Erweiterungsmodul: Maschinelle Übersetzung

Teil 2: Neuronale maschinelle Übersetzung

Helmut Schmid

Stand: 3. Juli 2019

Überblick

- von log-linearen Modellen zu neuronalen Netzen
- Embeddings
- Aktivierungsfunktionen
- Training von neuronalen Netzen
- Rekurrente Netze
- LSTM
- Encoder-Decoder-Modelle für Übersetzung
- Attention
- Byte-Pair-Encoding
- Backtranslation
- Transformer
- Bilinguale Embeddings
- Unüberwachtes Training von Übersetzungssystemen
- Adversarial Networks ("gegnerische" Netzwerke)

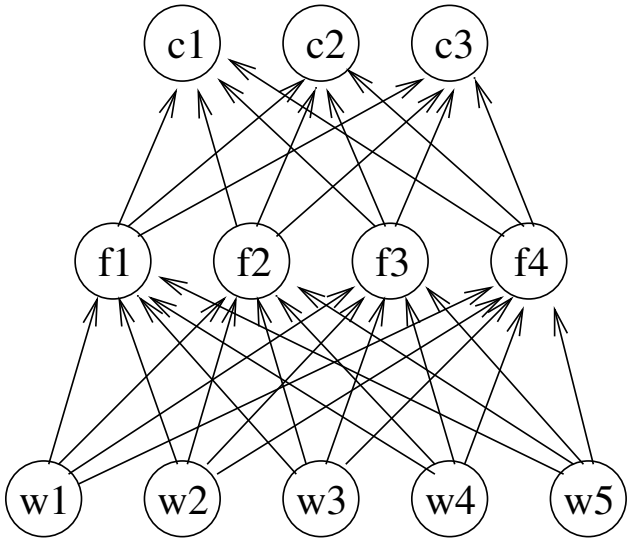
Lernen von Repräsentationen

- Modelle mit log-linearen Klassifikatoren arbeiten mit einer manuell definierten Menge von Merkmalen
- Gute Merkmale zu finden ist schwierig.
- Die gefundene Menge ist selten optimal.
- Die Suche muss wiederholt werden, wenn sich die Aufgabe ändert.

Könnte man die Merkmale automatisch lernen?

Von log-linearen Modellen zu neuronalen Netzen

grafische Darstellung eines log-linearen Modelles



Klassen: c1, c2, c3
Merkmale: f1, f2, f3, f4
Eingabewörter: w1, w2, w3, w4, w5

Jedes Merkmal f_k wird aus w_1^5 berechnet

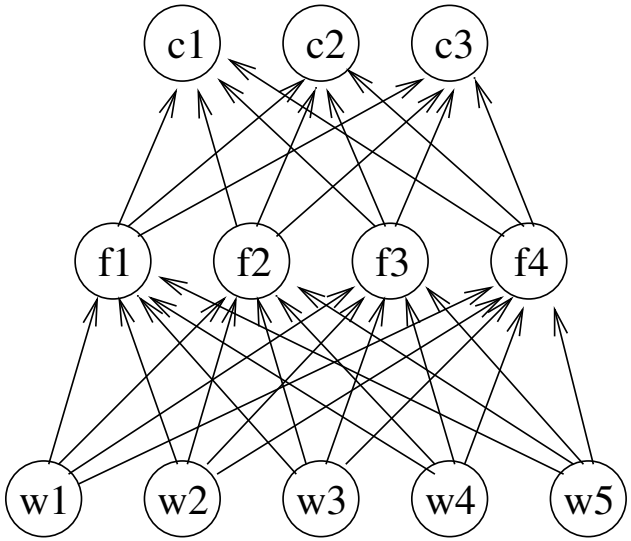
Jede Klasse c_k wird aus f_1^4 berechnet mit

$$c_k = \frac{1}{Z} e^{\sum_i \theta_{ki} f_i} = \mathit{softmax}\left(\sum_i \theta_{ki} f_i\right)$$

Anmerkung: Hier sind die Merkmale unabhängig von den Klassen. Stattdessen gibt es für jedes Merkmal klassenspezifische Gewichte.

Von log-linearen Modellen zu neuronalen Netzen

grafische Darstellung eines log-linearen Modelles



Klassen: c1, c2, c3
Merkmale: f1, f2, f3, f4
Eingabewörter: w1, w2, w3, w4, w5

Jedes Merkmal f_k wird aus w_1^5 berechnet

Jede Klasse c_k wird aus f_1^4 berechnet mit

$$c_k = \frac{1}{Z} e^{\sum_i \theta_{ki} f_i} = softmax\left(\sum_i \theta_{ki} f_i\right)$$

Anmerkung: Hier sind die Merkmale unabhängig von den Klassen. Stattdessen gibt es für jedes Merkmal klassenspezifische Gewichte.

Idee: Die Merkmale f_i werden analog zu den Klassen c_i berechnet/gelernt:

$$f_k = act\left(\sum_i \theta_{ki} w_i\right) = act(net_k)$$

$\Rightarrow$ neuronales Netz

Embeddings

$$f_k = act\left(\sum_i \theta_{ki} w_i\right)$$

Die Eingabe w_i muss eine numerische Repräsentation des Wortes sein.

Möglichkeiten

Embeddings

$$f_k = act\left(\sum_i \theta_{ki} w_i\right)$$

Die Eingabe w_i muss eine numerische Repräsentation des Wortes sein.

Möglichkeiten

1. binärer Vektor (1-hot-Repräsentation)
   - Beispiel: $w_i^{hot} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)$
   - Dimension = Vokabulargröße
     $\Rightarrow$ Alle Wörter sind sich gleich ähnlich/unähnlich

Embeddings

$$f_k = act\left(\sum_i \theta_{ki} w_i\right)$$

Die Eingabe w_i muss eine numerische Repräsentation des Wortes sein.

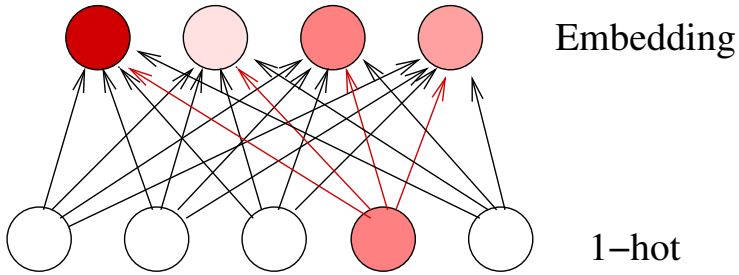
Möglichkeiten

1. binärer Vektor (1-hot-Repräsentation)
   - Beispiel: $w_i^{hot} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)$
   - Dimension = Vokabulargröße
     $\Rightarrow$ Alle Wörter sind sich gleich ähnlich/unähnlich

2. reeller Vektor (verteilte Repräsentation, distributed representation)
   - Beispiel: $w_i^{dist} = (3.275, -7.295, 0.174, 5.7332)$
   - Vektorlänge beliebig wählbar
   - Ziel: Ähnliche Wörter durch ähnliche Vektoren repräsentieren

Embeddings

Wir können die 1-hot-Repräsentation in einem neuronalen Netz auf die verteilte Repräsentation (Embedding) abbilden:



Da nur ein 1-hot-Neuron den Wert 1 und alle anderen den Wert 0 haben, ist die verteilte Repräsentation identisch zu den Gewichten des aktiven Neurons. (Die Aktivierungsfunktion ist hier die Identität $act(x) = x$)

Die Embeddings werden dann ebenfalls im Training gelernt.

Lookup-Tabelle

Wir können die verteilten Repräsentationen der Wörter zu einer Matrix L (Embedding-Matrix, Lookup-Tabelle) zusammenfassen. Multiplikation eines 1-hot-Vektors mit dieser Matrix liefert die verteilte Repräsentation.

$$\boldsymbol{w}^{dist} = L\boldsymbol{w}^{hot}$$

Matrix-Vektor-Multiplikation:

$$\begin{array}{cccccc} & & & & & 0 & \\ & & & & & 0 & \\ & & & & & 1 & \boldsymbol{w}^{hot} \\ & & & & & 0 & \\ & & & & & 0 & \end{array}$$

$$L \begin{pmatrix} 3.3 & 8.5 & 7.3 & 1.9 & -7.7 \\ 4.8 & 5.1 & 4.7 & -5.3 & 3.1 \\ -8.6 & 8.7 & 9.5 & 3.5 & 5.6 \\ 6.9 & 6.4 & -4.3 & 5.7 & 3.3 \\ 7.7 & 6.7 & -8.2 & 9.7 & -9.1 \end{pmatrix} \begin{pmatrix} 7.3 \\ 4.7 \\ 9.5 \\ -4.3 \\ -8.2 \end{pmatrix} \boldsymbol{w}^{dist}$$

Aktivierungsfunktionen

Neuron: $$f_k = act\left(\sum_i \theta_{ki} w_i\right) = act(net_k)$$

In der Ausgabeebene eines neuronalen Netzes wird (wie bei log-linearen Modellen) oft die Softmax-Aktivierungsfunktion verwendet.

$$c_k = softmax(net_k) = \frac{1}{Z} e^{net_k} \quad \text{mit } Z = \sum_{k'} e^{net_{k'}}$$

Spezialfall: 2 Klassen

SoftMax: $c_k = \frac{1}{Z} e^{net_k}$ mit $net_k = \sum_i \theta_{ki} f_i = \boldsymbol{\theta}_k^T \boldsymbol{f} = \boldsymbol{\theta}_k \cdot \boldsymbol{f}$

Im Falle von zwei Klassen c_1 und c_2 ergibt sich:

$$\begin{aligned} c_1 &= \frac{e^{\boldsymbol{\theta}_1 \cdot \boldsymbol{f}}}{e^{\boldsymbol{\theta}_1 \cdot \boldsymbol{f}} + e^{\boldsymbol{\theta}_2 \cdot \boldsymbol{f}}} \frac{e^{-\boldsymbol{\theta}_1 \cdot \boldsymbol{f}}}{e^{-\boldsymbol{\theta}_1 \cdot \boldsymbol{f}}} \\ &= \frac{1}{1 + e^{-(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) \cdot \boldsymbol{f}}} \end{aligned}$$

Nach einer Umparametrisierung erhalten wir die Sigmoid-Funktion

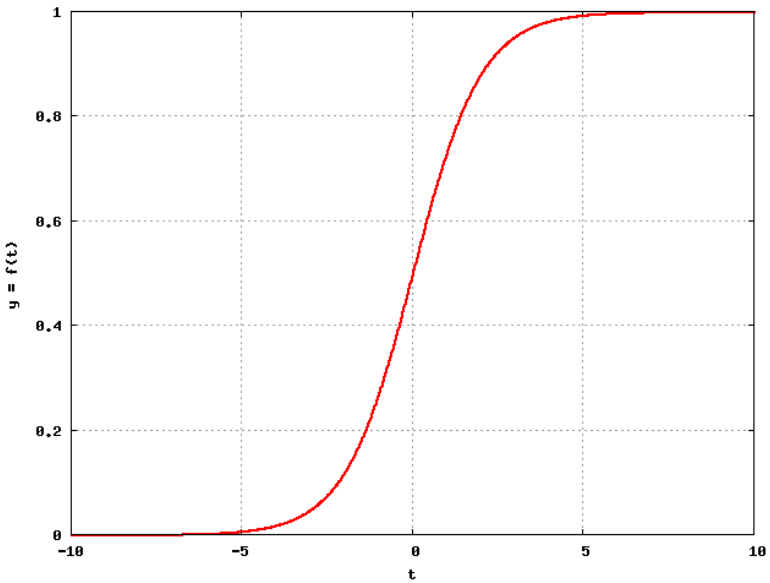
$$\begin{aligned} c_1 &= \frac{1}{1 + e^{-\theta f}} \quad \text{mit } \theta = \theta_1 - \theta_2 \\ c_2 &= 1 - c_1 \end{aligned}$$

$\Rightarrow$ Bei 2-Klassen-Problemen genügt ein Neuron.

Sigmoid-Funktion

Die Sigmoid-Funktion wird häufig als Aktivierungsfunktion der Neuronen in den versteckten Ebenen (hidden layers) genommen.

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



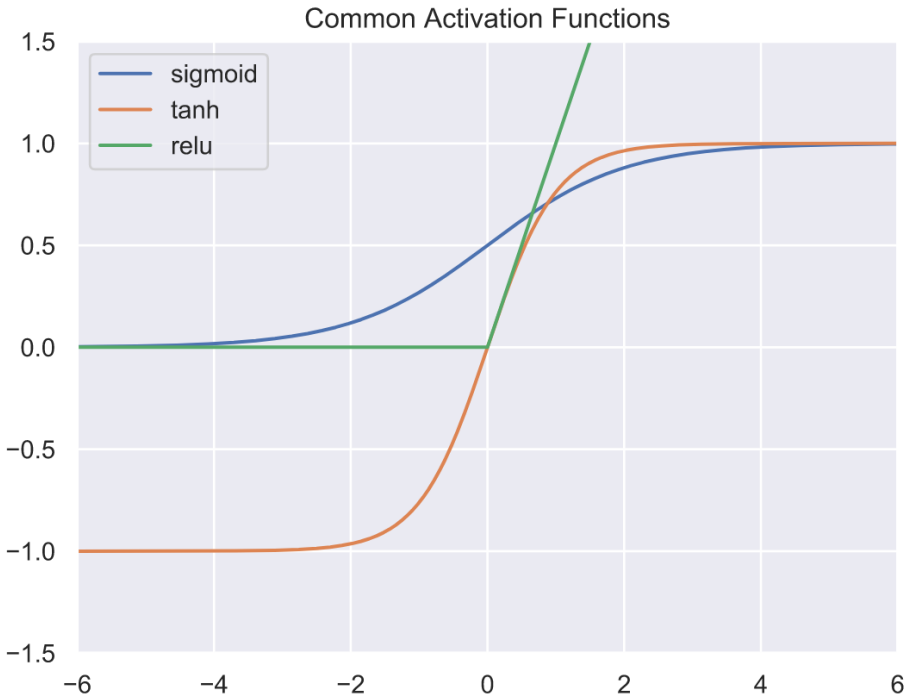
Man könnte also sagen, dass die versteckten Neuronen die Wahrscheinlichkeiten von binären Merkmalen berechnen.

Verschiedene Aktivierungsfunktionen

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

tanh: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$

ReLU: $ReLU(x) = max(0, x)$



Tangens Hyperbolicus tanh: skalierte und verschobene Sigmoidfunktion, oft besser als Sigmoidfunktion, liefert 0 als Ausgabe, wenn alle Eingaben 0 sind

ReLUs (Rectified Linear Units): einfach zu berechnen, bei 0 nicht differenzierbar.

Ableitungen der Aktivierungsfunktionen

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

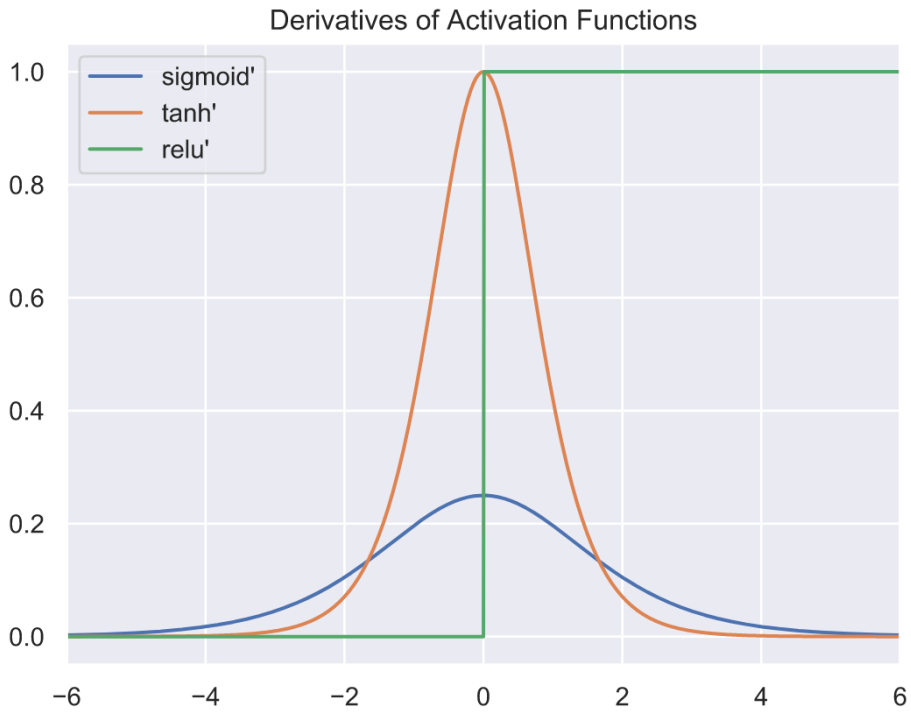
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

tanh: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$

$$tanh'(x) = 1 - \tanh(x)^2$$

ReLU: $ReLU(x) = max(0, x)$

$$ReLU'(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x < 0 \end{cases}$$



Warum ist die Aktivierungsfunktion wichtig?

Ohne (bzw. mit linearer) Aktivierungsfunktion gilt:

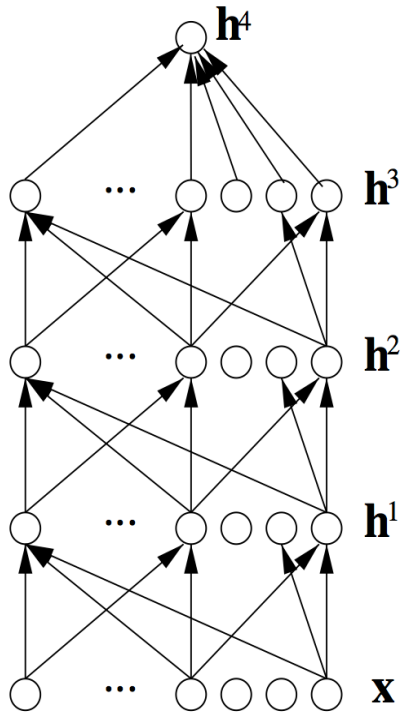
- Jede Ebene eines neuronalen Netzes führt eine lineare Transformation der Eingabe durch: $y = Wx$
- Eine Folge von linearen Transformationen kann aber immer durch eine einzige lineare Transformation ersetzt werden:

$$y = W_3 W_2 W_1 x = (W_3 W_2 W_1) x = Wx$$

$\Rightarrow$ Jedes mehrstufige neuronale Netz mit linearer Aktivierungsfunktion kann durch ein äquivalentes einstufiges Netz ersetzt werden.

$\Rightarrow$ Mehrstufige Netze machen also nur Sinn, wenn nicht-lineare Aktivierungsfunktionen verwendet werden.
Ausnahme: Embedding-Schicht mit gekoppelten Parametern

Neuronale Netze



Die Aktivierung a_k eines Neurons ist gleich der Summe der gewichteten Eingaben e_i moduliert durch eine Aktivierungsfunktion *act*.

$$a_k = act(\sum_i w_{ik} e_i + b_k)$$

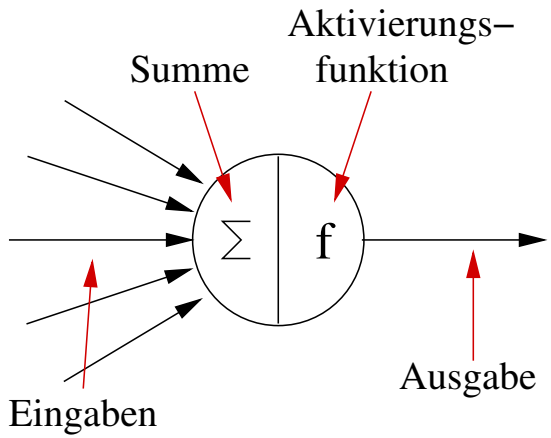
Der zusätzliche Biasterm b_k ermöglicht, dass bei einem Nullvektor als Eingabe die Ausgabe nicht-null ist.

Vektorschreibweise $a_k = act\left(\boldsymbol{w_k}^T \boldsymbol{e} + b_k\right)$
Matrixschreibweise $\boldsymbol{a} = act\left(W\boldsymbol{e} + \boldsymbol{b}\right)$

wobei die Funktion *act* elementweise angewendet wird:

$$act([z_1, z_2, z_3]) = [act(z_1), act(z_2), act(z_3)]$$

trainierbare Parameter: $W, \boldsymbol{b}$



Training von neuronalen Netzen

Neuronale Netzwerke werden (ähnlich wie CRFs) trainiert, indem eine Zielfunktion, welche die Qualität der aktuellen Ausgabe misst, optimiert (d.h. minimiert oder maximiert) wird.

Die Optimierung erfolgt mit stochastischem Gradientenabstieg oder -anstieg.

Dazu wird die Ableitung der Zielfunktion nach den Netzwerkparametern (Gewichte und Biaswerte) berechnet, mit der Lernrate multipliziert und zu den Parametern addiert (Gradientenanstieg) bzw. davon subtrahiert (Gradientenabstieg).

Für N Iterationen

$$\theta := \theta + \eta \nabla LL_\theta(D)$$

mit den trainierbaren Parametern θ, Lernrate η, Trainingsdaten D, Log-Likelihood $LL_\theta(D)$ und Gradient $\nabla LL_\theta(D)$

Zielfunktionen

Die verwendeten Ausgabefunktionen und zu optimierenden Zielfunktionen hängen von der Aufgabe ab.

- **Regression:** Ein Ausgabeneuron, welches eine Zahl liefert
  Beispiele: Vorhersage des Sentiments eines Textes auf einer Skala von 0 bis 10 oder Bewertung der Grammatikalität eines Satzes

  Aktivierungsfunktion der Ausgabeneuronen: $o = act(x) = e^x$
  Kostenfunktion: $(y - o)^2$ quadrierter Abstand zwischen gewünschter Ausgabe y und tatsächlicher Ausgabe o

Zielfunktionen

Die verwendeten Ausgabefunktionen und zu optimierenden Zielfunktionen hängen von der Aufgabe ab.

- **Regression:** Ein Ausgabeneuron, welches eine Zahl liefert
  Beispiele: Vorhersage des Sentiments eines Textes auf einer Skala von 0 bis 10 oder Bewertung der Grammatikalität eines Satzes

  Aktivierungsfunktion der Ausgabeneuronen: $o = act(x) = e^x$
  Kostenfunktion: $(y - o)^2$ quadrierter Abstand zwischen gewünschter Ausgabe y und tatsächlicher Ausgabe o

- **Klassifikation:** n mögliche disjunkte Ausgabeklassen repräsentiert durch je ein Ausgabeneuron
  Beispiel: Vorhersage der Wortart eines Wortes, oder des nächsten Wortes in einer Wortfolge

  Aktivierungsfunktion der Ausgabeneuronen: $o = softmax(x)$
  Kostenfunktion: $\log(y^T o)$ (Log-Likelihood, Crossentropie)

Deep Learning

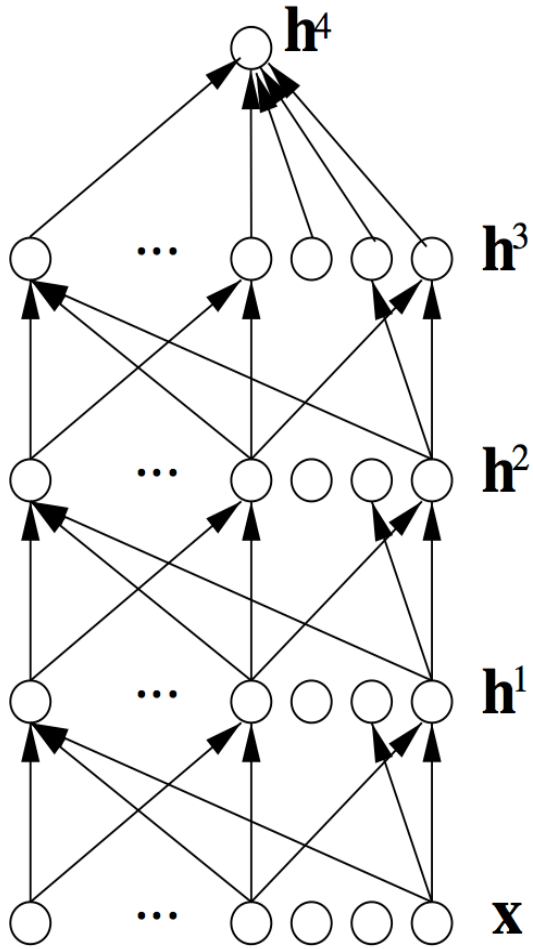
Neuronale Netze mit mehreren versteckten Ebenen nennt man tiefe Netze → Deep Learning.

Bei der Anwendung eines neuronalen Netzes werden die Eingabewerte **x** über die Neuronen der versteckten Ebenen $h_1, ..., h_3$ bis zur Ausgabeebene h_4 propagiert.

Je höher die versteckte Ebene, desto komplexer sind die darin repräsentierten Merkmale.

Im Training werden alle Parameter (Gewichte, Biases) gleichzeitig optimiert. Dazu wird der Gradient der Zielfunktion an den Ausgabeneuronen berechnet und bis zu den Eingabeneuronen zurückpropagiert.

→ Backpropagation



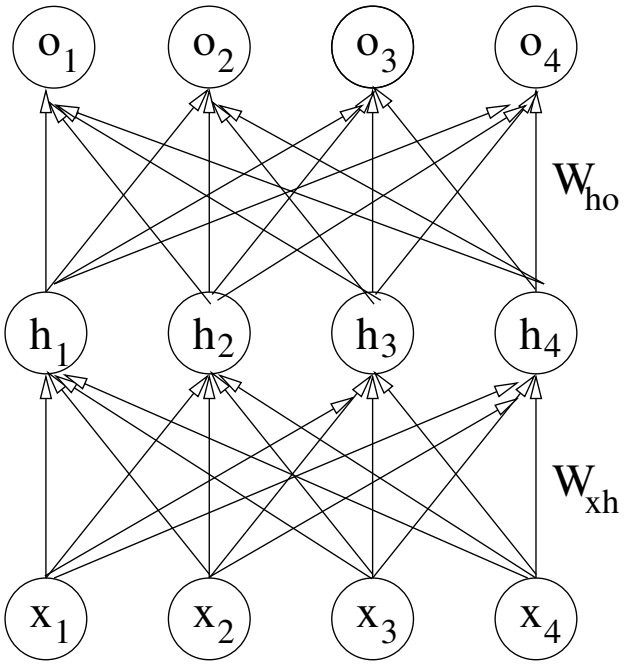
Arbeitsweise eines neuronalen Netzes

- Forward-Schritt: Die Aktivierung der Eingabeneuronen wird über die Neuronen der versteckten Ebenen zu den Neuronen der Ausgabeebene propagiert.
- Backward-Schritt: Im Training wird anschließend die zu optimierende Zielfunktion berechnet und deren zu den Eingabeneuronen zurückpropagiert.

Forward Propagation

Ein neuronales Netz mit Eingabevektor $\boldsymbol{x}$, verstecktem Vektor $\boldsymbol{h}$ und skalarem Ausgabewert o kann wie folgt definiert werden:

$$
\begin{aligned}
\boldsymbol{h}(\boldsymbol{x}) &= \tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) \\
\boldsymbol{o}(\boldsymbol{x}) &= \text{softmax}(W_{oh}\boldsymbol{h} + \boldsymbol{b}_o) \\
&= \text{softmax}(W_{oh}\tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o)
\end{aligned}
$$



Das Training maximiert die Daten-Log-Likelihood

$$
\begin{aligned}
LL(\boldsymbol{x}, \boldsymbol{y}) &= log(\boldsymbol{y}^T \boldsymbol{o}) \\
&= log(\boldsymbol{y}^T \text{softmax}(W_{oh}\tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o))
\end{aligned}
$$

$\boldsymbol{y}$ repräsentiert die gewünschte Ausgabe als 1-hot-Vektor.

Backward-Propagation

Für die Parameteroptimierung müssen die Ableitungen des Ausdruckes

$$LL(\boldsymbol{x}, \boldsymbol{y}) = log(\boldsymbol{y}^T \text{softmax}(W_{oh} \tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o))$$

bzgl.der Parameter W_{hx}, $\boldsymbol{b}_h$, W_{oh} und $\boldsymbol{b}_o$ berechnet werden.

Backward-Propagation

Für die Parameteroptimierung müssen die Ableitungen des Ausdruckes

$$LL(\boldsymbol{x}, \boldsymbol{y}) = log(\boldsymbol{y}^T \text{softmax}(W_{oh} \tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o))$$

bzgl.der Parameter W_{hx}, $\boldsymbol{b}_h$, W_{oh} und $\boldsymbol{b}_o$ berechnet werden.

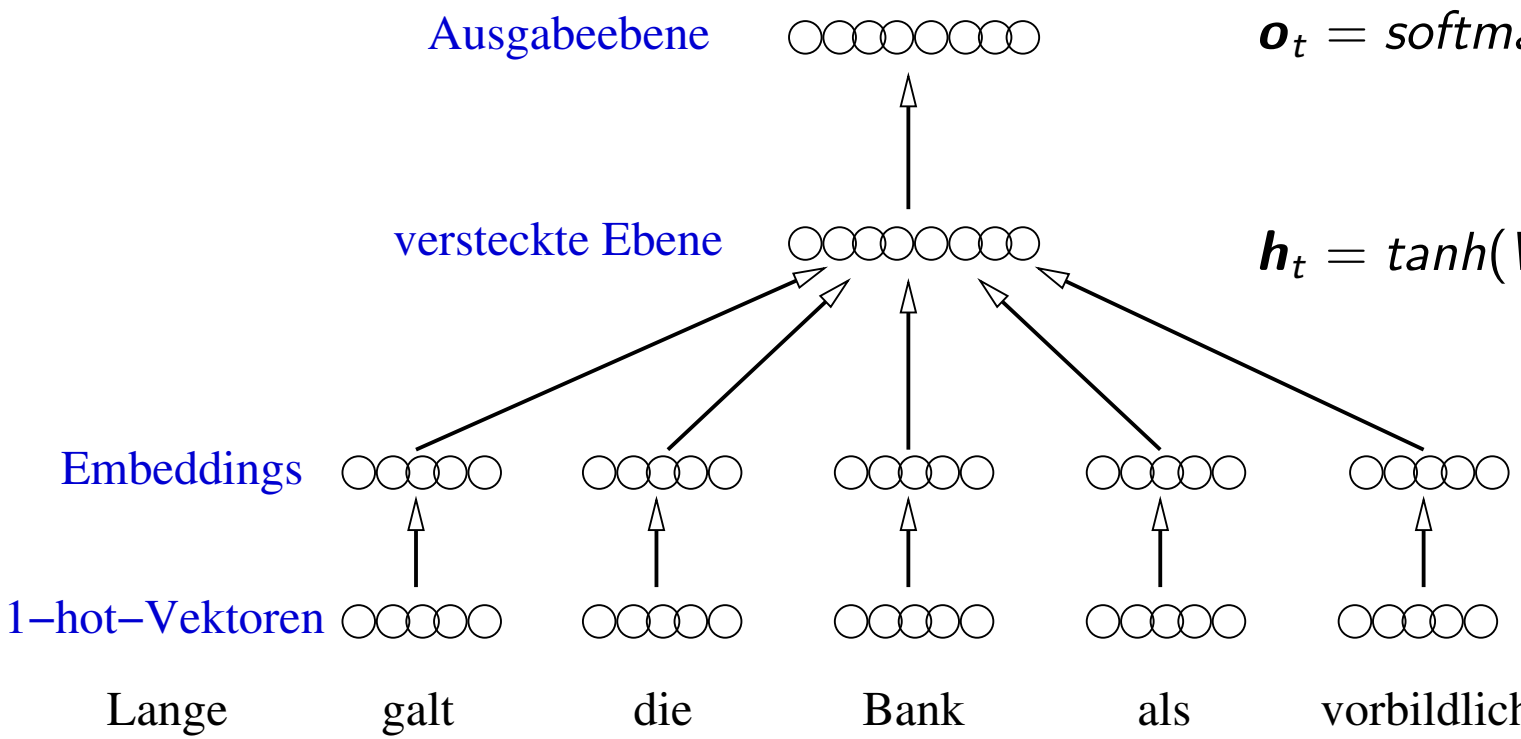
Am besten benutzt man dazu **automatische Differentiation**.

Anschließend Anpassung der Parameter θ mit Gradientenanstieg:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \nabla LL_{\boldsymbol{\theta}_t}$$

θ_t steht dabei für die trainierbaren Parameter W_{hx}, W_{oh}, b_h und b_o.

Einfaches neuronales Netz für Wortart-Annotation



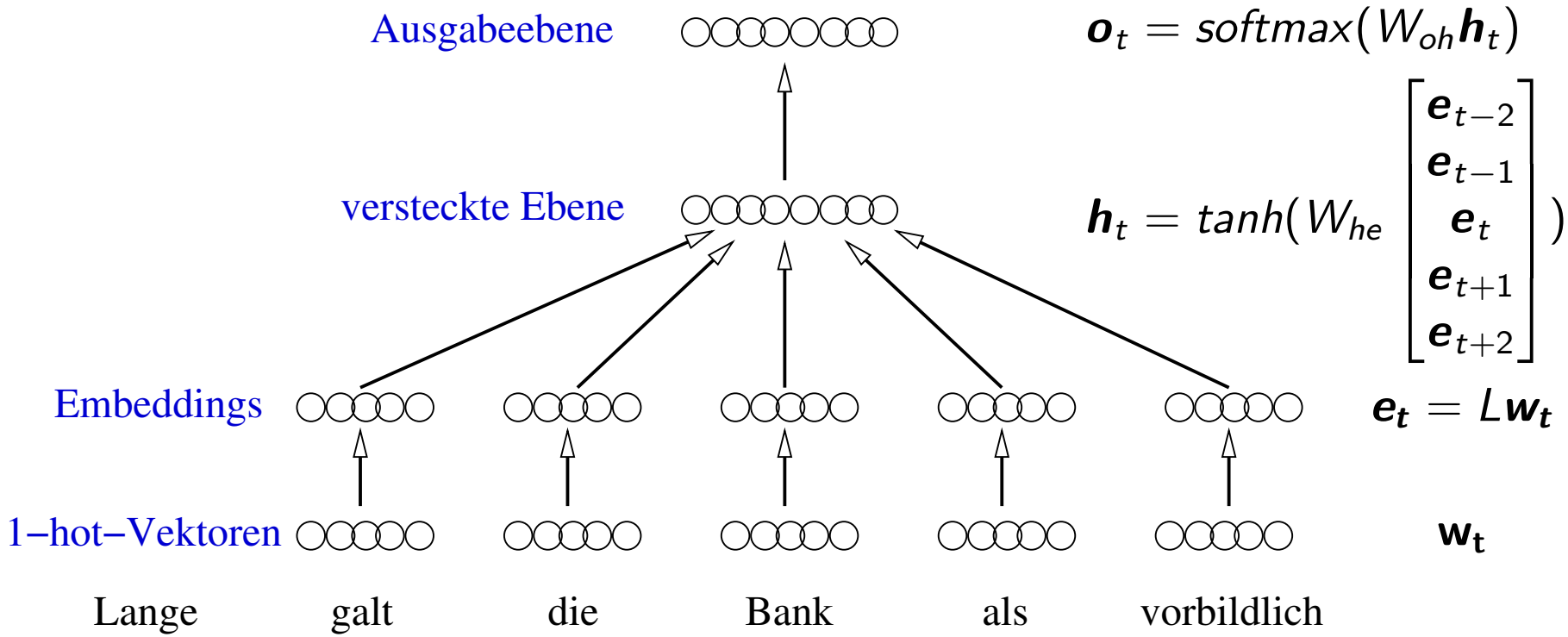
$$\boldsymbol{o}_t = softmax(W_{oh}\boldsymbol{h}_t)$$

$$\boldsymbol{h}_t = tanh(W_{he}\begin{bmatrix}\boldsymbol{e}_{t-2}\\ \boldsymbol{e}_{t-1}\\ \boldsymbol{e}_t\\ \boldsymbol{e}_{t+1}\\ \boldsymbol{e}_{t+2}\end{bmatrix})$$

$$\boldsymbol{e_t} = L\boldsymbol{w_t}$$

$$\mathbf{w_t}$$

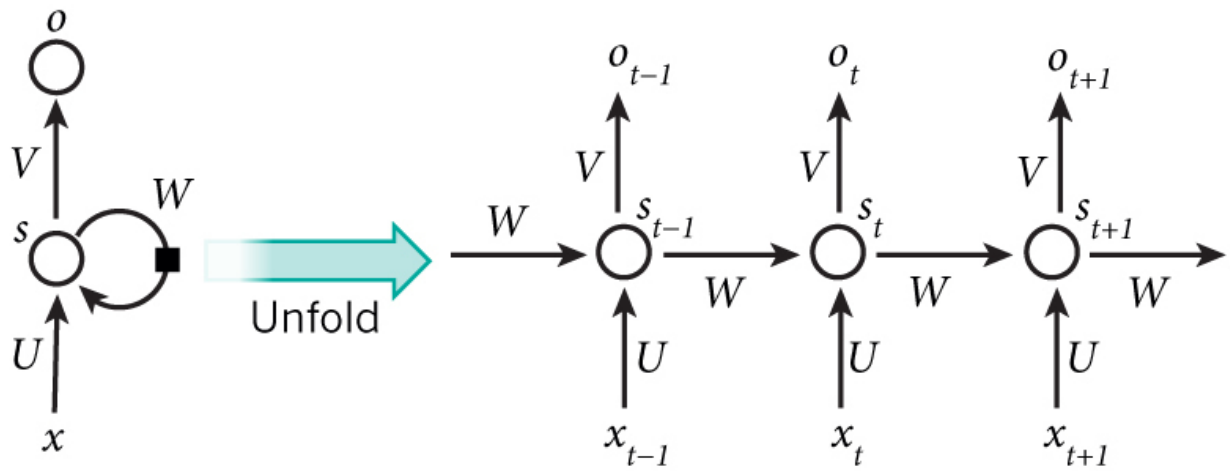
Einfaches neuronales Netz für Wortart-Annotation



"Feedforward"-Netze haben ein beschränktes Eingabefenster und können daher keine langen Abhängigkeiten erfassen.

→ rekurrente Netze

Rekurrente Neuronale Netze

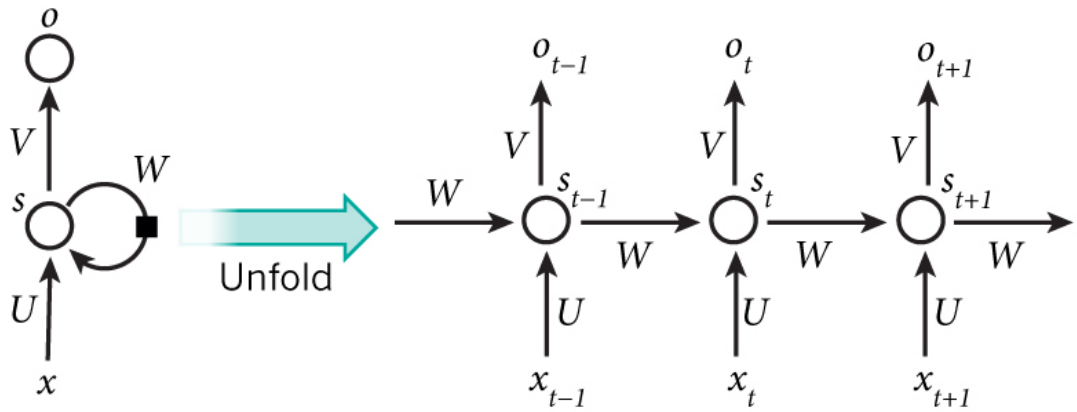


http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns

Ein RNN ist äquivalent zu einem sehr tiefen Feedforward-NN mit gekoppelten Gewichtsmatrizen.

Das neuronale Netz lernt, relevante Information über die Vorgängerwörter im Zustand s_t (= Aktivierungen der versteckten Ebene) zu speichern.

Rekurrentes neuronales Netz für Wortart-Annotation

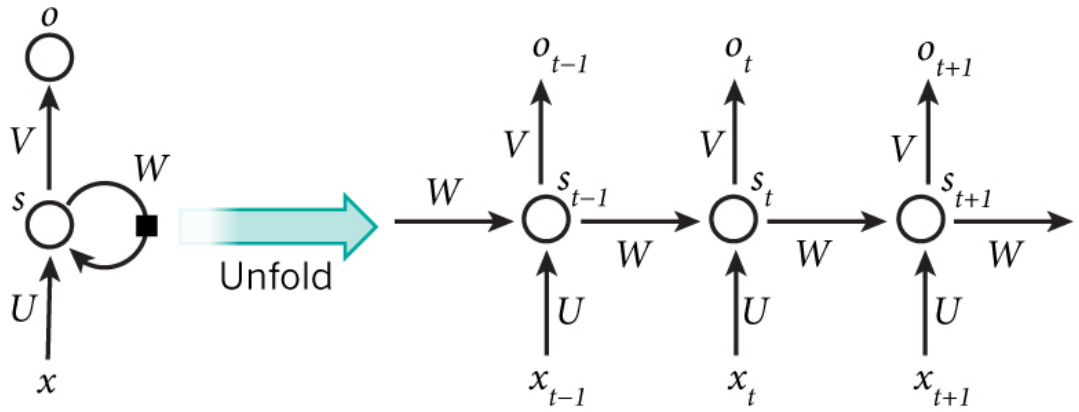


In jedem Schritt

- wird ein Wort x_t gelesen
- ein neuer Hidden State s_t berechnet
- eine Wahrscheinlichkeitsverteilung über mögliche Tags o_t ausgegeben

(Das Netzwerk hat noch keine Information über den rechten Kontext. Dazu später mehr.)

Backpropagation Through Time (BPTT)

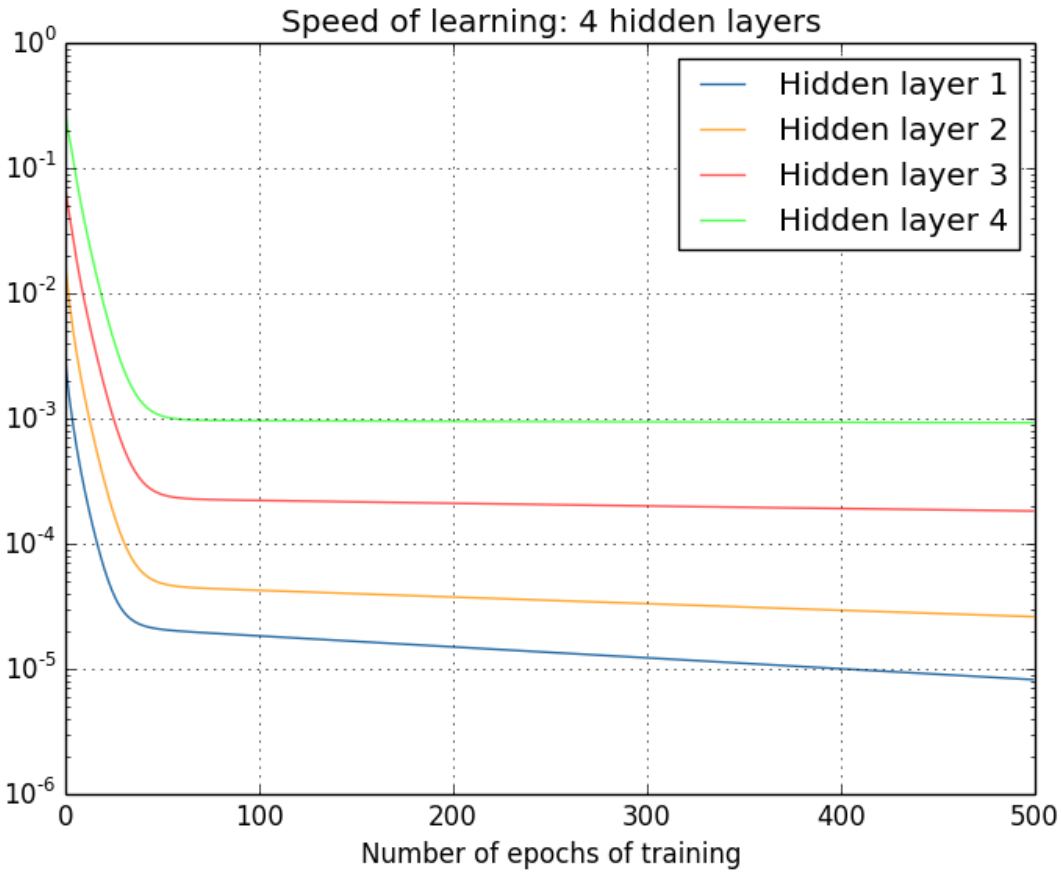


- Zum Trainieren wird das rekurrente Netz zu einem Feedforward-Netz aufgefaltet
- Der Gradient wird von den Ausgabeneuronen o_t zu den versteckten Neuronen s_t und von dort weiter zu den Eingabeneuronen x_t und den vorherigen versteckten Neuronen s_{t-1} propagiert

  $\Rightarrow$ Backpropagation through time
- An den versteckten Neuronen werden zwei Gradienten addiert.

Verschwindender/Explodierender Gradient

Das Training tiefer neuronaler Netze ist schwierig, weil der Gradient beim Zurückpropagieren meist schnell kleiner (oder größer) wird.



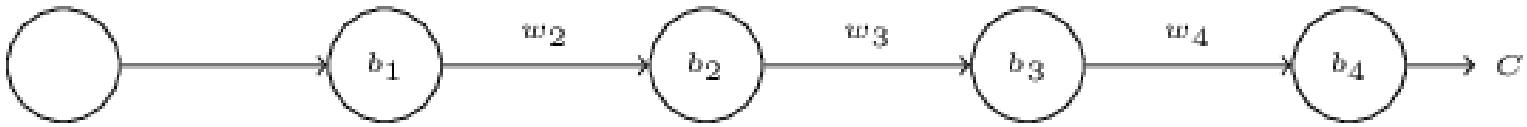
`http://neuralnetworksanddeeplearning.com/chap5.html`

Verschwindender/Explodierender Gradient

Warum wird der Gradient exponentiell kleiner mit der Zahl der Ebenen?

Betrachten wir ein neuronales Netz mit 5 Ebenen und einem Neuron pro Ebene

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



http://neuralnetworksanddeeplearning.com/chap5.html

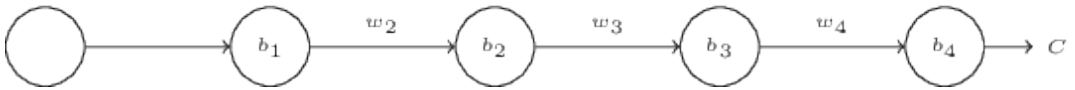
w_i ist ein Gewicht, b_i ein Bias, C die Kostenfunktion, a_i die Aktivierung eines Neurons, z_i die gewichtete Eingabe eines Neurons, σ' die Ableitung der Sigmoidfunktion

Der Gradient wird in jeder Ebene mit dem Ausdruck $w_i \times \sigma'(z_i)$ multipliziert.

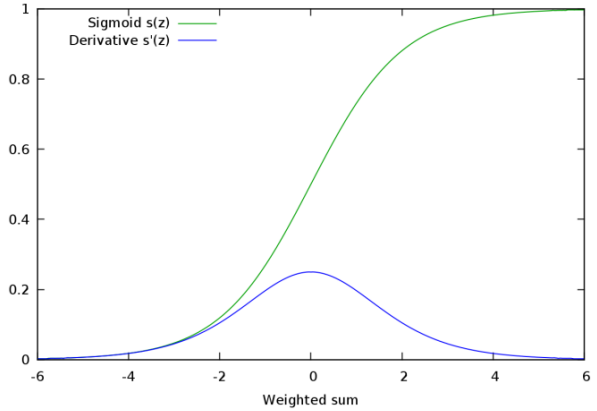
Wie groß ist dieser Ausdruck?

Verschwindender/Explodierender Gradient

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



http://neuralnetworksanddeeplearning.com/chap5.html



http://whiteboard.ping.se/MachineLearning/BackProp

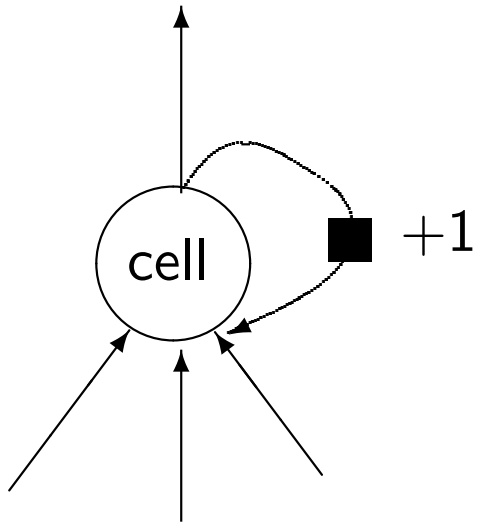
Wenn die Gewichte mit zufälligen Werten im Bereich [-1,1] initialisiert werden, dann gilt $|w_i \times \sigma'(z_i)| < 0.25$, da $|\sigma'(z_i)| < 0.25$.

Damit wird der Gradient exponentiell kleiner mit der Zahl der Ebenen.

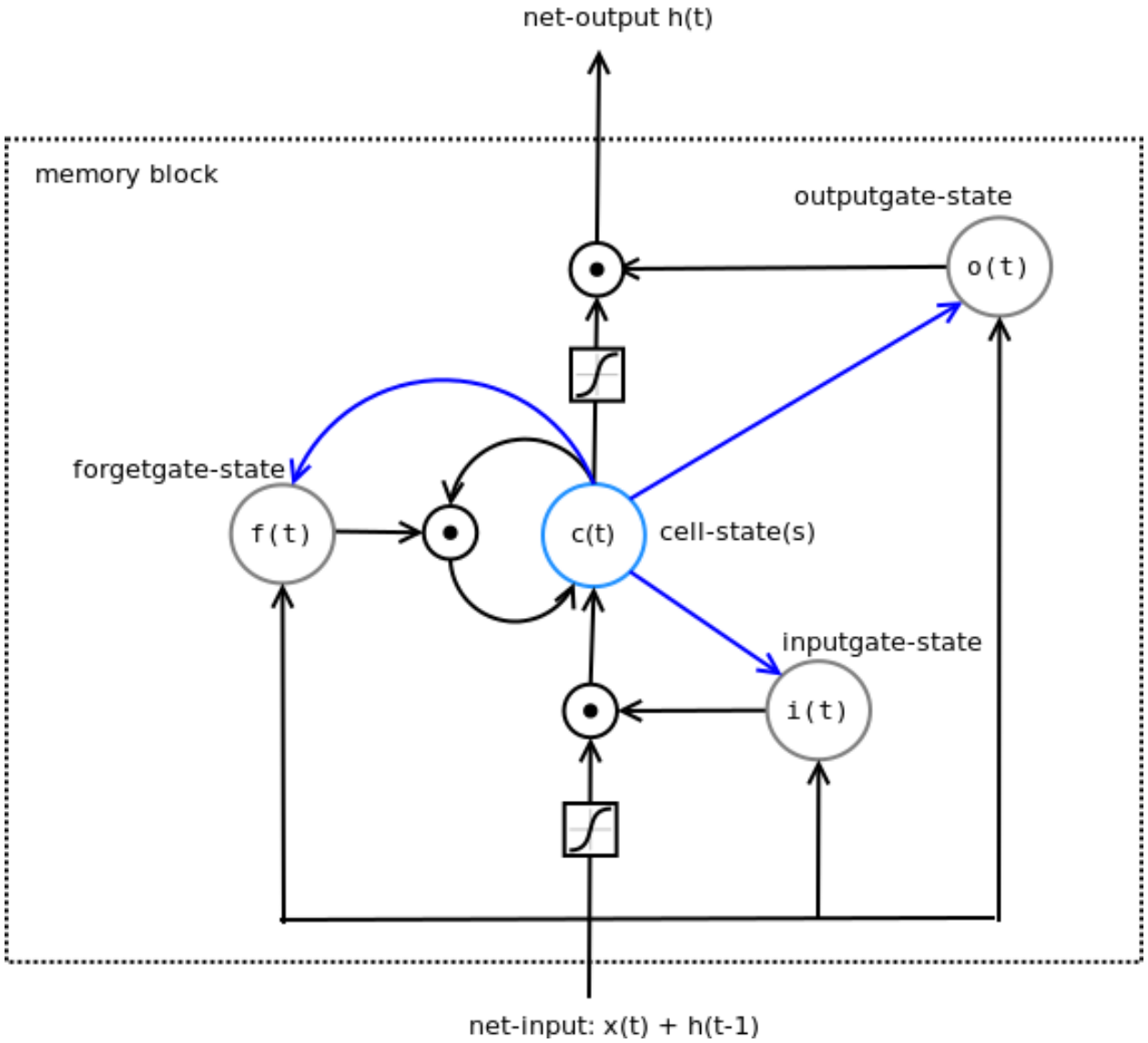
Wenn die Gewichte mit großen Werten initialisiert werden, kann der Gradient auch explodieren.

Long Short Term Memory

- entwickelt von Sepp Hochreiter und Jürgen Schmidhuber (TUM)
- löst das Problem der instabilen Gradienten für rekurrente Netze
- Eine Speicherzelle (cell) bewahrt den Wert des letzten Zeitschritts.
- Der Gradient wird beim Zurückpropagieren nicht mehr mit Gewichten multipliziert und bleibt über viele Zeitschritte erhalten.



Long Short Term Memory: Schaltplan



http://christianherta.de/lehre/dataScience/machineLearning/neuralNetworks/LSTM.php

Long Short Term Memory

Berechnung eines LSTMs (ohne Peephole-Verbindungen)

$$
\begin{aligned}
\mathbf{z_t} &= \tanh(W_z \mathbf{x_t} + R_z \mathbf{h_{t-1}} + \mathbf{b_z}) && \text{(input activation)} \\
\mathbf{i_t} &= \sigma(W_i \mathbf{x_t} + R_i \mathbf{h_{t-1}} + \mathbf{b_i}) && \text{(input gate)} \\
\mathbf{f_t} &= \sigma(W_f \mathbf{x_t} + R_f \mathbf{h_{t-1}} + \mathbf{b_f}) && \text{(forget gate)} \\
\mathbf{o_t} &= \sigma(W_x \mathbf{x_t} + R_o \mathbf{h_{t-1}} + \mathbf{b_o}) && \text{(output gate)} \\
\mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{z_t} && \text{(cell)} \\
\mathbf{h_t} &= \mathbf{o_t} \odot \mathit{tanh}(\mathbf{c_t}) && \text{(output activation)}
\end{aligned}
$$

mit $\mathbf{a} \odot \mathbf{b} = (a_1 b_1, a_2 b_2, ..., a_n b_n)$

Die LSTM-Zellen ersetzen einfache normale Neuronen in rekurrenten Netzen.

Man schreibt kurz:

$$Z = LSTM(X)$$

Long Short Term Memory

Vorteile

- löst das Problem mit instabilen Gradienten
- gute Ergebnisse in vielen Einsatzbereichen

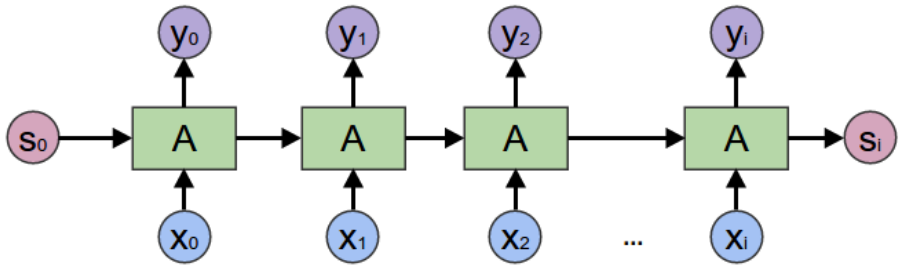
Nachteile

- deutlich komplexer als ein normales Neuron
- höherer Rechenzeitbedarf

Alternative

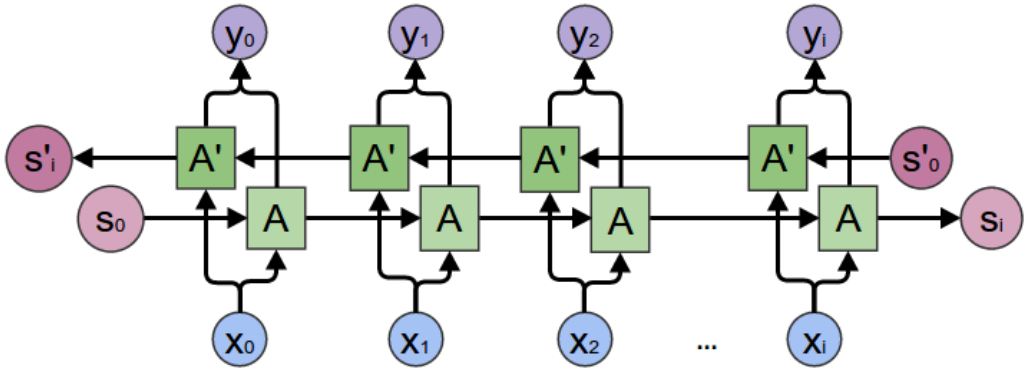
- Gated Recurrent Units (GRU) (von Cho et al.)
- etwas einfacher (nur 2 Gates)

Bidirektionale RNNs



colah.github.io/posts/2015-09-NN-Types-FP/img/RNN-general

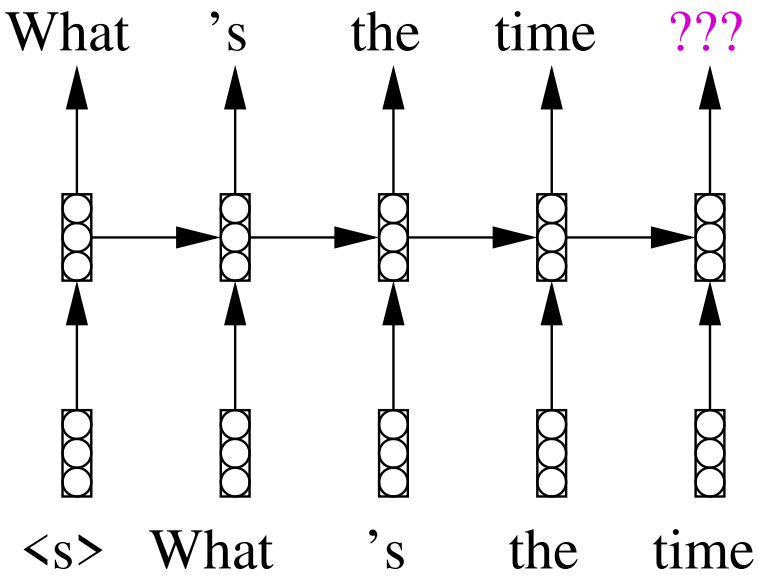
- Die rekurrenten Neuronen repräsentieren alle bisherigen Eingaben.
- Für viele Anwendungen ist aber auch eine Repräsentation der folgenden Eingaben nützlich. ⇒ bidirektionales RNN



colah.github.io/posts/2015-09-NN-Types-FP/img/RNN-bidirectional

- Bidirektionale RNNs können zu tiefen bidirektionalen RNNs gestapelt werden

LSTM-Sprachmodell



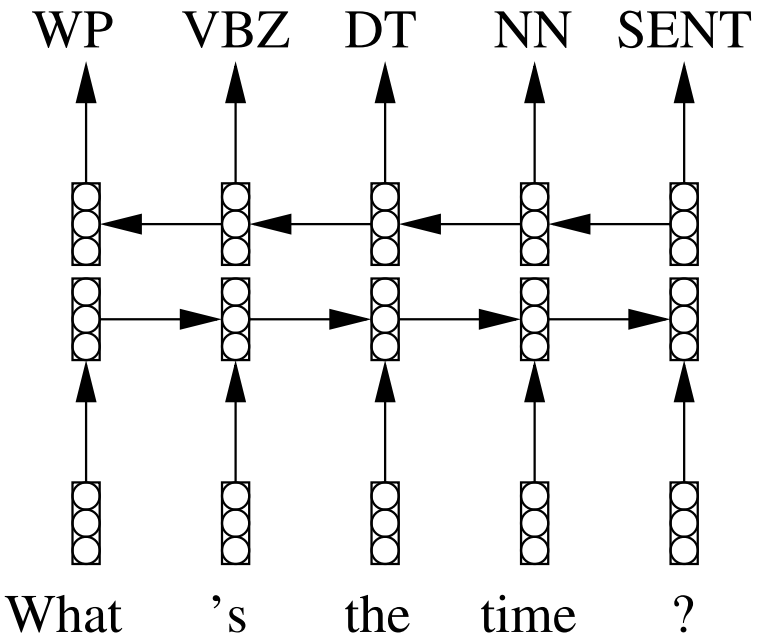
$$p(\mathbf{w}_{i+1}|\mathbf{w}_1^i) = \text{softmax}(W_{ho}\mathbf{h}_i)$$

$$(\mathbf{h}_i, \mathbf{c}_i) = \text{LSTM}(\mathbf{h}_{i-1}, \mathbf{c}_{i-1}, \mathbf{e}_i)$$

$$\mathbf{e}_i = \text{embedding}(w_i)$$

- Das LSTM berechnet eine Repräsentation $\mathbf{h_i}$ des bisherigen Satzes.
- Die Ausgabeebene berechnet aus dem aktuellen LSTM-Zustand $\mathbf{h_i}$ die Wahrscheinlichkeit $p(\mathbf{w}_{i+1}|\mathbf{w}_1^i)$, dass Wort w_{i+1} folgt.
- Im Gegensatz zum N-Gramm-Modell ist der Kontext nicht auf ein N-Gramm begrenzt.

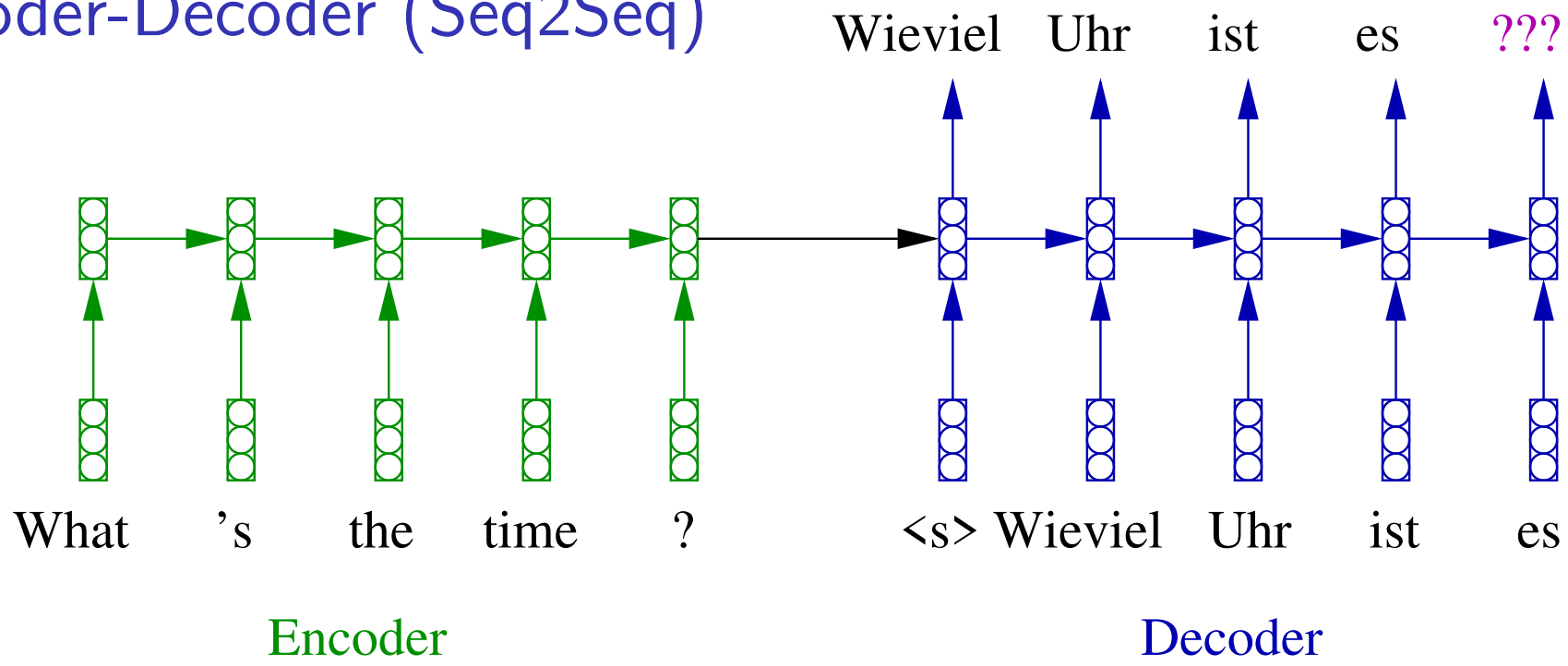
LSTM-Tagger



$$
\begin{aligned}
p(\mathbf{t}_i|\mathbf{w}_1^n) &= \text{softmax}(W_{ho}\mathbf{h}_i) \\
\mathbf{h}_i &= \vec{\mathbf{h}}_i \circ \overleftarrow{\mathbf{h}}_i \quad \text{(Konkatenation)} \\
(\vec{\mathbf{h}}_i, \vec{\mathbf{c}}_i) &= \text{LSTM}(\vec{\mathbf{h}}_{i-1}, \vec{\mathbf{c}}_{i-1}, \mathbf{e}_i) \\
(\overleftarrow{\mathbf{h}}_i, \overleftarrow{\mathbf{c}}_i) &= \text{LSTM}(\overleftarrow{\mathbf{h}}_{i+1}, \overleftarrow{\mathbf{c}}_{i+1}, \mathbf{e}_i) \\
\mathbf{e}_i &= \text{embedding}(w_i)
\end{aligned}
$$

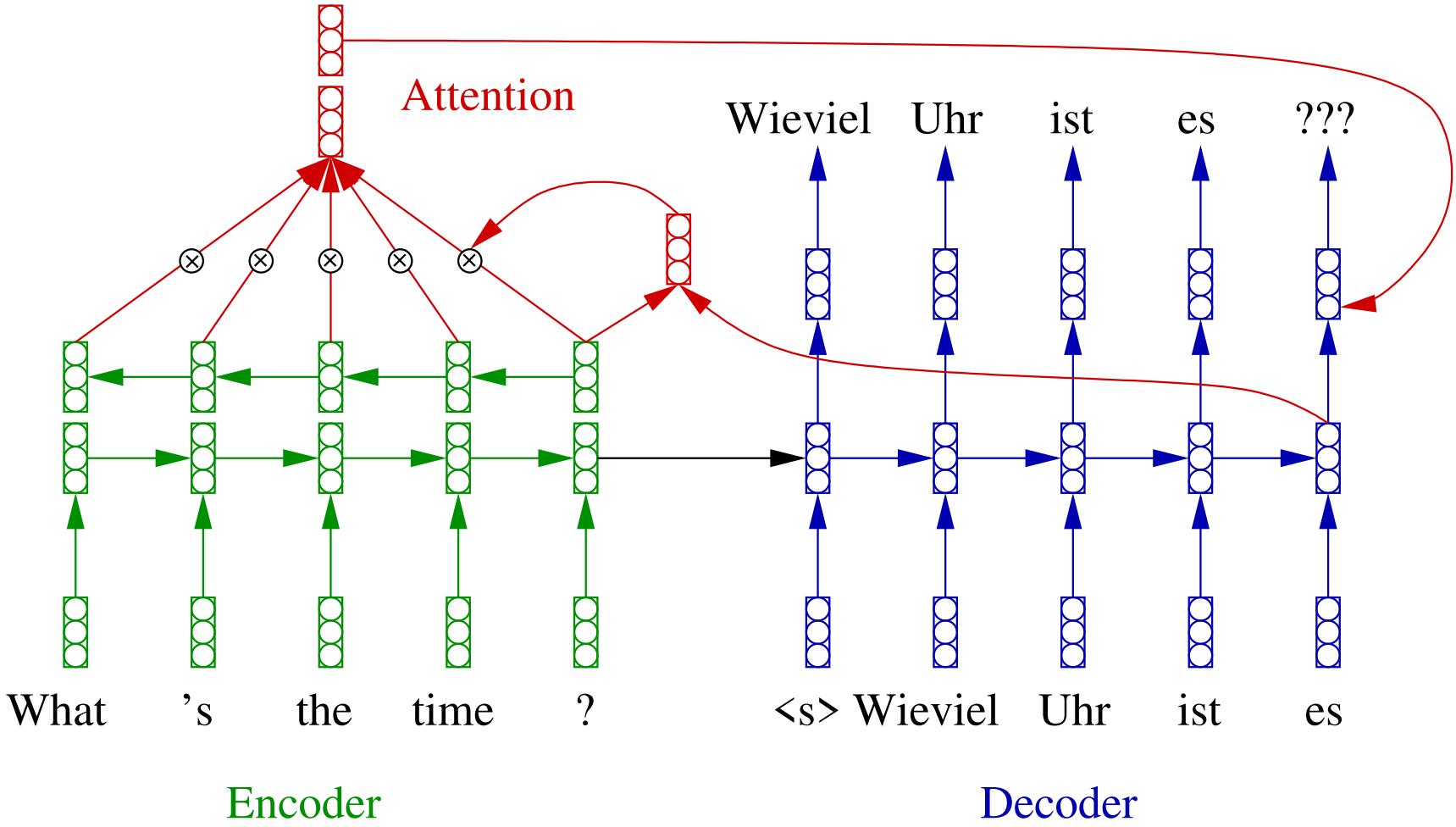
- Das bidirektionale LSTM berechnet eine kontextuelle Repräsentation für jedes Wort.
- Der BiLSTM-Tagger kann den gesamten Satzkontext berücksichtigen.

Encoder-Decoder (Seq2Seq)



- Der Encoder berechnet eine Repräsentation des ganzen Satzes.
- Der Decoder ist ein Sprachmodell, dessen Zustand mit der Encoder-Repräsentation initialisiert wird.
- Der Encoder verarbeitet die Eingabe oft in umgekehrter Reihenfolge, um das Training zu erleichtern. ("Wieviel ist dann näher bei "What".)
- **Problem:** Der Inhalt eines beliebig langen Satzes muss mit einem Vektor fester Länge repräsentiert werden.

Encoder-Decoder mit Attention



Attention-Mechanismus

- Der Encoder berechnet kontextuelle Repräsentationen für alle Wörter.
  $\Rightarrow$ Satzrepräsentation variabler Länge

- Das Attention-Modul erstellt für jeden Decoder-Zustand h_i^{dec} eine passende Zusammenfassung c_i der Eingabesatz-Repräsentation h^{enc}.

$$
\begin{aligned}
c_i &= \sum_j \alpha_{ij} h_j^{enc} \\
\alpha_i &= \text{softmax}(\mathbf{a}_i) \\
a_{ij} &= v_a \cdot \tanh(W_a(h_i^{dec} \circ h_j^{enc}))
\end{aligned}
$$

  v_a und W_a sind trainierbare Parameter.

- Es gibt auch andere Möglichkeiten, die Attention-Scores a_{ij} zu berechnen. Bspw.:

$$a_{ij} = h_i^{dec} \cdot W_a h_j^{enc}$$

Decoder-Ausgabeebene

- Die Ausgabewahrscheinlichkeiten $\mathbf{o}_{i+1}$ werden aus dem aktuellen Decoderzustand h_i^{dec}, dem Embedding e_i des letzten Ausgabesymbols y_i und dem Kontextvektor c_i berechnet.

$$\mathbf{o}_{i+1} \quad = \quad \text{softmax}(W_o(h_i^{dec} \circ c_i \circ e_i))$$

- Auch für die Berechnung der Ausgabewahrscheinlichkeiten gibt es noch viele andere Möglichkeiten.

Training

- Im Training wird die Log-Likelihood der Daten D maximiert:

$$LL(D) = \sum_{\mathbf{x},\mathbf{y} \in D} \sum_{i} log(y_i \cdot o_i(\mathbf{x}))$$

  wobei y_i das 1-hot-Encoding des i-ten Ausgabewortes ist.

- Diese Trainingsmethode wird als Teacher Forcing bezeichnet.
  Die Decoderzustände werden hier mit der korrekten Folge der Ausgabewörter berechnet.
- Beim Übersetzen wird dagegen die Wortfolge genommen, die der Decoder für die wahrscheinlichste hält.
- Für diese Situation wurde der Decoder eigentlich gar nicht trainiert.
  $\Rightarrow$ Exposure Bias
  $\Rightarrow$ Scheduled Sampling, Minimum Risk Training

NMT Decoding

Greedy Decoding

- Der Decoder verfolgt nur eine Übersetzungs-Hypothese und wählt in jedem Schritt das wahrscheinlichste Wort.

Beam Search

- Der Decoder verfolgt mehrere Übersetzungshypothesen parallel.
- In jedem Schritt werden für jede Hypothese die k besten nächsten Wörter bestimmt.
- Dann werden alle neuen, erweiterten Hypothesen verglichen und die n wahrscheinlichsten als neue Hypothesenmenge gewählt.
- typische Beamgröße: circa 5

Ensembles

- Je nach Initialisierung bekommt man im Training ein anderes NMT-System.
- Jedes so trainierte NMT-System stellt einen eigenen Experten dar.
- Man kann die Meinungen dieser Experten kombinieren
  bspw. durch Mittelung der Wortwahrscheinlichkeiten.
- Man spricht dann von einem Ensemble.
- Mit Hilfe von Ensembles kann die Übersetzungsgenauigkeit oft um mehrere BLEU-Punkte gesteigert werden.
- Der Rechenzeitbedarf steigt linear mit der Ensemblegröße.
- typische Ensemblegrößen: 4-8

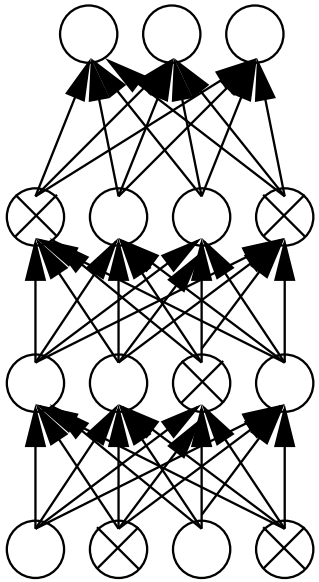
Overfitting: Regularisierung

Wie alle Klassifikatoren haben auch neuronale Netze Probleme mit Overfitting.

Lösung 1: Regularisierung der Gewichte

- neue Zielfunktion bei L1-Regularisierung: $LL_\theta(D) - |\theta|$
- neue Zielfunktion bei L2-Regularisierung: $LL_\theta(D) - \theta^2$

Overfitting: Dropout



Lösung 2: Dropout von Neuronen

- Die Aktivierung jedes einzelnen Neurons wird im Training zufällig mit einer Wahrscheinlichkeit von p auf 0 gesetzt.
- Im Decoding sind alle Neuronen aktiv. Zum Ausgleich dafür, dass nun die Summe der gewichteten Eingabeaktivierungen höher ist, werden die Aktivierungen mit (1-p) multipliziert.
- In der Ausgabeebene wird kein Dropout angewendet.

Dropout

Dropout ist die wirksamste Methode, um Overfitting bei neuronalen Netzen zu verhindern.

Erklärungsversuche, warum Dropout funktioniert:

- Dropout unterstützt, dass die von Neuronen repräsentierten Merkmale unabhängig voneinander sind.
- Das Netzwerk kann als eine Art Ensemble von exponentiell vielen Einzelnetzwerken interpretiert werden, die parallel trainiert werden.

Unbekannte Wörter

NMT-Systeme haben Probleme mit unbekannten Wörtern:

Beispiel: Kirschkuchen und Himbeertorte waren in den Trainingsdaten.

Nun soll Kirschtorte übersetzt werden. Das System weiß nichts über das Wort.

Mögliche Lösung: Pretraining der Wortembeddings auf monolingualen Daten mit einem Sprachmodell-System

Aber:

- Unbekannte Wörter gibt es auch auf der Ausgabeseite.
- Große Vokabulare erhöhen den Speicherplatzbedarf und machen NMT-Systeme langsam (wegen Softmax in Ausgabeebene).

Byte-Pair Encoding

- Bei Komposita wie Kirschtorte ist es naheliegend, sie in die Glieder (hier Kirsch und Torte) aufzuspalten.
- Dazu braucht man aber eine Morphologie.
- Byte-Pair-Encoding (BPE) (von Rico Sennrich) ist eine Alternative:
  - Spalte alle Wörter in einzelne Buchstaben auf.
  - Zähle, wie oft jedes Buchstabenpaar miteinander auftaucht.
  - Für N Iterationen
    - ★ Fasse das häufigste Buchstabenpaar in allen Wörtern zu einer Einheit zusammen.
    - ★ Aktualisiere die Paar-Häufigkeiten.
      (Nun sind es nicht mehr nur Buchstabenpaare.)
- Beispiel: reitet, reicht, riecht

Byte-Pair Encoding

- Die Größe des BPE-Vokabulars wird von der Zahl der Merge-Operationen bestimmt.
- BPE wird von den meisten NMT-Systemen verwendet.
- Ähnliche Methode: Word Pieces (Google)

Backtranslation

- Es gibt viel mehr monolinguale Daten als bilinguale Daten.
- Idee: halb überwachtes Training mit monolingualen Daten
  - Trainiere ein Target-Source-System auf den bilingualen Daten.
  - Übersetze damit monolinguale Targetdaten. (Backtranslation)
  - Füge die neu erzeugten Satzpaare zu den bilingualen Daten hinzu.
  - Trainiere ein Source-Target-System auf den kombinierten Daten.
- Man kann die Methode auch iterieren und zwei Systeme für Vorwärts- und Rückwärtsübersetzung abwechselnd auf der Kombination von biligualen Daten und frisch erzeugten Übersetzungen des anderen Systems trainieren.

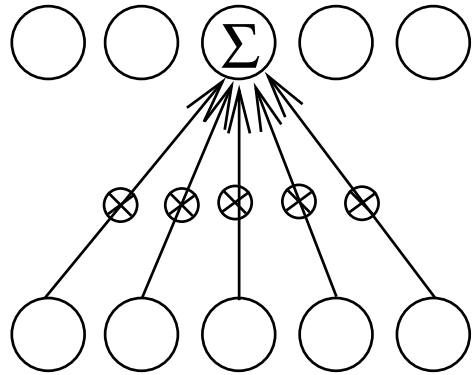
Transformer

- NMT-Systeme werden oft mit LSTMs implementiert.
- LSTMs haben aber den Nachteil, dass sie die Wortfolge sequentiell verarbeiten.
- Transformer können die Wörter parallel verarbeiten.
- Sie ersetzen RNNs durch Self-Attention.

Transformer: Überblick

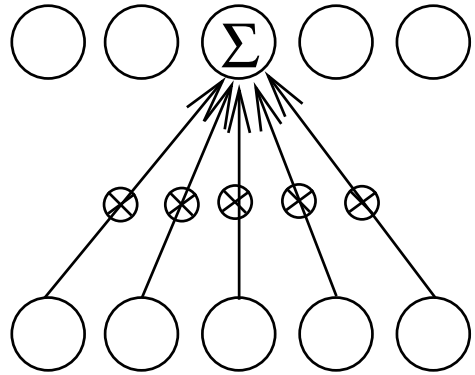
- Self Attention
- Positional Encodings
- Scaled Dot-Product Attention
- Multi-Head Attention
- Transformer-Architektur
- Residual Connections
- Layer Normalization

Transformer: Self Attention



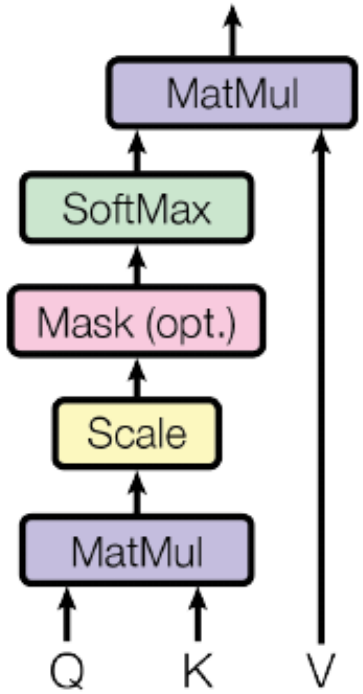
- "Normale" Attention berechnet eine Attention-Funktion über die Eingabepositionen für einen gegebenen Decoder-Zustand.
- Self-Attention berechnet eine Attention-Funktion über die Eingabepositionen für jeden einzelnen Encoder-Zustand.
- Jeder Zustand der Transformer-Ebene ist ein (mit Attention) gewichtetes Mittel der Zustände der darunterliegenden Ebene.
- Die Eingabe der ersten Transformerebene bildet die Folge der Wort-Embeddings.

Positional Encodings



- Self-Attention hat den Vorteil, dass alle Positionen direkt miteinander interagieren können.
- Self-Attention weiß aber im Gegensatz zu RNNs nichts über die Reihenfolge der Wörter. (Bag of Words)
- **Lösung:** Zu den Wort-Embeddings werden Positional Encodings addiert.
- Für jede absolute Position wird ein Encoding trainiert.

Scaled Dot-Product Attention



Eine effizient berechenbare Attention-Funktion

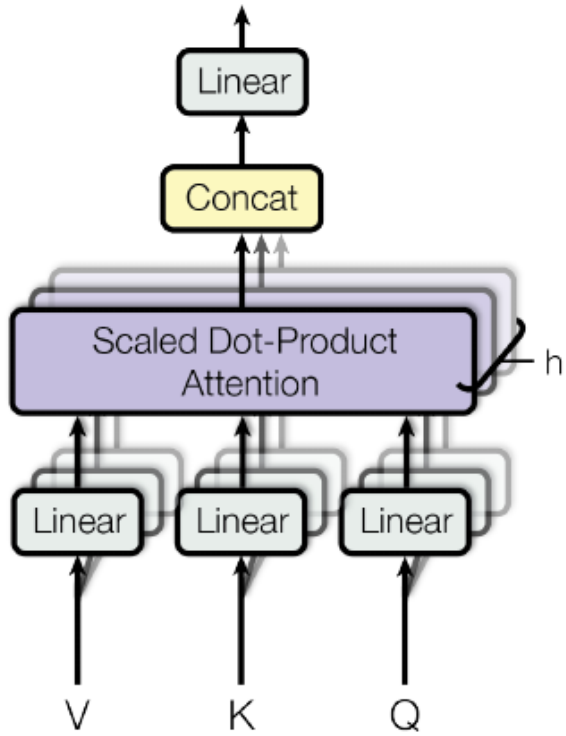
- Die Folge der Eingabezustände wird zu einer Matrix M zusammengefasst.
- Durch Multiplikation mit drei Matrizen wird M in eine Query-Matrix Q, eine Key-Matrix K und eine Value-Matrix V transformiert.
- Mit den drei Matrizen wird dann das Attention-Ergebnis berechnet:

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Der Scaling-Faktor $1/\sqrt{d_k}$ verhindert, dass durch die Summation so große Werte entstehen, dass die Aktivierungsfunktion (bspw. tanh) sättigt und der Gradient minimal wird. d_k ist der Länge der Query/Key-Vektoren.

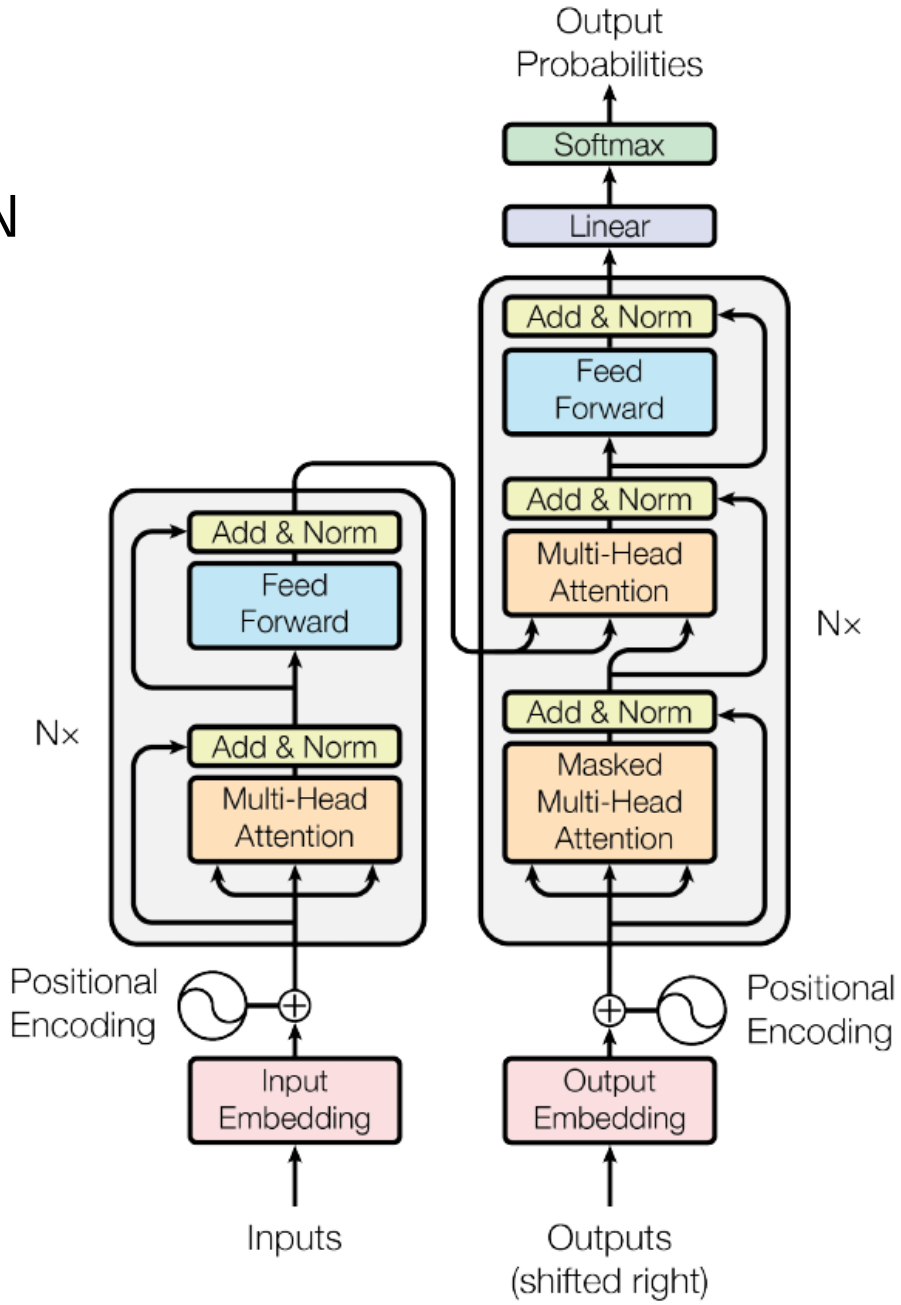
Multi-Head Attention

- Transformer wenden Attention mehrmals parallel an und konkatenieren die Ergebnisse.
- Der Ergebnisvektor wird dann noch mit einer trainierbaren Matrix transformiert.
- Jedes Attention-Modul kann sich hier auf andere Aspekte der Eingabe konzentrieren.
- Oft werden 8 Attention-Module verwendet.



Transformer-Architektur

- Encoder und Decoder werden mit je N Transformer-Ebenen implementiert.
- Jede Transformer-Ebene besteht aus
  - Multi-Head Self-Attention
  - Residual Connection
  - Layer Normalization
  - Feed-Forward-Ebene
  - Residual Connection
  - Layer Normalization
- Jede Decoder-Ebene hat zusätzlich
  - Encoder Attention
  - Eingabe-Maskierung

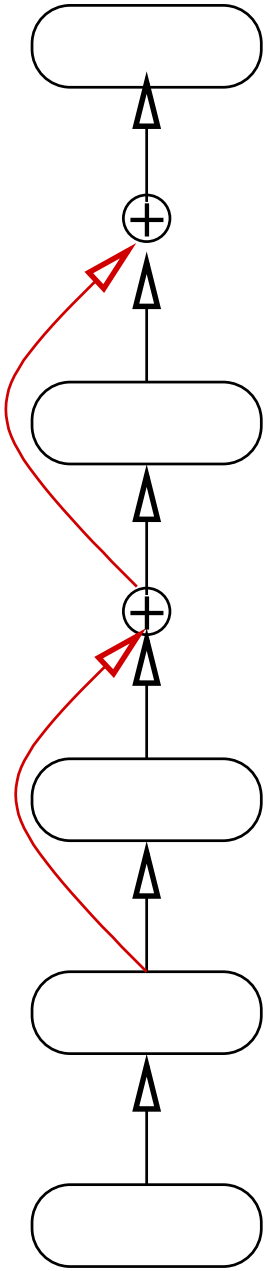


Transformer: Maskierung

- Der Decoder darf nicht in die Zukunft schauen
  (auch wenn im Training der ganze Ausgabesatz bekannt ist)
- Daher wird die Attention nur über die aktuelle Position und die vorherigen Positionen berechnet.
- Die zukünftigen Positionen werden maskiert.

Residual Connections

- Tiefe neuronale Netzwerke sind wegen des Vanishing-Gradient-Problems schwierig zu trainieren.
- Residual Connections reduzieren dieses Problem:
  - Alle Ebenen haben gleich viele Neuronen.
  - Bei jeder Ebene (außer der ersten) wird die Eingabe zur Ausgabe addiert.
  - Über diese zusätzlichen Verbindungen kann der Gradient besser zurückpropagiert werden.



Layer Normalization

Wenn die Aktivierungen der Neuronen einer Ebene groß sind, besteht die Gefahr, dass die Aktivierungen der Neuronen der nächsten Ebene gesättigt sind, was zu kleinen Gradienten führt und das Training erschwert.

Lösung: Layer Normalization

- Von den Aktivierungen der Neuronen wird ihr Mittelwert μ abgezogen.
- Dann werden sie durch die Standardabweichung σ geteilt.
- Schließlich wird noch ein trainierbarer Gain-Parameter g_i multipliziert..

$$\hat{a}_i = g_i \frac{a_i - \mu}{\sigma} \quad \text{mit} \quad \mu = \frac{1}{H}\sum_{i=1}^{H} a_i, \quad \sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i - \mu)^2}$$

Transformer

Vorteile

+ besser parallelisierbar als LSTM-basierte Systeme
+ bessere Ergebnisse

Nachteile

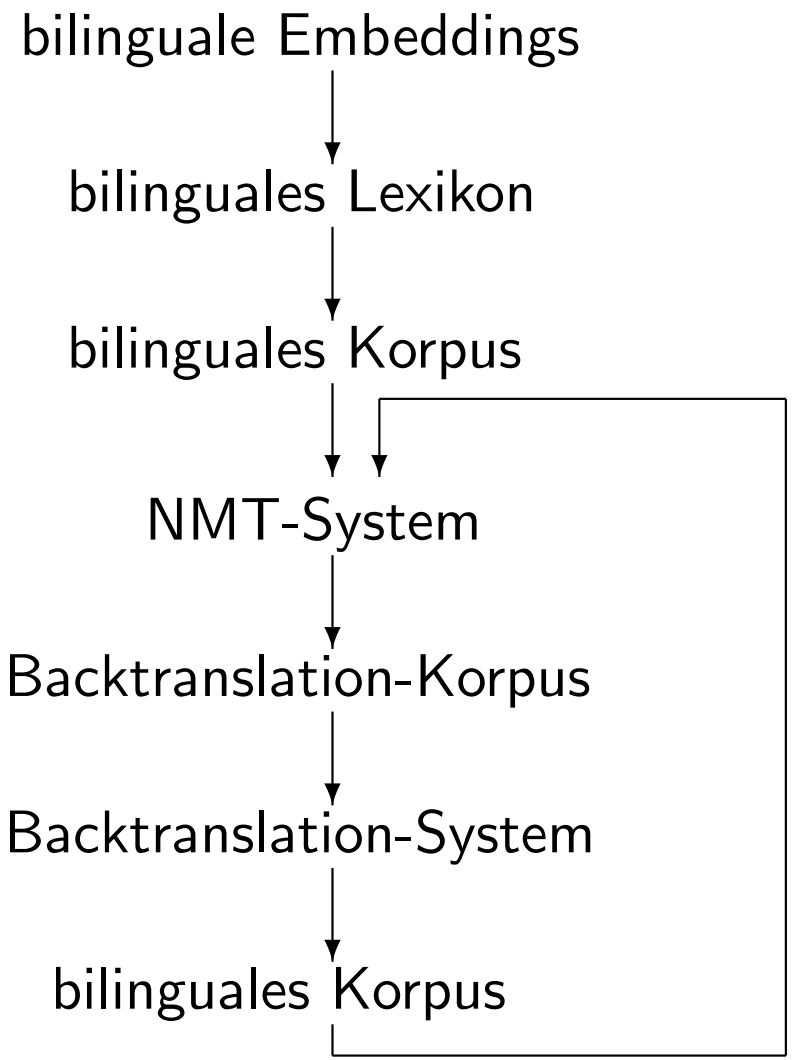
– komplexe Architektur
– speicherhungrig
– Ohne Tricks wie Layer Normalization und Residual Connections funktionieren Transformer nicht.
– Diese Tricks können auch bei LSTM-basierten Systemen die Genauigkeit verbessern.

Maschinelle Übersetzung ohne bilinguale Daten

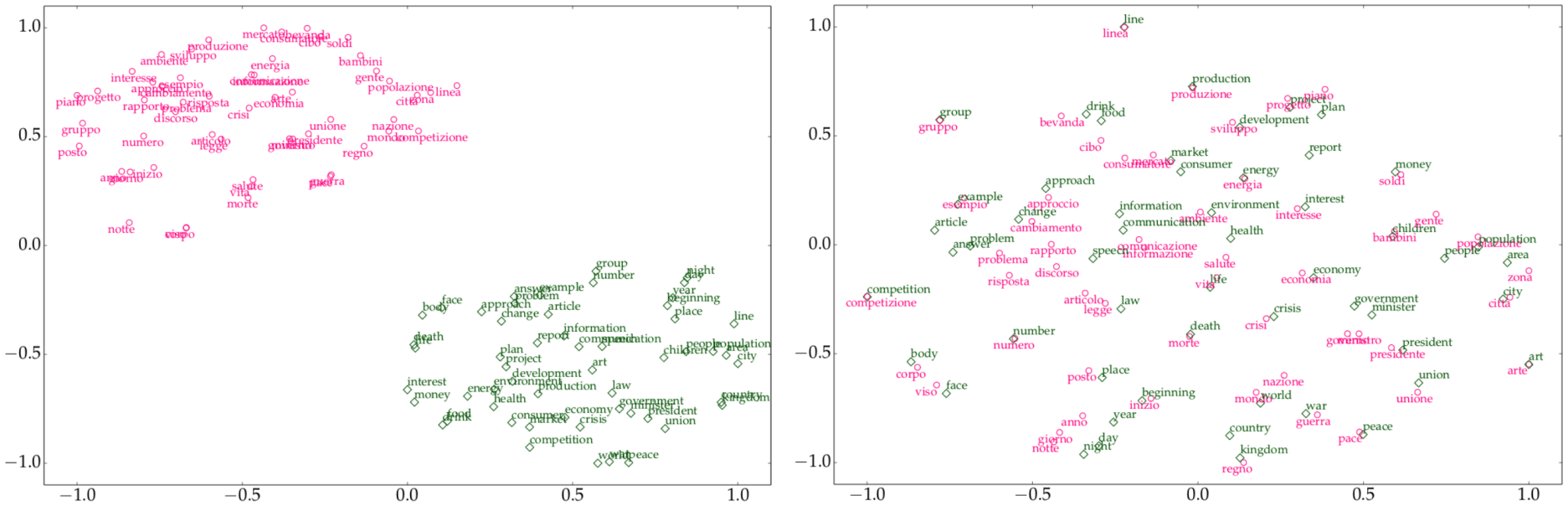
- Übersetzungssysteme werden meist auf Korpora mit vielen Millionen Wörtern trainiert.
- Für die meisten Sprachpaare stehen bilinguale Korpora dieser Größe nicht zur Verfügung.

- Könnte man einen Übersetzer unüberwacht nur mit monolingualen Korpora trainieren?

Unsupervised Neural Machine Translation



Bilinguale Embeddings

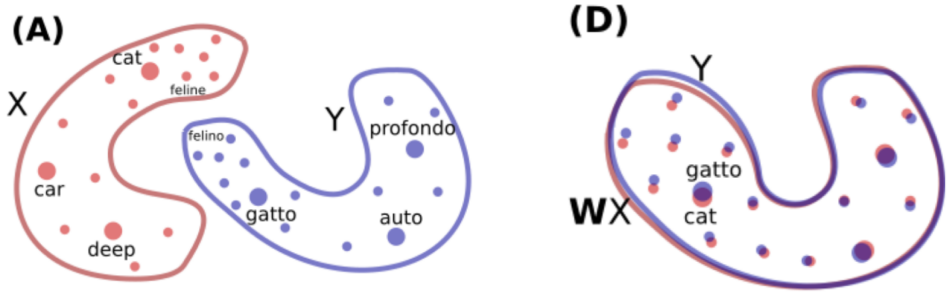


Quelle: `https://arxiv.org/pdf/1706.04902.pdf`

Ziele:

- Ähnliche Wörter sollen ähnliche Embeddings bekommen.
- Wörter und ihre Übersetzung sollen ähnliche Embeddings bekommen.

Bilinguale Embeddings



Quelle: https://arxiv.org/pdf/1710.04087.pdf

Mapping-Strategie

gegeben: zwei monolinguale Korpora, ein kleines bilinguales Lexikon L

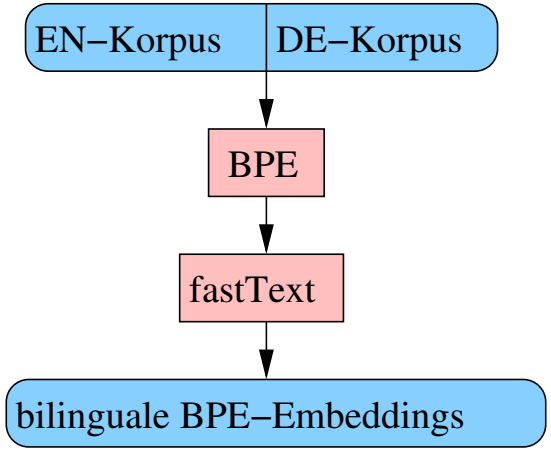
1. Embeddings für die Wörter der Sprache A auf einem monolingualem Korpus lernen (z.B. mit SkipGram oder FastText)
2. Analog Embeddings für die Wörter der Sprache B lernen
3. Embeddings der Sprache A durch Multiplikation mit einer Matrix W so transformieren, dass sich die Embeddings von Wörtern a und ihren Übersetzungen b möglichst ähnlich sind.

$$\hat{W} = \arg\min_W \sum_{(a,b)\in L} ||We_a - e_b||^2$$

Die Lösung kann mit einer Procrustes-Analyse berechnet werden.

Meist wird für W eine Orthogonalmatrix (mit $WW^T = W^TW = I$) genommen, weil dann die Abstände und Winkel erhalten bleiben.

Bilinguale Embeddings



Direkte Strategie (Lample, Ott et al.)

- Konkatenation der monolingualen Korpora
- Segmentierung der Wörter im neuen Korpus mit Byte-Pair-Encoding
- Lernen von BPE-Embeddings mit SkipGram oder FastText

- Die Methode funktioniert gut bei ähnlichen Sprachen mit vielen gemeinsamen BPEs.
- Die gemeinsamen BPEs bewirken eine Alignierung der Embedding-Räume.
- Hier wird kein bilinguales Lexikon benötigt.

Bilinguales Lexikon

Mit Hilfe der bilingualen Embeddings können bilinguale Wörterbücher erstellt werden.

Strategie

- Berechne für alle bilingualen Wort-Paare (a, b) die Ähnlichkeit ihrer bilingualen Embeddings $\text{sim}(\mathbf{e_a}, \mathbf{e_b})$.
- Berechne für jedes Wort a seine beste Übersetzung $\hat{b}$:

$$\hat{b} = \arg\max_{b \in V} \text{sim}(\mathbf{e_a}, \mathbf{e_b})$$

V = Vokabular der Zielsprache

Wie wird das Ähnlichkeitsmaß sim definiert?

Ähnlichkeitsmaße für Embeddings

- Kosinus-Ähnlichkeit

$$\text{sim}(x, y) = \frac{x \cdot y}{||x||_2 \, ||y||_2}$$

- Cross-Domain Similarity Local Scaling (CSLS)
  - reduziert Probleme mit Hubs und Anti-Hubs

Unüberwachtes Training von NMT-Systemen

- Bilinguale Embeddings werden trainiert.
- Daraus wird ein bilinguales Lexikon extrahiert.
- Damit wird ein bilinguales Korpus erzeugt.

  Zielsprachliche Sätze werden Wort für Wort mit dem Lexikon übersetzt.
- Auf diesem Korpus wird ein NMT-System trainiert.

Problem:

- Die synthetisierten quellsprachlichen Sätze haben die falsche (nämlich die zielsprachliche) Wortfolge.
- Das NMT-System lernt nur monotone Wort-für-Wort-Übersetzungen.
- Echte quellsprachliche Sätze werden schlecht übersetzt.

“Verrauschung” der quellsprachlichen Sätze

- Das NMT-System soll lernen, die korrekte Wortfolge zu generieren.
- Dazu werden die Wort-zu-Wortübersetzungen modifiziert durch
  - zufälliges Weglassen einzelner Wörter
  - zufälliges Umordnen der Wörter
    (innerhalb einer Maximaldistanz)
- Beispiel: er hat ein spannendes Buch gelesen

  he has a exciting book read

  he a book read exciting

“Verrauschung” der quellsprachlichen Sätze

Ergebnissatzpaar:

he a book read exciting ⇒ er hat ein spannendes Buch gelesen

- Durch Training auf den verrauschten Daten lernt das NMT-System
  - die korrekte zielsprachliche Wortfolge zu rekonstruieren und
  - fehlende (Funktions-)Wörter einzufügen
- Wichtig: Der Zielsatz muss ein korrekter zielsprachlicher Satz sein!
- Die Qualität des Quellsatzes ist weniger kritisch.

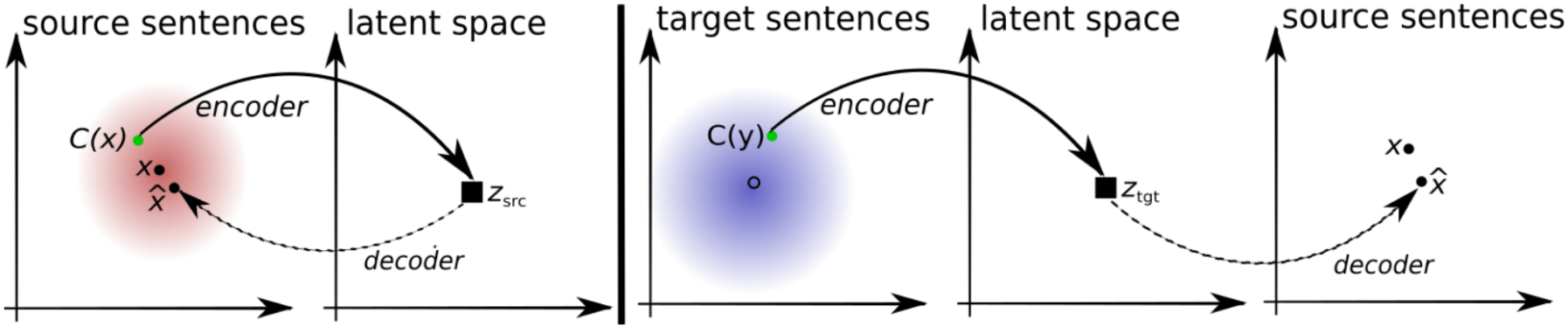
Iteration

- Mit dem trainierten NMT-System wird nun ein quellsprachliches Korpus übersetzt.
- Darauf wird ein NMT-System für die Rückwärtsübersetzung trainiert.
- Vorwärts- und Rückwärtsübersetzungen werden mit demselben System generiert.
- Das Starttoken (z.B. <english>) teilt dem Decoder die Zielsprache mit.

Problem: Der Encoder sieht nur rückübersetzte Sätze, die sich deutlich von echten quellsprachlichen Sätzen unterscheiden.

$\Rightarrow$ Der Encoder sollte auch auf echten Sätzen trainiert werden.

Denoising Autoencoding



Quelle: `https://arxiv.org/pdf/1711.00043.pdf`

- Das NMT-System wird zusätzlich darauf trainiert, aus verrauschten quellsprachlichen Sätzen die ursprünglichen quellsprachlichen Sätze zu rekonstruieren.
- Zum Verrauschen werden dieselben Operationen wie bei der Rückübersetzung verwendet.

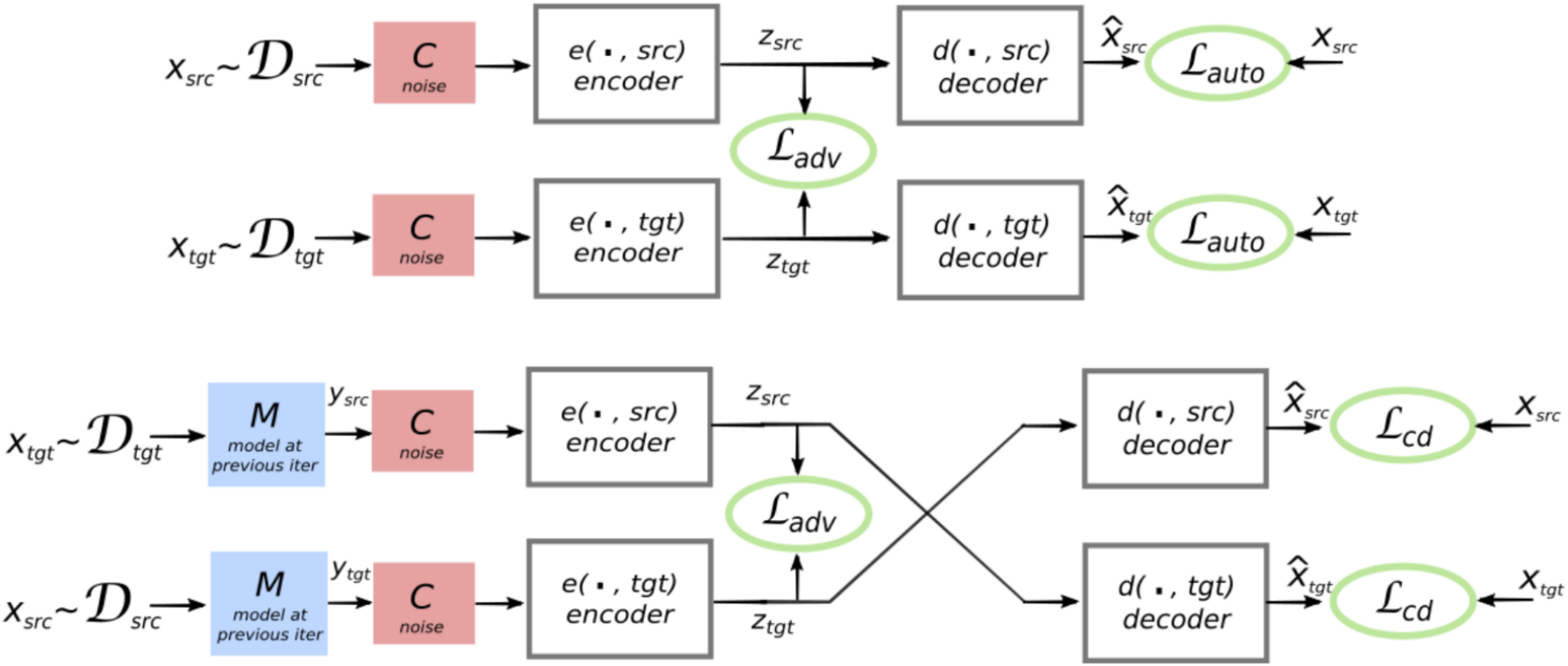
Problem: Damit der Decoder die Encoder-Repräsentationen für echte Sätze übersetzen kann, dürfen sich diese nicht von den Encoder-Repräsentationen für rückübersetzte Sätze unterscheiden.

Adversarial Networks

Ziel: Die Encoder-Repräsentationen für verrauschte echte Sätze und verrauschte Rückübersetzungen sollen ununterscheidbar werden.

- Es wird ein Gegner-Netz (Adversarial Network) trainiert, welches lernt, verrauschte echte Sätze und verrauschte Rückübersetzungen zu unterscheiden.
- Der Encoder wird trainiert, das Gegner-Netz auszutricksen, indem er die Repräsentationen der unterschiedlichen Eingabesätze ununterscheidbar macht.

Loss-Funktionen



Quelle: `https://arxiv.org/pdf/1711.00043.pdf`

- $x_{src} \sim \mathcal{D}_{src}$ wählt ein Element des quellsprachlichen monolingualen Korpus
- $e(\cdot, src)$ berechnet die Encoder-Repräsentationen z_{src}
- $d(\cdot, src)$ dekodiert den Zielsatz
- $\mathcal{L}_{cd}, \mathcal{L}_{auto}, \mathcal{L}_{adv}$: Losses für Übersetzung, Autoencoding, Gegner-Netzwerk

Pseudocode für unüberwachtes NMT-Training

Training($\mathcal{D}_{src}, \mathcal{D}_{tgt}, T$)

bilinguales Lexikon aus monolingualen Daten extrahieren (nach Conneau et al.)

$M^{(1)}$:= unüberwachte Wort-zu-Wort-Übersetzung mit dem bilingualen Lexikon

for t = 1 to T **do**

beide monolingualen Korpora mit $M^{(t)}$ übersetzen

// Training des Gegen-Netzwerkes und des NMT-Modelles

$(\theta_{enc}, \theta_{dec}, \theta_{discr}) = \theta = \arg\min_{\theta'} \mathcal{L}(\mathcal{D}, \theta')$

$M^{(t+1)} := e(\theta_{enc}) \circ d(\theta_{dec})$ // update MT model

return $M^{(T+1)}$

$\mathcal{D}$ steht für alle Daten $(\mathcal{D}_{src}, \mathcal{D}_{tgt})$

$\mathcal{L}(\mathcal{D}, \theta')$ ist die Summe der einzelnen Losses

Zusammenfassung

Wir haben gesehen, wie man

- bilinguale Embeddings lernt
- mit Hilfe der Embeddings ein Übersetzungslexikon erstellt
- mit dem Lexikon ein verrauschtes Übersetzungskorpus generiert
- mit dem Korpus einen Übersetzer trainiert
- einen Übersetzer und Rückübersetzer parallel trainiert
- mit Autoencoding die Ergebnisse verbessert
- mit Adversarial Training die Representationen angleicht

Vielen Dank!