Machine Learning for Natural Language Processing

Michael Gabler

6. August 2019

Inhaltsverzeichnis

1 Grundlagen

Neuronales Netzwerk ist Funktion, die auf Eingabedaten angewendet wird.
Optimierung durch Minimierung der Loss-Funktion
Loss-Funktion Maß, wie gut das Netzwerk Vorhersagen trifft. Berechnet sich aus Vorhersage und tatsächlichen Werten (Ground Truth).

- Euklidischer Loss, Mean-Squared-Error: $l_2 = \frac{1}{2N} \sum_i (f_\theta(x_i) - t_i)^2$
- Negative-Log-Likelihood, Cross-Entropy: $NLL = -\frac{1}{|D|} \sum_i \log[f_\theta(x_i)|_{t_i}]$

Konfusionsmatrix welche Klassen werden wie oft mit welcher Klasse verwechselt?

1.1 Training

Daten werden aufgeteilt in Train/Validierung/Test (z.B. 60/20/20)
Datenaugmentierung Generieren zusätzlicher Daten (z.B. bei Bildern) durch Spiegelung, Rotation, Skalierung, Anpassung Helligkeit und Farbe, etc.
Epoche Verarbeitung aller Trainingsdaten
Iteration Verarbeitung eines Batches
Batch Mehrere Trainingsbeispiele werden gerechnet bevor Gewichte einmal geupdated werden (z.B. 10 Beispiele pro Batch)
Learning Rate Faktor η, wie stark das Netzwerk durch die Deltas verändert werden soll (d.h. wie schnell es lernt bzw. seine Meinung ändert). Wird beim Update der Gewichte verwendet. **Evaluation auf Validierungsdaten** zur Anpassung der Hyperparameter (Learning-Rate, Netzstruktur, ...)
Evaluation auf Testdaten einmalig, um Genauigkeit des trainierten Netzes zu ermitteln
Forward-Pass Berechnen des Outputs des Netzwerkes für bestimmte Eingabedaten (z.B. ein Batch)
Backward-Pass Bilden der partiellen Ableitung für jeden Input in jedem Layer und Speichern der Werte als Deltas

Backpropagation am Beispiel

$$L = \frac{1}{2}\sum_{i=1}^{2}(y_i - t_i)^2$$

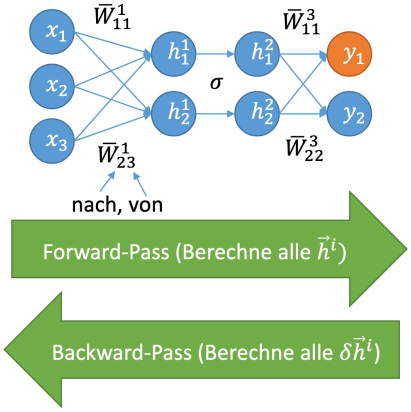
- Gesucht: Fehler an Knoten y_1:

$$\delta y_1 \overset{\text{def}}{=} \frac{\partial L}{\partial y_1} = y_1 - t_1$$

- analog, bzw. allgemein

$$\delta y_i \overset{\text{def}}{=} \frac{\partial L}{\partial y_i} = y_i - t_i$$

Entspricht dem „Fehler“ des Netzwerks



Berechnung der Gewicht-Deltas Bilden der partiellen Ableitung für jedes Gewicht jedes Layers und Speichern der Werte als Deltas. Zur Berechnung sind die Deltas der Outputs (siehe Backward-Pass) erforderlich. Für Batches werden die Deltas der Gewichte aufsummiert und nach dem Batch geupdated.

Backpropagation am Beispiel

$$\delta y_i \overset{\text{def}}{=} \frac{\partial L}{\partial y_i}, \qquad \vec{y} = \vec{h}^2 \cdot \bar{W}^3 + \vec{b}^3$$

- Gesucht: „Fehler“ des Gewichts $\bar{W}_{11}^3$:

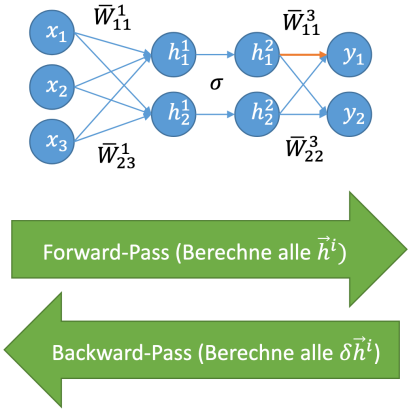
$$\delta \bar{W}_{11}^3 \overset{\text{def}}{=} \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial \bar{W}_{11}^3}$$

- analog, bzw. allgemein

$$\delta \bar{W}_{ij}^3 \overset{\text{def}}{=} \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial \bar{W}_{ij}^3}$$

- Einsetzen und Ableiten:

$$\delta \bar{W}_{ij}^3 = \delta y_i \cdot h_j^2$$



Update der Gewichte Die Gewichte werden nun geupdated, in dem die Deltas der Gewichte mit den ursprünglichen Gewichten verrechnet werden. Dafür wird ein Optimierungsverfahren verwendet (siehe unten).

Pre-Training Verwende Startgewichte, die bereits auf ähnlichen Daten trainiert wurden →

besser als zufällige Initialisierung.
Transferlearning Verwende vortrainiertes Netz und trainiere nur einzelne Schichten (z.B. letzte Schicht für Klassifikation) neu.

1.2 Optimizer

Optimierungsverfahren sind für das Update der Gewichte zuständig. Sie werden über die Learningrate η eingestellt.
Gradient Descent für Gewicht:

$$w'_{ij} = w_{ij} - \eta \cdot \delta w_{ij}$$

Momentum simuliert Ball, der die Funktion nach unten rollt:

$$w'_{ij} = w_{ij} + (mu \cdot v - \eta \cdot \delta w_{ij})$$

mit v aktuelle Ballgeschwindigkeit, mu wie viel v der Ball behalten soll
Nesterov Momentum verhindert hinauslaufen über Ziel (Minimum) (Problem bei Momentum) $\rightarrow$ verwende Lookahead für Gradient an nächster Position

$$w'_{ij} = w_{ij} + (mu \cdot v - \eta \cdot \frac{df}{dx}(w_{ij} + mu \cdot v))$$

Adagrad Verwendet Cache pro Parameter, um variable, sich optimierende Learningrate zu nutzen $\rightarrow$ kleinere Learningrate für große Gewichtsupdates und umgekehrt

$$cache_{ij} += \delta w_{ij}^2$$

$$w'_{ij} = w_{ij} - \frac{\eta}{\sqrt{cache_{ij}} + \epsilon} \cdot \delta w_{ij}$$

RMSProp Sanftere Learningrate/Cache-Anpassung als Adagrad. Update wie bei Adagrad.

$$cache'_{ij} = decayrate \cdot cache_{ij} + (1 - decayrate) \cdot \delta w_{ij}^2$$

Adam-Optimizer RMSProp mit Momentum, funktioniert oft am bestehen

$$m' = \beta_1 \cdot m + (1 - \beta_1) \cdot \delta w_{ij}$$

$$v' = \beta_2 \cdot v + (1 - \beta_2) \cdot \delta w_{ij}^2$$

$$w'_{ij} = w_{ij} - \frac{\eta}{\sqrt{v'} + \epsilon} \cdot m'$$

1.3 Regularisierung

Methoden um Overfitting vorzubeugen

- Rauschen auf Eingabedaten, Gewichten, Ausgabe (Zufallswerte hinzufügen)
- Datenaugmentierung

- Early-Stopping: Laufende Validierung auf Validierungsset, Trainingsabbruch wenn Accuracy abnimmt
- Dropout (siehe Layer)

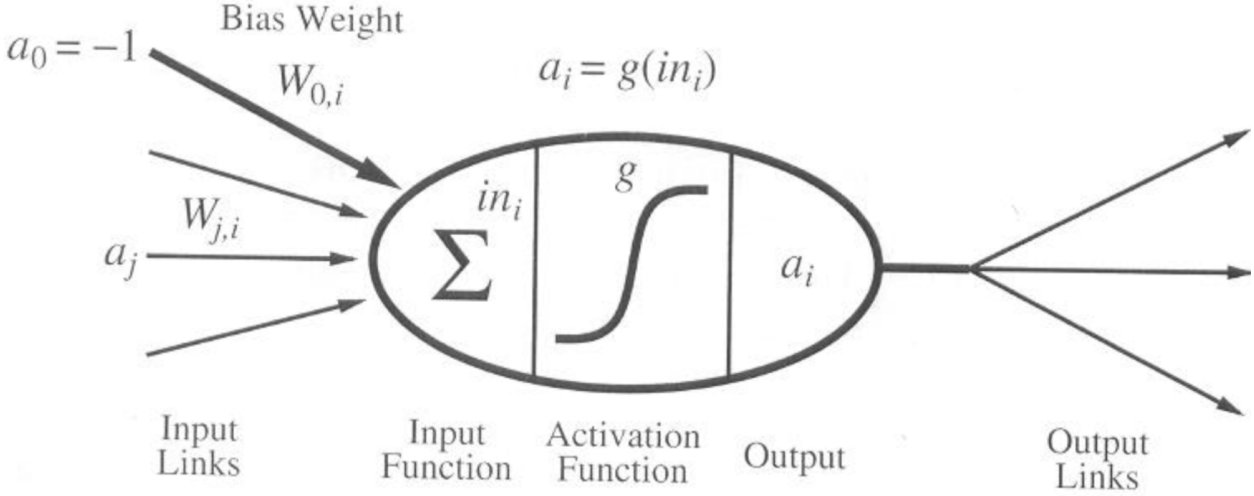
L2 Regularisierung Zusatzterm in Lossfunktion: $L_{reg} = L + \frac{1}{2}\lambda \sum_w w^2$
Weight Decay Zusatzterm zur Ableitung der Lossfunktion

1.4 Initializer

Es gibt mehrere Ansätze die Gewichte vor dem Training initial zu belegen.
Random zufällige Belegung → Normalisierung mit Anzahl der Gewichte n: $x = \frac{random}{\sqrt{n}}$
Glorot/Xavier Funktioniert am besten für Sigmoid oder Tanh Aktivierung: $x = \frac{random}{\sqrt{\frac{2}{n_{in}+n_{out}}}}$
He/Kaiming Funktioniert am besten für ReLU Aktivierung: $x = \frac{random}{\sqrt{\frac{2}{n_{in}}}}$

1.5 Netzarchitektur

Perceptron (= künstliches Neuron) stellt lineare Trennung (binäre Klassifikation) dar. Kann Funktionen AND, OR und NOT lernen, nicht aber XOR.



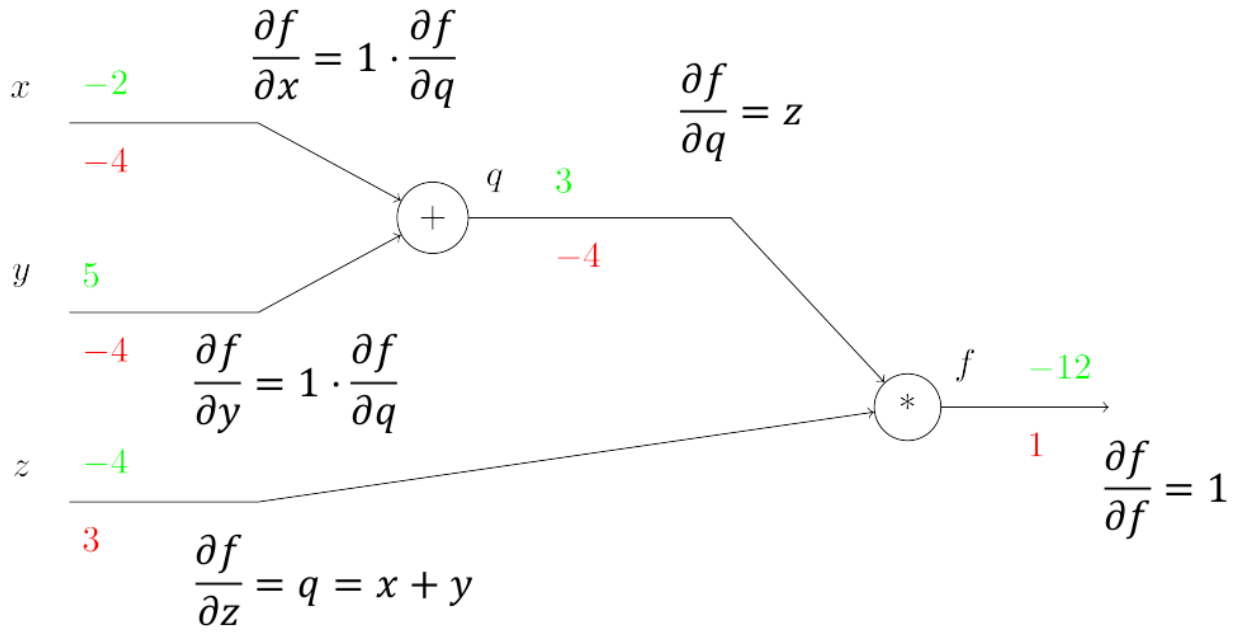
mit Inputs a_j gewichtet mit W_{ji} (ergeben zusammen die Gewichtsmatrix W), addiert mit Bias b, Outputs a_i und der Aktivierungsfunktion g mit der der Output berechnet wird.

$$Ausgabe_{Schicht(x)} = g(Eingabe_{Schicht(x)} \cdot W + b) = Eingabe_{Schicht(x+1)}$$

Aktivierungsfunktion muss bei Multi-Layer-Perceptronen (MLP) nicht-linear sein, da sonst nicht mehr Information gespeichert werden kann.

$$(x \cdot W + b) \cdot V + a = x \cdot W \cdot V + b \cdot V + a = x \cdot W' + b'$$

Circuit Diagram Darstellung eines neuronalen Netz als Graph mit Basis-Rechenoperationen. Kann für Forward- und Backwardpass verwendet werden.



Prediction Temperature Skaliert die Wahrscheinlichkeiten des Softmax. Wenn über Wahrscheinlichkeiten gesampled wird (z.B. bei Textgenerierung) können damit eher sichere oder eher unsichere Vorhersagen gemacht werden.

$$pred' = pred^{\frac{1}{temp}}$$

mit $temp \in [0, 1]$

2 Layer

Mögliche Schichten aus denen ein neuronales Netz bestehen kann.

2.1 Fully Connected / Dense

Voll verbundene Schicht, d.h. jeder Input landet in jedem Neuron. Wird z.B. als letzter Layer zur Klassifikation verwendet, um Vektor mit Logits für Klassen auszugeben.
Parameter $X \in \mathbb{R}^{1 \times a}$: Eingabe, $W \in \mathbb{R}^{a \times n}$: Gewichtsmatrix, $b \in \mathbb{R}^{1 \times n}$: Bias, n: Anzahl der Neuronen, $Y \in \mathbb{R}^{1 \times n}$: Ausgabe

Forward-Pass

$$Y = X \cdot W + b$$

Backward-Pass

$$\delta X = \delta Y \cdot W^T$$

Calculate Delta Weights

$$\frac{\delta L}{\delta W} = X^T \cdot \delta Y$$

$$\frac{\delta L}{\delta b} = \delta Y$$

2.2 Aktivierungsfunktion

Elementweise Anwendung

Funktion	**Forward**	**Backward**
tanh	$Y = \tanh(X)$	$\delta X = (1 - \tanh^2(X)) \odot \delta Y$
Sigmoid	$Y = \sigma(X) = \frac{1}{1+e^{-X}}$	$\delta X = (\sigma(X) \cdot (1 - \sigma(x))) \odot \delta Y$
ReLU	$Y = ReLU(X)$	$\delta X = ReLU'(X) \odot Y$

ReLU: $ReLU(x) = x > 0?x : 0$, $ReLU'(x) = x > 0?1 : 0$

2.3 Softmax

Normalisiert eine Menge von Werten, sodass deren Summe 1 ergibt. Wird zur Berechnung der Wahrscheinlichkeitsverteilung bei Klassifikation verwendet.

Forward-Pass

$$Y = softmax(X) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

Backward-Pass

$$\delta X = \delta Y \cdot \begin{bmatrix} \frac{\delta y_1}{\delta x_1} & \dots & \frac{\delta y_n}{\delta x_1} \\ \dots & \ddots & \dots \\ \frac{\delta y_1}{\delta x_n} & \dots & \frac{\delta y_n}{\delta x_n} \end{bmatrix}$$

2.4 Dropout

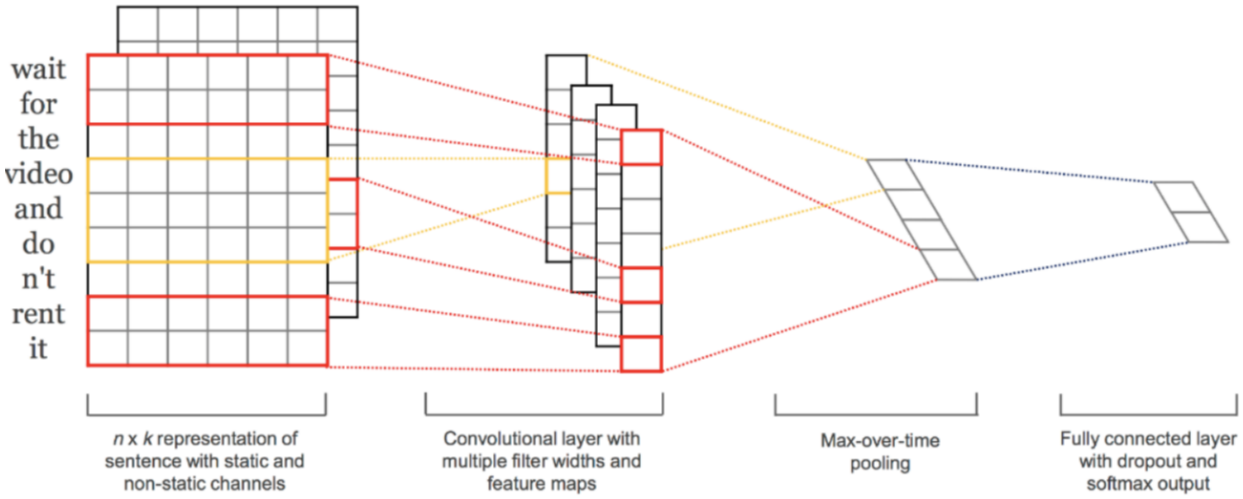
Für Regularisierung verwendet. Ein Teil der Gewichte wird zufällig je Durchlauf auf 0 gesetzt (deaktiviert). Wird nicht trainiert, sollten alle Gewichte weitergegeben, aber durch die Dropout-Rate geteilt werden, da sonst eine zu hohe Aktivierung der nächsten Schicht statt findet.

Parameter d: Dropout-Rate gibt den prozentualen Anteil der zu deaktivierenden Gewichte an.

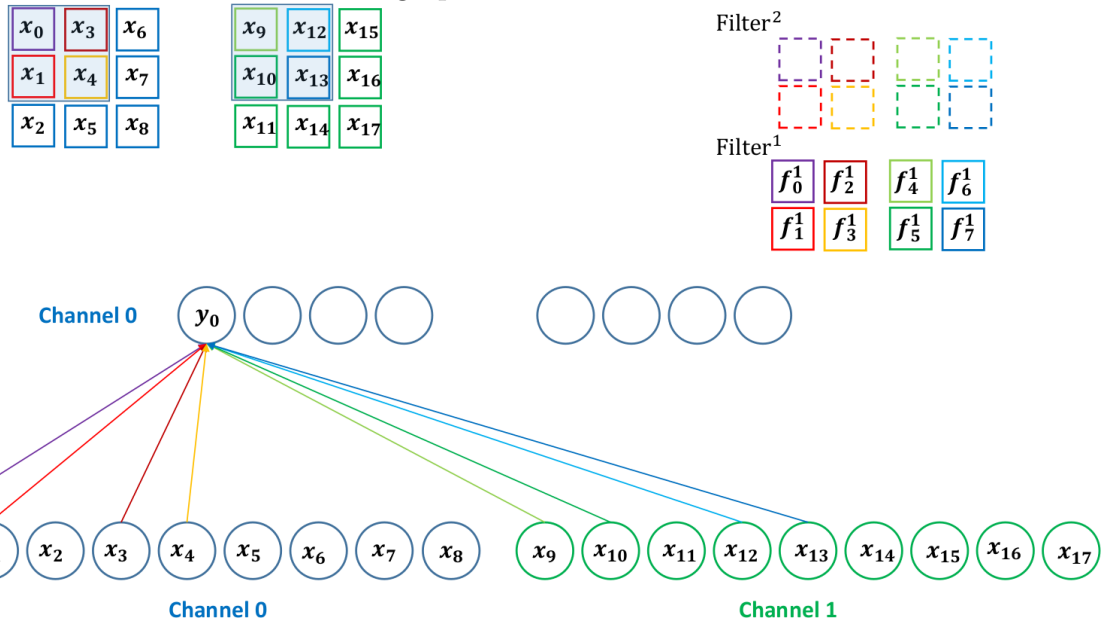
2.5 Convolutional

Extrahiert Features aus einer Matrix. Wird oft in der Bildklassifizierung verwendet oder NLP zur Satzklassifikation (Kernel Größe der Embeddings × Anzahl betrachteter Wörter).

Beispiel für NLP

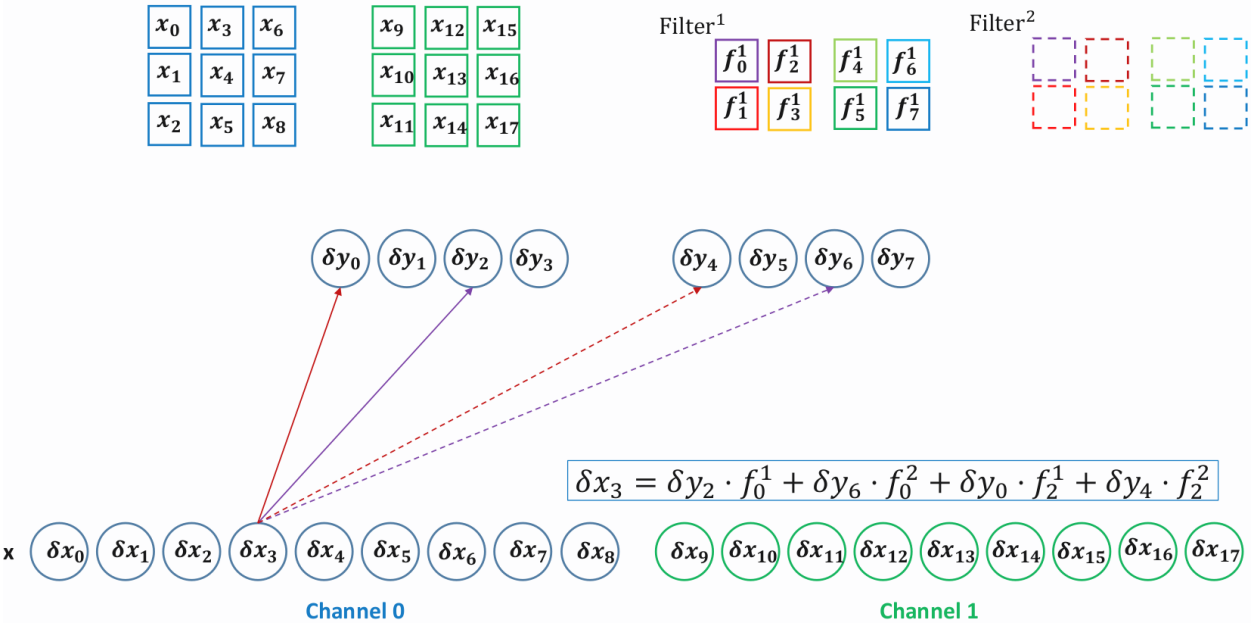


Parameter
$X \in \mathbb{R}^{h \times w \times d}$: Eingabe
(fh, fw): Filtergröße
fn: Anzahl Filter
$F \in \mathbb{R}^{fh \times fw \times fd \times fn}$: Filtertensor
$b \in \mathbb{R}^{fn}$: Bias (ein Wert pro Ausgabechannel, wird auf jeden Wert des Channels addiert)
$Y \in \mathbb{R}^{(h-fh+1) \times (w-fw+1) \times fn}$: Ausgabe
Optionale Parameter Stride: wie weit der Filter verschoben wird (default = 1), Dilation: zusätzliche 0en im Filter (Filter über nicht benachbarte Elemente)
Padding Covolution verkleinert die Daten. Um dies zu verhindern, kann die Eingabe gepadded werden (Hinzufügen von 0en am Rand der Eingabe).
Half-Padding: Ausgabegröße = Eingabegröße, $ph = \lfloor \frac{fh}{2} \rfloor$, $pw = \lfloor \frac{fw}{2} \rfloor$
Full-Padding (Full-Convolution): Ausgabegröße > Eingabegröße, $ph = fh - 1$, $pw = fw - 1$
Forward-Pass $*$ bezeichnet die Faltungsoperation



$$Y = X * F + b$$

Backward-Pass $*_F$ bezeichnet die Full-Convolution



$$\delta X = \delta Y *_F rot_{h,w}^{180}(trans(F))$$

Calculate Delta Weights $*_{ch}$ bezeichnet die Channel-Wise-Convolution, f ist ein Element des Filtertensors

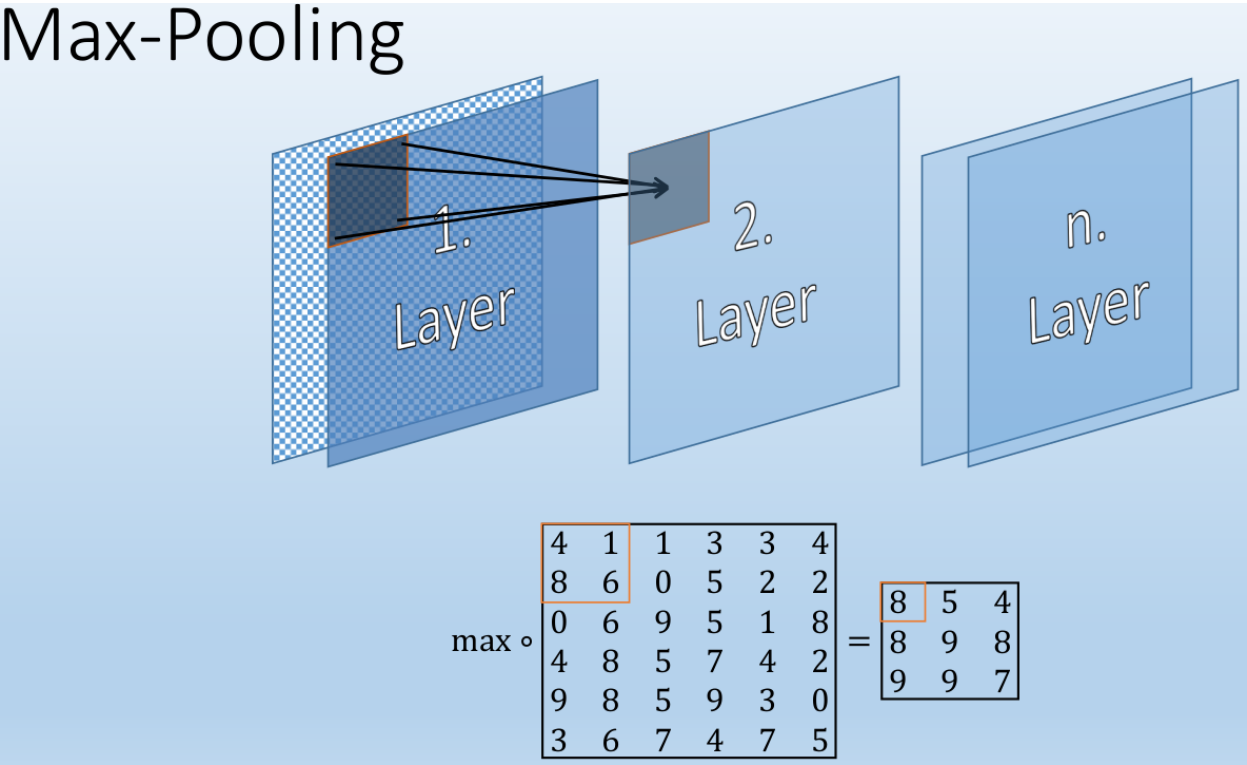
$$\frac{\delta L}{\delta f} = X *_{ch} \delta Y$$

$$\frac{\delta L}{\delta b_f} = \sum_i \delta y_{i,f}$$

2.6 Pooling

Reduziert die Werte innerhalb eines Filters auf einen Wert (z.B. Maximum oder Durchschnitt). Filter wird ohne Überlappung (mit Stride) verschoben.

Forward-Pass



Backward-Pass Nur die Elemente, die im Forward-Pass an der Berechnung des Outputs beteiligt waren, bekommen anteilig oder ganz das Delta des jeweiligen Outputs.

$$\delta x_i = (x_i == max)?\delta y : 0$$

Maximum over time pooling Übernimmt den größten Wert pro Outputvektor eines Filters

2.7 Vanilla RNN

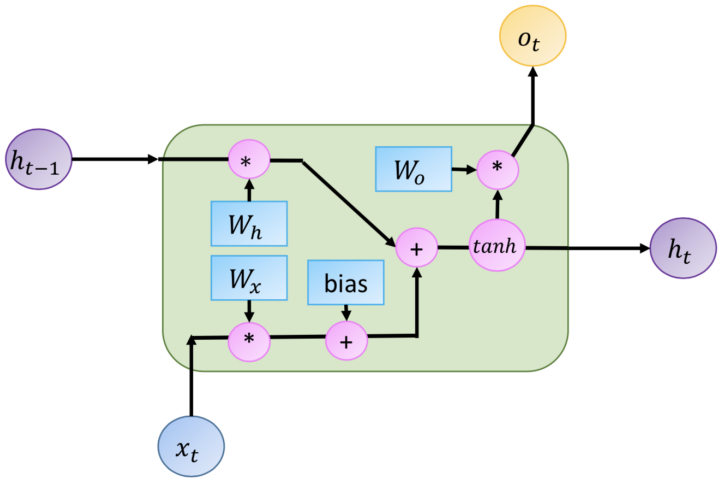
Recurrent Neuronal Networks (RNNs) werden verwendet, um Daten unterschiedlicher Länge zu verarbeiten. Dabei wird ein Hidden State h in jedem Schritt um Informationen der Eingabe ergänzt.

Parameter

$[x_0, \cdots, x_n] = X \in \mathbb{R}^{n \times m}$: Eingabevektoren mit jeweils Länge m
g: Zwischenvariable nach der Addition vor tanh
h: Hidden State (h_{-1} kann mit Nullen initialisiert werden)
W_x, W_h, W_o: Gewichtsmatrizen
b: Bias
$[o_0, \cdots, o_n] = O \in \mathbb{R}^{n \times l}$: Ausgabevektoren mit jeweils Länge l

Funktionsweise

The Vanilla Recurrent Cell (one possibility)



$$h_t = \tanh(h_{t-1} \cdot W_h + x_t \cdot W_x + bias)$$
$$o_t = h_t \cdot W_o$$

Forward-Pass

$$h_t = \tanh(h_{t-1} \cdot W_h + x_t \cdot W_x + b)$$
$$o_t = h_t \cdot W_o$$

Backward-Pass

$$\delta h_t = \delta o_t \cdot W_o^T + \delta g_{t+1} \cdot W_h^T$$
$$\delta g_t = \delta h_t \cdot (1 - \tanh^2(g_t)) = \delta h_t \cdot (1 - h_t^2)$$
$$\delta x_t = \delta g_t \cdot W_x^T$$

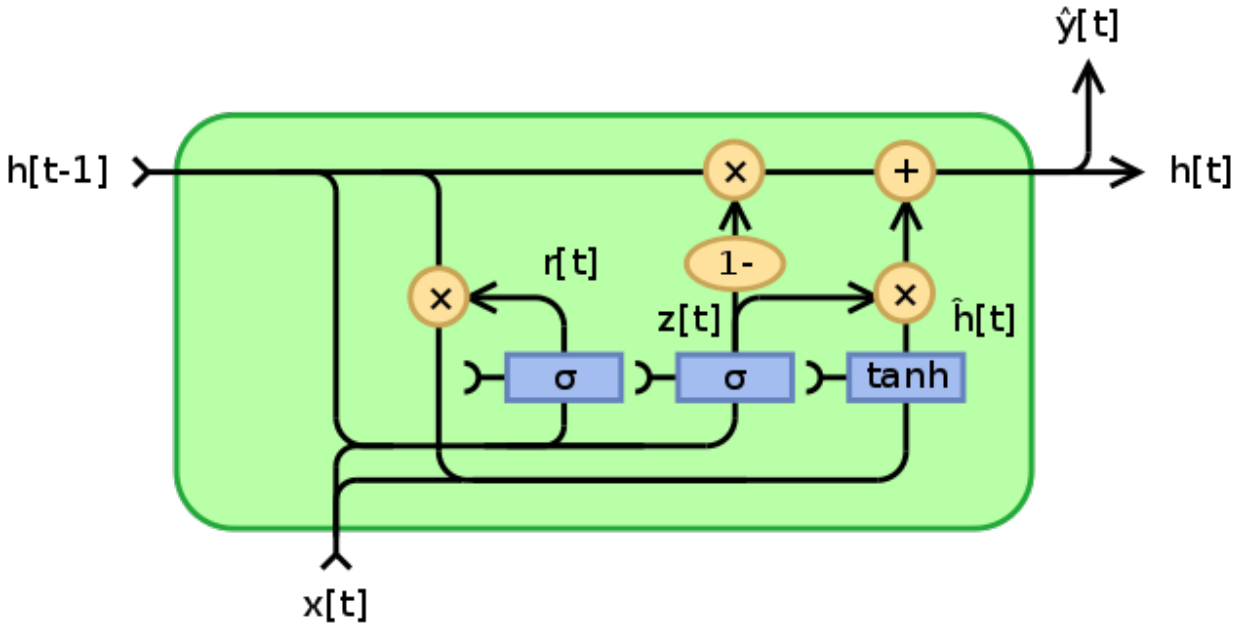
Calculate Delta Weights

$$\delta W_h = \sum_t h_{t-1}^T \cdot \delta g_t$$
$$\delta W_x = \sum_t x_t^T \cdot \delta g_t$$
$$\delta W_o = \sum_t h_t^T \cdot \delta o_t$$
$$\delta b = \sum_t \delta g_t$$

2.8 Gru

Verwendet zwei Gates (wie g im Vanilla RNN), um zu bestimmen, was aus dem alten Internal State h übernommen wird und was von den aktuell verarbeiteten Daten hinzugefügt wird.

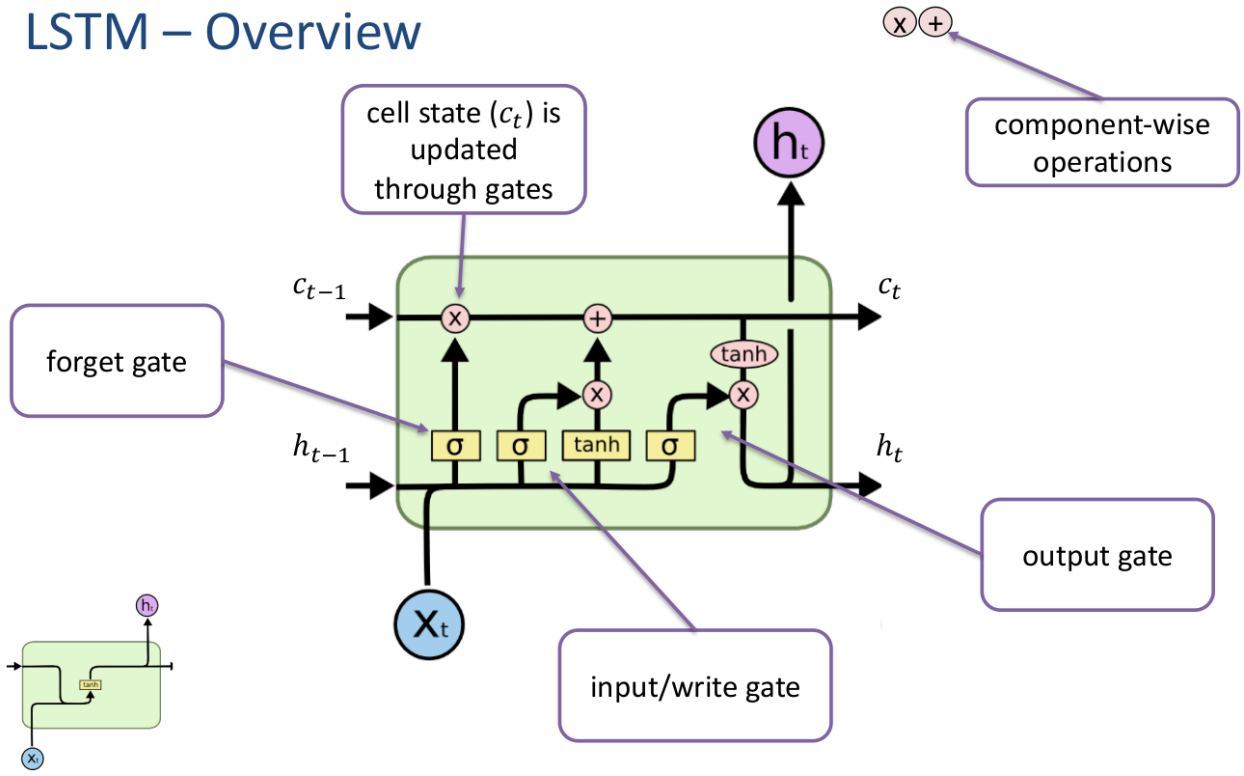
Funktionsweise



2.9 LSTM

Verwendet drei Gates (Forget, Input, Output) und zusätzlichen State c

Funktionsweise



2.10 Highway Layer

Verwendet ein Gate, das bestimmt, ob der Layer wie ein Fully Connected Layer funktioniert oder lediglich den Input weitergibt. Wird verwendet bei sehr tiefen Netzwerken (> 100 Layer), um Daten über lange Zeit/viele Layer hinweg nutzbar zu machen.
Forward-Pass

$$Y = (X \cdot W + b) \cdot T(x) + x \cdot (1 - T(x))$$
$$T(x) = \sigma(x \cdot W_T + b_T)$$

3 Loss-Funktionen

X: Eingabe der Loss-Funktion bzw. Ausgabe des Netzes (Prediction)
T: Erwartetes Ergebnis (Ground Truth)

3.1 Cross Entropy

Negative-Log-Likelihood, Cross-Entropy
Forward-Pass

$$L = -\sum_i t_i \cdot \log(x_i)$$

Backward-Pass

$$\frac{\delta L}{\delta x_i} = -\frac{t_i}{x_i}$$

3.2 Mean Squared Error

Euklidischer Loss, Mean-Squared-Error, l_2-Loss
Forward-Pass

$$L = \sum_i \frac{1}{2}(x_i - t_i)^2$$

Backward-Pass

$$\frac{\delta L}{\delta x_i} = x_i - t_i$$

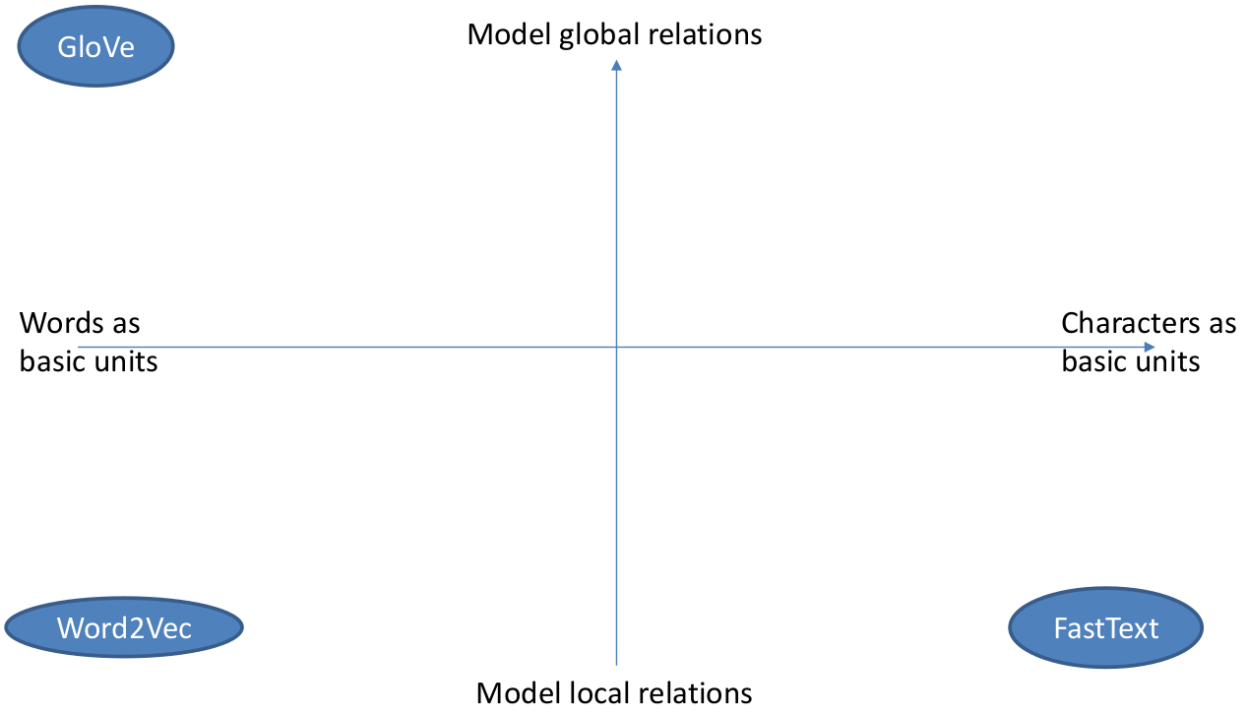
4 Textrepräsentation

Wie kann Text (Wörter, Buchstaben) für NLP-Tasks mit neuronalen Netzen repräsentiert werden?
Dokument beschreibt eine Menge an Text, z.B. Satz, Tweet, Buch, etc.
Vokabular alle möglichen Wörter in einem bestimmten Kontext (z.B. Duden für deutsche Sprache)
Bag of Words simpler Ansatz, Vektor der für jedes Wort des Vokabulars einen Wert beinhaltet, Anzahl der Worthäufigkeit im Dokument, Reihenfolge der Wörter im Dokument wird nicht betrachtet
→ funktioniert gut, aber verliert viel Information über Kontext

One-hot-vector repräsentiert ein Wort, Vektor hat einen Eintrag für jedes Wort des Vokabulars, Eintrag für zu repräsentierendes Wort ist 1 alle anderen 0
→ Reihenfolge durch Sequenz von one-hot-vectors abbildbar, aber Ähnlichkeiten von Wörtern werden nicht berücksichtigt
Cosine-Similarity Ähnlichkeit von Wörtern über Cosinus Berechnen

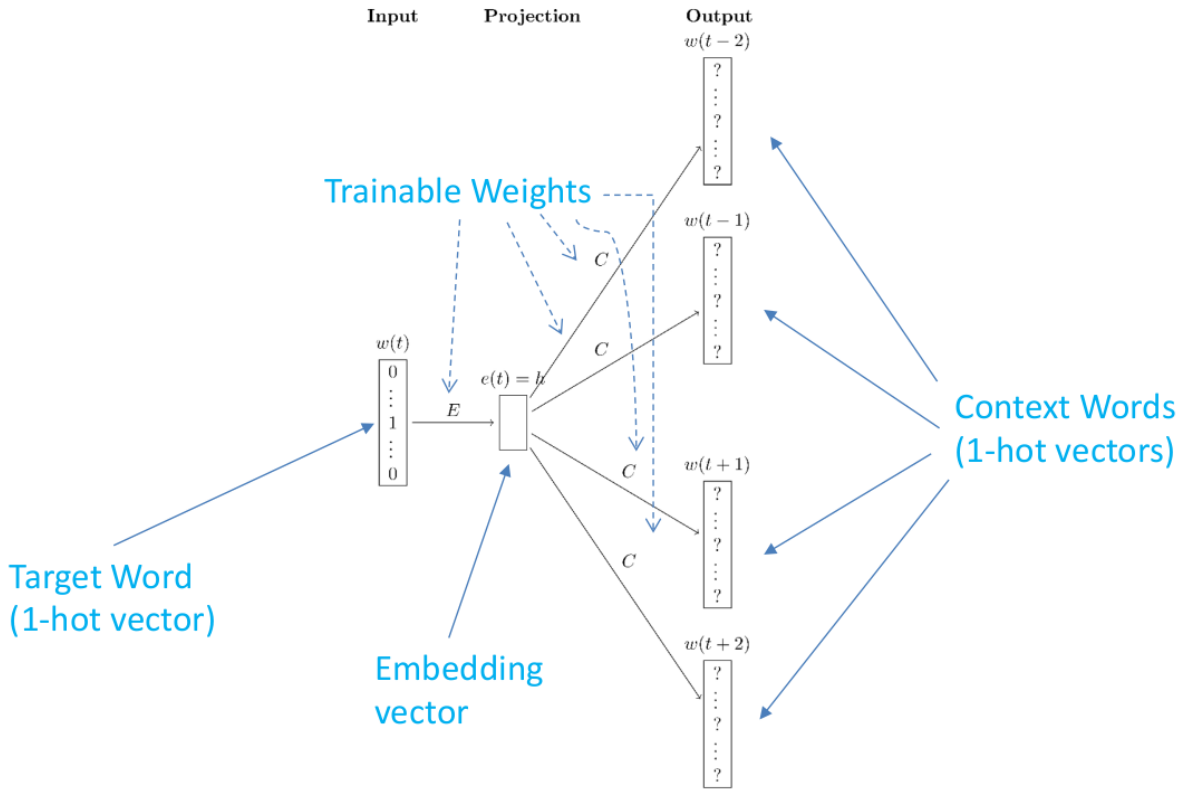
$$sim_{cos}(x, y) = \frac{\sum_{i=1}^{N} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{N} x_i^2} \cdot \sqrt{\sum_{i=1}^{N} y_i^2}}$$

WordNet handgelabelter Datensatz mit Ähnlichkeitsbeziehungen von Wörtern → nicht genau genug, unvollständig, menschlicher Bias
Word Embedding Vektor, der ein Wort repräsentiert. Bekannte Verfahren Embeddings zu erzeugen:

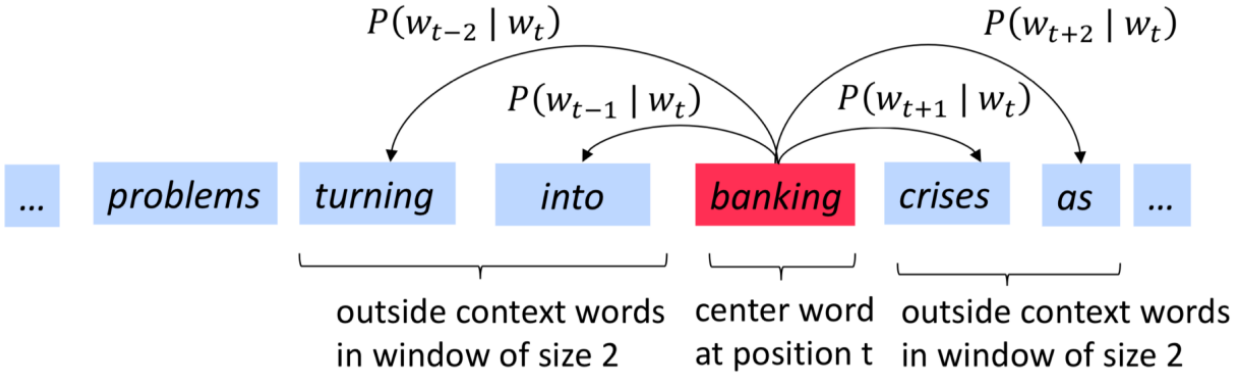


4.1 Word2Vec

Annahme: Ähnliche Wörter kommen im ähnlichen Kontext vor d.h. sie sind in Texten häufig von den gleichen Wörtern umgeben.
Berechnung von Word Embeddings (feste Länge) anhand der häufig dieses Wort umgebenden Wörter → ähnliche Wörter bekommen ähnliche Vektoren
Word2Vec wird selbst auf Beispieltexten trainiert



CBOW Center Wort durch Kontextwörter vorhersagen
Skip-gram Kontextwörter durch Center Wort vorhersagen



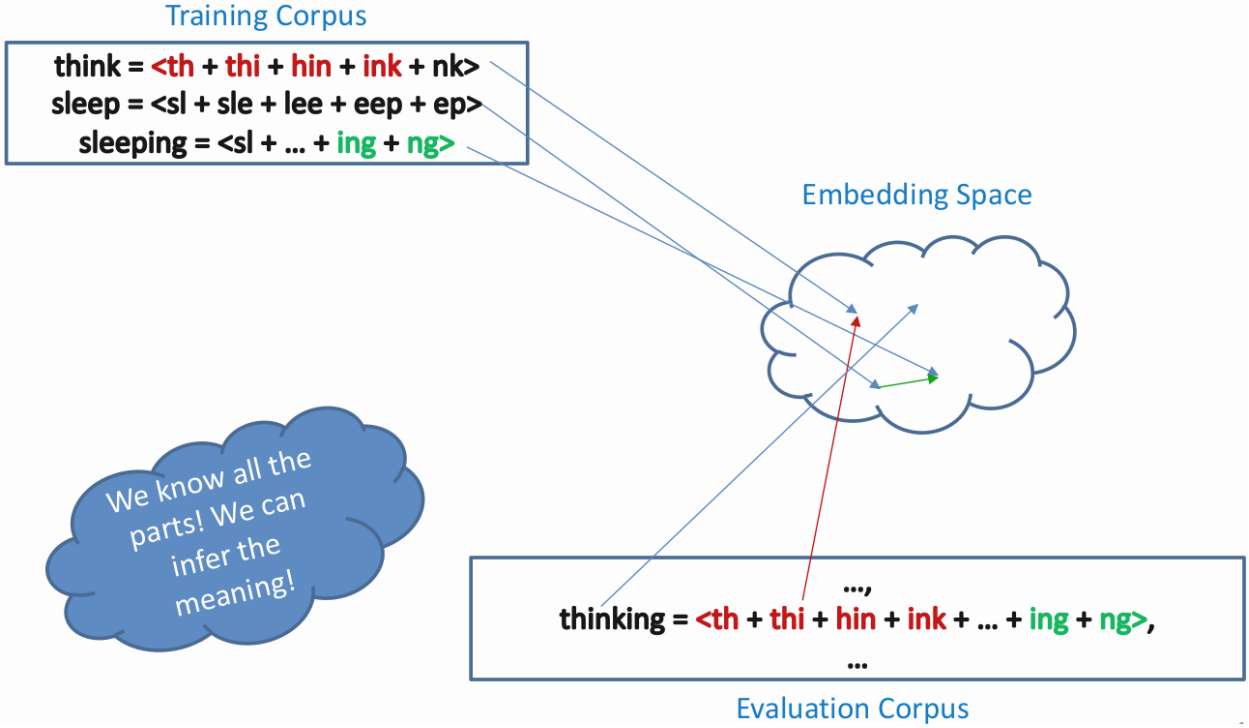
4.2 GloVe (Global Vectors)

Berechnung von Word Embeddings über eine Matrix X, die für einen Wortkorpus die Häufigkeiten angibt, mit der zwei Wörter zusammen auftreten. Verwendet globale Abhängigkeiten statt lokale wie bei Word2Vec.
Neue, unbekannte Wörter werden auf gleichen UNKNOWN-Vektor abgebildet (wie bei Word2Vec).

4.3 FastText

Funktioniert wie Word2Vec auf lokalem Kontext, aber verwendet Bustaben n-Gramme (Buchstabenketten der Länge n) statt ganze Wörter. Word Embeddings werden durch Aufsummieren der Vektoren

ihrer n-Gramme erzeugt.
→ kann neue Wörter besser abbilden, da n-Gramme vermutlich bekannt (z.B. bei konjugierten Verben)

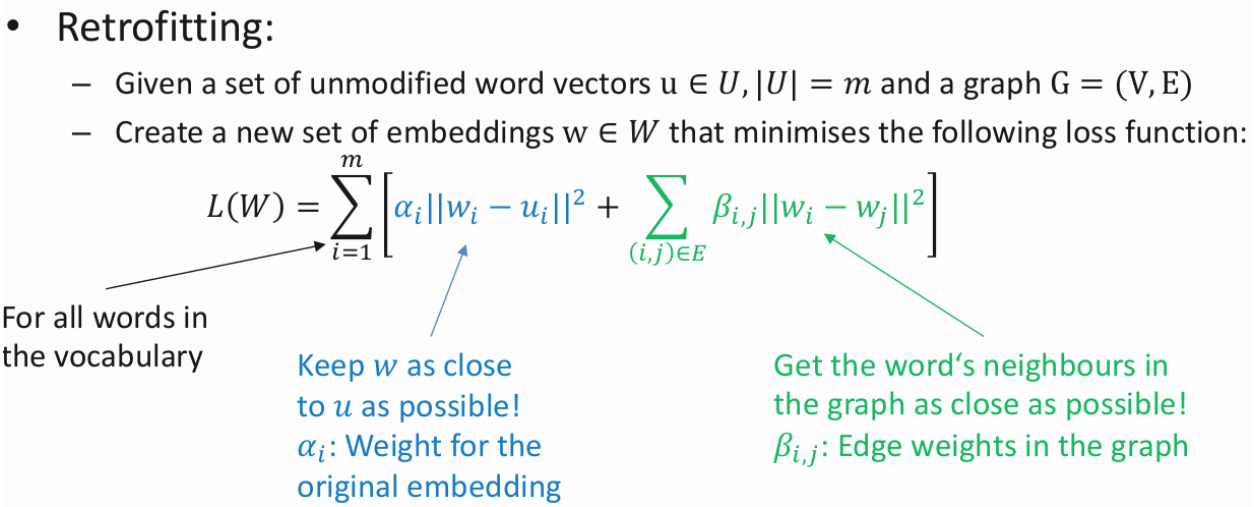


4.4 Weltwissen

Word Embeddings werden durch die vorgestellen Verfahren nur auf einen bestimmten Korpus angepasst. Menschen verstehen Sprache besser durch Einbeziehen von Hintergrundinformationen, sog. Weltwissen.
Quelle für Weltwissen ist z.B. Dbpedia (maschinenlesbare Form der Wikipedia), Wikidata, BabelNet, ConceptNet (verbindet Begriffe durch gewichtete, beschriftete Beziehungen/Kanten, z.B. Synonym, TeilVon, ...)
Pointwise mutual information Treten zwei Wörter öfters zusammen auf als wenn sie unabhängig wären?

$$PMI(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

ConceptNet Numberbatch Anreichern von existierenden Embeddings (Word2Vec, GloVe oder FastText) mit Weltwissen aus ConceptNet → Abbildung durch neuronales Netz mit Retrofitting Loss-Funktion



4.5 Evaluation

Mehrere Word Embeddings können auf verschiedene Weise miteinander verglichen werden.
Extrinsisch Evaluation durch Verwenden in einem NLP-Task
Intrinsisch hand-annotierten Evaluationsdatensatz (Human Intuition Datasets HID, z.B. WS-353, MEN-3000, ...) mit Wortpaaren verwenden, der Ähnlichkeitsgrad vorhersagt, z.B. r(table, chair) > r(duck, plane)
→ Word Embeddings können auch auf HIDs nachtrainiert werden

5 Anwendungsfelder für NLP

5.1 Textklassifikation

Abbilden eines Textes auf ein Label
Sentimentanalyse Ausgabe eines Stimmungswertes für einen Text meist zwischen -1 (negativ) und 1 (positiv).

5.2 Textgenerierung

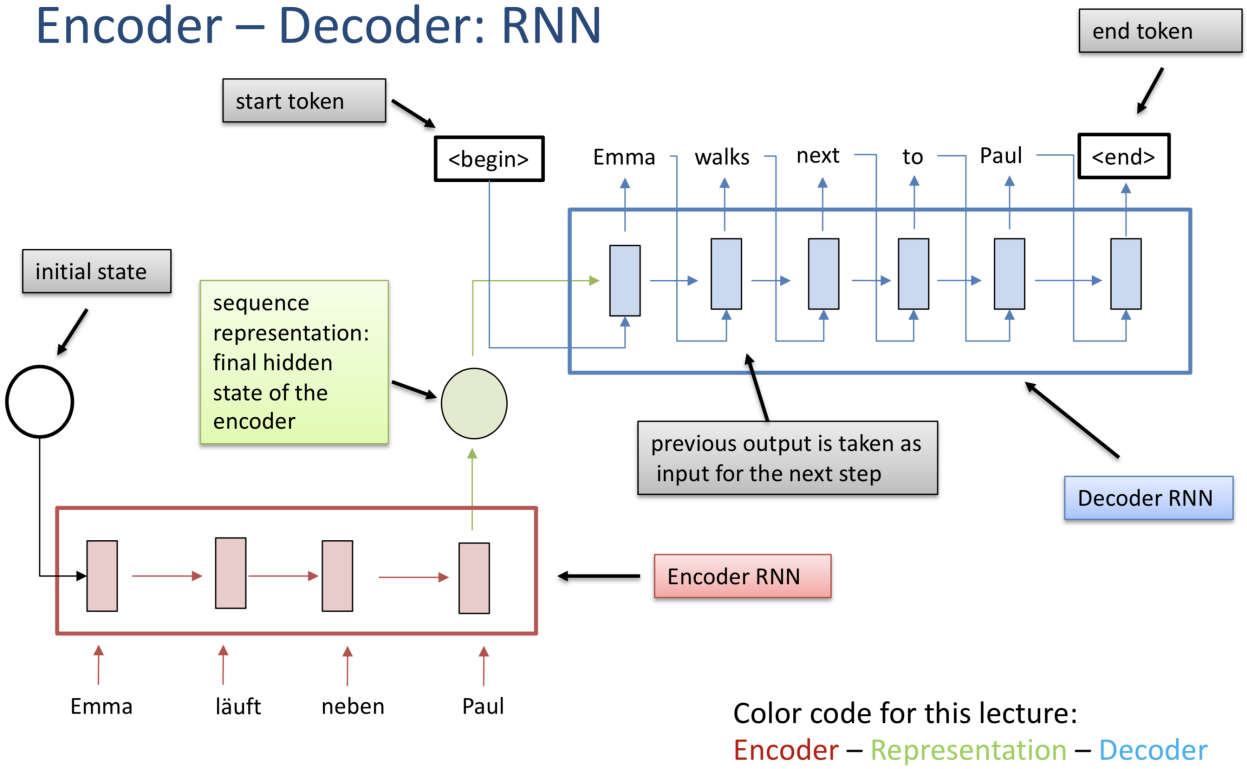
Führt vorgegebenen Text weiter. Perfekter Text Generator besteht Turing Test

5.3 Machine Translation

Maschinelle Übersetzung verwendet RNNs wodurch folgende Probleme entstehen:

- unterschiedliche Satzstellung bei verschiedenen Sprachen
- ein Inputwort entspricht nicht gleich einem Outputwort (z.B. neben ⇒ next to)

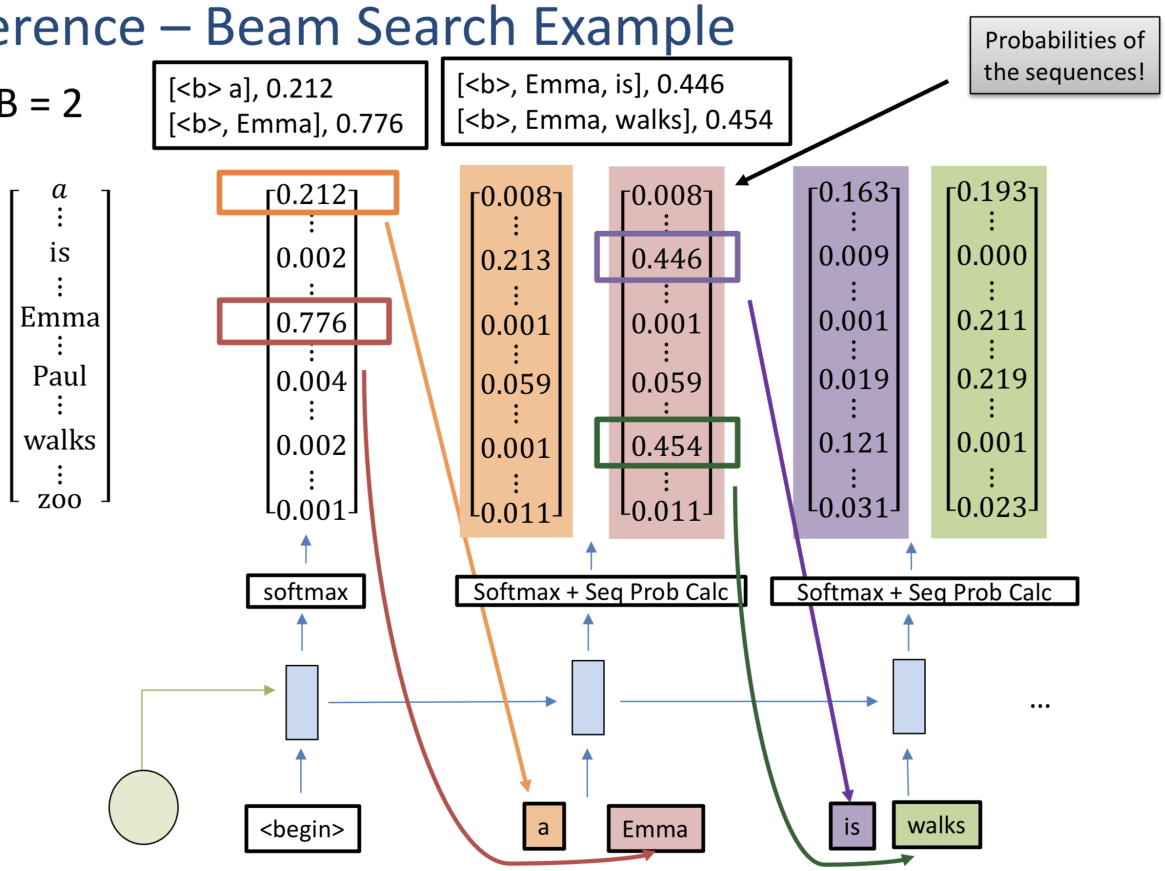
→ **Encoder-Decoder-Architektur** liest gesamten Satz mit einem RNN ein und erzeugt mit anderem RNN die Ausgabe



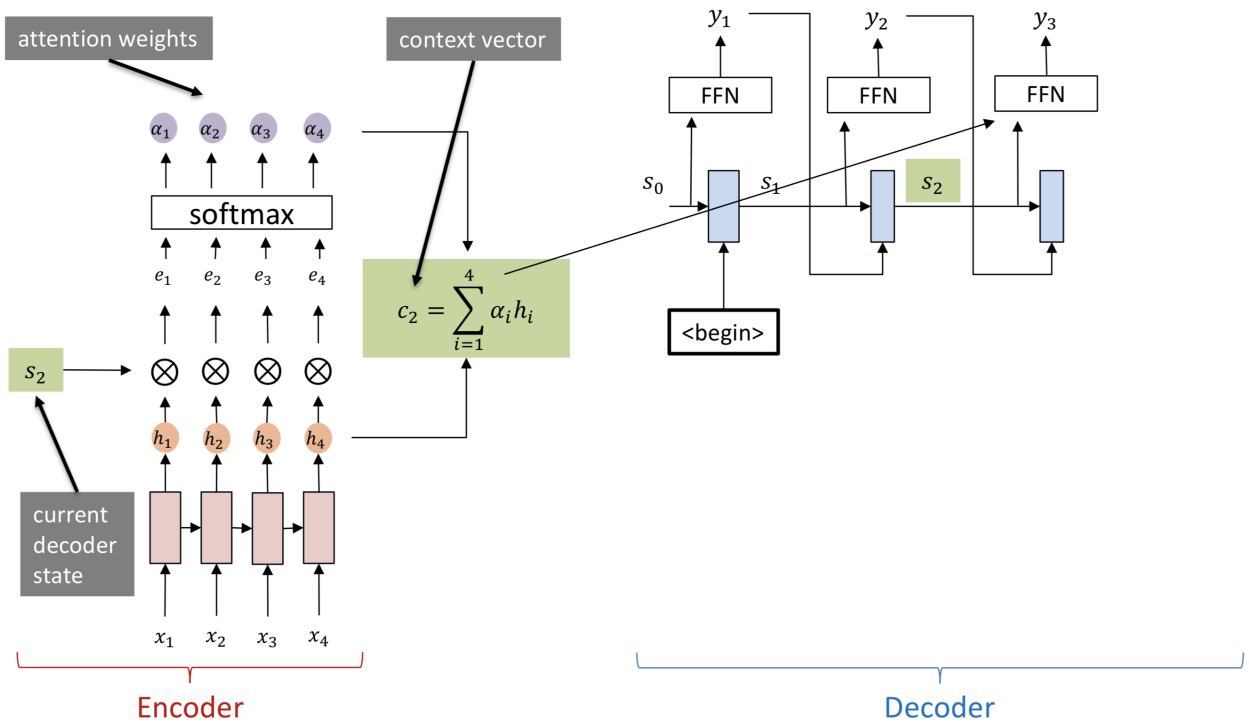
Teacher Forcing Ausgabewort ist Eingabe für nächsten Decoderschritt. Um Fehler zu vermeiden, solle die Eingabe für den nächsten Schritt das vorherige Wort der Ground Truth sein.
→ sollte nur zu Beginn des Trainings verwendet werden, dann Wort der vorherigen Vorhersage, da Netzwerk sich sonst darauf verlässt etwas richtiges zu bekommen
Greedy Search immer wahrscheinlichstes Wort nehmen
Beam Search betrachte die B Sequenzen mit der höchsten Wahrscheinlichkeit und entscheide später für die beste ($B = 1 \rightarrow$ Greedy-Search)

Inference – Beam Search Example

- B = 2



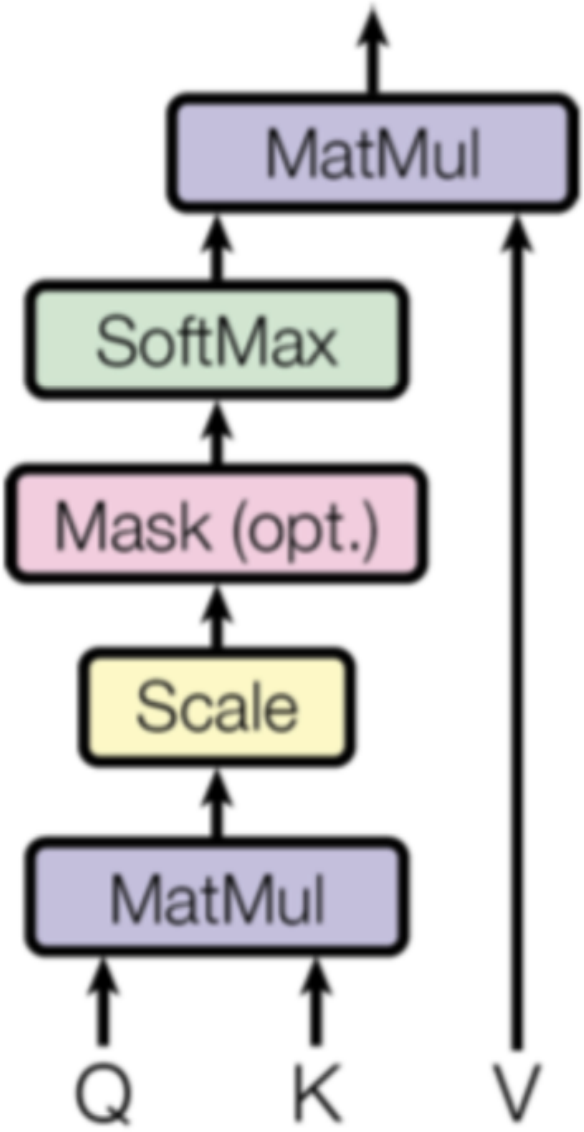
BiRNN zwei RNNs im Encoder, die Sequenz vorwärts bzw. rückwärts bekommen. Letzter State beider RNNs wird konkatiniert und ist Encoder State.
Attention Mechanismus einzelner State nach Encoder ist Informationsbottleneck → verwende alle Hidden States des Decoders gewichtet als Input für jeden Decoderschritt. Der Decoder-Hidden-State gibt die Gewichtung/Attention vor.



Evaluation von Übersetzung Es gibt keine Single Source of Truth (mehrere Übersetzungen möglich und korrekt)

- Unigram Precision: $s = \frac{m}{w_t}$, m Wörter die in Ground Truth und Ausgabe auftreten, w_t Länge der Maschinenübersetzung
- Bilingual evaluation understudy (BLEU): arbeitet auf n-Grammen (Wortketten), limitiert Anzahl auf Vorkommen in Zielübersetzung, wird gegen mehrere mögliche Übersetzungen evaluiert und Score gemittelt

Transformer neue neuronale Netzarchitektur auf Basis von Attention ohne RNNs → gut parallelisierbar



$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V$$

$$Q = X \cdot W_1, K = X \cdot W_2, V = X \cdot, W_3$$

Attention wird komplett aus Input berechnet, es wird eine gewichtete Summe des Inputs erzeugt

Multihead Attention Berechne Q, K, V mehrmals mit unterschiedlichen Gewichtsmatrizen W_i, um verschiedenen Fokus zu bekommen.

Evolved Transformer Netzarchitektur wird auch als neuronales Netz mittrainiert.