Anwendungen der KI

– Sommersemester 2018 –

Kapitel 08: Neuronale Netze III

Prof. Adrian Ulges
B.Sc. {Angewandte, Medien-, Wirtschafts-}informatik, ITS
Fachbereiche DCSM
Hochschule RheinMain

Outline

1. Term Embeddings

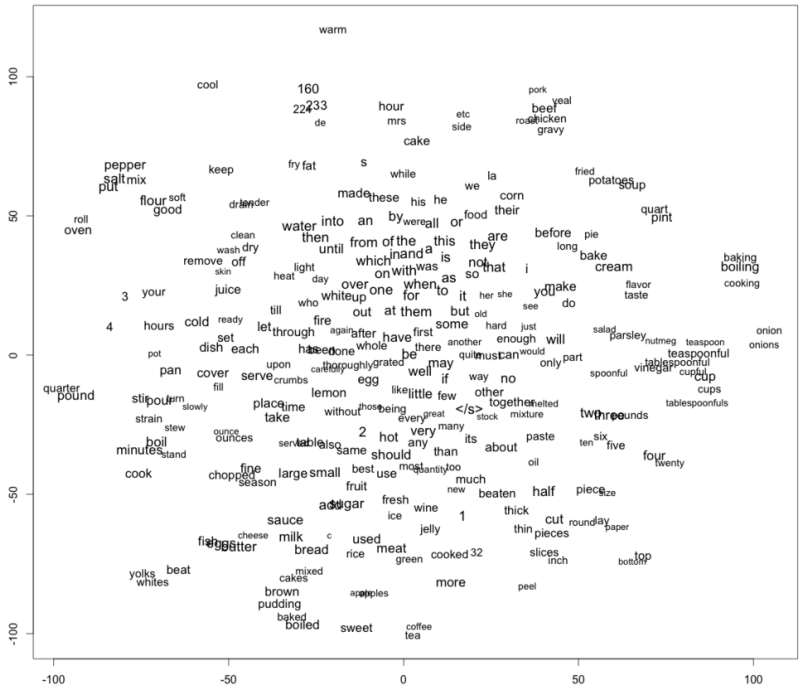
2. NLP-Modell 2: CNN

3. NLP-Model 3: Rekurrente Netze

Embeddings und Sprachmodelle

Lernen von Embeddings

- Embeddings sollen **ähnliche Terme** auf **ähnliche Vektoren** abbilden
- **Möglichkeit 1**: Überwachtes Training, z.B. auf Beispielfragen
- **Problem**: Überwachtes Training erfordert **gelabelte Daten**, die oft stark begrenzt sind
- **Möglichkeit 2** *(hier)*: **Unüberwachtes Training** *auf großen Daten mit Sprachmodellen*



Embeddings und Sprachmodelle

"You shall know a word by the company it keeps." *J.R.Firth (1957)*

Definition (Sprachmodell)

Es sei V ein Vokabular von Termen, und $(w_1, .., w_k)$ sei ein k-Gramm über V. Dann modelliert ein Sprachmodell die Beziehung zwischen Wort w_k und seinem Kontext $w_1, ..., w_{k-1}$, also die Wahrscheinlichkeitsverteilung

$$P(w_k | w_1, ..., w_{k-1})$$

Beispiel

The cat sat on the ?

→ "roof", "bed" sind in diesem Kontext wahrscheinlich

→ "walked", "faith" sind unwahrscheinlich

Sprachmodelle (cont'd)

Idee

- Wir verwenden als Sprachmodell ein neuronales Netz
  (hier: Das populäre Word2Vec- bzw. SkipGram-Modell)
- Das Netz lernt auf (potenziell großen) **Text-Korpora**
- Das Netz sagt für jeden Term w vorher, **welche anderen Terme** im Kontext von w wahrscheinlich sind.
- Hierfür verwendet das Netz **Embeddings** E!

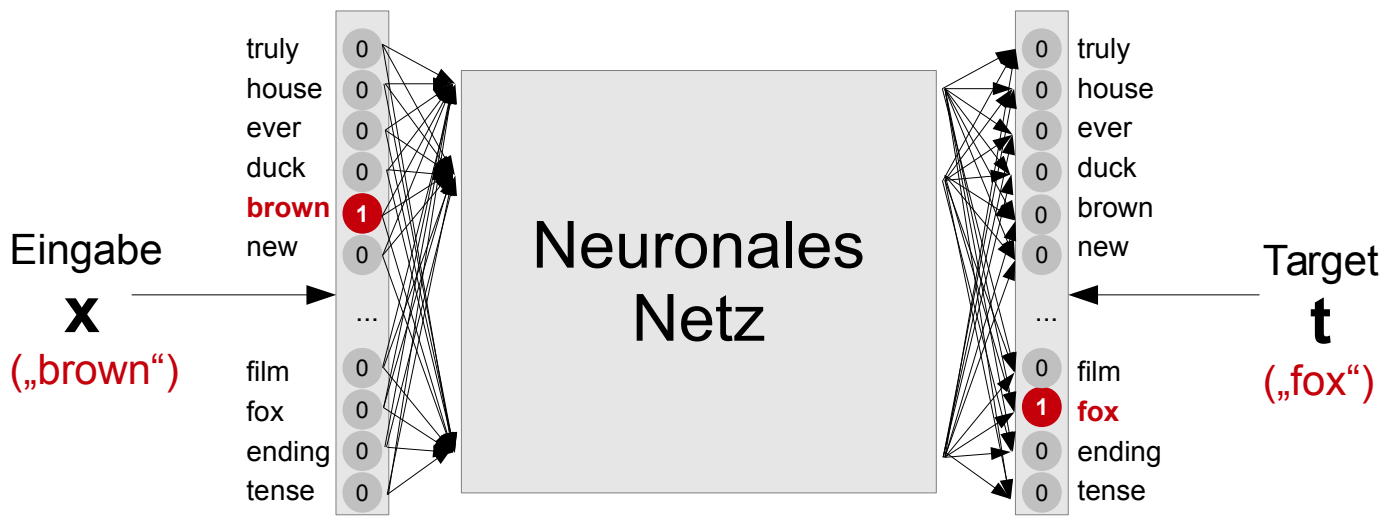
Das Word2Vec-Modell [1]

- Gegeben: Großer **Textkorpus** *(bis zu 1.6 Mrd. Terme [2])*
- Wir samplen **lokale Term-Paare** aus dem Korpus:

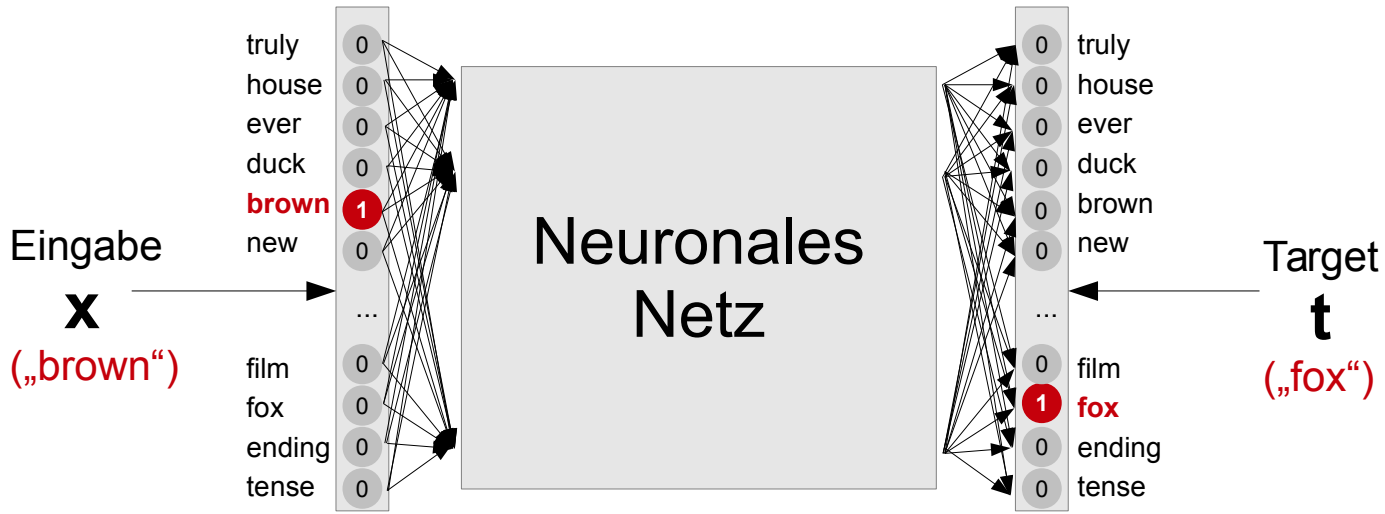
the quick brown fox jumped over the lazy dog

	Term-Paare
the quick	(the,quick)
the **quick** brown	(quick,the), (quick,brown)
quick **brown** fox	(brown,quick), (brown,fox)
brown **fox** jumped	(fox,brown), (fox,jumped)
fox **jumped** over	(jumped,fox), (jumped,over)
...	

- Aus jedem Termpaar (w_1, w_2) wird ein **Trainings-Sample**: Gegeben w_1, soll das Netz w_2 ausgeben!



Das Word2Vec-Modell [1]



Anmerkungen

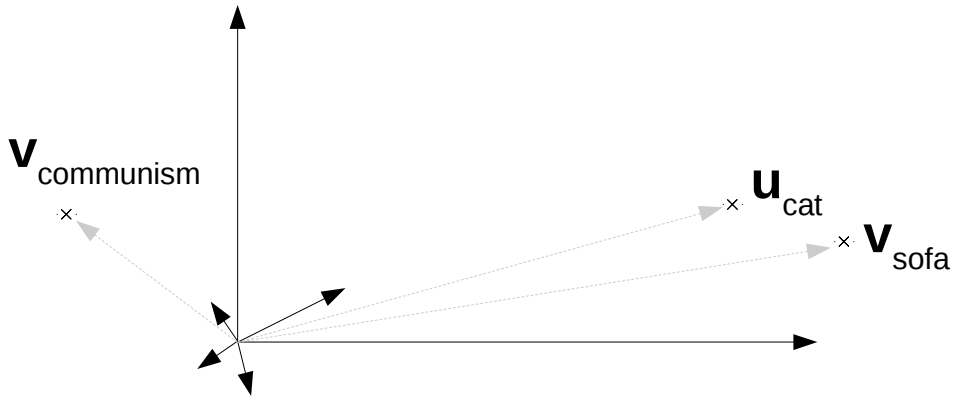
- Die Trainingsdaten enthalten **widersprüchliche** Samples, z.B. (brown,quick) und (brown,fox). Dies ist aber kein Problem: Das neuronale Netz findet einen “Kompromiss”.
- Die Ausgabewerte bedeuten: Gegeben den Eingabeterm w_1, was ist die **Wahrscheinlichkeit**, dass in der Umgebung Ausgabeterm w_2 auftaucht?

Das Word2Vec-Modell [1]

Anmerkungen

- **Beispiel**: In den Trainingsdaten taucht `soviet` häufiger mit `union` auf als mit `yeti`. Also wird bei Eingabe von `soviet` das Netz den Ausgabeknoten `union` deutlich stärker aktivieren.
- Die **Anzahl der Eingabe- und Ausgabeknoten** entspricht der Größe des **Vokabulars** *(bis zu mehreren Millionen!)*

Das Word2vec-Modell



- Jeder Term t besitzt zwei **Embeddings** $\mathbf{u}_t, \mathbf{v}_t \in \mathbb{R}^{300}$
- Ein einfacher Sigmoid wird verwendet um $P(t_2|t_1)$ zu schätzen

$$P(t_2|t_t) := \sigma\Big(< \mathbf{u}_{t_1}, \mathbf{v}_{t_2} > \Big) = \frac{1}{1 + exp(- < \mathbf{u}_{t_1}, \mathbf{v}_{t_2} >)}$$

- Ziel **Lerne** die Embeddings durch **Backpropagation**!

Das Word2vec-Modell: Illustration

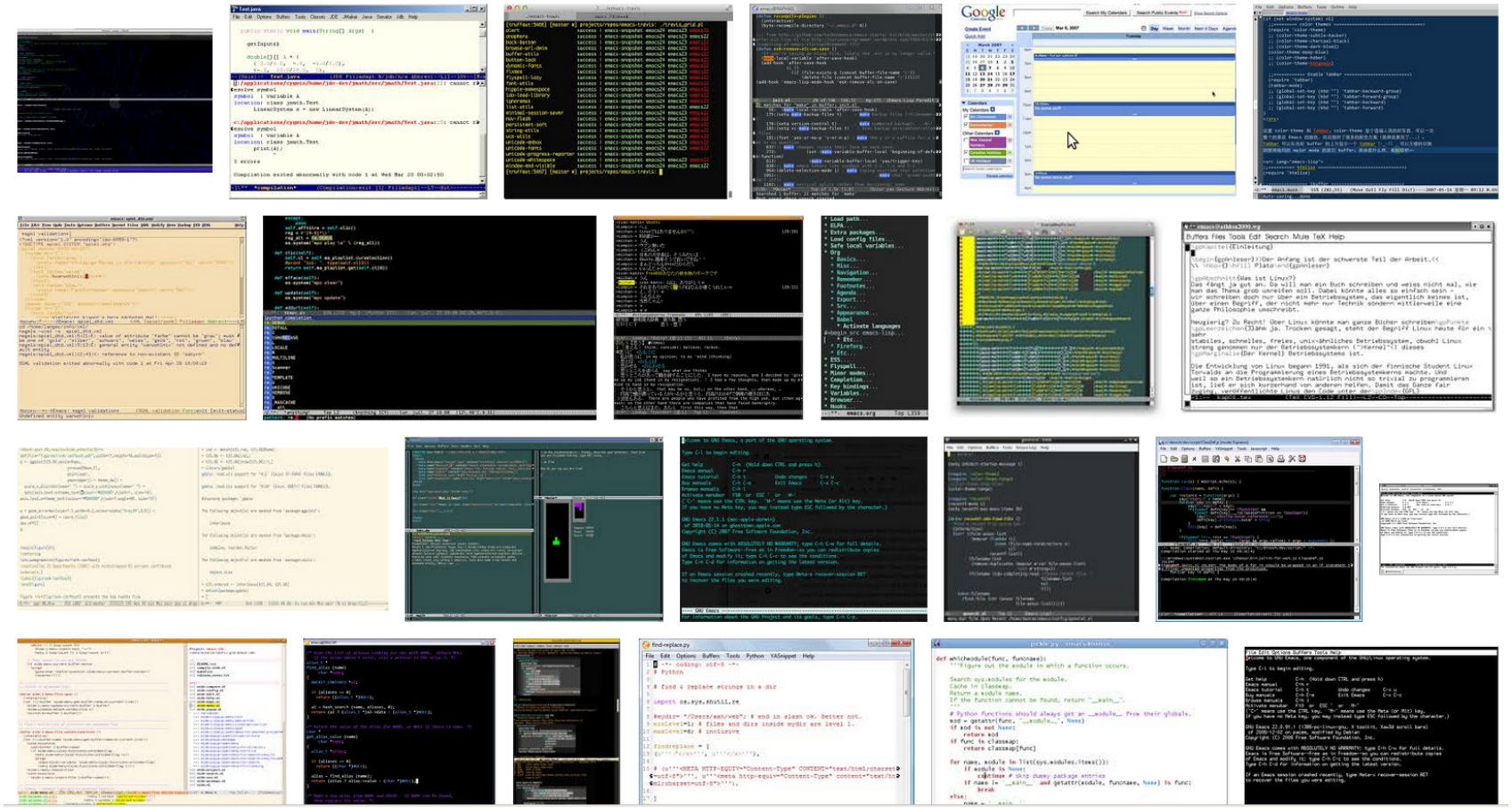
Das Word2vec-Modell: Lernen

Das Word2vec-Modell: Lernen

Das Word2vec-Modell: Lernen

Das Word2vec-Modell: Lernen (Illustration)

Term Embeddings: Code-Beispiel (`word2vec.py`)



- Mini-Text-Korpus *(Bier und Wein in ähnlichen Kontexten)*
- Formeln von oben händisch implementiert.

Embeddings im Skip-Gram-Modell

Liefert das Modell "gute" Embeddings?

Unser Ziel war, dass semantisch ähnliche Terme w_1, w_2 *(z.B. "politician" und "president")* ähnliche Embeddings besitzen sollten. **Ist das so?**

- → Die Terme w_1, w_2 tauchen in oft **ähnlichen Kontexten** auf.
- → Das **Skip-Gram-Netz** muss für beide Terme **ähnliche Ergebnisse** produzieren.
- → Hierzu bildet das Netz beide Terme auf **ähnliche Embeddings** ab, d.h. $e(w_1) \approx e(w_2)$.

Skip-Gram: Anmerkungen und Tricks

Fenstergröße

- Wie wird die **Fenstergröße** des Kontexts gewählt, aus dem wir unsere Termpaare ziehen (z.B. ± 5 *Terme*)?
- Üblicher Weise führen größere Fenster eher zu **thematisch orientierten** Ähnlichkeiten *(“dog”, “bark”, “leash”)*, kleinere Fenster zu **syntaktisch** orientierten Ähnlichkeiten *(“walking”, “approaching”, “driving”)*

Effizienz-Tricks

Wie ermöglichen wir ein effizientes Training auf **großen Eingabedaten**?

1. Identifiziere Phrasen (“Boston Globe”)
2. Reduziere die Samples für häufige Terme (“the”, “said”, ...)

Skip-Gram: Beispiele & Software

Beispiel-Cluster: Welche Terme besitzen ähnliche Embeddings? [3]

Cluster	Wörter
1	flourless babka strudels cheesecake rugelach kugel tortes torte shortcake stollen panettone macaroons pumpernickel cornbread quiches
2	eatery steakhouses restaurant steakhouse eateries
3	cooking bakers baking cooks cook baker chefs cookery recipes cookbooks
4	wines varietals vintages bottlings chardonnays
5	sugo pesce salade beurre boeuf cacciatore tartine moules grissini saltimbocca choucroute pomme primi pommes mignon

Korpusgröße und Dimensionalität[1]

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Software

- Implementierungen: C[2], Java/Scala/Spark[3], Python[4]

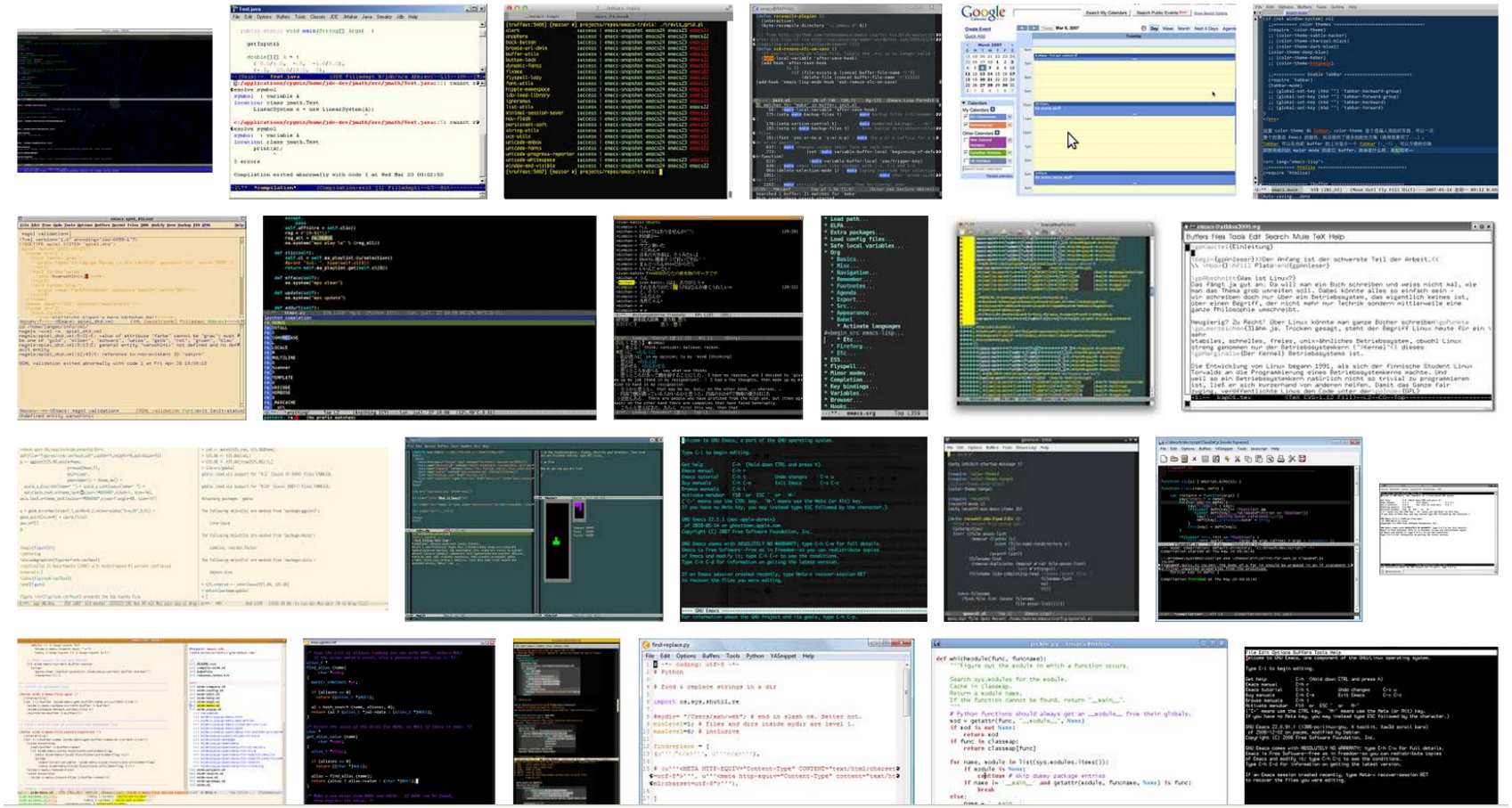
[1] Accuracy gemessen auf Word-Assoziations-Aufgabe: "X verhält sich zu Y wie Z zu ..." [2]

[2] https://github.com/tmikolov/word2vec

[3] https://deeplearning4j.org/word2vec

[4] Gensim: http://radimrehurek.com/gensim, Tensorflow: https://www.tensorflow.org/tutorials/word2vec

Term Embeddings: Code-Beispiel



- Fertig trainierte Vektoren
- kleiner Webservice als Wrapper
- Ähnlichkeiten zwischen Termen

Outline

1. Term Embeddings

2. NLP-Modell 2: CNN

3. NLP-Model 3: Rekurrente Netze

Convolutional Neural Networks

Beispiel: Question Classification

- Bezieht sich eine Frage auf eine Person / einen Ort / eine Mengenangabe / ... ?
- Verschiedene lokale Indikatoren in der Frage zeigen dies an!
  - *" Who was the inventor of the light bulb?"*
  - *" What politician became the first EU president?"*
  - *" Name a wife of Henry VIII."*
- Indikatoren können an verschiedenen Positionen auftauchen
  - *"Who was the inventor of the light bulb?"*
  - *"After World War II, who was BRD's 1st president?"*

Idee: Convolutional Neural Networks (CNN)

- "Suche" nach lokalen Indikatoren mit sogenannten *Filtern* (*"Faltung", engl. "Convolution"*)

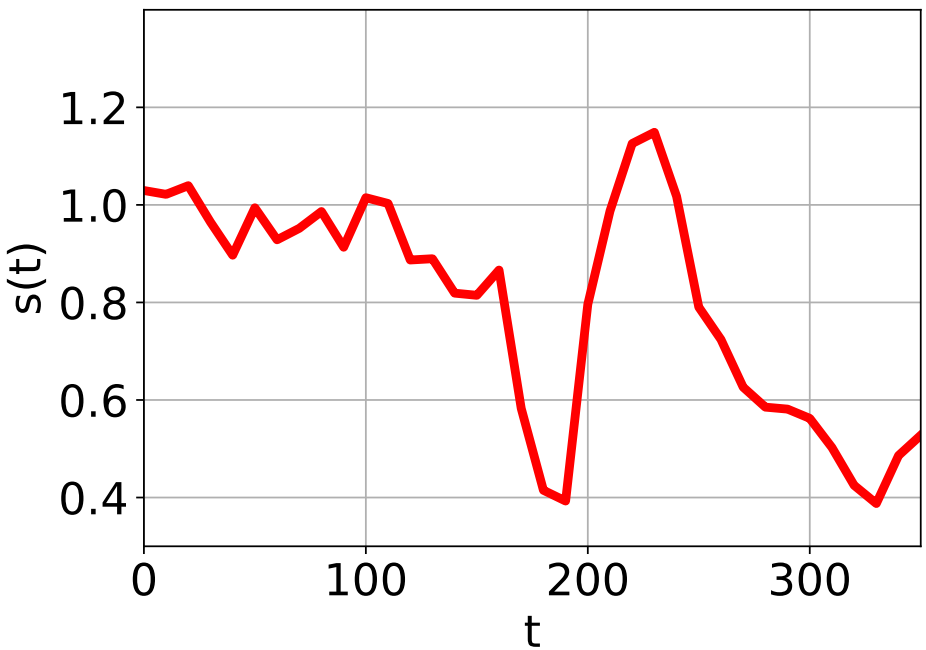
[Signalverarbeitung] Signale und Filter

Definition (Signal)

Gegeben $M \in \mathbb{N}^+$, *nennen wir* $s : \mathbb{Z} \to \{1, ..., M\}$
ein (diskretes 1D-) **Signal**.

Anmerkungen

- Konzeptuell hat das Signal keinen Anfang und kein Ende, in der Praxis schon.
- **Beispiele**: Audio-Signale, Zeitreihen, Merkmals-Vektoren



[Signalverarbeitung] Signale und Filter

Definition ((FIR-)Filter)

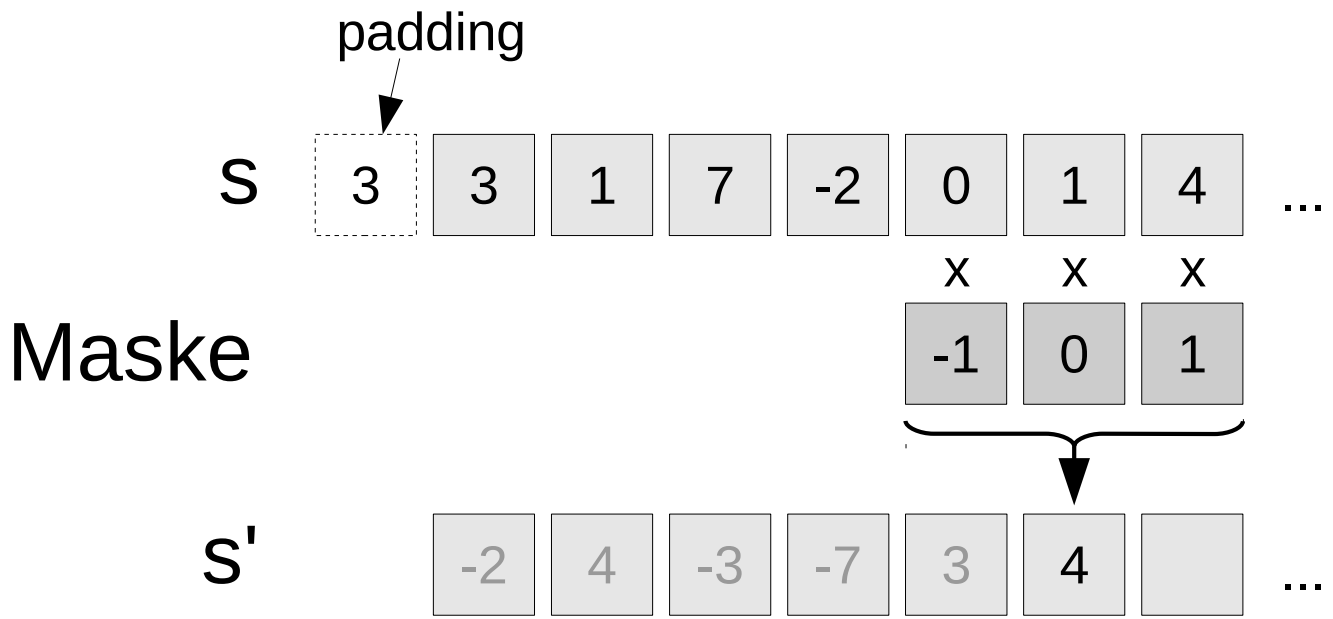
Es sei s ein Signal, $M \in \mathbb{N}$, und

$$(w_{-M}, w_{-M+1}, \ldots, w_{-1}, w_0, w_1, \ldots, w_M) \in \mathbb{R}^{2M+1}$$

eine sogenannte **Filter-Maske**. *Dann nennen wir die Abbildung $s \mapsto s'$ mit*

$$s'(t) = \sum_{\tau=-M}^{M} s(t-\tau) \cdot w_\tau$$

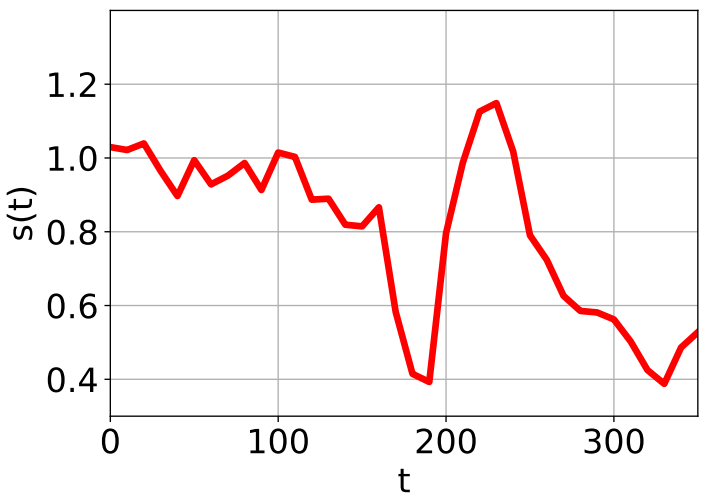
ein~~en~~(!) Finite-Impulse-Response (FIR)-Filter.

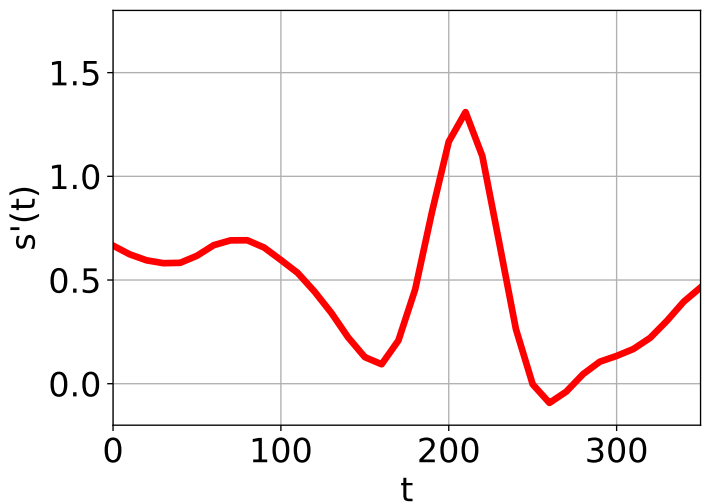


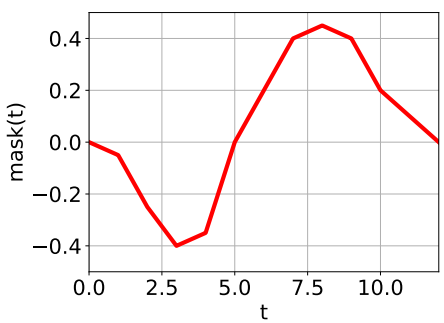
Filter = Detektoren

- Die Werte des gefilterten Signals s' sind dort am **größten**, wo das Eingabesignal s am besten auf das Filter **“passt”**.
- Indem wir die **Filtermaske geschickt** wählen, wird das Filter zu einem **Detektor**.

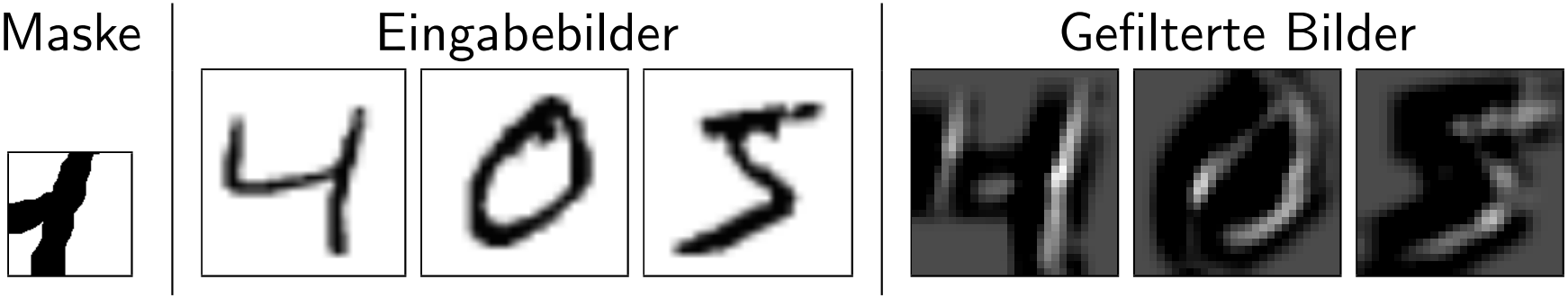
Beispiel (1D)







Filter = Detektoren (Beispiel in 2D)



CNN-Idee

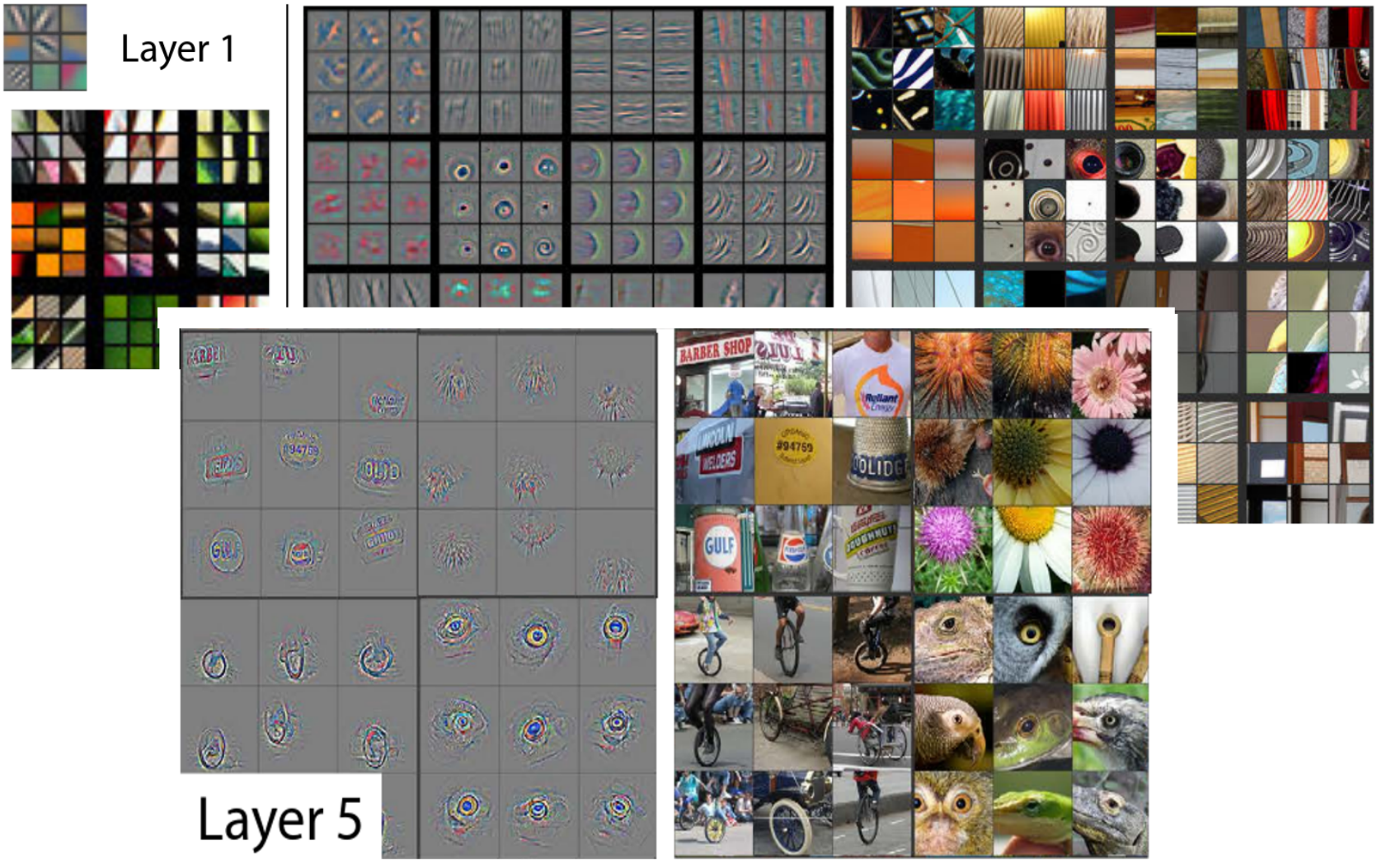
- **Schicht 1**: Filtere das Bild mit Masken / Detektoren
- **Schicht 2**: Klassifiziere auf Basis des gefilterten Signals
- **CNNs lernen ihre Filter mittels Backpropagation!**

Anmerkungen

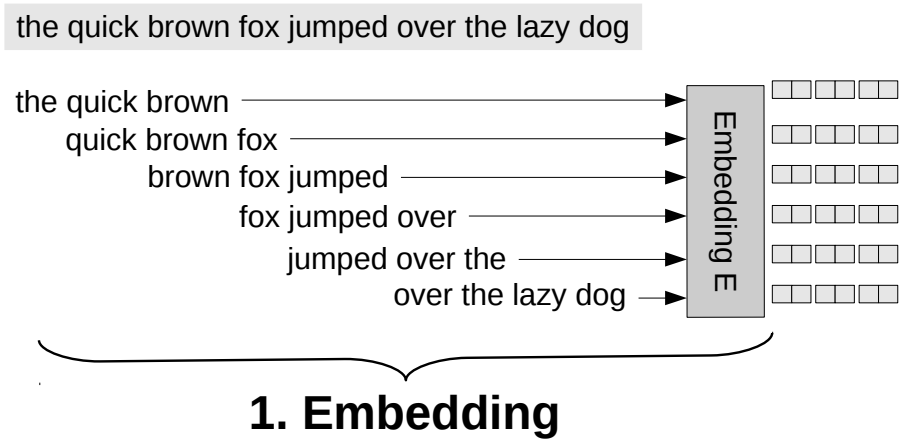
- CNNs kommen eigentlich aus dem Bereich der **Bildanalyse**. Dort haben sie in den letzten 5 Jahren die **Erkennung von Objekten in Bildern** revolutioniert.
- CNNs sind ein Kernkonzept des aktuellen Trends **“Deep Learning”**.

Deep CNNs Bilder: [4]

Durch das **Stacking** mehrer Schichten können wir immer **abstraktere Merkmale** entdecken!



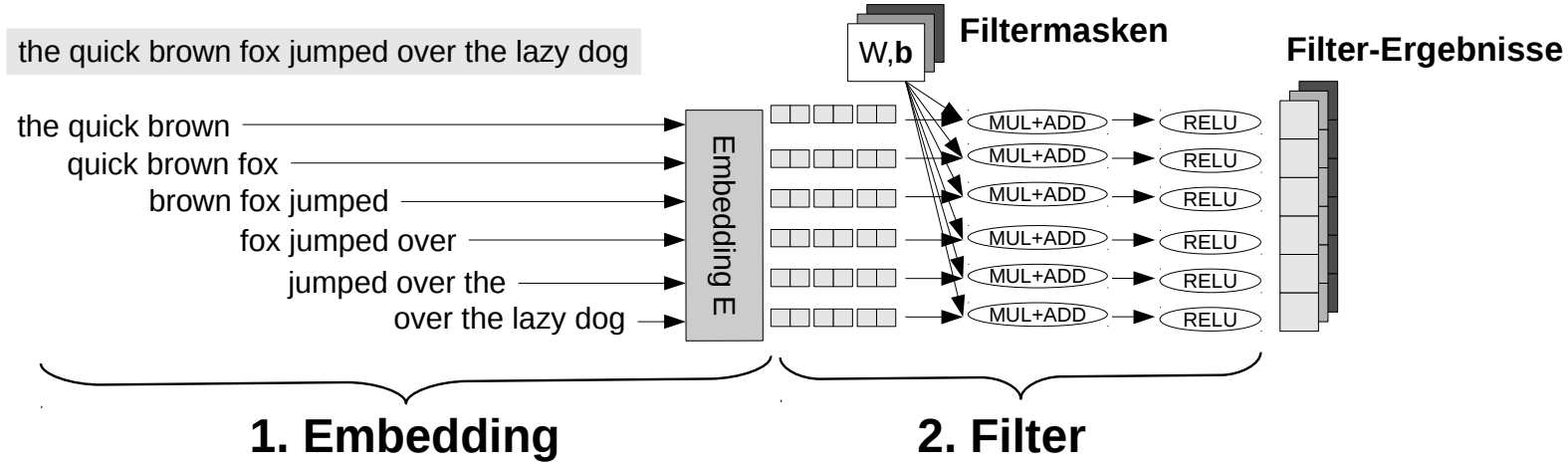
CNN-Beispiel-Architektur



1. Embedding

- Bilde n-Gramme der Größe $2M + 1$ *(hier: $M = 1$)*
- Ermittle das Embedding jedes Terms *(hier: $e(''fox'') \in \mathbb{R}^2$)*
- Konkateniere die Embeddings jedes n-Gramms
  (hier: $e(''quick'') \circ (''brown'') \circ (''fox'') \in \mathbb{R}^6$)

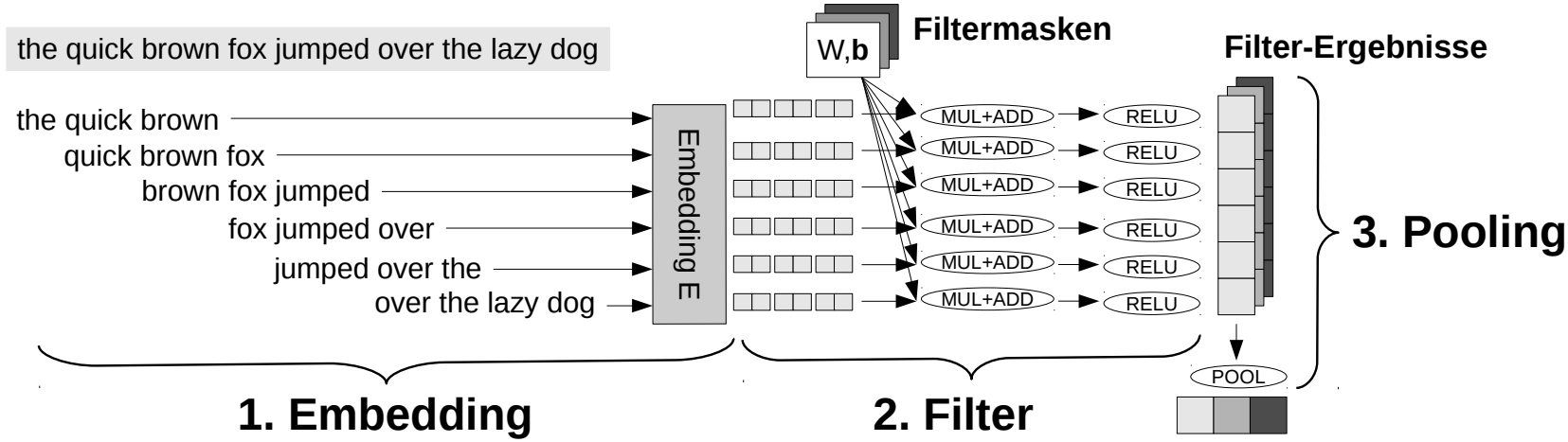
CNN-Beispiel-Architektur



2. Filter

- Wende eine **Filtermaske** $\mathbf{w} \in \mathbb{R}^6$ an *(+ Bias, Aktivierung)*
- Die Maske detektiert für jedes Fenster, ob ein **lokaler Indikator** vorliegt *(z.B. "who was ..."?)*
- Wir erhalten für jedes n-Gramm eine **Aktivierung** $\in \mathbb{R}$.
- Wir wenden **mehrere Filter** an, die **verschiedene Indikatoren** entdecken *(hier: 3 Filter = Grau-Schattierungen)*.

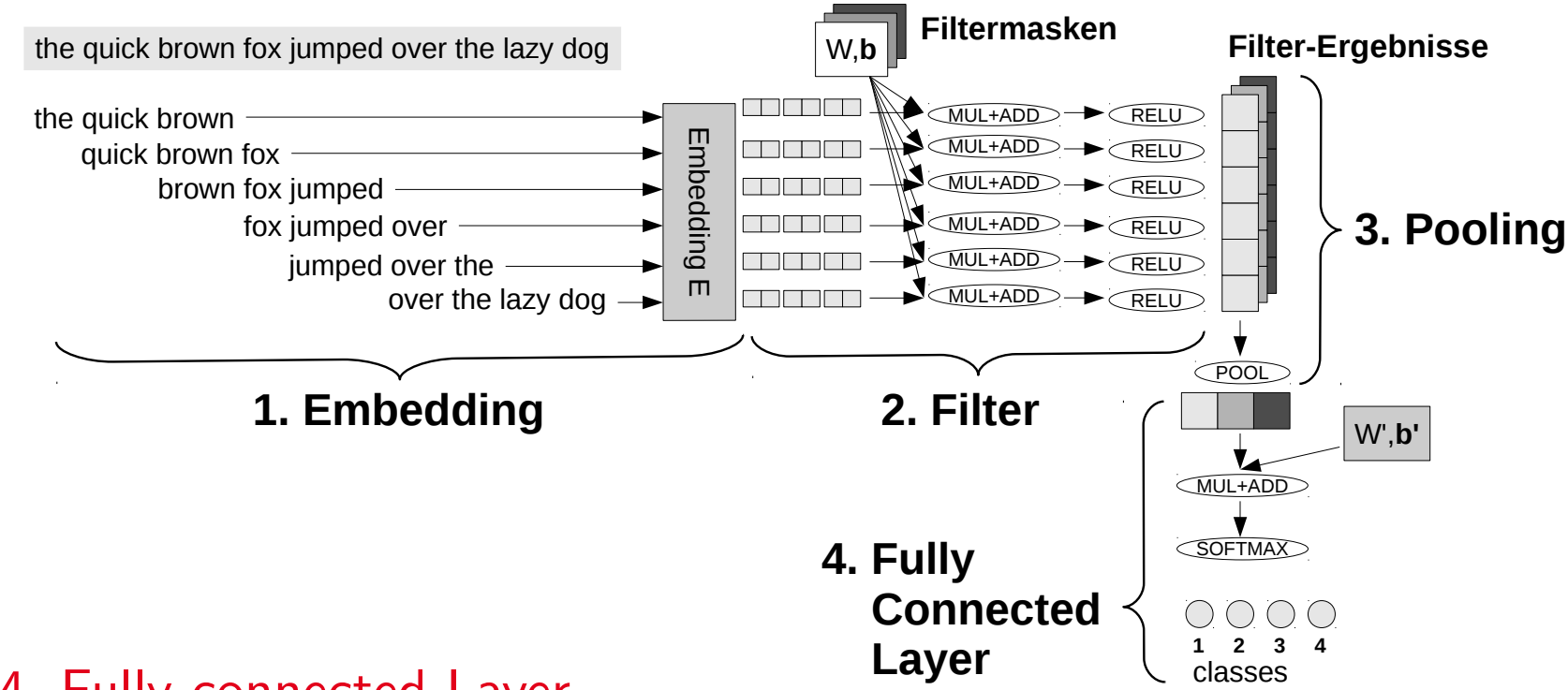
CNN-Beispiel-Architektur



3. Pooling

- Uns interessiert **welche Filter** für den Eingabesatz hohe Aktivierungen liefern, aber nicht **wo** im Satz.
- Deshalb wenden wir ein sogenanntes **Max-Pooling** an: Für jedes Filter reduzieren wir sämtliche Aktivierungen auf ihr **Maximum**.
- Aus 6 Werten (da 6 n-Gramme) wird **einer**.

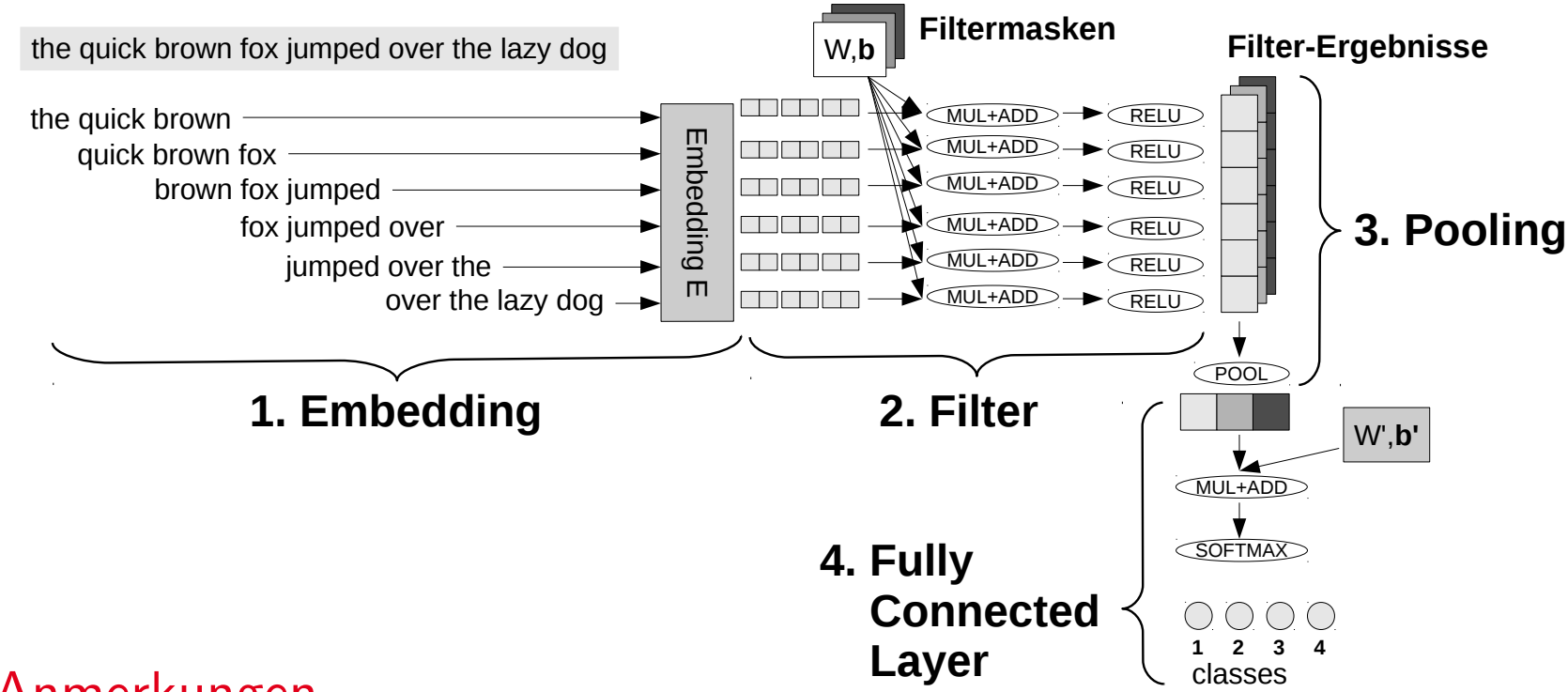
CNN-Beispiel-Architektur



4. Fully-connected Layer

- Die gepoolten Aktivierungen sagen aus, auf **welche Filter** der Eingabesatz anspricht *(d.h., welche* **lokalen Indikatoren** *vorhanden sind)*.
- Die letzte Schicht fällt auf Basis dieser Werte die Entscheidung für eine **Klasse** *(z.B. "PERSON")*.

CNN-Beispiel-Architektur



Anmerkungen

- Das komplette Netz wird mittels **Backpropagation** aus gelabelten Beispielsätzen **gelernt**!
- Dies betrifft das **Embedding** (E), sämtliche **Filter** ($W, \mathbf{b}$) und die **finale Schicht** ($W', \mathbf{b}'$).
- Durch die Embeddings **generalisiert** das Netz: Spricht ein Filter an auf "Welcher **Präsident** war..", dann auch auf "Welcher **Politiker** war..."?

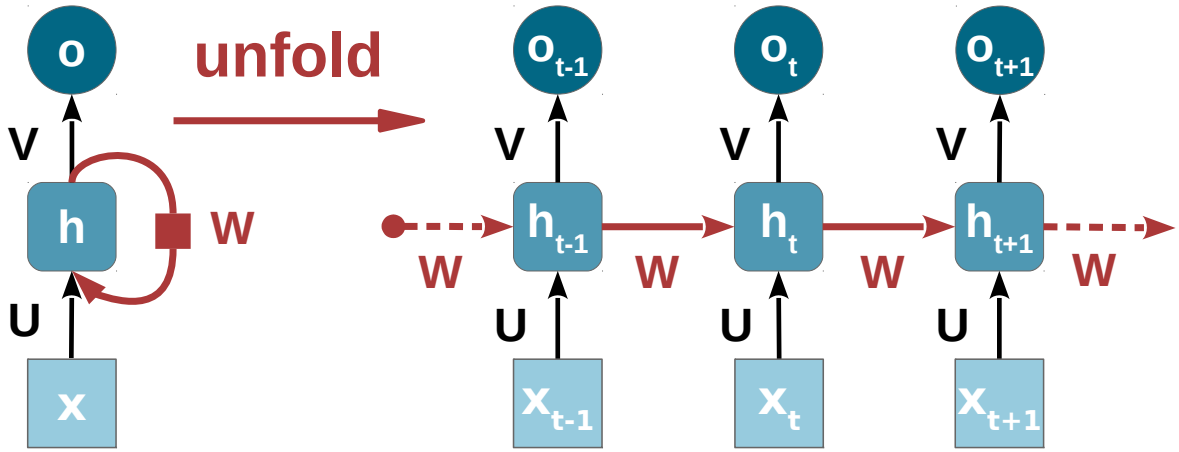
Outline

1. Term Embeddings

2. NLP-Modell 2: CNN

3. NLP-Model 3: Rekurrente Netze

Rekurrente Netze



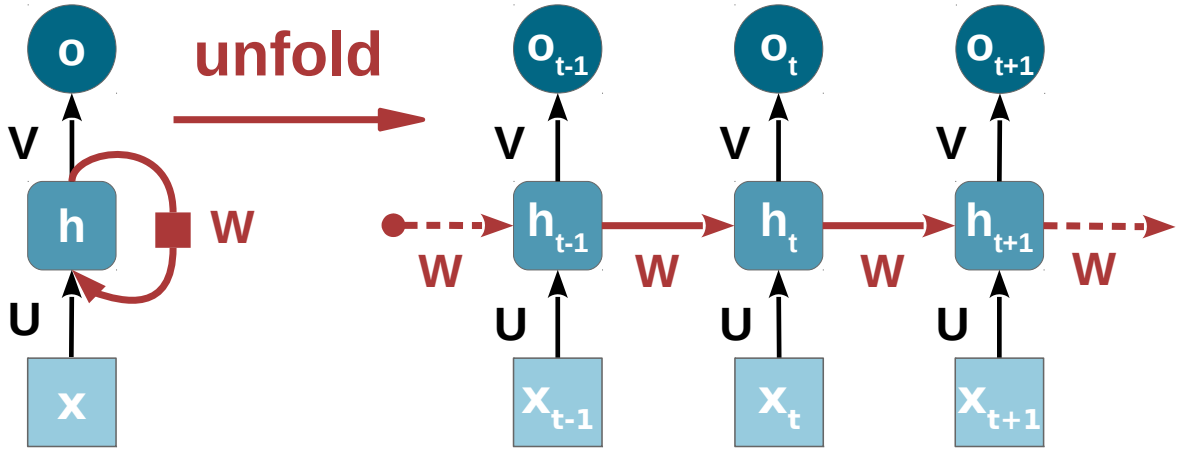
Motivation

- Bisherige Modelle verarbeiten nur **einzelne Items** (Embeddings) bzw. lokale Umgebungen / n-Gramme (CNNs).
- Um einen ganzen Satz zu verstehen, müsste man eigentlich die **komplette Wortsequenz** berücksichtigen.

What animal, after..., was discovered to breed in caves 1978?

- **Rekurrente Netze** verarbeiten komplette Eingaben sequentiell!

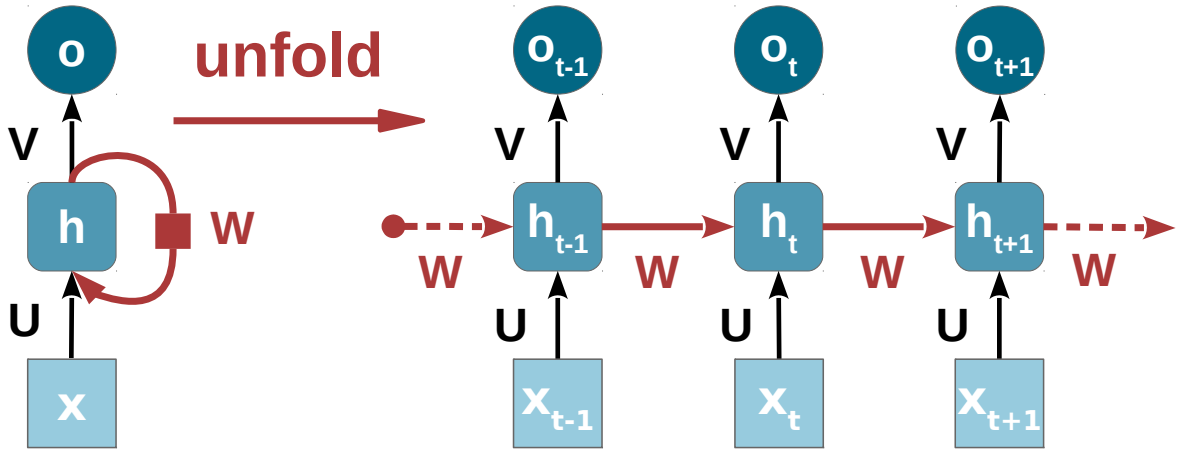
Rekurrente Netze: Vorgehen



Modell (Einfaches Elman Network)

- Gegeben ist eine Sequenz von Eingabevektoren $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ *(z.B. die Embeddings zu den Termen eines Satzes).*
- Das Netz generiert Ausgabevektoren $\mathbf{o}_1, ..., \mathbf{o}_n$.
- Interne Zustandsvektoren $\mathbf{h}_1, ..., \mathbf{h}_n$ codieren die *(für das Problem relevante)* Historie zum jeweiligen Zeitpunkt:
  - War das letzte Wort "not"? Kam bereits "who" im Satz vor?
  - Beginnt gerade ein neuer Nebensatz? ...

Rekurrente Netze: Vorgehen



Verarbeitungsschritte (Elman Network) zum Zeitpunkt t

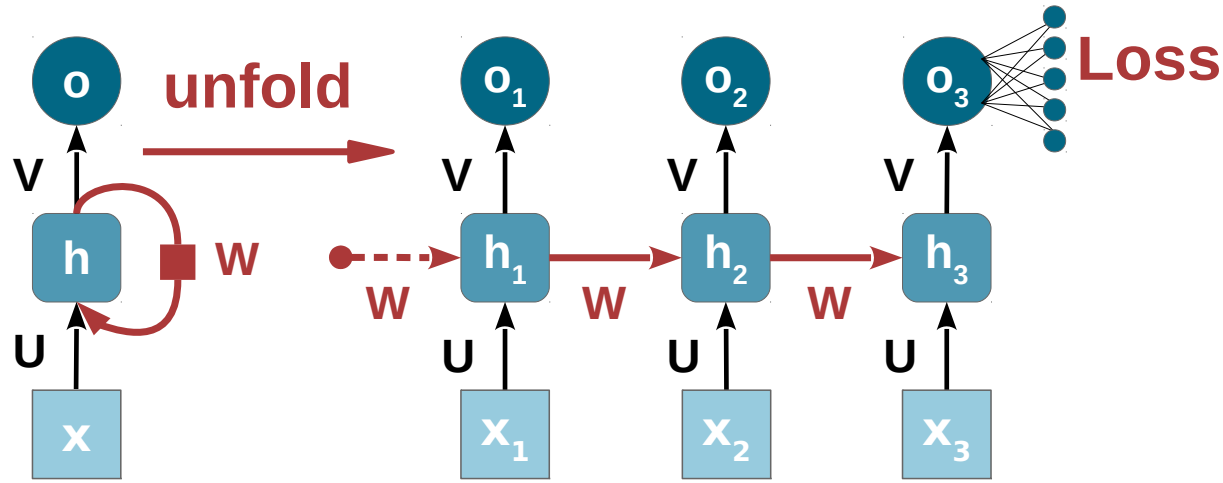
1. Berechne den neuen internen Zustand $\mathbf{h}_t$

$$\mathbf{h}_t = f_h\Big(W \cdot \mathbf{h}_{t-1} + U \cdot \mathbf{x}_t + \mathbf{b}_h\Big)$$

2. Berechne die Ausgabe $\mathbf{o}_t$

$$\mathbf{o}_t = f_o\Big(V \cdot \mathbf{h}_t + \mathbf{b}_o\Big)$$

Rekurrente Netze: Vorgehen



Rekurrente Netze: Lernen

- Welche **Information** in den Zuständen $\mathbf{h}_t$ gespeichert wird, wird per Backpropagation gelernt!
- Zu **lernen**: Matrizen U, V, W *(und Biases* $\mathbf{b}_h, \mathbf{b}_o$*)*.
- Wir stellen uns hierzu das Netz "aufgefaltet" *(engl. unfolded)* vor. Zu jedem Zeitpunkt t wird für jeden Parameter eine Ableitung berechnet *(und diese werden addiert)*.
- Dieses Verfahren heißt "**Backpropagation-Through-Time**".

Rekurrente Netze: Variante

Herausforderung: Abhängigkeiten über große Distanz

> *"The* **clouds***, which passed along so slowly that [...], lingered in the* **sky***."*

- Wie heben wir Informationen möglichst lange auf? Das Modell muss sein "**Erinnern**" und "**Vergessen**" managen!

Übliche Lösung: **Gates**

- Wir entscheiden (für jede Dimension), wieviel Prozent des internen Zustandes **erinnert** werden sollen ...

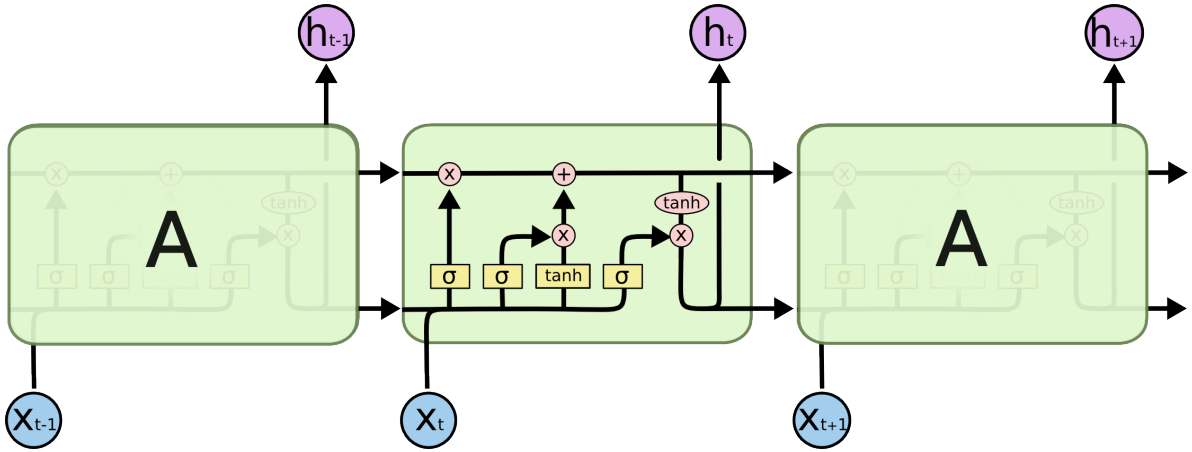
$$\mathbf{g}_t = \sigma\Big(W' \cdot \mathbf{h}_{t-1} + U' \cdot \mathbf{x}_t + \mathbf{b}'_h\Big)$$

- ... gewichten den Zustand mit dem Wert des Gates $\mathbf{g}_t$ ...

$$\mathbf{h}'_t := \mathbf{h}_t \odot \mathbf{g}_t$$

- ... und verwenden ab nun $\mathbf{h}'_t$.
- Das Gate-Verhalten wird per **Backpropagation** mitgelernt!

Long-Short-Term Memory Networks (LSTMs)[5]

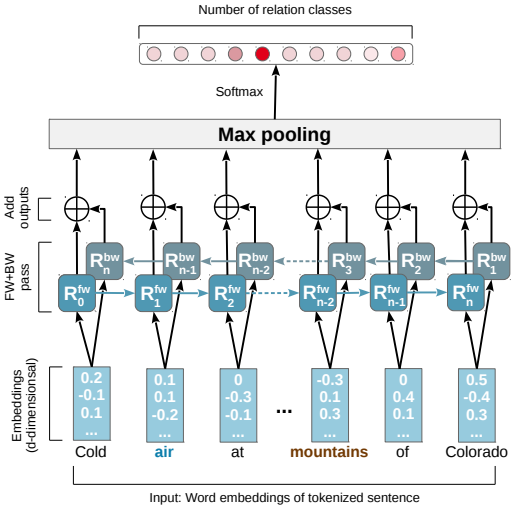


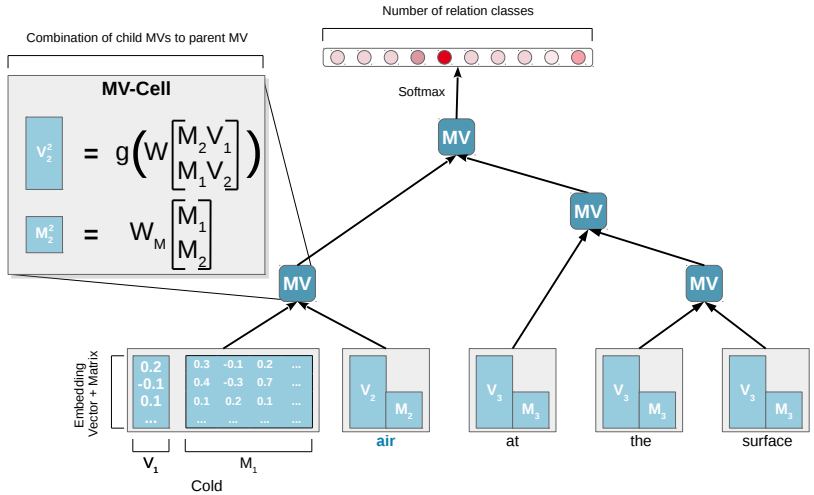
Long-Short-Term Memory Networks (LSTMs)

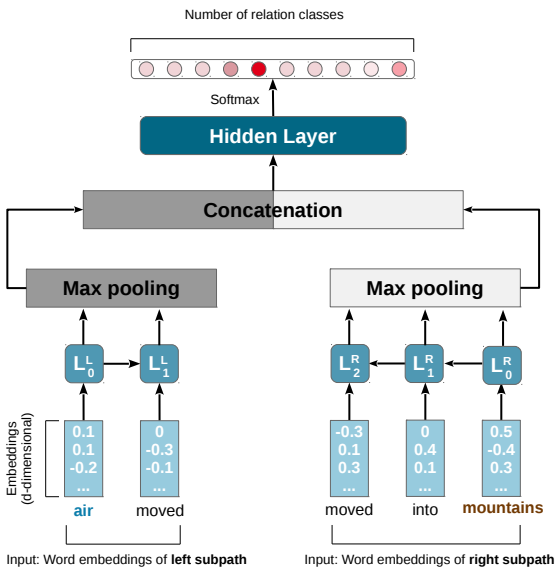
- Eine sehr erfolgreiche rekurrente Standard-Architektur mit drei verschiedenen Gates.
- Die **Gates** bestimmen ...
  - ... was vom neuen Input $\mathbf{x}_t$ ignoriert wird *(input gate)*
  - ... was vom alten Zustand $\mathbf{h}_{t-1}$ behalten wird *(forget gate)*
  - ... was vom alten Zustand $\mathbf{h}_{t-1}$ die Ausgabe beeinflusst *(output gate)*
- Einfachere Varianten: GRUs *(Gated Recurrent Units)*.

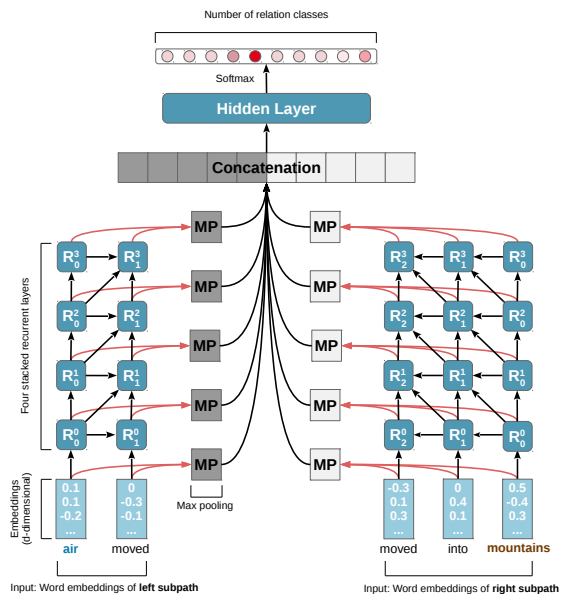
[5] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Ausblick: Relation Classification mit Neural Networks[6]









[6] Master-Projekt Markus Eberts, Felix Hamann (WS17/18)

References I

[1] McCormick, Chris.
A Primer on Neural Network Models for Natural Language Processing.
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/, retrieved: Apr 2017).

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean.
Efficient estimation of word representations in vector space.
CoRR, abs/1301.3781, 2013.

[3] Stephan Richter.
Word Embeddings für Information Retrieval und die Klassifikation von Dokumenten.
Bachelor's Thesis, RheinMain University of Applied Sciences, 2017.

[4] Matthew D. Zeiler and Rob Fergus.
Visualizing and Understanding Convolutional Networks.
CoRR, abs/1311.2901, 2013.