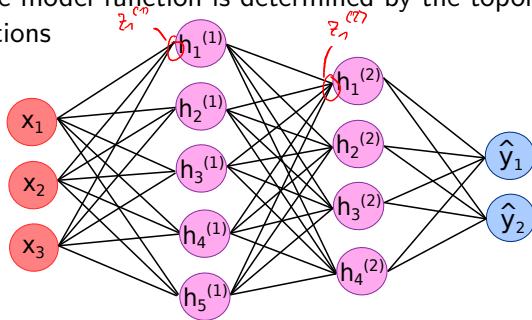


Source: Christian Bergler, Dissertation "Deep Learning Applied To Animal Linguistics", 2023
Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Einführung

Multi-Layer Perceptron

Network Architecture

- The structure of the model function is determined by the topology of the network and the activation functions



- Forward propagation of an input sample $x \in \mathbb{R}^3$ w.r.t. the given architecture:

vector \rightarrow $z^{(1)} = W^{(1)T}x + b^{(1)}$, followed by the network activation: $h^{(1)} = f^{(1)}(z^{(1)})$

- $z^{(2)} = W^{(2)T}h^{(1)} + b^{(2)}$, followed by the network activation: $h^{(2)} = f^{(2)}(z^{(2)})$
- $z^{(3)} = W^{(3)T}h^{(2)} + b^{(3)}$, followed by the network activation: $\hat{y} = f^{(3)}(z^{(3)})$

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Model Function & Forward Propagation

Let L be the number of layers of the mesh (without the input layer). Then the model function for an input $\mathbf{x} \in \mathbb{R}^p$ is calculated by

$$\begin{aligned} \mathbf{z}^{(1)} &:= (\mathbf{W}^{(1)})^T \mathbf{x} + \mathbf{b}^{(1)}, & \mathbf{h}^{(1)} &= f^{(1)}(\mathbf{z}^{(1)}), \\ \vdots & & \vdots & \\ \mathbf{z}^{(l)} &:= (\mathbf{W}^{(l)})^T \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, & \mathbf{h}^{(l)} &= f^{(l)}(\mathbf{z}^{(l)}), \quad \text{for } l = 2, \dots, L-1 \\ \vdots & & \vdots & \\ \mathbf{z}^{(L)} &:= (\mathbf{W}^{(L)})^T \mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}, & \hat{\mathbf{y}} &= f^{(L)}(\mathbf{z}^{(L)}) \end{aligned}$$

Handwritten note: "output layer davor" with an arrow pointing to the $\mathbf{h}^{(l-1)}$ term in the equation for $\mathbf{z}^{(l)}$.

- Where $f^{(1)}, \dots, f^{(L)}$ are the activation functions of the individual layers
- Loss/Error/Objective Function: $L(\theta) = L(\hat{\mathbf{y}})$

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Question: How do you choose the number of layers and the number of nodes per layer?

- The number of layers and the number of nodes per layer are hyperparameters
- Additional layers and nodes increase the number of parameters (dimensionality) and can lead to overfitting
- Neural networks should therefore always be regularized (stand-alone lecture about “regularization”) *ReLU, Gradient Clipping, Early stopping, ...*
- A middle way must be found between too many and too few parameters θ
- In most cases, only experience or trial and error will help

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

General: The choice of the architecture and activation functions depends on the task and data situation!

MLP for Classification:

- Output layer with n nodes, where n is the number of classes
- Softmax activation at the output layer
- Frequently: Cross entropy (CE) loss as error function

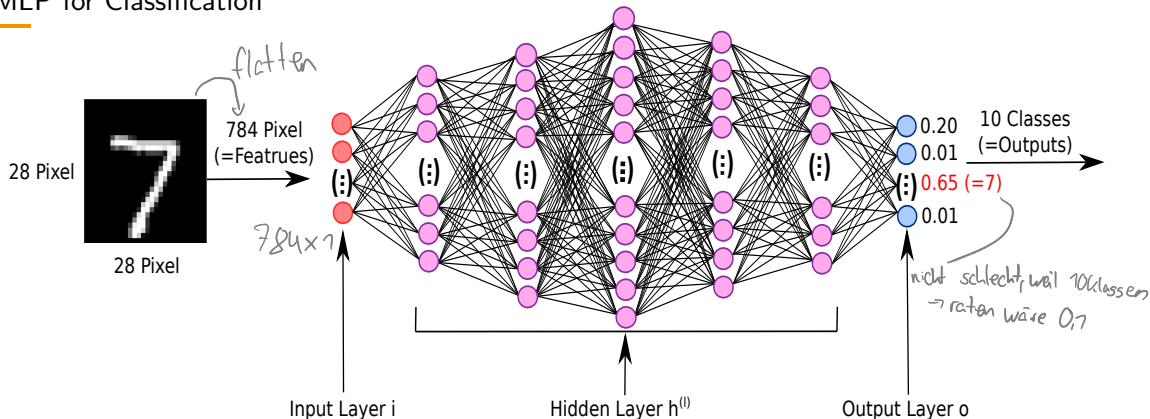
MLP for Regression

- Output layer with one node
- Linear activation at the output layer
- Frequently: Mean Squared Error (MSE) as loss function

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Multi-Layer Perceptron

MLP for Classification



- Transforms a given input $x \in \mathbb{R}^{784}$ via forward propagation (forward path) into a final output feature representation $\hat{y} \in \mathbb{R}^{10} \rightarrow 10 \text{ Classes} = 10 \text{ Numbers (0-9)}$

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

The choice of loss function depends on the task and data situation. The following loss functions are frequently used in deep learning

Regression

- Mean Squared Error
- Mean Absolute Error
- Huber

Classification

- (Binary) Cross Entropy
- Hinge
- Kullback-Leibler Divergence

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Definition

Using a neural network with just a single hidden layer (together with a sigmoid activation), any continuous function can be approximated with arbitrary accuracy

Notes:

Wieviele Nodes? Ja Xor-Problem

- The above description is not a rigorous formulation of the theorem in the mathematical sense. To understand it, we lack the basic mathematical knowledge (e.g. convergence in function spaces).
- The theorem is not necessarily of practical relevance, as it does not specify how many nodes the hidden layer must have.
- However, after the first „AI winter“ it has promoted the „confidence“ in deep learning.

The training of neural networks typically uses the gradient (descent) method (or a variant thereof)

- For this purpose, the partial derivatives of the loss function w.r.t. the weights must be determined
- Therefor, the so-called **backpropagation algorithm (backward path)** is used, which is based on the chain rule
- The loss function is obtained by summing all samples of the training dataset. Subsequently, the contribution of a sample to the loss function is analyzed

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Forward Propagation

- Forward propagation is the evaluation of the model for a given sample $x \in \mathbb{R}^p$, meaning the calculation of $f(x)$, while f describes the network as function
- The sample is feeded to the network as model input, while the input is successively propagated forward through the model, computing affine linear mappings and the respective activation until the final output layer

Backward Propagation

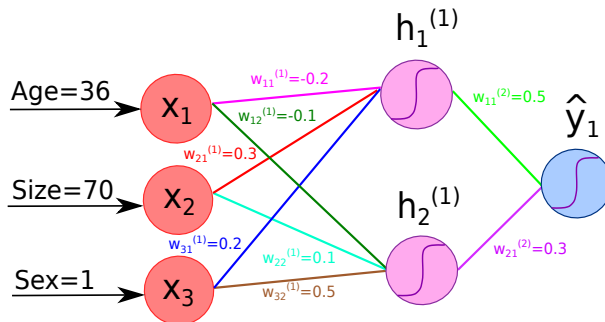
- Forward propagation takes place from the input layer to the output layer
- For a given sample, the error can be measured at the output layer (\rightarrow Loss, determined by the pre-defined loss function $L(\theta)$)
- This can be propagated back from the output layer to the input layer layer, in order to compute (using the chain rule) the partial derivatives of the loss loss function w.r.t. the weights/parameters \rightarrow This process is called back(ward) propagation

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Multi-Layer Perceptron

Forward Propagation

- The network uses the sigmoid function in all layers as the respective activation function
- Furthermore, bias terms are omitted for simplification
- Perform the forward propagation for the input $x = [36, 70, 1]^T$



Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

Additional Notes

$$h_1^{(1)} = x_1 \cdot w_{11}^{(1)} + x_2 \cdot w_{21}^{(1)} + x_3 \cdot w_{31}^{(1)} = 36 \cdot 0.2 + 70 \cdot 0.3 + 7 \cdot 0.2 = 74$$

$$\sigma(74) = 1$$

$$h_2^{(1)} = 36 \cdot 0.1 + 70 \cdot 0.1 + 7 \cdot 0.5 = 39$$

$$\sigma(39) = 0.58$$

$$\hat{y}_1 = h_1^{(1)} \cdot w_{11}^{(2)} + h_2^{(1)} \cdot w_{21}^{(2)} = 7 \cdot 0.5 + 0.58 \cdot 0.3 = 0.794$$

$$\sigma(0.794) = 0.69$$

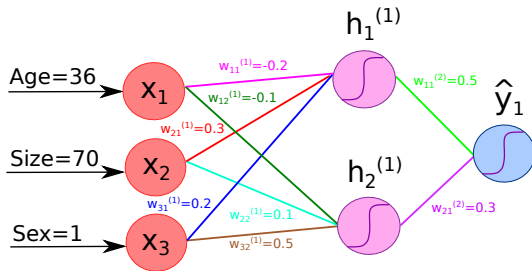
- Forward Propagation:

- $z^{(1)} = W^{(1)T}x + b^{(1)}$ and $h^{(1)} = \sigma(z^{(1)})$

- $z^{(2)} = W^{(2)T}h^{(1)} + b^{(2)}$ and $\hat{y} = \sigma(z^{(2)})$

- Computing the Loss:

- MSE-Loss: $L(\theta) = L(\hat{y}, y) = L_{\theta=(W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)})}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$



- Backward propagation via "Chain Rule"!

Source: OTH-AW, Electrical Engineering, Media and Computer Science, Fabian Brunner – Vorlesung Deep Learning, Multi-Layer Perceptron

$$\frac{1}{2}(\hat{y} - y)^2$$

Backward propagation – Partial Derivatives $L(\theta)$ w.r.t to $\theta (= W, b)$

$$\frac{\partial L}{\partial W^{(2)}} = \underbrace{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}}}_{\delta^{(2)} := \frac{\partial L}{\partial z^{(2)}}} \frac{\partial z^{(2)}}{\partial W^{(2)}} = (\hat{y} - y) \sigma'(z^{(2)}) h^{(1)}$$

Output
Gradient loss \rightarrow Zahl

$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial b^{(2)}} = (\hat{y} - y) \sigma'(z^{(2)}) \cdot 1$$

Wb + b

$$\frac{\partial L}{\partial W^{(1)}} = \underbrace{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial z^{(1)}}}_{\delta^{(1)} := \frac{\partial L}{\partial z^{(1)}}} \frac{\partial z^{(1)}}{\partial W^{(1)}} = (\hat{y} - y) \sigma'(z^{(2)}) W^{(2)} \sigma'(z^{(1)}) x$$

$$\frac{\partial L}{\partial b^{(1)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial b^{(1)}} = (\hat{y} - y) \sigma'(z^{(2)}) W^{(2)} \sigma'(z^{(1)}) \cdot 1$$

Backward propagation – Observations

- For $\delta^{(1)} = \frac{\partial L}{\partial z^{(1)}}$ and $\delta^{(2)} = \frac{\partial L}{\partial z^{(2)}}$ the following relation is given:

$$\delta^{(1)} = \delta^{(2)} \frac{\partial z^{(2)}}{\partial z^{(1)}} = \delta^{(2)} W^{(2)} \sigma'(z^{(1)})$$

- For $\delta^{(1)}$ and $\delta^{(2)}$ the partial derivatives of L according to the weights can be calculated as follows:

$$\begin{aligned} \frac{\partial L}{\partial W^{(1)}} &= \delta^{(1)} \frac{\partial z^{(1)}}{\partial W^{(1)}} , & \frac{\partial L}{\partial b^{(1)}} &= \delta^{(1)} \frac{\partial z^{(1)}}{\partial b^{(1)}} , \\ \frac{\partial L}{\partial W^{(2)}} &= \delta^{(2)} \frac{\partial z^{(2)}}{\partial W^{(2)}} , & \frac{\partial L}{\partial b^{(2)}} &= \delta^{(2)} \frac{\partial z^{(2)}}{\partial b^{(2)}} \end{aligned}$$

- This two-step procedure (first recursive calculation of the quantities $\delta^{(l)}$ from „right to left“, then calculation of the desired derivatives) is transferred to general MLPs with several units per hidden layer

Recursive Calculation of Variables $\delta^{(l)} = \frac{\partial L}{\partial z^{(l)}}$:

$$\delta^{(L)} = \nabla_{\hat{y}} L \odot (f^{(L)})'(z^{(L)}), \quad = \frac{\partial L}{\partial z^{(L)}} \quad \text{elementwise multiplication}$$
$$\delta^{(l)} = \mathbf{W}^{[l+1]} \delta^{[l+1]} \odot (f^{(l)})'(z^{(l)}), \quad l = L - 1, \dots, 1 \quad \text{activation for each element in vector}$$

Note: where \odot denotes the dot product between matrices.

Calculation of the Partial Derivatives with respect to the Variables $\delta^{(l)}$

$$\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)}, \quad l = 1, \dots, L$$

$$\frac{\partial L}{\partial w_{kj}^{(l)}} = h_k^{(l-1)} \delta_j^{(l)}, \quad l = 1, \dots, L$$

Summary

- From softmax regression to multi-layer networks
- Activation functions
- Forward propagation
- Backpropagation

Outlook

- Exercise: Implementation from scratch
- Initialisation of the weights
- Regularization
- Deep learning in PyTorch

Further Questions?



<https://www.oth-aw.de/hochschule/ueber-uns/personen/bergler-christian/>

Source: <https://emekaboris.medium.com/the-intuition-behind-100-days-of-data-science-code-c98402cdc92c>