Computer-Linguistische Anwendungen

CLA | B.Sc. | LMU

Vorlesung:
Neuronale Netze

Philipp Wicke, PhD
Centrum für Sprach- und Informationsverarbeitung
Ludwig-Maximilians-Universität München
pwicke@cis.lmu.de

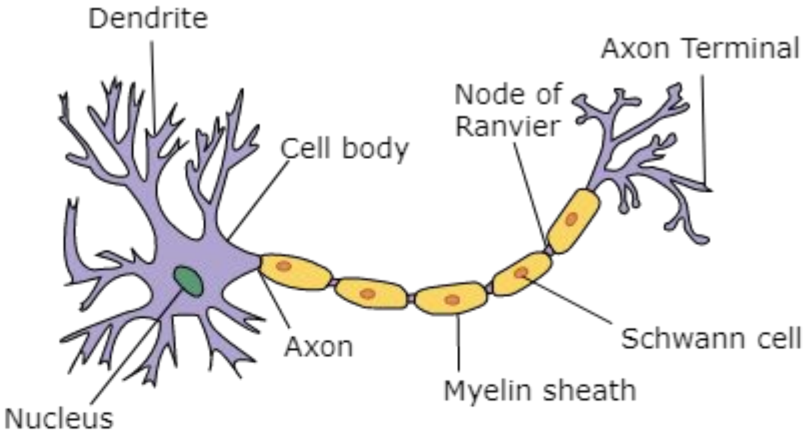
Übersicht

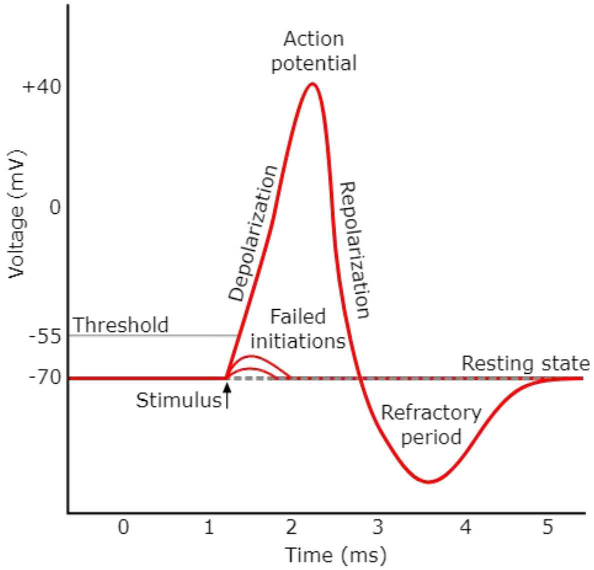
Themen der Vorlesung

- Einführung Neuronale Netze
- Anwendungsbeispiel
- Recurrent Neural Networks (RNNs)
- Convolutional Neural Networks (CNNs)

Neuronale Netze

Einführung

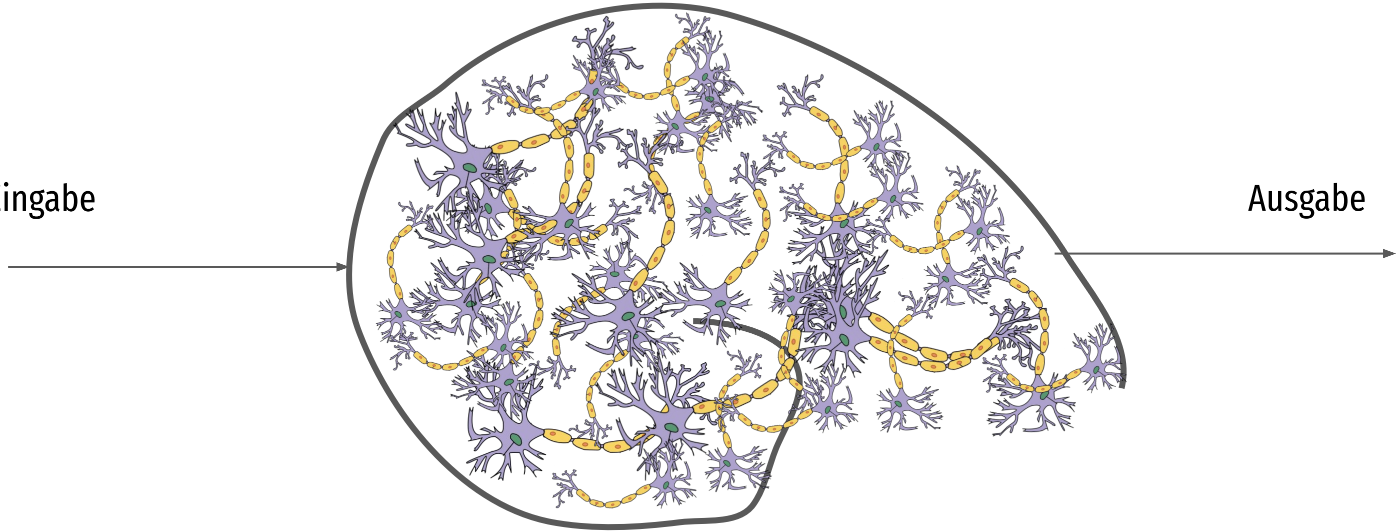




https://commons.wikimedia.org/wiki/File:Neuron.svg https://commons.wikimedia.org/wiki/File:Action_potential.svg

Neuronale Netze

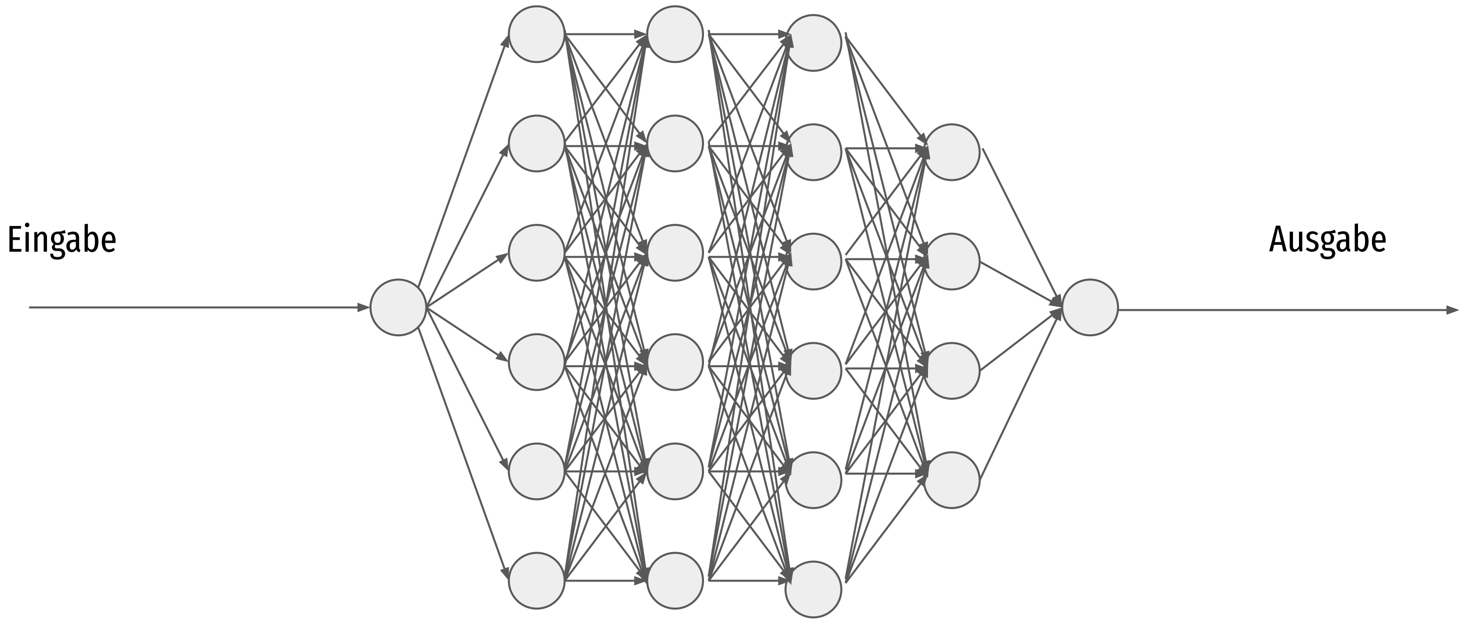
Einführung



https://commons.wikimedia.org/wiki/File:Neuron.svg https://commons.wikimedia.org/wiki/File:Action_potential.svg

Neuronale Netze

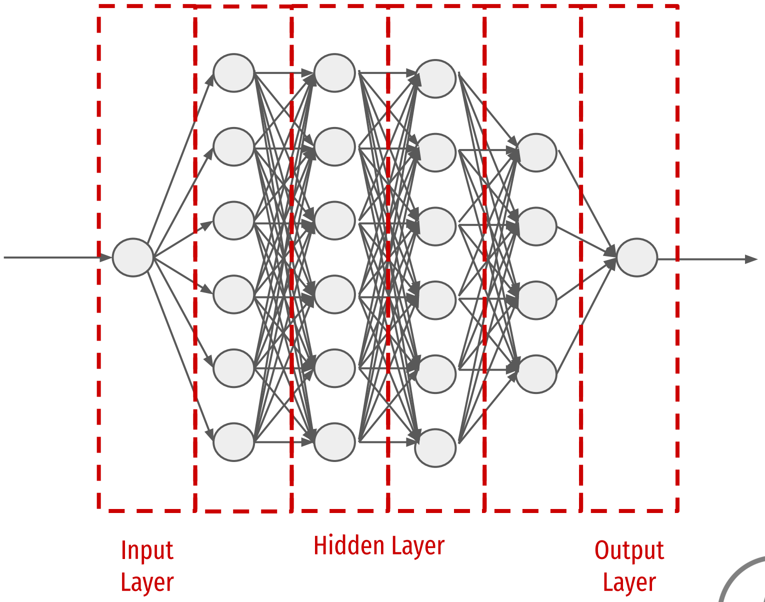
Einführung



Neuronale Netze

Layers

- Ein neuronales Netz setzt sich aus mehreren Schichten an Neuronen mit ihren dazugehörigen Gewichten zusammen. Diese Schichten nennt man **Layer**.
- Der ***Input-Layer*** bestimmt die Eingabe, den Anfang des Datenstroms in das Netzwerk
- Der ***Output-Layer*** hat dann die notwendige Größe für die Prädiktion / Generation / Klassifikation.
- Jeder Layer nimmt einen **Vektor** (oder Matrix, oder Tensor (mehrdimensionales Array)).
- Die Größe der Ausgabe muss nicht mit der Größe der Eingabe übereinstimmen (e.g. wird ein Bild klassifiziert: Eingabe -> Bild, Ausgabe -> Klassen Label)



Neuronale Netze

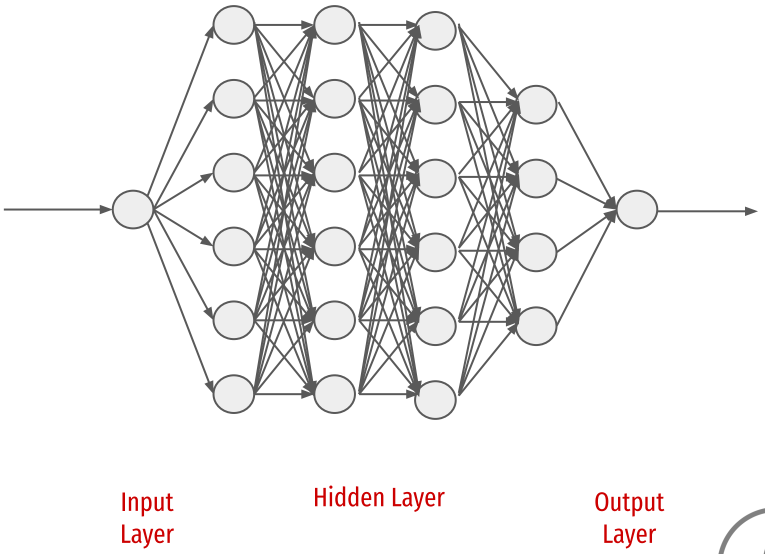
Training

- Zu Beginn werden die Gewichte des neuronalen Netzes alle **zufällig** gewählt

- Wenn nun ein Datenpunkt eingegeben wird, wird das Signal mit unterschiedlichen Gewichten berechnet und eine zufällige Ausgabe wird erfolgen.

Z.B. wenn auf einem Bild Hunde und Katzen unterschieden werden soll und beispielsweise ein Katzenbild gezeigt wird, dann ist die Wahrscheinlichkeit das ein untrainiertes Netzwerk das richtige Label findet 50%.

- Da für die Trainingsdaten ein Label existiert, kann dem Netzwerk gemeldet werden, welchen **prediction error** es begangen hat und dadurch die Gewichte verstärken oder schwächen



Neuronale Netze

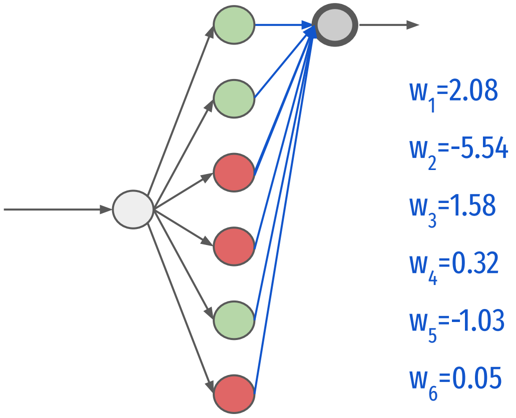
Training

- Für die Berechnung der Aktivierung wird die **gewichtete Summe** der eingehenden Signale gebildet

  Die unterliegende Optimierung hier lautet: wie müssen die Gewichte (w_N) angepasst werden, damit die Aktivierung für jeden Input das richtige Label liefert?

$$\sum_{i=1}^{n} w_i a_i = w_1 a_1 + w_2 a_2 + w_3 a_3 + \ldots + w_n a_n$$

  Mit a_N als eingehenden Aktivitäts-Signal des verbundenen Neuron.



Neuronale Netze

Training

- Da nun als gewichtete Summe Werte größer als 1 und kleiner als 0 resultieren können, die Aktivität eines Neurons aber mit 1 oder 0 (Feuern/nicht-Feuern) kodiert wird, wird eine **nichtlineare** Funktion gewählt, welche die Werte auf das Intervall 0-1 abbildet.

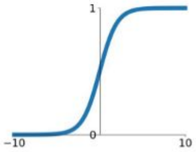
- Hier wird oft die **Sigmoid** Funktion gewählt:

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

$$\sigma(\sum_{i=1}^{n} w_i a_i) = \sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 + \ldots + w_n a_n)$$

Sigmoid

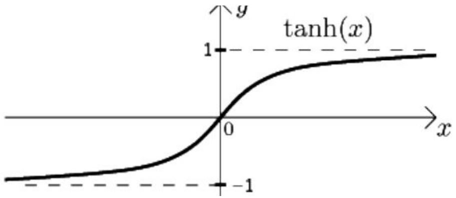
$\sigma(x) = \frac{1}{1+e^{-x}}$



Neuronale Netze

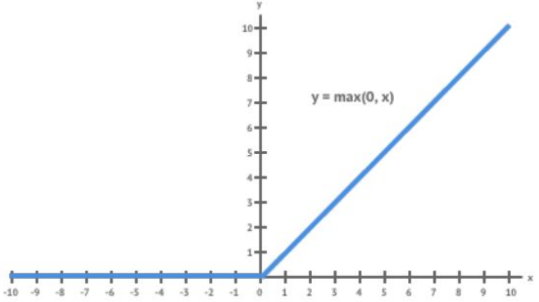
Alternative nichtlineare Funktionen

Tanh:



$y_i = \tanh(x_i) = 2\sigma(2x_i) - 1$
Wie Logistic Sigmoid, aber Wertebereich zwischen -1 . . . 1

ReLu: $\boldsymbol{y_i} = \max(0, \boldsymbol{x_i})$

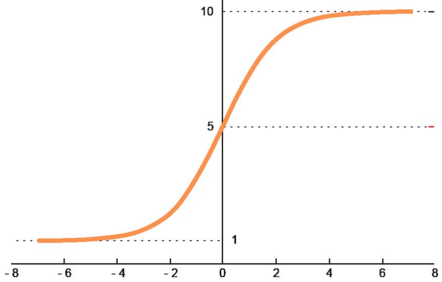


Rectified Linear Unit

Sparse activation: Bei einem zufällig initialisierten Netzwerk sind nur ungefähr die Hälfte der Neuronen aktiv.

In ReLu Neuronen kann es manchmal dazu kommen das sie einen Zustand erreichen in dem sie für alle Eingaben inaktiv werden ("sterben")

Softmax:



Normiert die Ausgabe der vorangehenden Layer zu einer Wahrscheinlichkeitsverteilung
Meist fur Vorhersage-Layer verwendet (WK für Ausgabe-Klassen)

$$\boldsymbol{y_i} = \frac{e^{(x_i)}}{\sum_j e^{(x_j)}}$$

Neuronale Netze

Training: Backpropagation

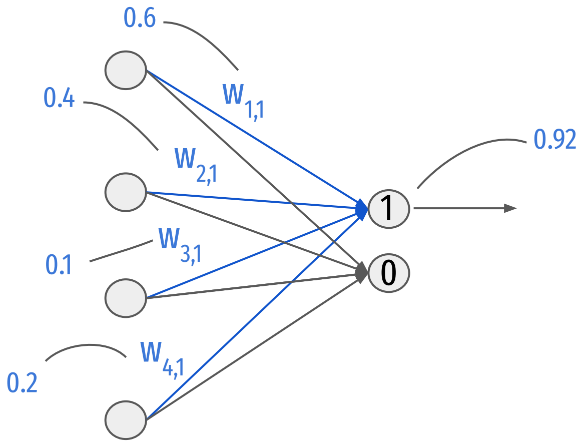
- Das eingehende Signal propagiert durch das Netzwerk bis es bei dem **output layer** ankommt und dort dem verfügbaren Label (hier: 0 oder 1) eine Wahrscheinlichkeit zuweist.

- Da für die Daten im **trainings set** für die das **ground truth** label existiert, kann am Ende des Trainings Schrittes der **error** als Differenz zwischen tatsächlicher (**target**) und predizierter (**output**) Wahrscheinlichkeit des Labels errechnet werden.

  $error_i = target_i - output_i$ (auch hier wird der squared error bevorzugt)

  $error_i = ½ (target_i - output_i)^2$

- Mit diesem Fehler möchten wir nun die **Gewichte** so adjustieren, dass der Fehler möglichst gering wird



Neuronale Netze

Training: Backpropagation

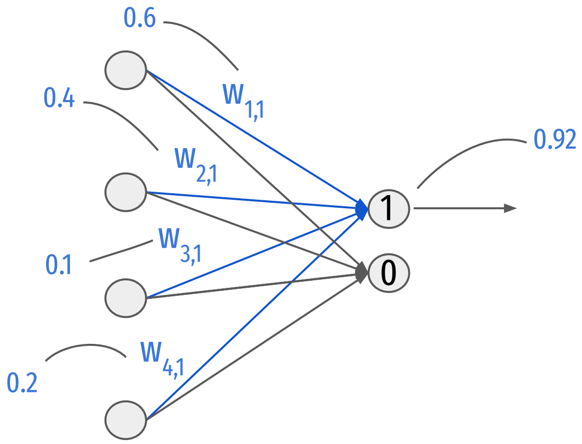
- Sei im Beispiel der **output** für “1” bei 0.92 und das **target** label tatsächlich “1” dann liegt der Fehler bei:

  error = 1 - 0.92 = 0.08

- Dieser Fehler soll nun auf die (hier) 4 verantwortlichen **Gewichte** ($w_{1,1}$, $w_{2,1}$, $w_{3,1}$, $w_{4,1}$) verrechnet werden. Es ergibt aber mehr Sinn diesen Fehler nicht gleichmäßig, sondern **proportional nach Gewichtswerten** zu verrechnen.

  Für $w_{1,0}$ hieße dies: $\text{error} \cdot \frac{w_{11}}{\sum_{i=1}^{4} w_{i1}}$

  oder: 0.08 * (0.6 / (0.6+0.4+0.1+0.2)) = 0.037



Neuronale Netze

Training: Backpropagation

Für die **Aktivierungsfunktion** wurde eine **nichtlineare**, **ableitbare** Funktion gewählt. Um den Fehler nun zurückzusenden (backpropagation), wird nun nicht mehr der **error** minimiert, sondern die **Ableitung** der **Fehlerfunktion.**

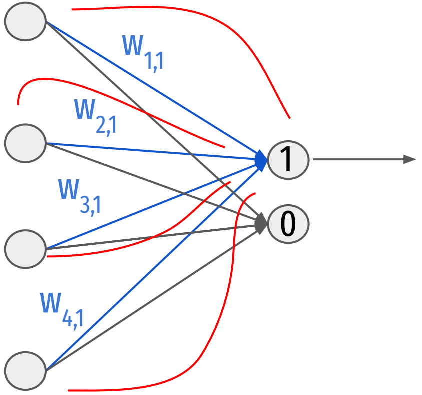
Diese gibt die Richtung des absteigenden Fehlers ab, hin welche die Gewichte optimiert werden sollen (**Gradient Descent**).

Ableitung des Fehlers bezüglich der Gewichte: $$\frac{\partial E}{\partial w_{ij}}$$

Fehlerfunktion: $$E = \sum_{j=1}^{n} \frac{1}{2}(t_j - o_j)^2$$

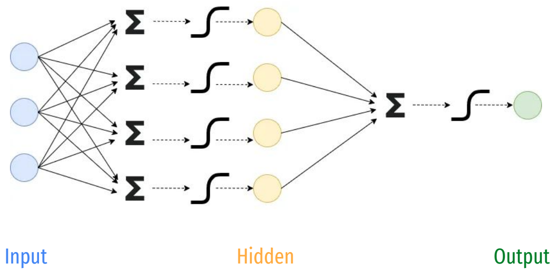
Fehlerfunktion mit Ableitung: $$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \frac{1}{2} \sum_{j=1}^{n}(t_j - o_j)^2$$

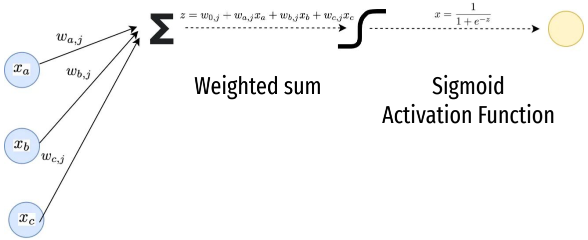
mit t: target und o: output



Neuronale Netze

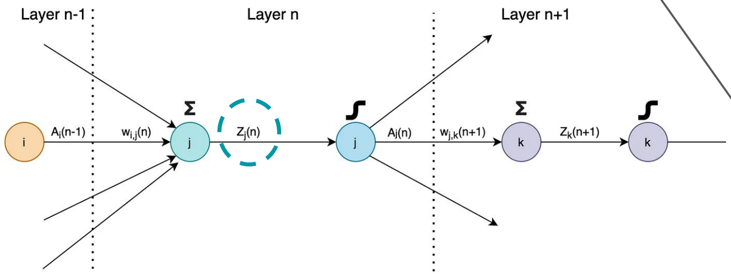
Backpropagation: Herleitung





$$w_i = w_i - \boxed{a}\boxed{\frac{\partial E(D,\mathbf{w})}{\partial w_i}}$$

Ziel des **Gradient Descent** ist es den Fehlerterm zu verringern und die Weights dementsprechend anzupassen. Wenn die **Steigung der Fehlerfunktion** negativ ist, wollen wir die Weights proportional stärker gewichten, mit einer Schrittgröße **a**.



Wir erweitern erneut die ∂E/∂z mit Hilfe der Kettenregel. Für ∂z/∂w erinnern wir uns daran, dass z_j die Summe aller Gewichte und Aktivierungen aus der vorherigen Schicht zu Neuron j ist. Die Ableitung nach dem Gewicht w_i,j ist daher einfach A_i(n-1).

$$\frac{\partial E(D,\mathbf{w})}{\partial w_{i,j}} = \frac{\partial E(D,\mathbf{w})}{\partial z_j(n)} \frac{\partial z_j(n)}{\partial w_{i,j}}$$

$$= \boxed{\frac{\partial E(D,\mathbf{w})}{\partial A_j(n)} \frac{\partial A_j(n)}{\partial z_j(n)}} \; A_i(n-1)$$

$$= \boxed{-2(y - \mathbf{xw})} \; A_i(n-1)$$

with error function: $E = (y - \mathbf{xw})^2$

$$\mathrm{w}_{i,j} \leftarrow w_{i,j} - a\frac{\partial E(D,\mathbf{w})}{\partial w_{i,j}}$$

$$\mathrm{w}_{i,j} \leftarrow w_{i,j} - a[-2(y - \mathbf{xw}) \; A_i(n-1)] \text{ last layer}$$

$$\mathrm{w}_{i,j} \leftarrow w_{i,j} - a\left[\sum_k \left[\frac{\partial E(D,\mathbf{w})}{\partial z_k(n+1)} \quad w_{j,k}(n+1)\right] A_i(n-1)\right] \text{ all other layers}$$

Neuronale Netze

Training: Backpropagation

Kettenregel

$$f(x) = u(v(x))$$

$$\underbrace{f'(x)}_{\text{Ableitung}} = \underbrace{u'(v(x))}_{\text{Äußere Ableitung}} \cdot \underbrace{v'(x)}_{\text{Innere Ableitung}}$$

Folgende Herleitung:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \frac{1}{2} \sum_{j=1}^{n} (t_j - o_j)^2$$

Error-Funktion mit Ableitung

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \frac{1}{2} (t_j - o_j)^2$$

Wenn für jede einzelne Ausgabe der Fehler separat berechnet wird kann die Summe verschwinden.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial w_{ij}}$$

Anwendung der Kettenregel

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

Sigmoid-Aktivierungsfunkt.

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x))$$

Ableitung der Sigm.-Funkt.

$$\frac{\partial E}{\partial w_{ij}} = (t_j - o_j) \cdot \sigma(\sum_{i=1}^{m} w_{ij} h_i) \cdot (1 - \sigma(\sum_{i=1}^{m} w_{ij} h_i)) \cdot o_j$$

Neuronale Netze

Anwendungsbeispiel: Neuronales Netz das handgeschriebene Zahlen erkennen kann (in Keras).