# Neural Networks: Architectures and Applications for NLP

*Session 02*

Julia Kreutzer

8. November 2016

Institut für Computerlinguistik, Heidelberg

# Overview

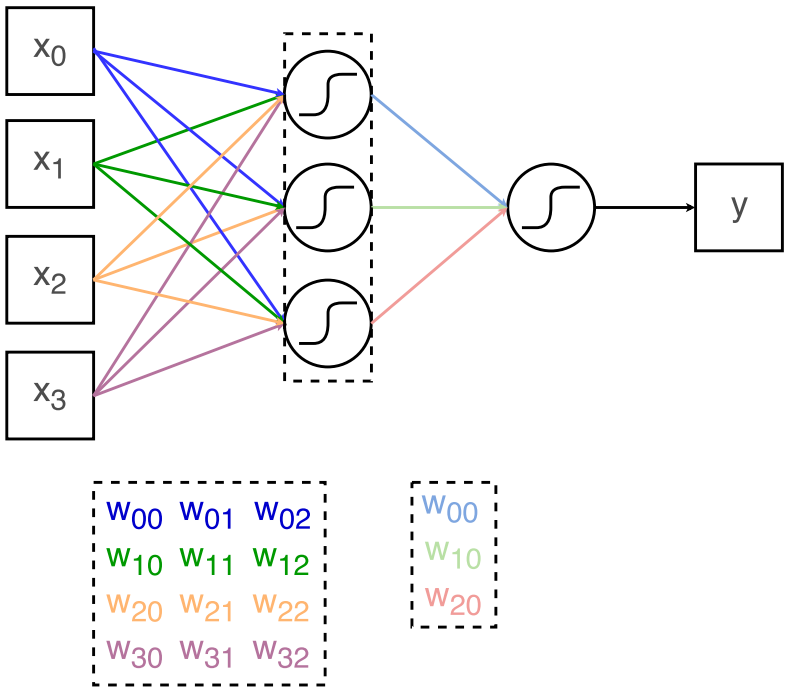
1. Recap

2. Backpropagation

3. Ausblick

# Recap

# Multi-Layer-Perceptron: Regression




$$NN(\mathbf{x}) = \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

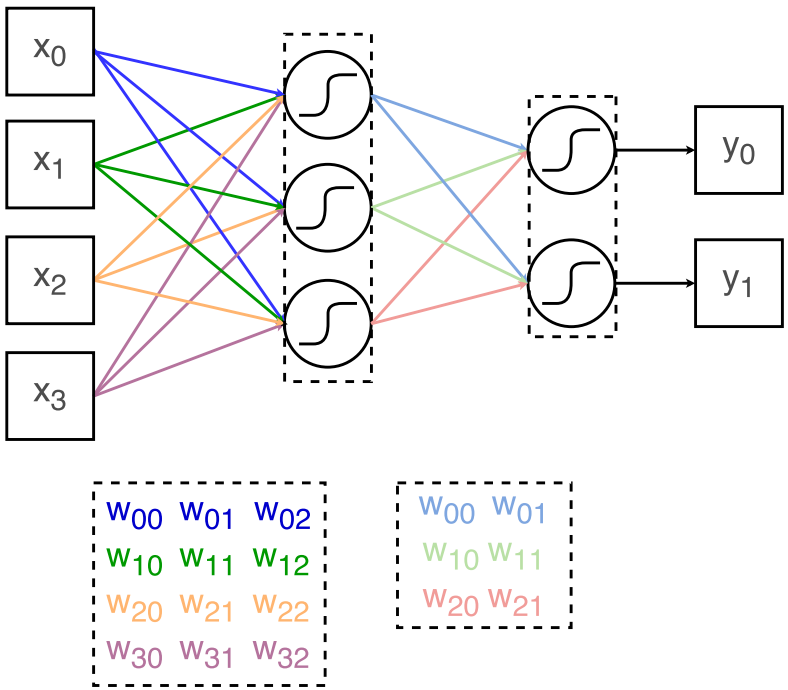
Regression: $NN(\mathbf{x}) \in \mathbb{R}$

Beispiel:

$\mathbf{W}_1 \in \mathbb{R}^{4x3}, \mathbf{b}_1 \in \mathbb{R}^3,$

$\mathbf{W}_2 \in \mathbb{R}^{3x1}, \mathbf{b}_2 \in \mathbb{R}^1$

# Multi-Layer-Perceptron: Binäre Klassifikation




$$\mathrm{NN}(\mathbf{x}) = \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

Binäre Klassifikation:

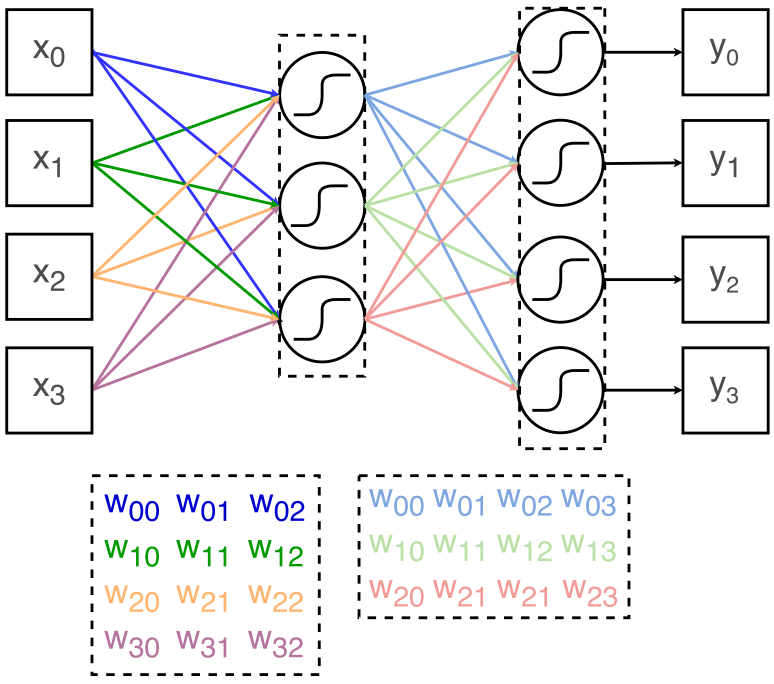
$\mathrm{NN}(\mathbf{x}) \in \mathbb{R}^2$

Beispiel:

$\mathbf{W}_1 \in \mathbb{R}^{4x3}, \mathbf{b}_1 \in \mathbb{R}^3,$

$\mathbf{W}_2 \in \mathbb{R}^{3x2}, \mathbf{b}_2 \in \mathbb{R}^2$

# Multi-Layer-Perceptron: Multinomiale Klassifikation




$$\mathrm{NN}(\mathbf{x}) = \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

Multinomiale Klassifikation:
$\mathrm{NN}(\mathbf{x}) \in \mathbb{R}^k$

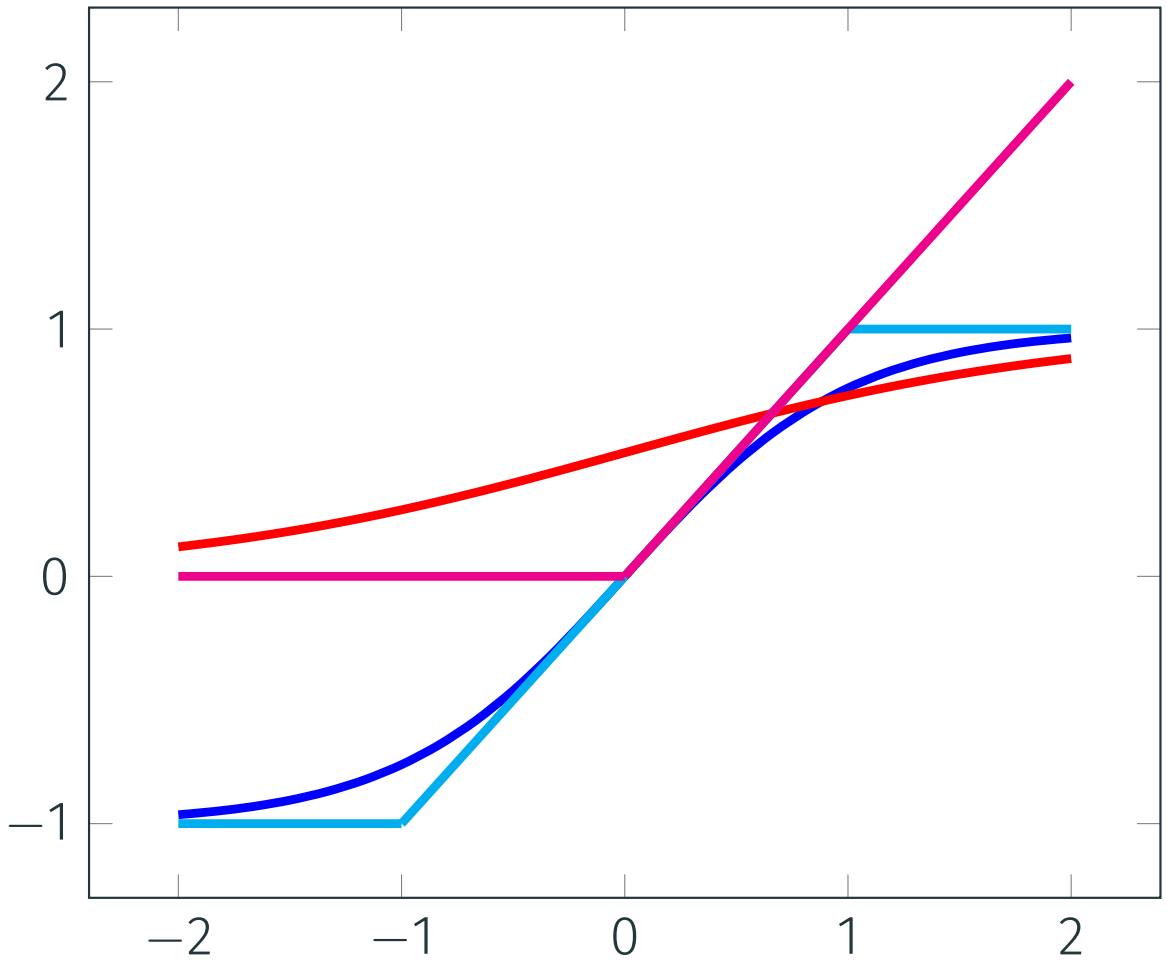
$\Rightarrow$ $k$ Ausgabeneuronen liefern Scores für $k$ Klassen

# Aktivierungsfunktionen

| | $f(z)$ | $f'(z)$ | $f(z) \in$ |
|---|---|---|---|
| `sigmoid` | $\frac{1}{1+exp(z)}$ | $\sigma(z)(1-\sigma(z))$ | $(0,1)$ |
| `tanh` | $\frac{exp(2z)-1}{exp(2z)+1}$ | $1-tanh^2(z)$ | $(-1,+1)$ |
| `hardtanh` | $\begin{cases} -1 & z < -1 \\ 1 & z > 1 \\ z & \text{otherwise} \end{cases}$ | $\begin{cases} x & -1 \leq z \leq 1 \\ 0 & \text{otherwise} \end{cases}$ | $(-1,+1)$ |
| `ReLU`[1] | $\max(0,z)$ | $\begin{cases} 1 & z > 0 \\ 0 & \text{otherwise} \end{cases}$ | $(0,\infty)$ |

[1] Rectified Linear Units, auch: hinge

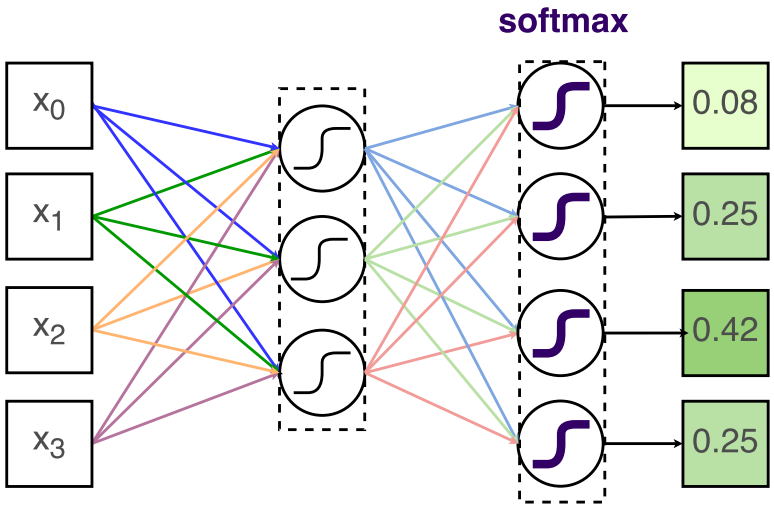
# Aktivierungsfunktionen




tanh
hardtanh
sigmoid
ReLU

# softmax-Funktion

softmax erlaubt die Interpretation der Ausgabe als Wahrscheinlichkeitsverteilung über die $k$ Klassen:




$$\texttt{softmax}(o_i) = \frac{exp(o_i)}{\sum_j^k exp(o_j)}$$

$$P(\hat{y} = i|\mathbf{x}) = \texttt{softmax}(o_i)$$
$$\sum_i^k P(\hat{y} = i|\mathbf{x}) = 1$$

# Klassifikation: Von der Netzausgabe zum Fehler

Fehlerfunktionen vergleichen Verteilungen ($P(y = i|\mathbf{x})$ vs. $P(\hat{y} = i|\mathbf{x})$) oder Scores für Klassen (z.B. multinomial 1 vs. $o_i$).
Evaluationsmaße vergleichen meist die Klassenzuordnung ($y$ vs. $\hat{y}$).

**Beispiel**: Klassifikation mit 4 Klassen

| | |
|---|---|
| Gegeben: | korrekte Klasse $y = 1$ |
| Als One-Hot-Vektor: | $\mathbf{y} = [0\,, 1\,, 0\,, 0]$ |
| Netzausgabe: | $\hat{\mathbf{y}} = [0.08\,, 0.25\,, 0.42\,, 0.25]$ |
| Vorhergesagte Klasse: | $\arg\max_i \hat{y}_i = 2$ |

# Gradient

Der Gradient einer Funktion gibt die Richtung des steilsten Anstiegs an.

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Die Einträge $\frac{\partial f}{\partial x_i}$ sind partielle Ableitungen von $f$ in $x_i$-Richtung.

# Batch Gradient Descent

**Algorithm 1** Batch Gradient Descent

**Input:** Trainingset $(\mathbf{x_i}, y_i) \in D$

**Output:** $\theta^* = \arg\min R_D(\theta)$

1: **for** $1 \rightarrow$ N **do** ▷ N epochs

2: $\theta \leftarrow \gamma \frac{1}{|D|} \sum_{i}^{|D|} \frac{\partial L(f_\theta(\mathbf{x_i}), y_i)}{\partial \theta}$ ▷ Average gradients of training loss

3: **end for**

# Stochastic Gradient Descent (SGD) i

**Algorithm 2** Stochastic Gradient Descent

**Input:** Trainingset $(\mathbf{x_i}, y_i) \in D$

**Output:** $\theta^* = \arg\min R_D(\theta)$

1: **while** stopping criterion not met **do**

2: Sample $(\mathbf{x_i}, y_i)$ from $D$

3: $\theta \leftarrow \gamma \frac{\partial L(f_\theta(\mathbf{x_i}), y_i)}{\partial \theta}$ ▷ Gradients of **one** training sample

4: **end while**

Stopping criteria: convergence, fixed number of iterations, ...

# Stochastic Gradient Descent (SGD) ii

In practice:

**Algorithm 3** Stochastic Gradient Descent (shuffle)

**Input:** Trainingset $(\mathbf{x_i}, y_i) \in D$

**Output:** $\theta^* = \arg\min R_D(\theta)$

1: **for** $1 \rightarrow$ N **do** ▷ N epochs
2: Shuffle $D$ ▷ Stochasticity
3: **for** $i = 1 \ldots |D|$ **do** ▷ |D| iterations
4: $\theta \leftarrow \gamma \frac{\partial L(f_\theta(\mathbf{x_i}), y_i)}{\partial \theta}$ ▷ Gradients of **one** training sample
5: **end for**
6: **end for**

# Minibatch Gradient Descent i

**Algorithm 4** Minibatch Gradient Descent

**Input:** Trainingset $(\mathbf{x_i}, y_i) \in D$

**Output:** $\theta^* = \arg\min R_D(\theta)$

1: **while** stopping criterion not met **do**

2: Sample minibatch of size $s$ from $D$

3: $\theta \leftarrow \gamma \frac{1}{s} \sum_{i}^{s} \frac{\partial L(f_\theta(\mathbf{x_i}), y_i)}{\partial \theta}$ ▷ Average gradients of minibatch

4: **end while**

# Minibatch Gradient Descent ii

In practice:

**Algorithm 5** Minibatch Gradient Descent (shuffle)

**Input:** Trainingset $(\mathbf{x_i}, y_i) \in D$

**Output:** $\theta^* = \arg\min R_D(\theta)$

1: **for** $1 \rightarrow$ N **do** ▷ N epochs
2: Shuffle $D$
3: Split $D$ into minibatches of size $s$
4: **for** $i = 1 \ldots \frac{|D|}{s}$ **do** ▷ $\frac{|D|}{s}$ iterations
5: $\theta \leftarrow \gamma \frac{1}{S} \sum_{i}^{S} \frac{\partial L(f_\theta(\mathbf{x_i}), y_i)}{\partial \theta}$ ▷ Average gradients of minibatch
6: **end for**
7: **end for**

# Gradient Descent Varianten [Bottou, 2004]

1. Batch
   - Garantierte Konvergenz zu einem Minimum (konvex: global, non-konvex: lokal)
   - Redundanz bei großen Trainingsets
   - Aufwendige Berechnung der Gradienten

# Gradient Descent Varianten [Bottou, 2004]

1. Batch
   - Garantierte Konvergenz zu einem Minimum (konvex: global, non-konvex: lokal)
   - Redundanz bei großen Trainingsets
   - Aufwendige Berechnung der Gradienten
2. Stochastic
   - schnellere Berechnung
   - Online-Adaption
   - höhere Varianz der Gradienten
   - garantierte Konvergenz mit kleiner werdendem $\eta$

# Gradient Descent Varianten [Bottou, 2004]

1. Batch
   - Garantierte Konvergenz zu einem Minimum (konvex: global, non-konvex: lokal)
   - Redundanz bei großen Trainingsets
   - Aufwendige Berechnung der Gradienten
2. Stochastic
   - schnellere Berechnung
   - Online-Adaption
   - höhere Varianz der Gradienten
   - garantierte Konvergenz mit kleiner werdendem $\eta$
3. Minibatch
   - Varianzreduktion im Vergleich zu SGD
   - effiziente Berechnung mit NN-Libraries
   - zusätzlicher Hyperparameter: Größe der Minibatches

# Ableitungsregeln

| | $f(x)$ | $f'(x)$ |
|---|---|---|
| Konstante | $c$ | $0$ |
| Vielfaches | $c \cdot f(x)$ | $c \cdot f'(x)$ |
| Summe | $g(x) + h(x)$ | $g'(x) + h'(x)$ |
| Produkt | $g(x) \cdot h(x)$ | $g'(x) \cdot h(x) + g(x) \cdot h'(x)$ |
| **Kettenregel** | $\mathbf{g(h(x))}$ | $\mathbf{g'(h(x)) \cdot h'(x)}$ |
| Potenz | $x^n$ | $n \cdot x^{n-1}$ |
| Exponentialfunktion | $\exp(x)$ | $\exp(x)$ |
| Logarithmusfunktionen | $\ln(x)$ | $\frac{1}{x}$ |
| | ${}^a\log x$ | $\frac{1}{x \ln a}$ |

# Backpropagation

# Backpropagation: Idee

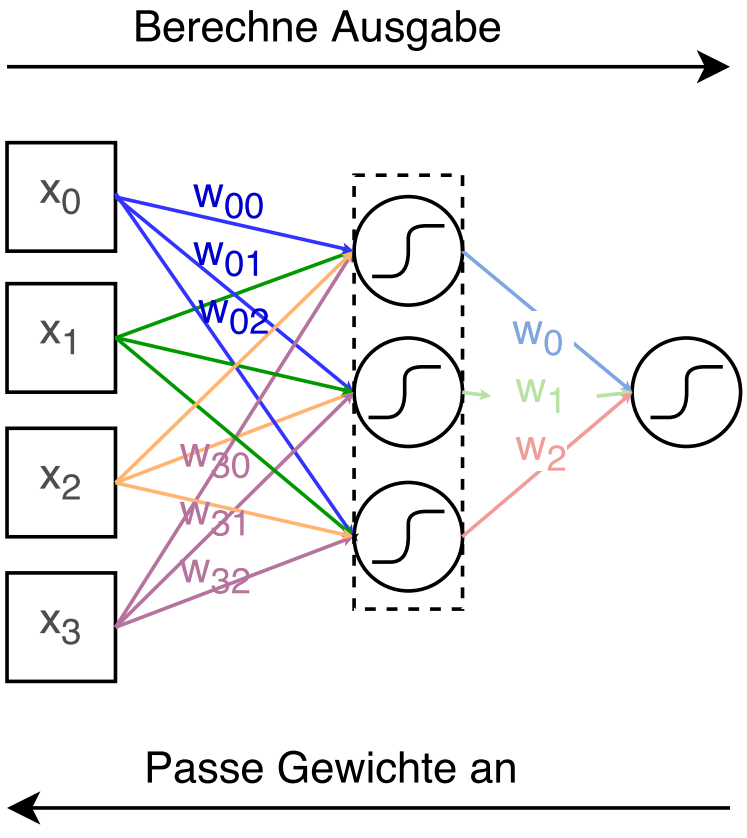
Lernprozess: Klassifizierungsfehler $\Rightarrow$ Update der Gewichte

Welche Gewichte eines tiefen Netzwerks müssen wie geändert werden?

**Backpropagation**:

- Kettenregel zur Berechnung der Gradienten
- Wiederverwenden von Zwischenergebnissen

# Backpropagation: Beispiel




1. Berechne

$$\begin{aligned}\mathrm{NN}(\mathbf{x}) &= \sigma(\mathbf{W}_2 \cdot \mathbf{h})) \\ &= \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot \mathbf{x})) \\ &= \sigma(\textstyle\sum_j w_j \cdot h_j) \\ &= \sigma(\textstyle\sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i))\end{aligned}$$

$$L(\mathrm{NN}(\mathbf{x}), y) = \tfrac{1}{2}(y - \mathrm{NN}(\mathbf{x}))^2$$

# Backpropagation: Beispiel

2. Berechne
   a) Für jedes Gewicht in der letzten Schicht:

$$\begin{aligned}
\frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial w_j} &= \frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial \mathrm{NN}(\mathbf{x})} \frac{\partial \mathrm{NN}(\mathbf{x})}{\partial w_j} \\
&= (\mathbf{y} - NN(\mathbf{x})) \frac{\partial \sigma(\sum_j w_j \cdot h_j)}{\partial w_j} \\
&= (\mathbf{y} - NN(\mathbf{x})) \frac{\partial \sigma(\sum_j w_j \cdot h_j)}{\sum_j w_j \cdot h_j} \frac{\partial \sum_j w_j \cdot h_j}{\partial w_j} \\
&= (\mathbf{y} - NN(\mathbf{x}))\, \sigma'(\textstyle\sum_j w_j \cdot h_j)\, h_j
\end{aligned}$$

# Backpropagation: Beispiel

2. Berechne
   b) Für jedes Gewicht in der ersten Schicht:

$$
\begin{aligned}
\frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial w_{ij}} &= \frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial \mathrm{NN}(\mathbf{x})} \frac{\partial \mathrm{NN}(\mathbf{x})}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x})) \frac{\partial \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i))}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x})) \frac{\partial \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i))}{\partial \sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i)} \frac{\sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i)}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x}))\, \sigma'(\textstyle\sum_j w_j \cdot h_j)\, \frac{\partial \sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i)}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x}))\, \sigma'(\textstyle\sum_j w_j \cdot h_j)\, w_j\, \frac{\partial \sigma(\sum_i w_{ij} \cdot x_i)}{\partial \sum_i w_{ij} \cdot x_i} \frac{\partial \sum_i w_{ij} \cdot x_i}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x}))\, \sigma'(\textstyle\sum_j w_j \cdot h_j)\, w_j\, \sigma'(\sum_i w_{ij} \cdot x_i)\, x_i
\end{aligned}
$$

# Backpropagation: Beispiel

2. Berechne
   a) Für jedes Gewicht in der letzten Schicht:

$$\begin{aligned}
\frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial w_j} &= \frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial \mathrm{NN}(\mathbf{x})} \frac{\partial \mathrm{NN}(\mathbf{x})}{\partial w_j} \\
&= (\mathbf{y} - NN(\mathbf{x})) \frac{\partial \sigma(\sum_j w_j \cdot h_j)}{\partial w_j} \\
&= (\mathbf{y} - NN(\mathbf{x})) \frac{\partial \sigma(\sum_j w_j \cdot h_j)}{\sum_j w_j \cdot h_j} \frac{\partial \sum_j w_j \cdot h_j}{\partial w_j} \\
&= (\mathbf{y} - NN(\mathbf{x}))\, \sigma'(\textstyle\sum_j w_j \cdot h_j)\, h_j
\end{aligned}$$

# Backpropagation: Beispiel

2. Berechne
   b) Für jedes Gewicht in der ersten Schicht:

$$
\begin{aligned}
\frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial w_{ij}} &= \frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial \mathrm{NN}(\mathbf{x})} \frac{\partial \mathrm{NN}(\mathbf{x})}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x})) \frac{\partial \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i))}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x})) \frac{\partial \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i))}{\partial \sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i)} \frac{\sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i)}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x}))\, \sigma'(\textstyle\sum_j w_j \cdot h_j) \frac{\partial \sum_j w_j \cdot \sigma(\sum_i w_{ij} \cdot x_i)}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x}))\, \sigma'(\textstyle\sum_j w_j \cdot h_j)\, w_j \frac{\partial \sigma(\sum_i w_{ij} \cdot x_i)}{\partial \sum_i w_{ij} \cdot x_i} \frac{\partial \sum_i w_{ij} \cdot x_i}{\partial w_{ij}} \\
&= (\mathbf{y} - NN(\mathbf{x}))\, \sigma'(\textstyle\sum_j w_j \cdot h_j)\, w_j\, \sigma'(\sum_i w_{ij} \cdot x_i)\, x_i
\end{aligned}
$$

# Backpropagation: Deltas

Gradienten der Gewichte eines Neurons setzen sich aus dem skalierten Fehler aus den oberen Schicht und dem Output der unteren Schicht zusammen:

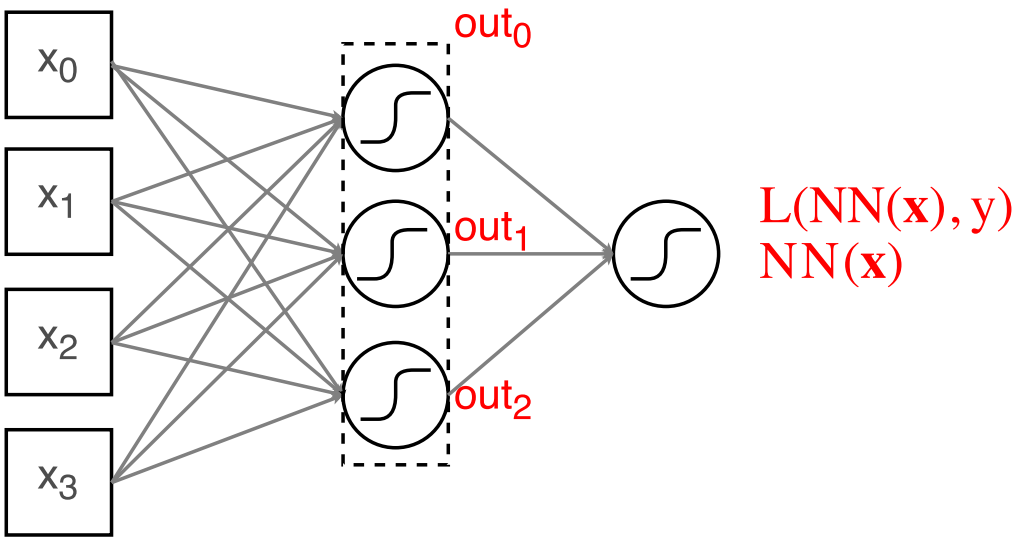
$$\begin{aligned}\frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial w_{ij}} &= \frac{\partial L(\mathrm{NN}(\mathbf{x}), y)}{\partial \sum_i w_{ij} \cdot x_i} \frac{\partial \sum_i w_{ij} \cdot x_i}{\partial w_{ij}} \\ &= \delta_j \, out_i\end{aligned}$$

Update-Regel:

$$w_{ij} \mathrel{-}= \eta \, \delta_j \, out_i$$

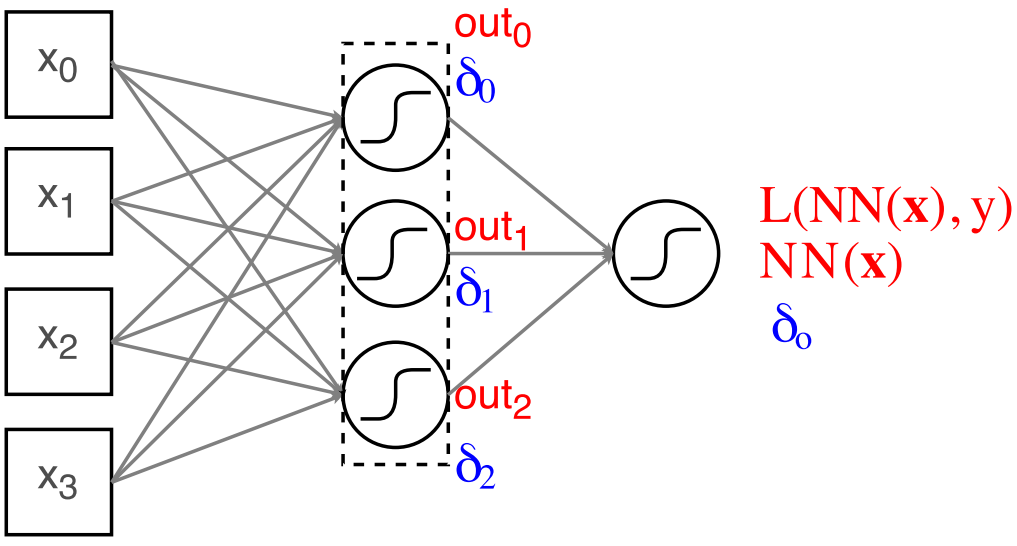
# Backpropagation: Algorithmus i

1. **Forward Pass** vom Input- zum Output-Layer:
   - Berechne die Ausgabe *out* für jedes Neuron
   - Berechne die Netzausgabe $\mathrm{NN}(\mathbf{x})$
   - Berechne den Fehler $L(\mathrm{NN}(\mathbf{x}), y)$



# Backpropagation: Algorithmus ii

2. **Backward Pass** vom Output- zum Input-Layer:
   - Berechne den Fehler-Term $\delta$ für jedes Neuron
   - Berechne das Gewichtsupdate $\Delta w_{ij} = -\eta\, \delta_j\, out_i$



# Computation Graph Backprop

Repräsentation des **NN als DAG**:

- Knoten entsprechen Tensor-Operationen oder Variablen (”tensor”)
- Kanten geben die Weitergabe der Werte an (”flow”)
- Topologische Sortierung für Backpropagation
- Mehr: Chris Olah’s Blogpost über Backprop mit Computation Graphs

# Computation Graph – MLP

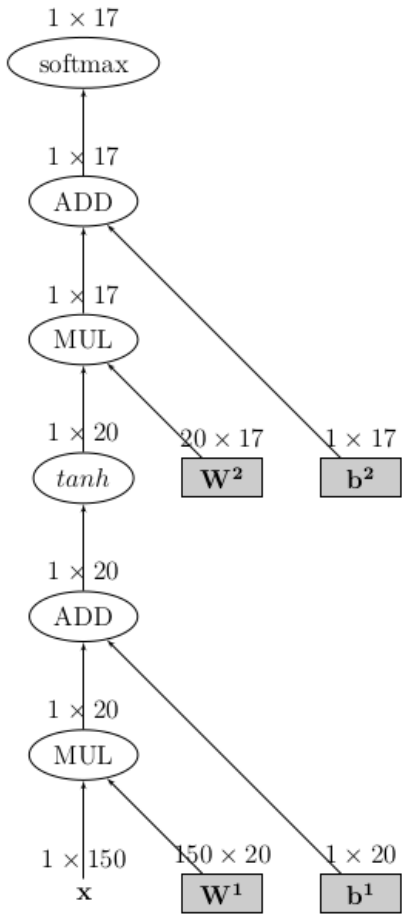



**Abbildung 1:** Graph ohne konkretes Input [Goldberg, 2015]

# Computation Graph – MLP

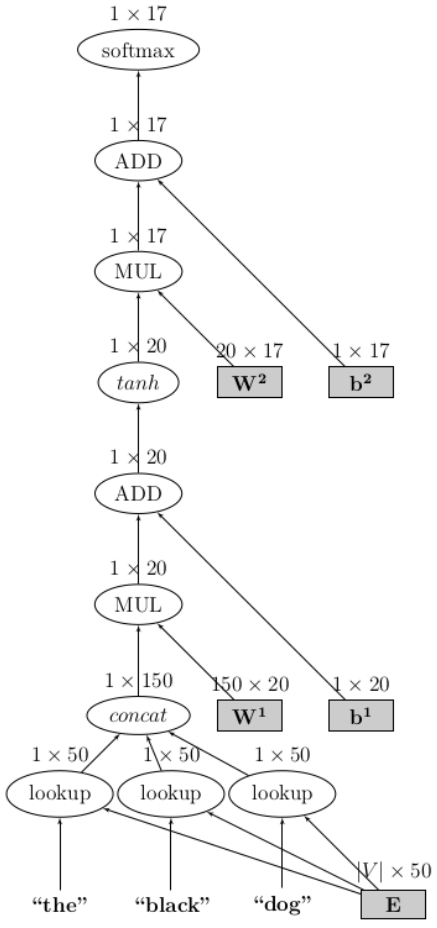



**Abbildung 2:** Graph mit konkretem Input [Goldberg, 2015]

# Computation Graph – MLP

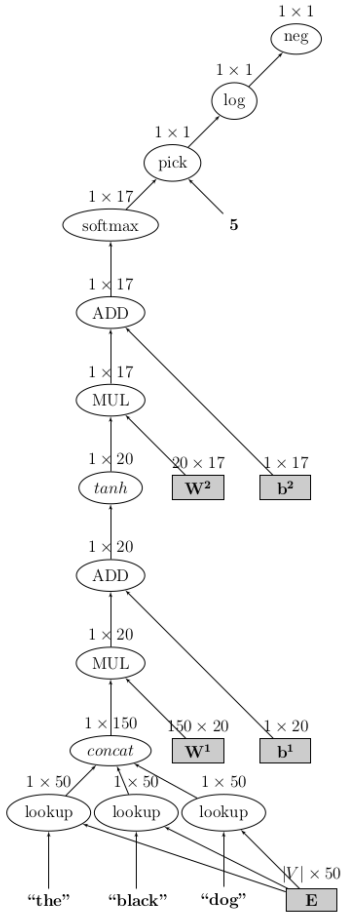



**Abbildung 3:** Graph mit Input und Fehlerberechnung [Goldberg, 2015]

# Ausblick

# Nächstes Mal

- How to train your NN
- Probleme
- NLP-Anwendungen

# Wer nicht lesen will muss hören

Machine Learning Podcasts:

- This Week in Machine Learning and AI
- Talking Machines
- Learning Machines 101

# References i


Bottou, L. (2004).
**Stochastic learning.**
In *Advanced lectures on machine learning*, pages 146–168. Springer.

Goldberg, Y. (2015).
**A primer on neural network models for natural language processing.**
*arXiv preprint arXiv:1510.00726*.