

Recurrent Neural Networks

(Rekurrentes neuronales Netz)

Übersicht

- Wir haben Neuronale Netze verwendet, um Klassifizierungs- und Regressionsprobleme zu lösen, aber wir haben noch nicht gesehen, wie Neuronale Netze mit Sequenzinformationen umgehen können.
- Hierfür verwenden wir Recurrent Neural Networks (RNN).

Inhalt



- RNN Theorie
- Basis-Handbuch RNN
- Vanishing Gradients (Verschwindende Gradienten)
- LSTM und GRU Einheiten
- Zeitreihen mit RNN
- Zeitreihen-Übung / Lösungen
- Wort2Vec Algorithmus

Los geht's!

Recurrent Neural Networks

Theorie

Beispiele für Sequenzen



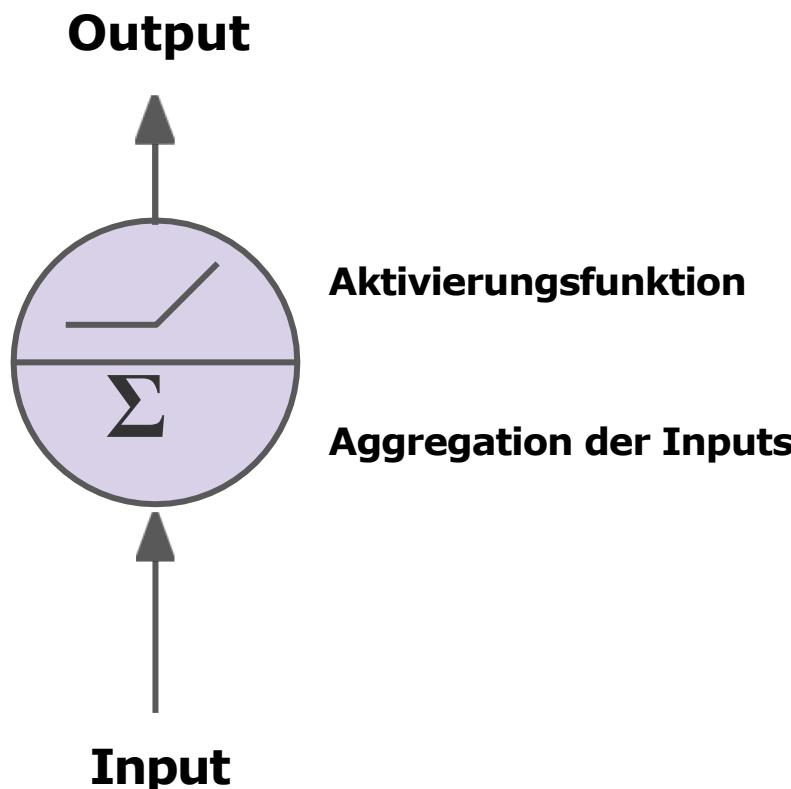
- Zeitreihen-Daten (Verkauf)
- Sätze
- Audio/Musik
- Auto Trajektorien
- Maschinendaten

Recurrent Neural Networks

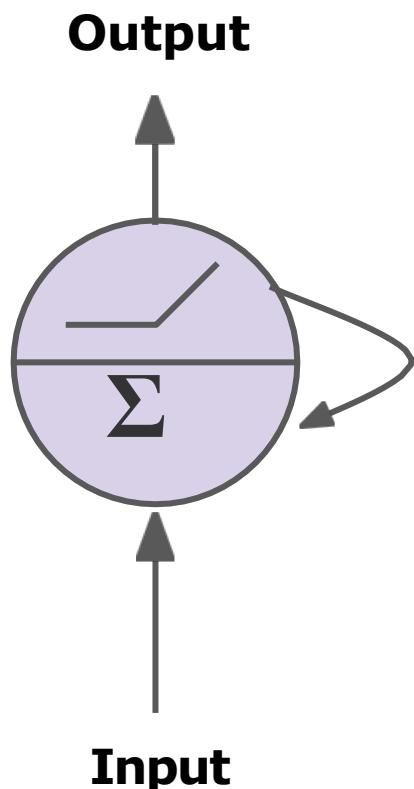


- Stellen wir uns eine Sequenz vor:
[1,2,3,4,5,6]
- Wäre es möglich, eine ähnliche Sequenz vorherzusagen, die um einen Zeitschritt in die Zukunft verschoben wurde?
[2,3,4,5,6,7]

Normales Neuron im Feed Forward Netzwerk

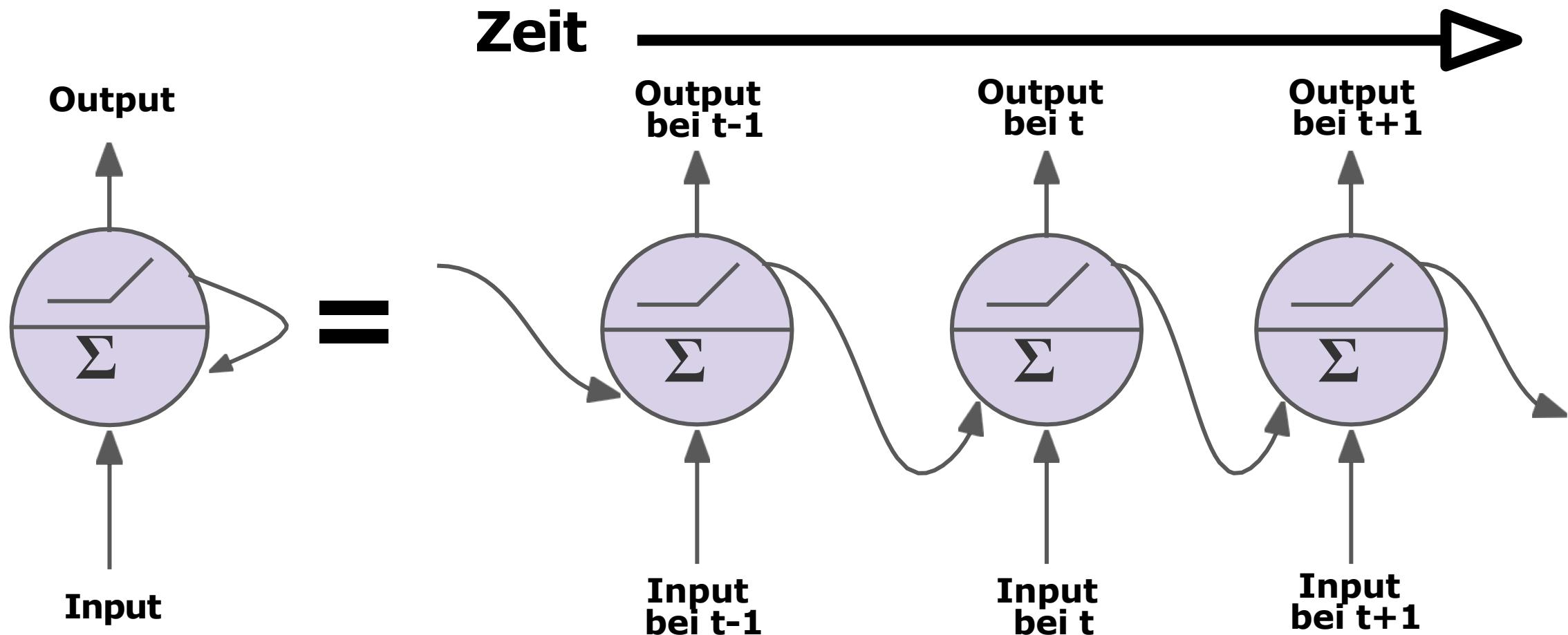


Rekurrentes Neuron



- Sendet die Ausgabe an sich selbst zurück!
- Mal sehen, wie das im Laufe der Zeit aussieht.

Rekurrentes Neuron



Memory Cells (Speicherzellen)



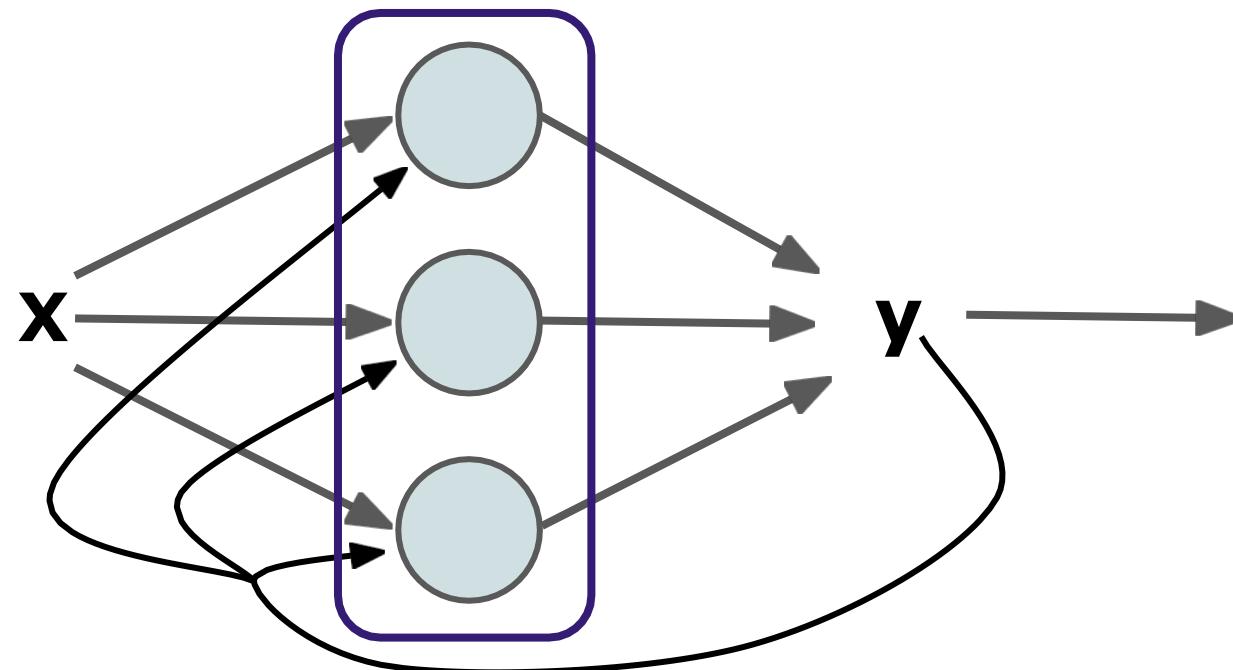
- Zellen, die eine Funktion von Inputs aus früheren Zeitschritten sind, werden auch als *memory cells* (Speicherzellen) bezeichnet.
- RNN sind auch in ihren Ein- und Ausgängen flexibel, sowohl für Sequenzen als auch für einzelne Vektorwerte.

Recurrent Neurons (Rekurrente Neuronen)

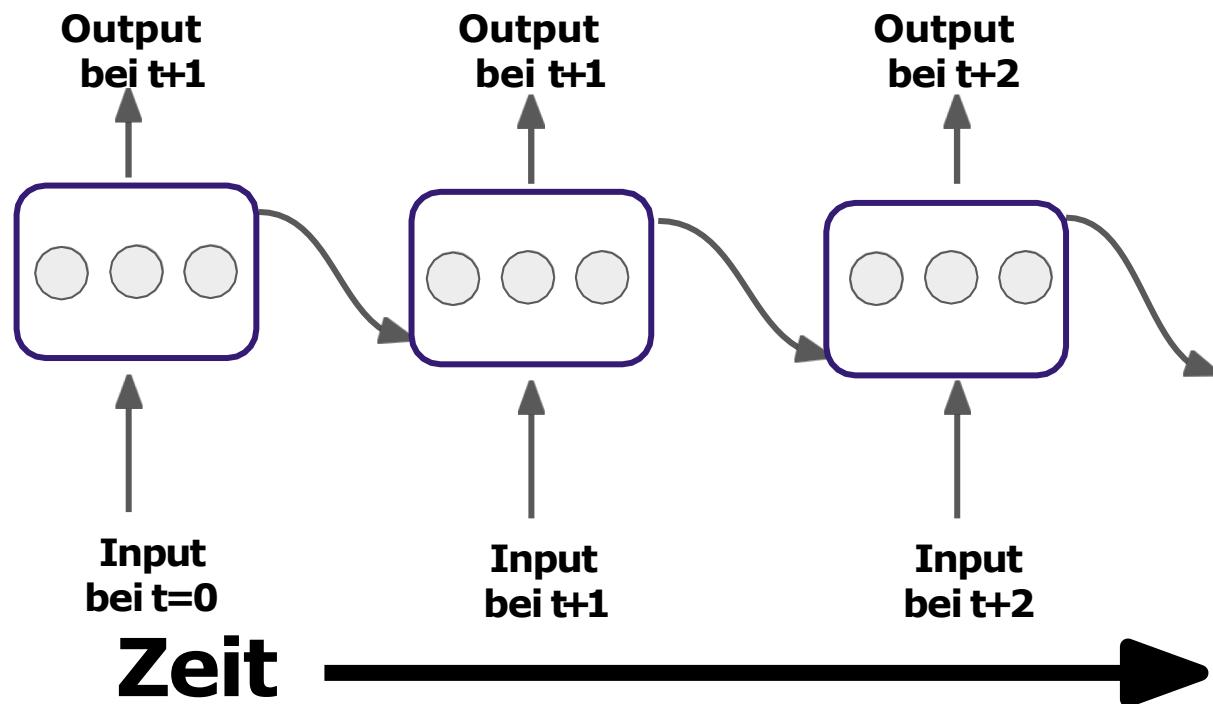


- Wir können auch ganze Schichten von Rekurrenten Neuronen erzeugen....

RNN-Schicht mit 3 Neuronen



“Unrolled” Layer

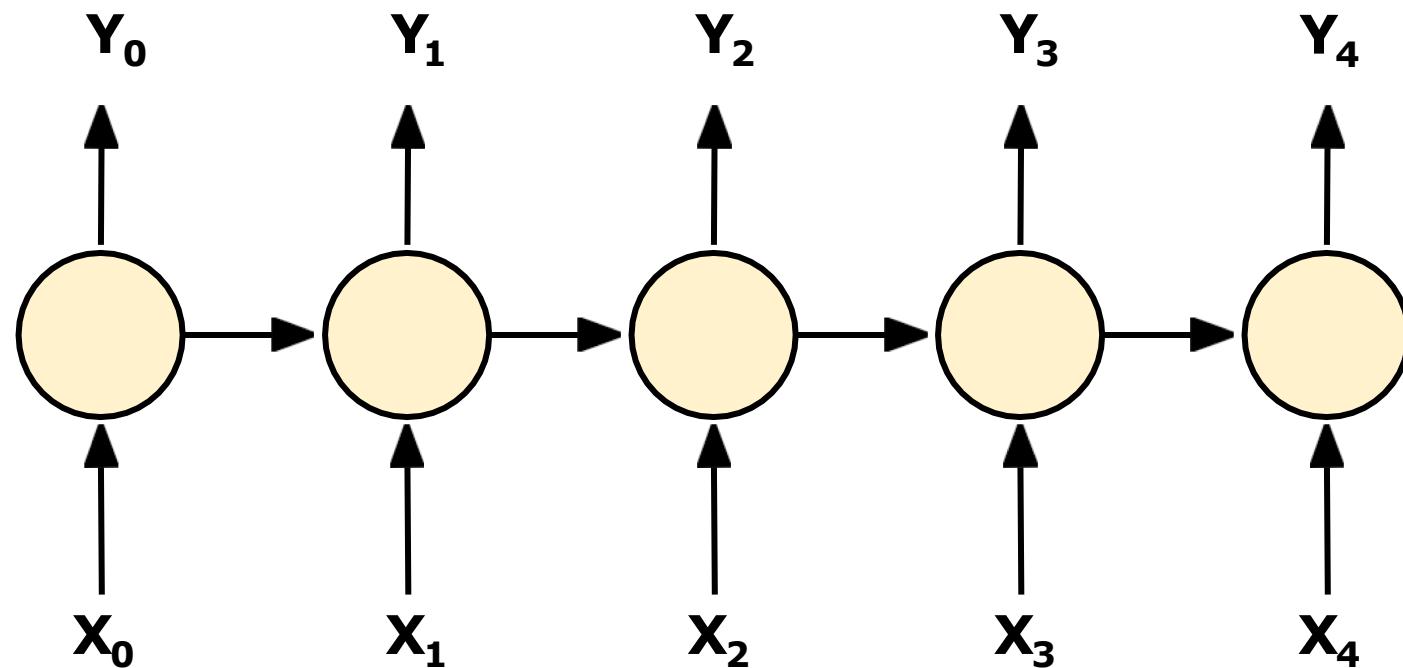


Recurrent Neural Networks

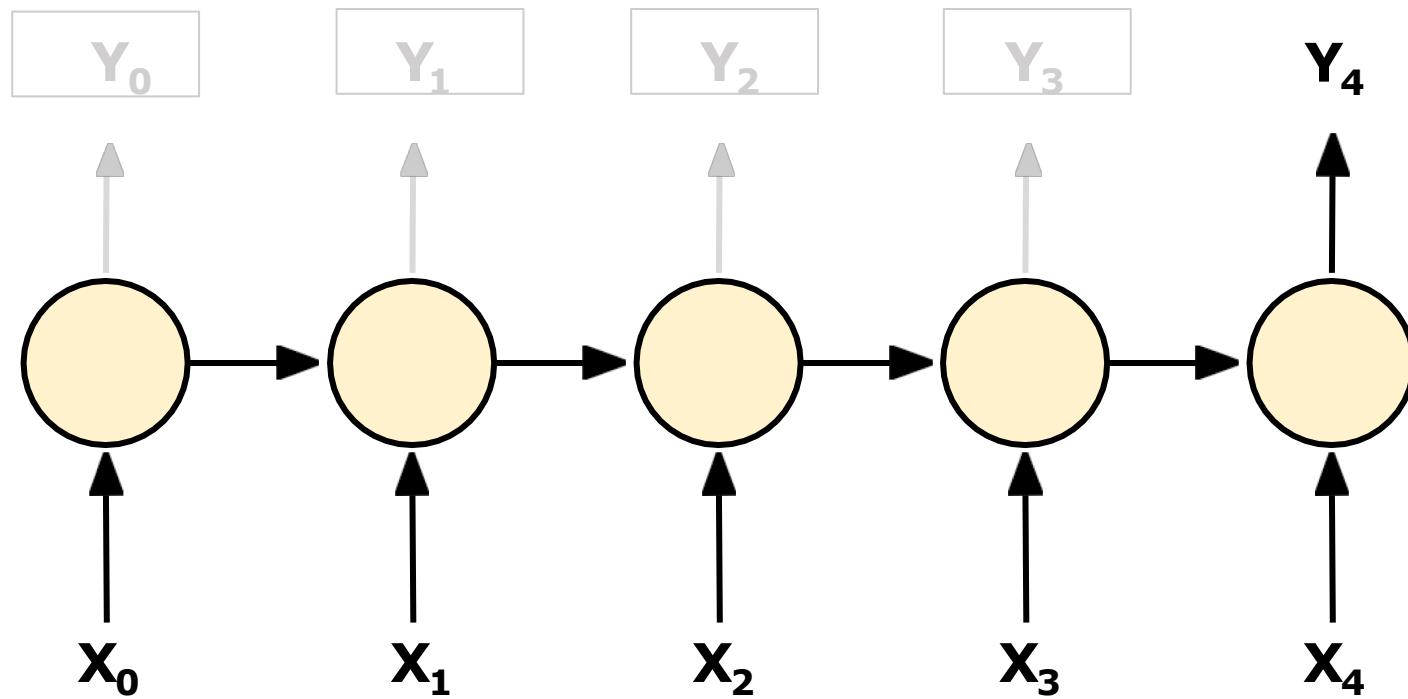


- RNN sind auch in ihren Ein- und Ausgängen sehr flexibel.
- Lass uns ein paar Beispiele anschauen.

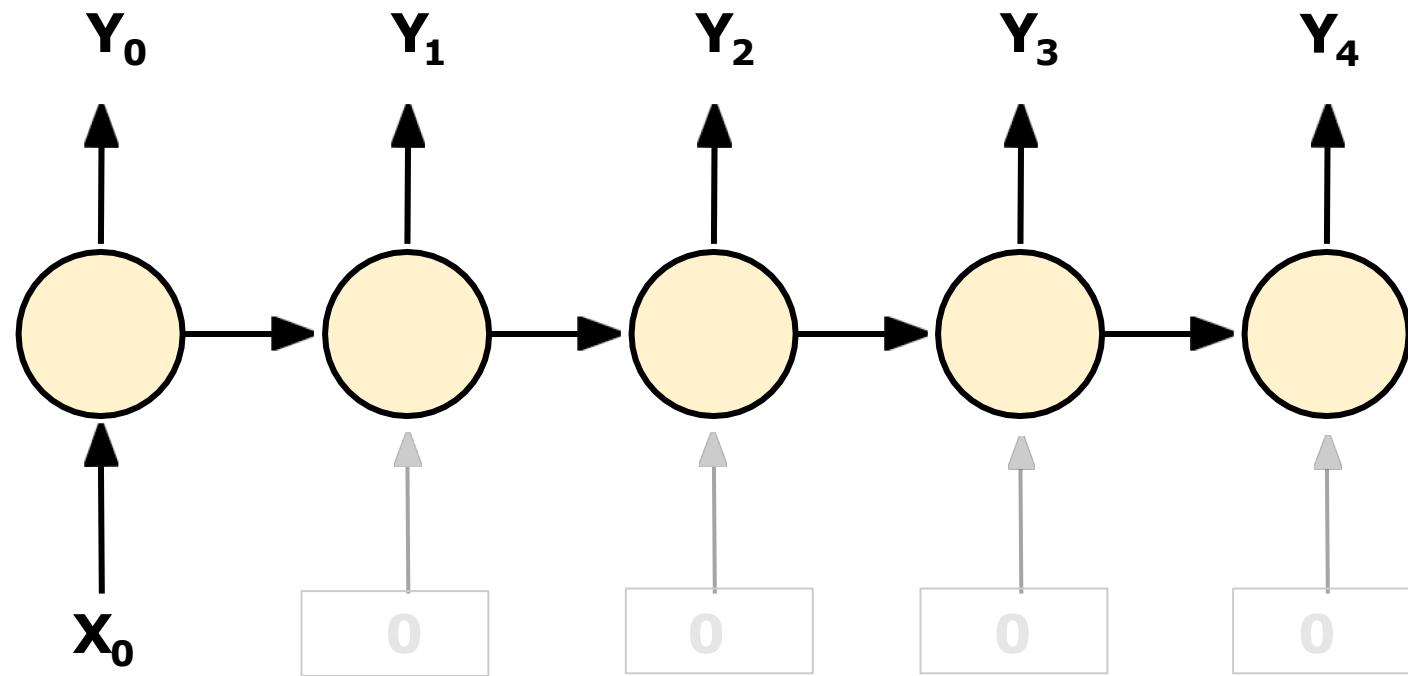
Sequenz zu Sequenz



Sequenz zu Vektor



Vektor zu Sequenz



Recurrent Neural Network Modell



- Lass uns untersuchen, wie wir ein einfaches RNN-Modell in TensorFlow manuell erstellen können.
- Danach werden wir sehen, wie man TensorFlow's eingebaute RNN API-Klassen verwendet.

Recurrent Neural Networks

Manuelle Erstellung

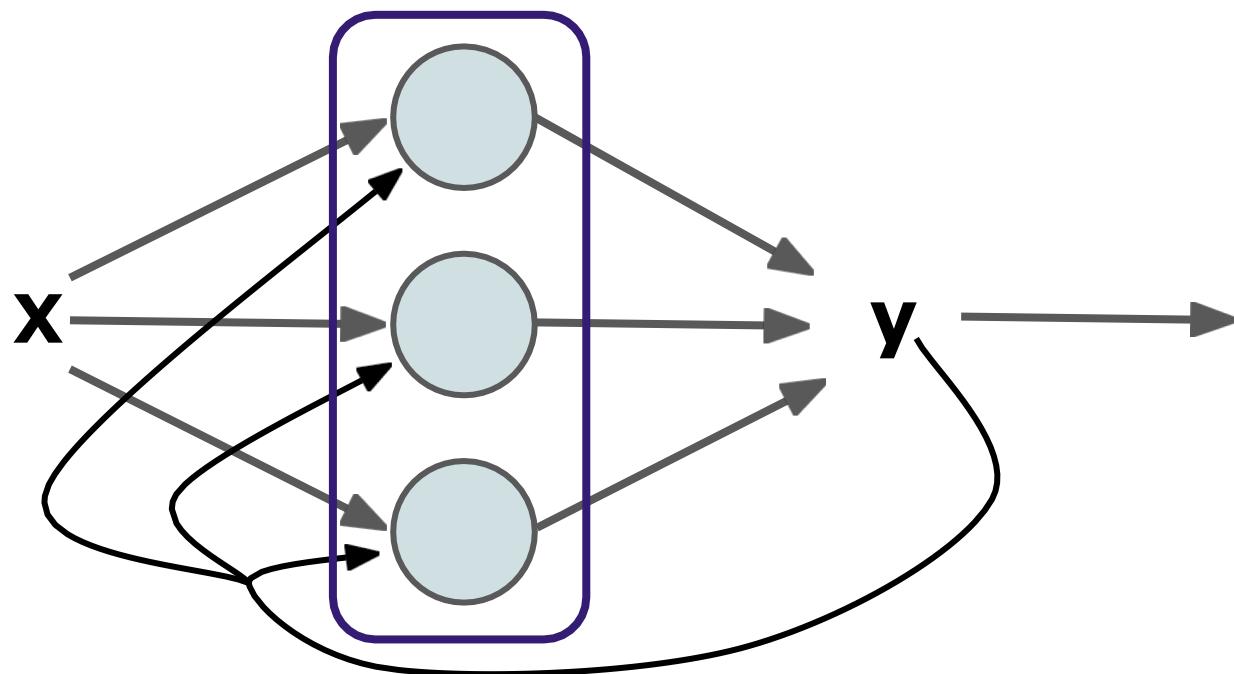
Übersicht



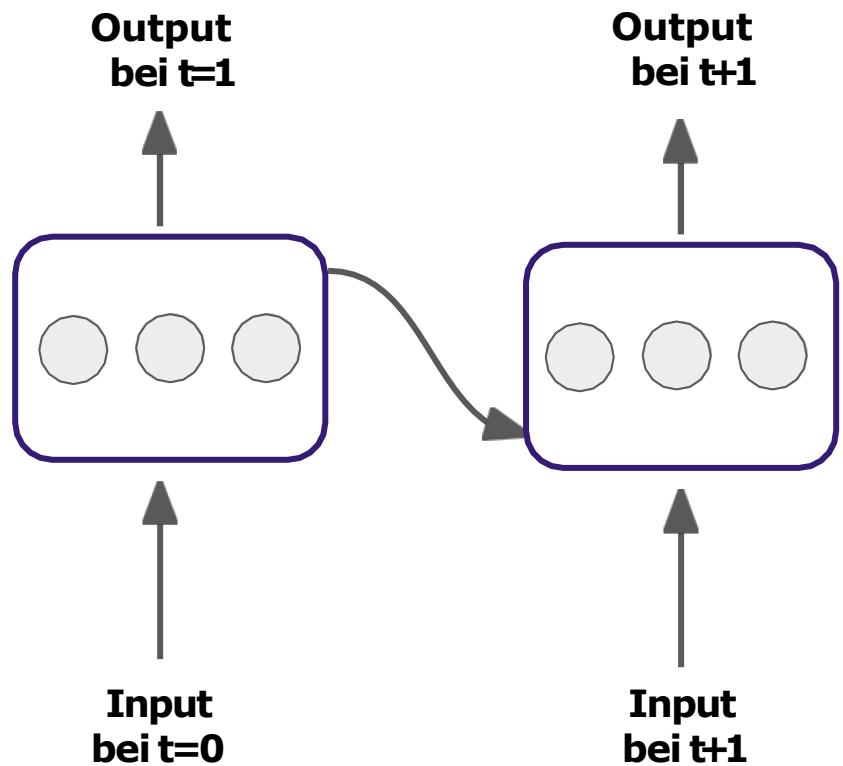
- In dieser Lektion erstellen wir manuell eine 3 Neuronen RNN Schicht mit TensorFlow.
- Im Vordergrund steht dabei das Eingabeformat der Daten.

RNN mit TensorFlow

- Wir bauen die folgende RNN-Schicht:



„Unrolled“ Layer



RNN

- Wir beginnen mit der Ausführung der RNN für 2 Datensätze, $t=0$ und $t=1$.
- Jedes rekurrierende Neuron hat 2 Sets von Gewichten:
 - W_x für Eingangsgewichte auf X
 - W_y für Gewichte am Ausgang des ursprünglichen X

Beispiel für RNN Daten

t=0	t=1	t=2	t=3	t=4
[The,	brown,	fox,	is,	quick]
[The,	red,	fox,	jumped,	high]

```
words_in_dataset[0] = [The, The]
words_in_dataset[1] = [brown, red]
words_in_dataset[2] = [fox, fox]
words_in_dataset[3] = [is, jumped]
words_in_dataset[4] = [quick, high]
```

num_batches = 5, batch_size = 2, time_steps = 5

Vanishing Gradients (Verschwindende Gradienten)

Vanishing Gradients

- Die Backpropagation (Rückführung) erfolgt rückwärts von der Ausgabe- zur Eingabeschicht, wodurch sich der Fehlergradient ausbreitet.
- Bei tieferen Netzwerken (Deep Networks) können Probleme durch *Backproagation, Vanishing und Exploding* Gradienten entstehen.

Vanishing Gradients



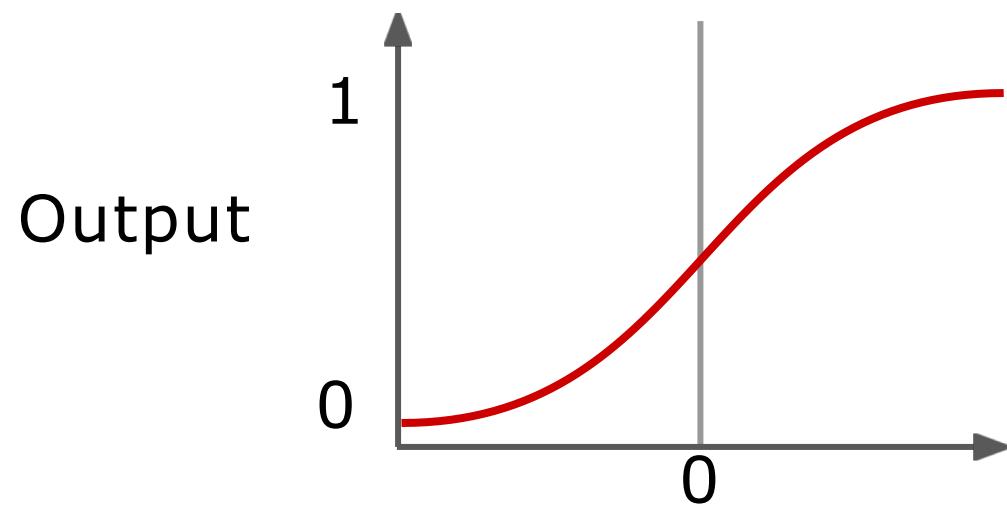
- Wenn man zu den "unteren" Schichten zurück geht, werden die Gradienten oft kleiner, was dazu führt, dass sich die Gewichtungen auf den unteren Ebenen nie ändern.
- Das Gegenteil kann auch vorkommen, Gradienten „explodieren“ auf dem Rückweg und verursachen Probleme.

Vanishing Gradients



- Lass uns nun besprechen, warum das passieren könnte und wie wir es beheben können.
- Im nächsten Teil der Vorlesung werden wir dann diskutieren, wie sich diese Probleme speziell auf RNN auswirken und wie man LSTM und GRU verwendet, um sie zu beheben.

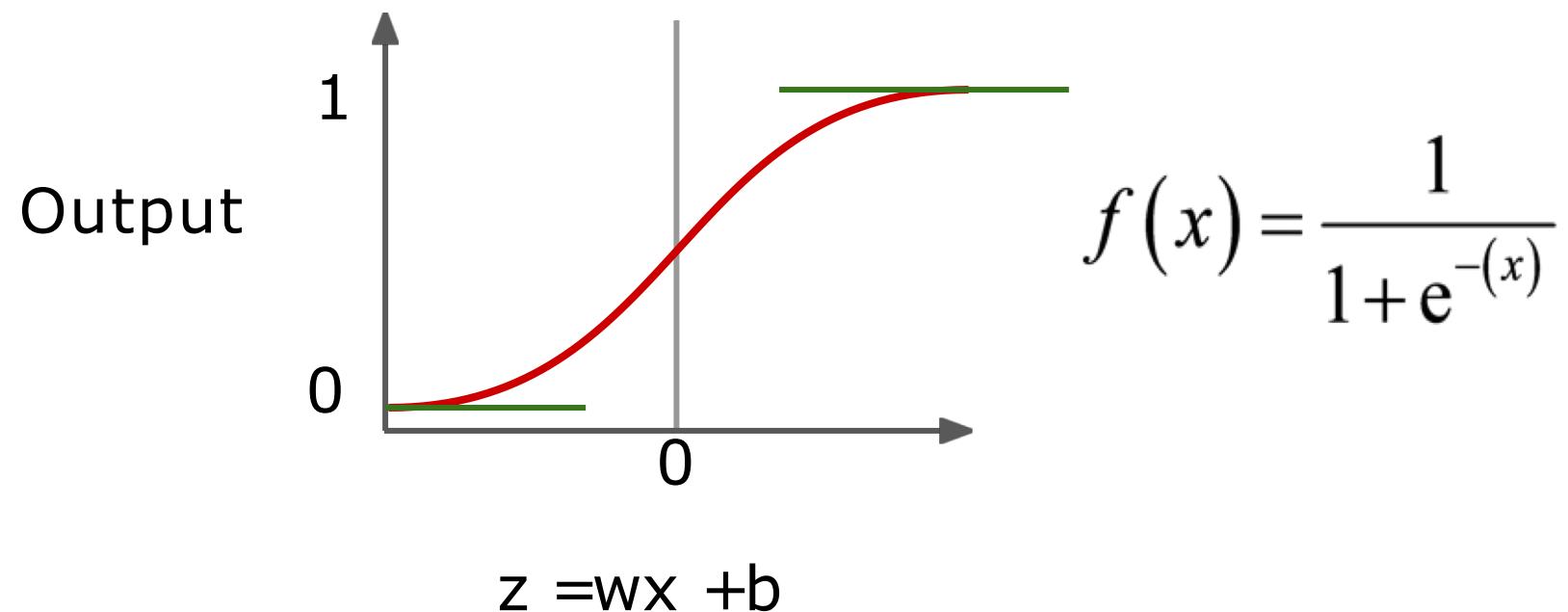
Warum passiert das?



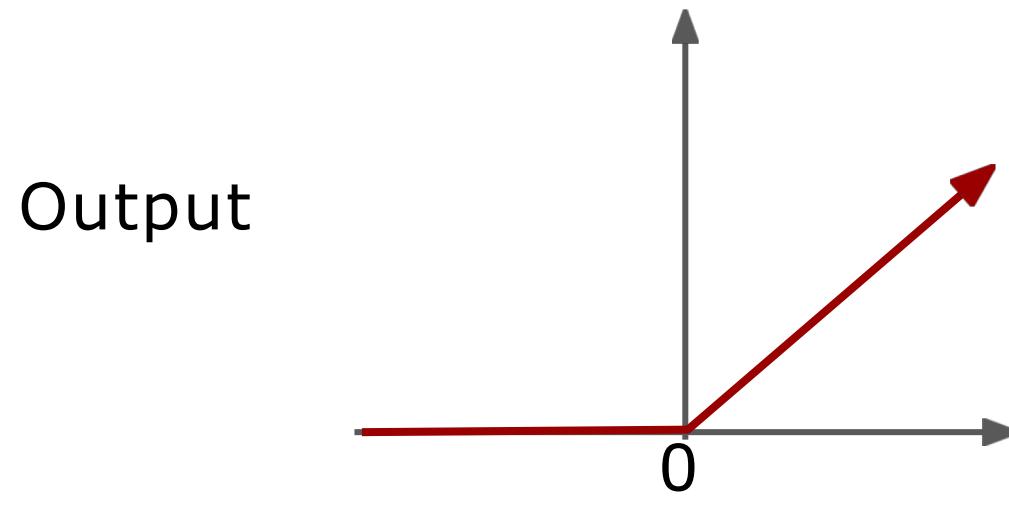
$$z = w x + b$$

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

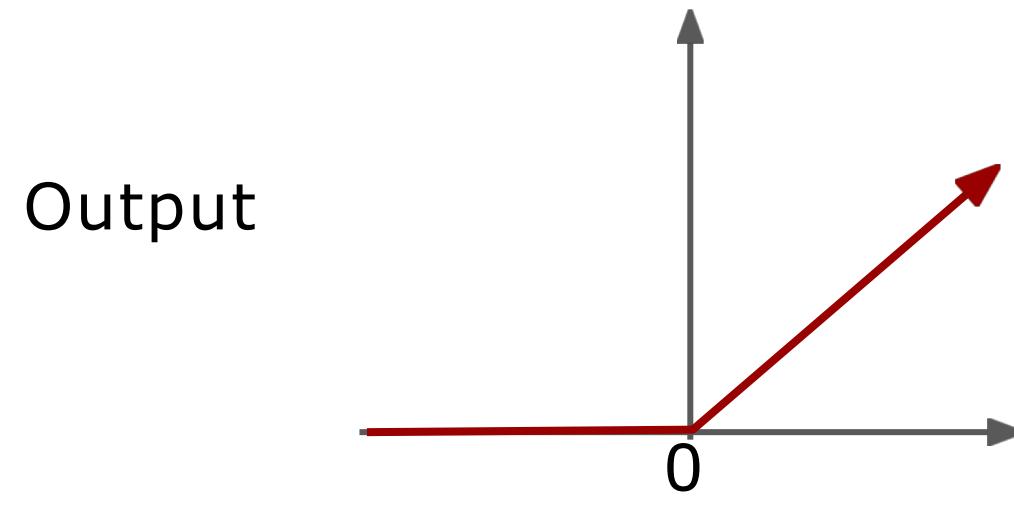
Warum passiert das?



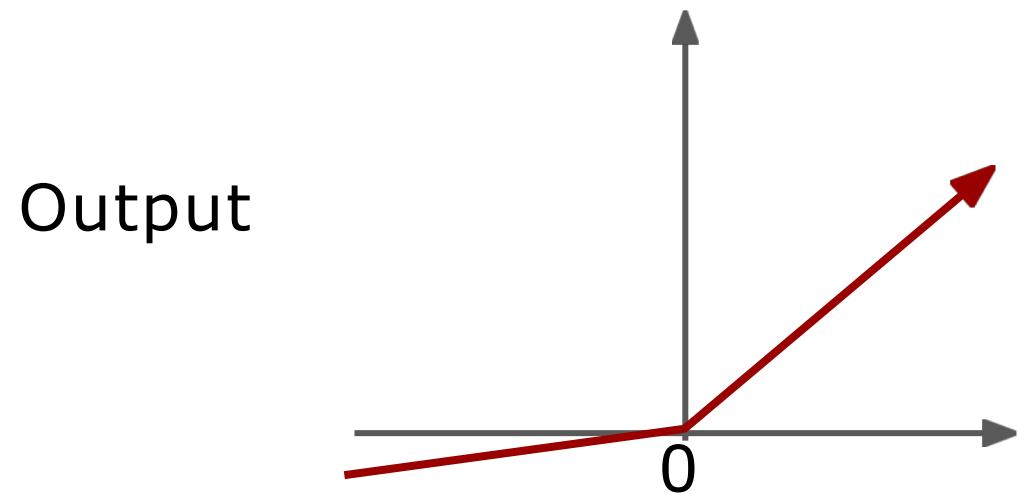
Verwendung verschiedener Aktivierungsfunktionen



Das ReLu sättigt keine positiven Werte

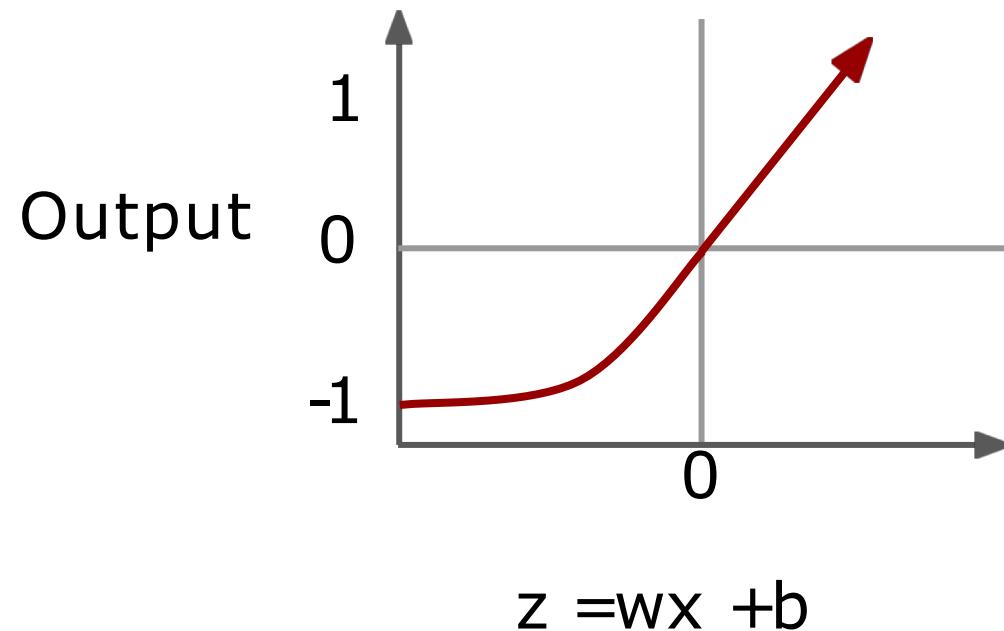


“Leaky” ReLU



Exponential Linear Unit (ELU)

(Exponentielle Lineareinheit)



Batch Normalization (Chargennormalisierung)



- Eine andere Lösung ist die Chargennormalisierung, bei der das Modell jede Charge anhand des Chargenmittelwerts und der Standardabweichung normalisiert.

Gradienten-Clipping



- Neben der Chargennormalisierung haben die Entwickler auch das "Gradienten-Clipping" eingesetzt, bei dem Gradienten vor Erreichen einer vorgegebenen Grenze abgeschnitten werden (z.B. Gradienten zwischen -1 und 1).

RNN für Time Series



- RNN für Zeitreihen haben ihre eigenen Herausforderungen mit den Gradienten. Wir werden jetzt spezielle Neuroneneinheiten (Neuron Units) kennenlernen, um diese Probleme zu beheben.

LSTM und GRU

LSTM und GRU



- Viele der bisher vorgestellten Lösungen für das Verschwinden von Gradienten können auch für RNN gelten: verschiedene Aktivierungsfunktionen, Chargennormalisierungen, etc....
- Aufgrund der Dauer der Zeitreiheneingabe können diese jedoch das Trainieren verlangsamen.

LSTM und GRU



- Eine mögliche Lösung wäre, nur die Zeitschritte für die Vorhersage zu verkürzen, aber das macht das Modell schlechter bei der Vorhersage längerer Trends.

LSTM und GRU

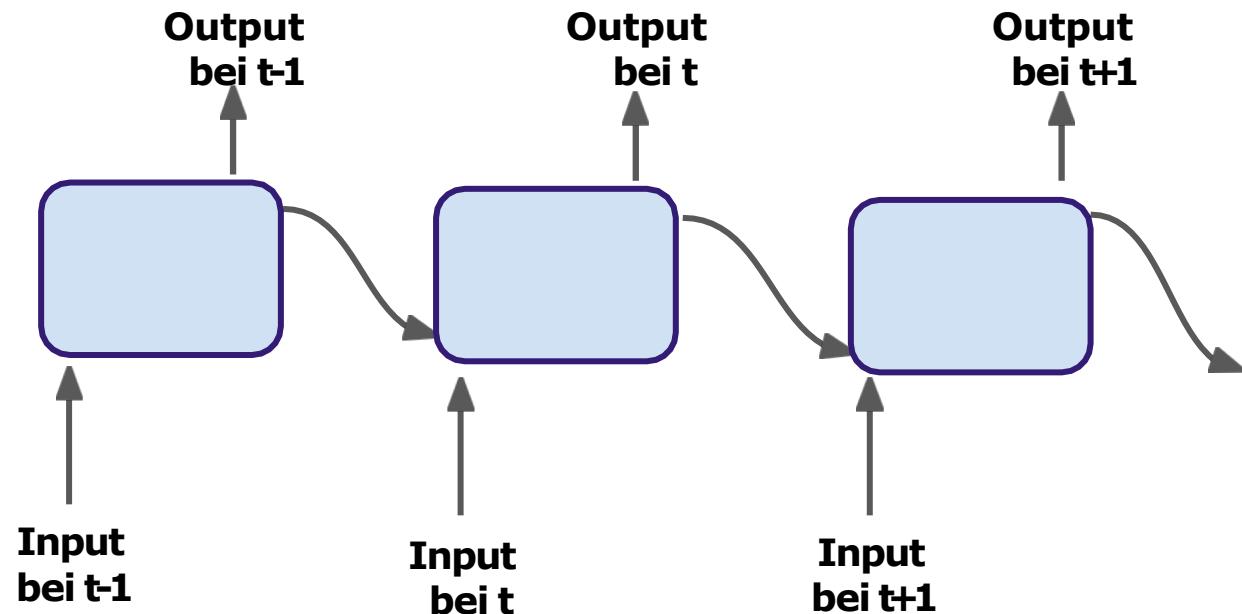
- Ein weiteres Problem von RNN's ist, dass nach einer Weile das Netzwerk beginnt die ersten Eingänge zu "vergessen", da Informationen bei jedem Schritt durch die RNN verloren gehen.
- Wir brauchen eine Art „Langzeitgedächtnis“ (long-term-memory) für unsere Netzwerke.

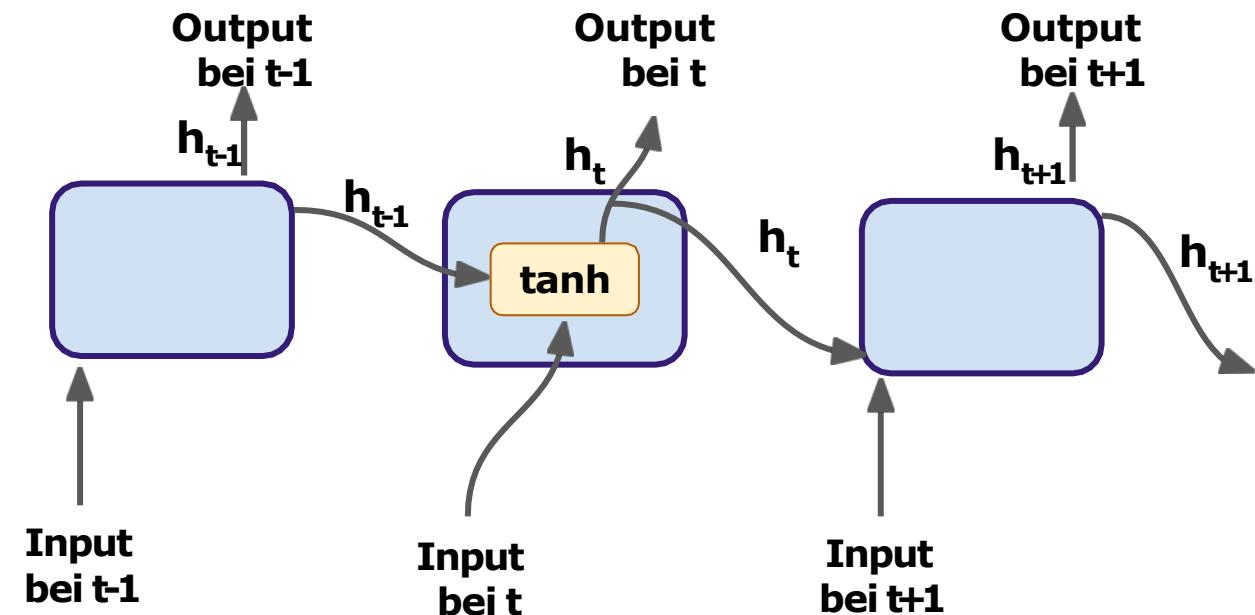
LSTM-Zelle



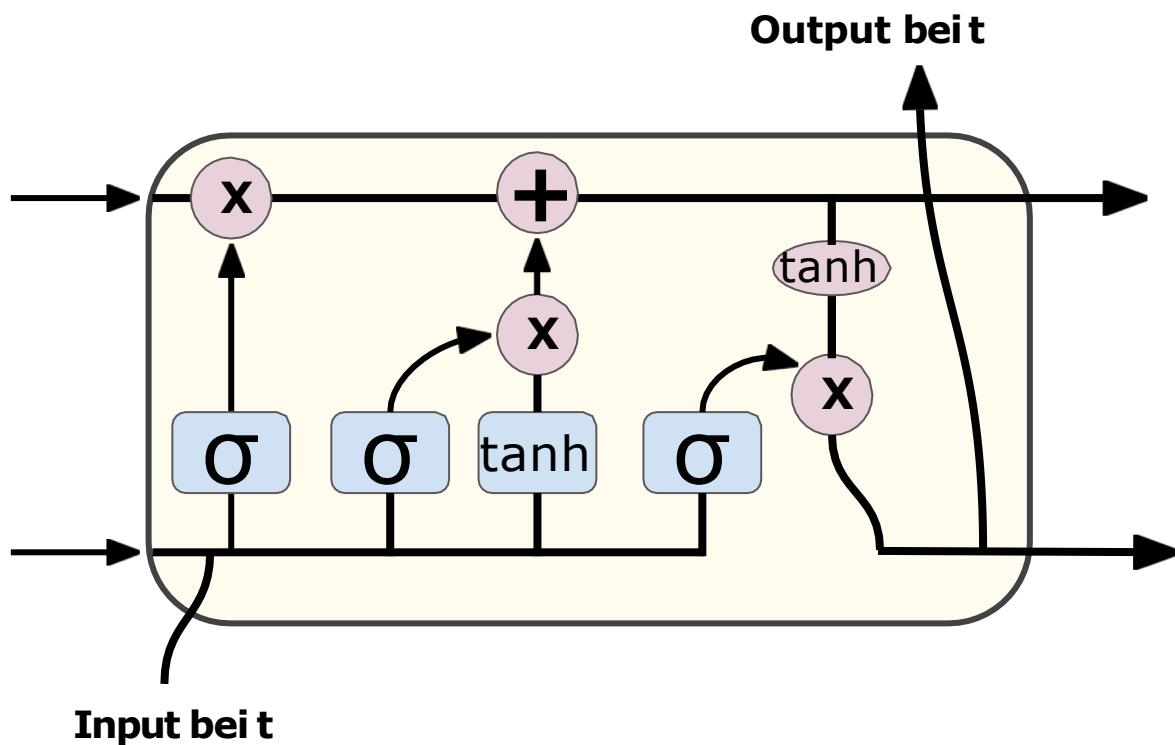
- Die LSTM-Zelle (Long Short-Term Memory) wurde entwickelt, um diese RNN-Probleme zu lösen.
- Gehen wir die Funktionsweise einer LSTM-Zelle durch!

Ein Typisches RNN





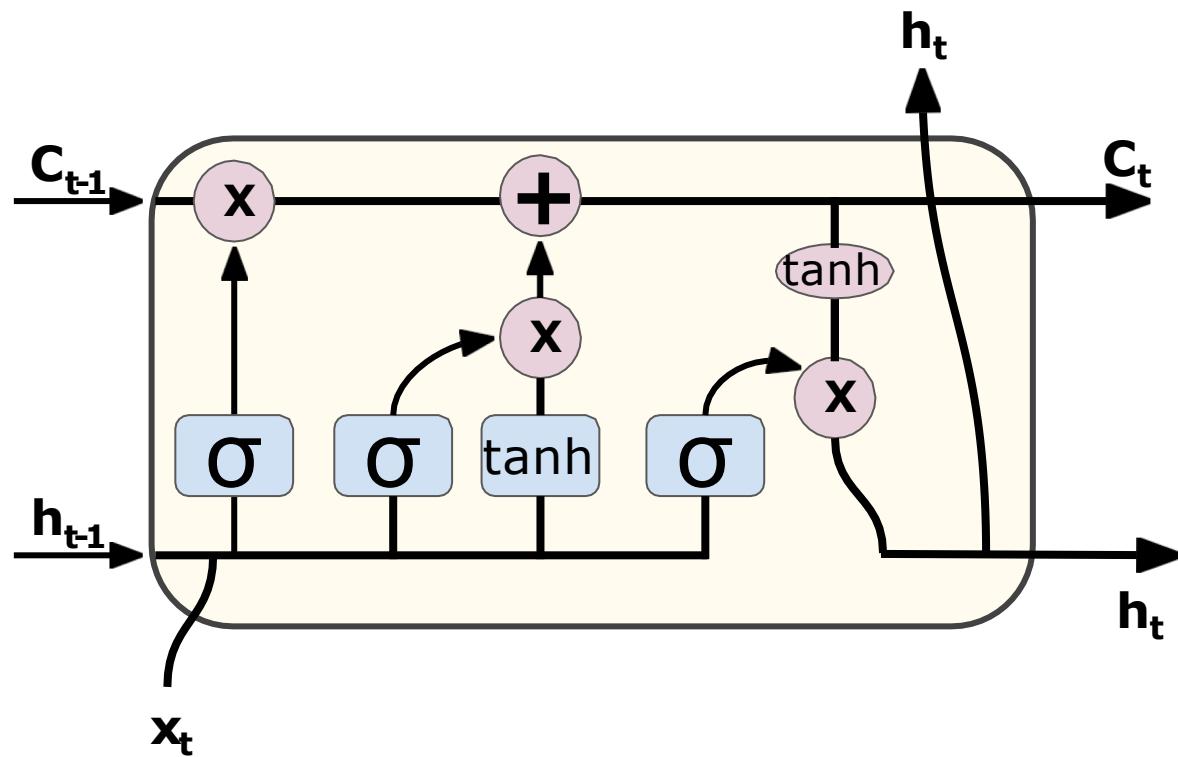
LSTM Zelle



- Hier sehen wir die ganze Zelle
- Gehen wir den Prozess durch!

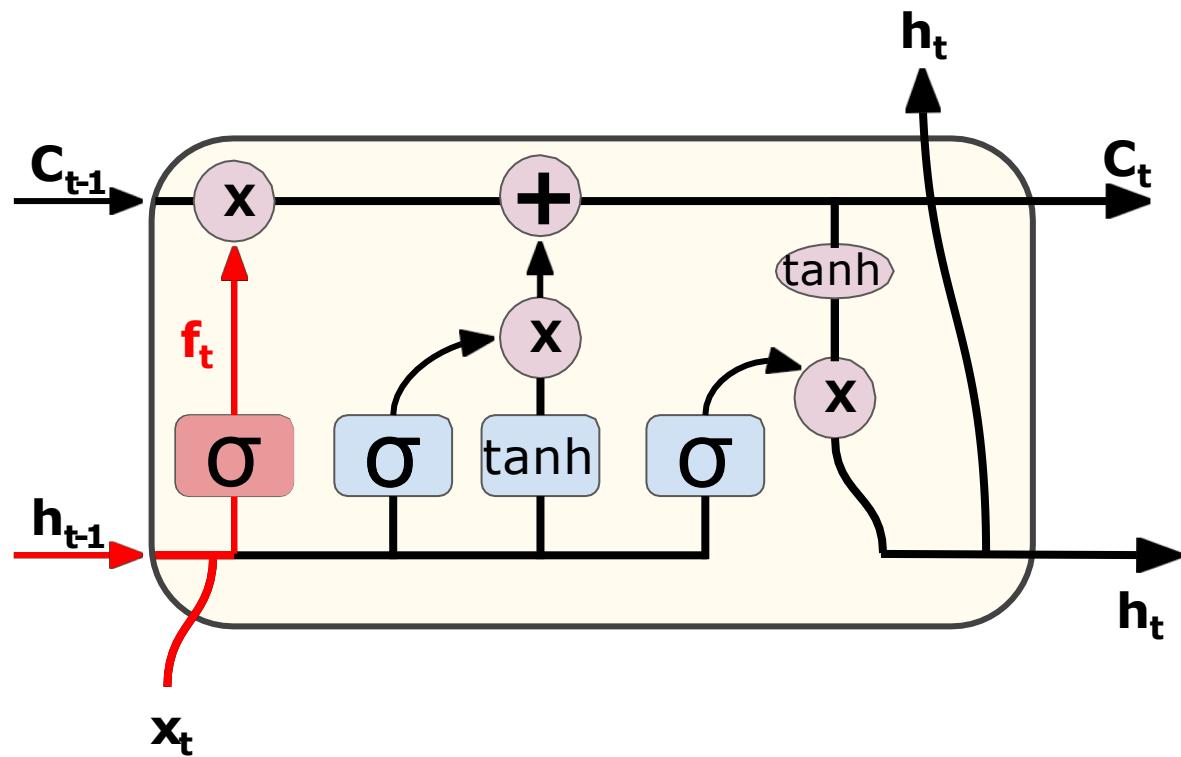
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTM Zelle



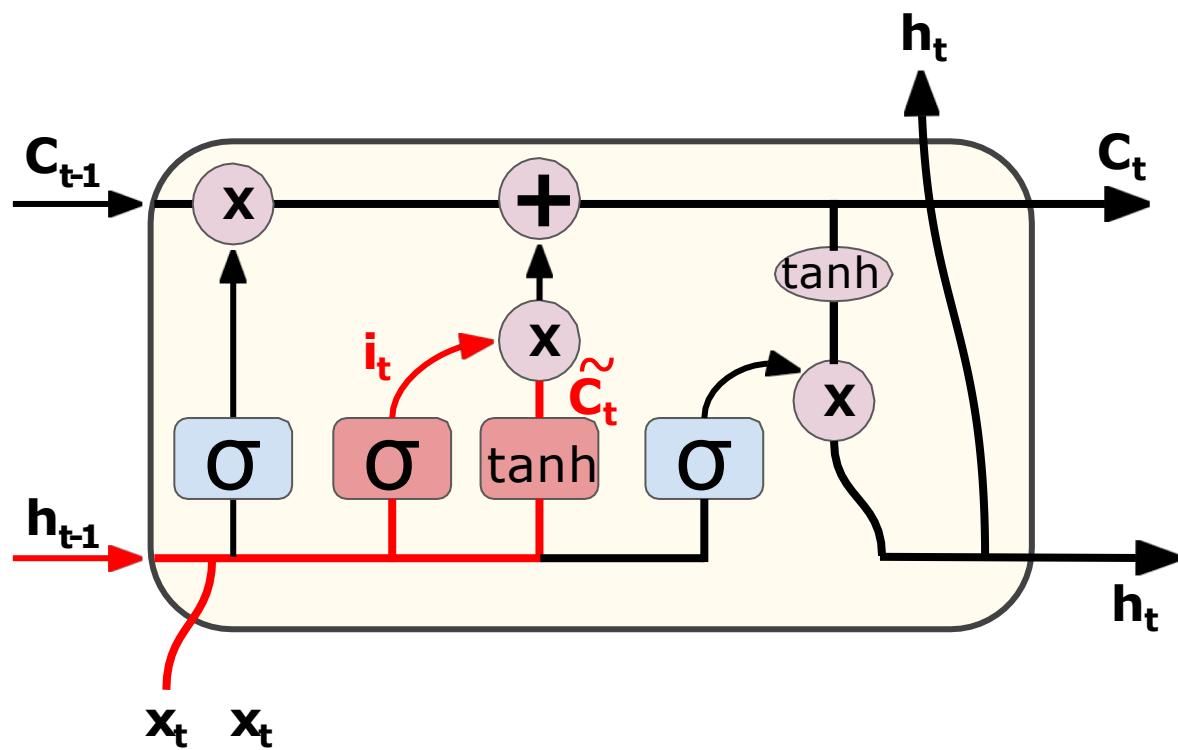
- Hier sehen wir die ganze Zelle
- Gehen wir den Prozess durch!

LSTM Zelle



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

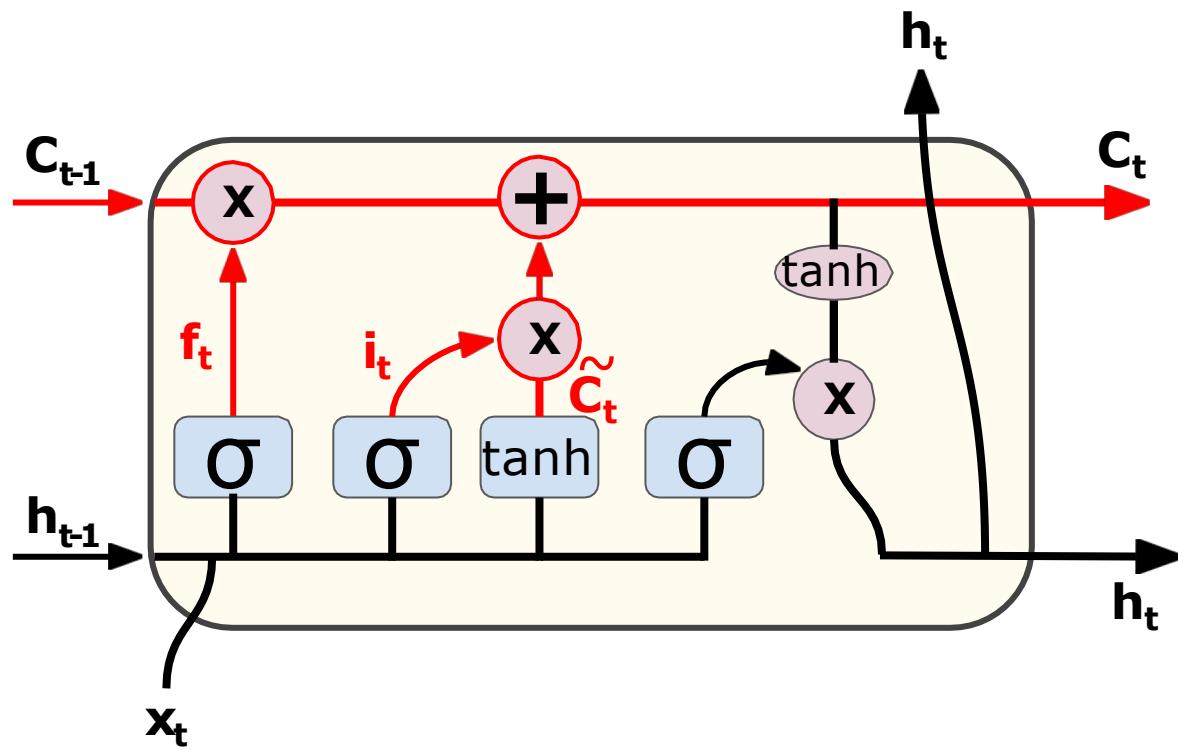
LSTM Zelle



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

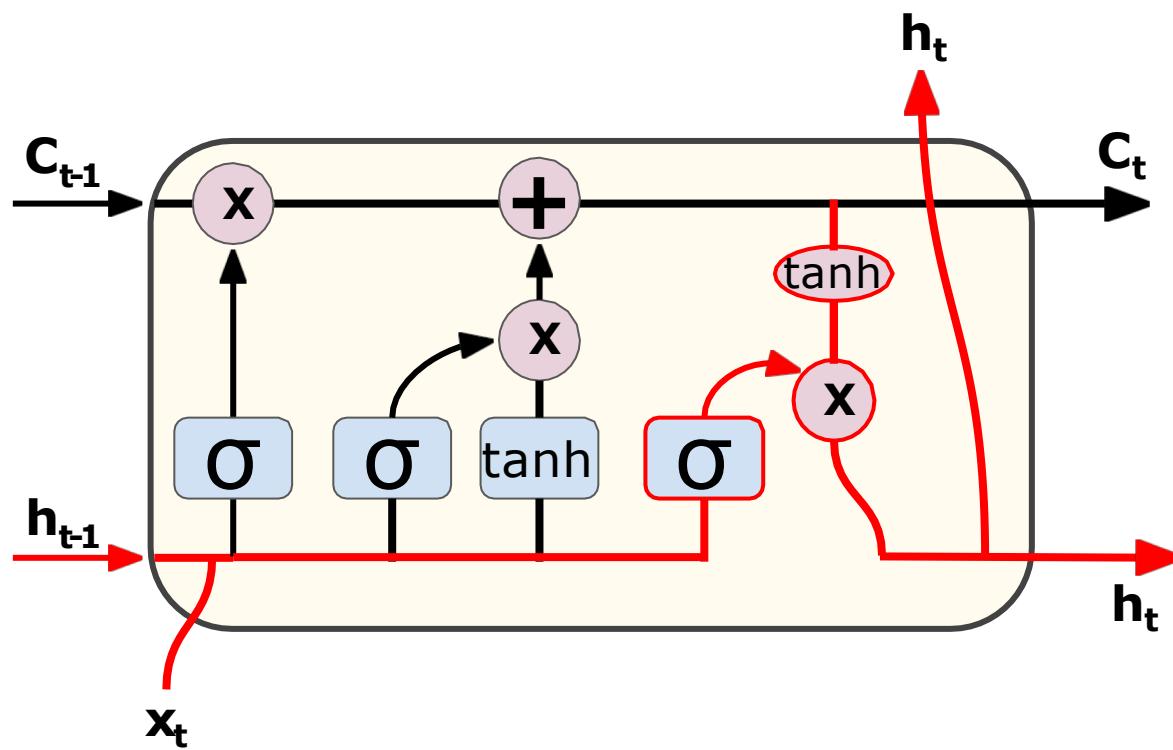
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

LSTM Zelle



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

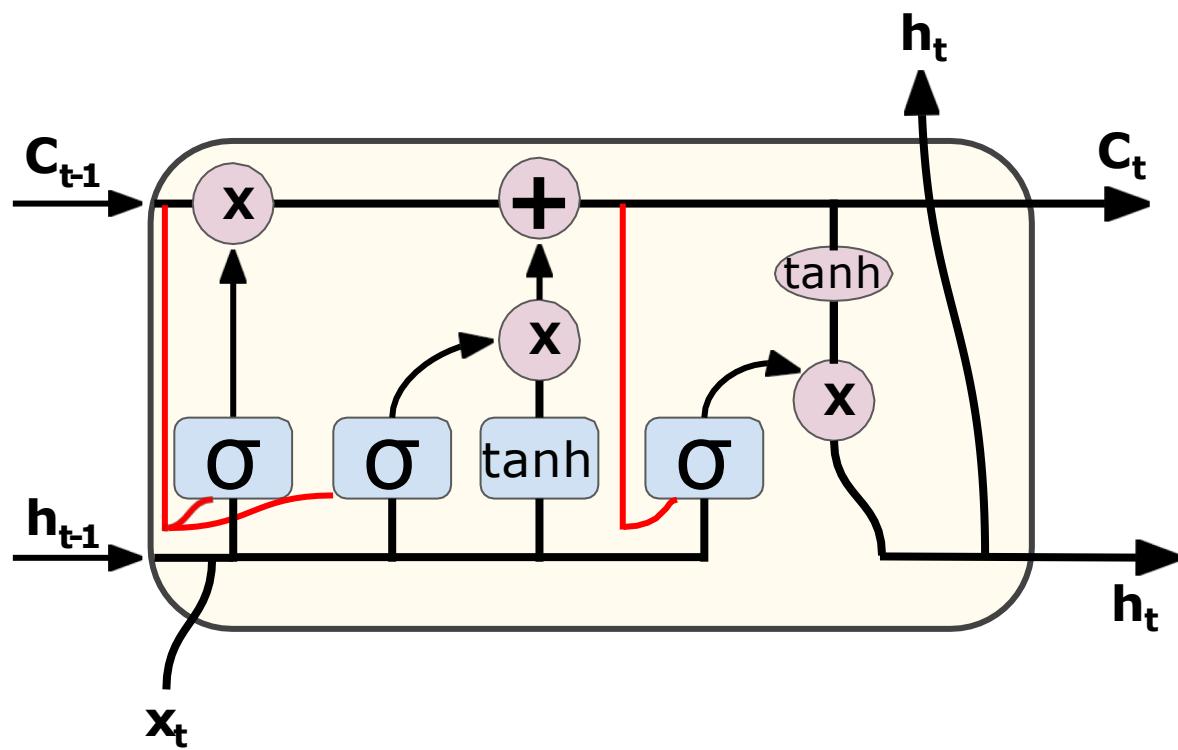
LSTM Zelle



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Eine LSTM-Zelle mit "Peepholes"

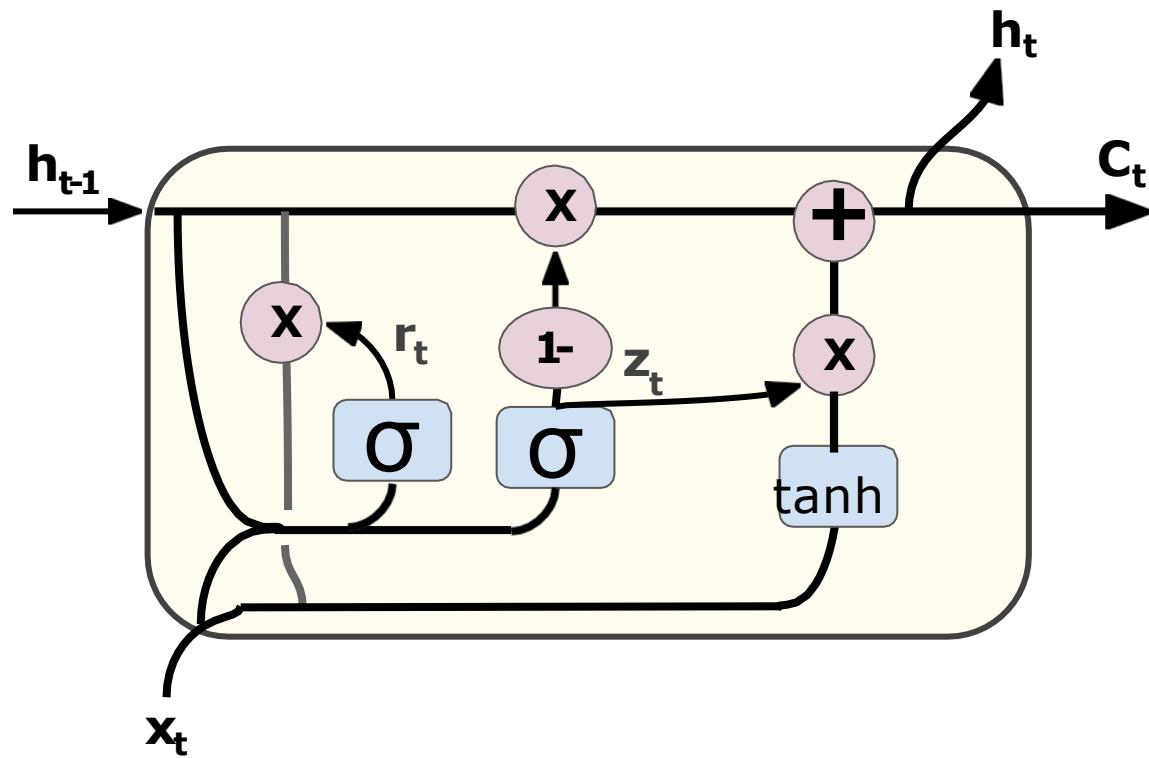


$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

Gated Recurrent Unit (GRU)



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

TensorFlow RNN API



- Glücklicherweise sind diese Neuronenmodellen bei TensorFlow in eine nette API integriert. Das macht es uns einfach diese ein- und auszutauschen.
- Als nächstes werden wir die Verwendung dieser TensorFlow RNN API für die Vorhersage und Generierung von Zeitreihen untersuchen!

RNN mit TensorFlow API

RNN mit TensorFlow API



- Jetzt verstehen wir verschiedene Verbesserungen für RNNs, daher verwenden wir die in TensorFlow integrierte tf.nn Funktions-API, um Sequenzprobleme zu lösen!

RNN mit TensorFlow API



- Erinnerst du dich noch an unsere ursprüngliche Sequenzübung:
[1,2,3,4,5,6]
- Können wir vorhersagen, dass sich die Reihenfolge um einen Zeitschritt nach vorne verschoben hat?
[2,3,4,5,6,7]

RNN mit TensorFlow API



- Was ist mit dieser Zeitreihe?
[0,0.84,0.91,0.14,-0.75,-0.96,-0.28]

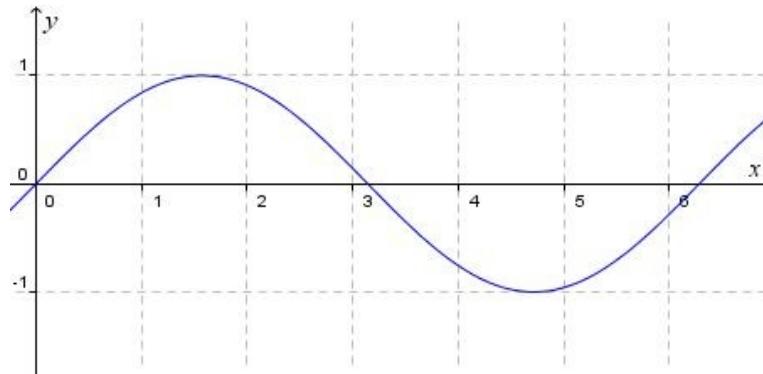
RNN mit TensorFlow API

- Was ist mit dieser Zeitreihe?

[0,0.84,0.91,0.14,-0.75,-0.96,-0.28]

- Es ist eigentlich nur $\text{Sin}(x)$:

[0.84,0.91,0.14,-0.75,-0.96,-0.28,0.65]



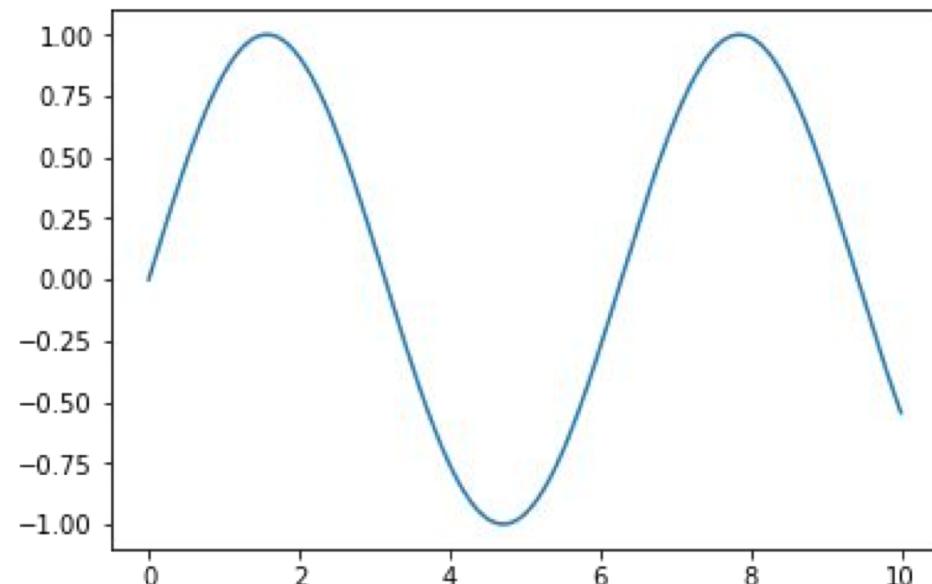
RNN mit TensorFlow API



- Wir beginnen mit der Erstellung eines RNN, das versucht, eine Zeitreihe vorherzusagen, die über eine Einheit in die Zukunft verschoben wurde.
- Dann werden wir versuchen, neue Sequenzen mit einer Seed-Serie zu generieren.

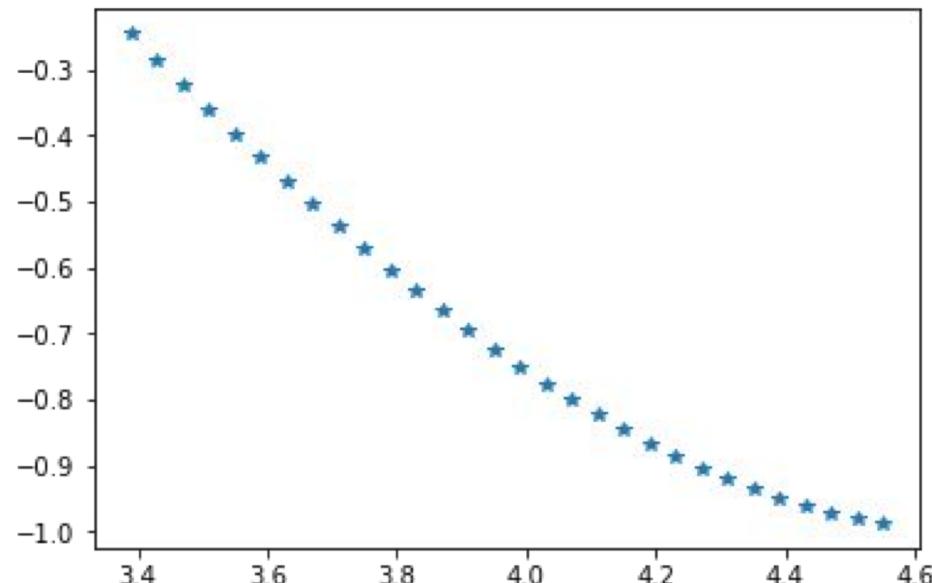
RNN mit TensorFlow API

- Wir werden zuerst eine einfache Klasse erstellen, um $\sin(x)$ zu erzeugen und auch zufällige Batches von $\sin(x)$ zu erfassen.



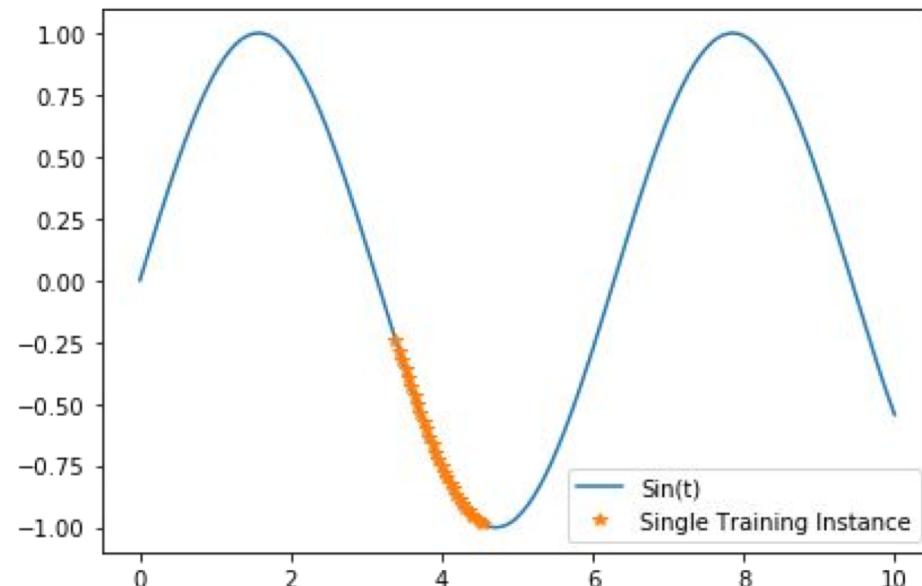
RNN mit TensorFlow API

- Wir werden zuerst eine einfache Klasse erstellen, um $\sin(x)$ zu erzeugen und auch zufällige Batches von $\sin(x)$ zu erfassen.



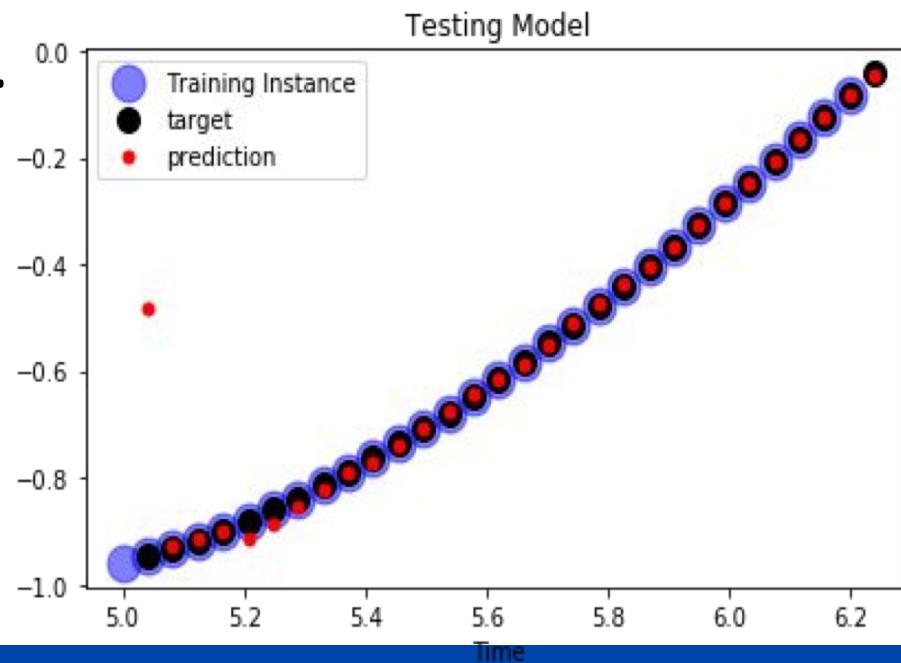
RNN mit TensorFlow API

- Wir werden zuerst eine einfache Klasse erstellen, um $\sin(x)$ zu erzeugen und auch zufällige Batches von $\sin(x)$ zu erfassen.



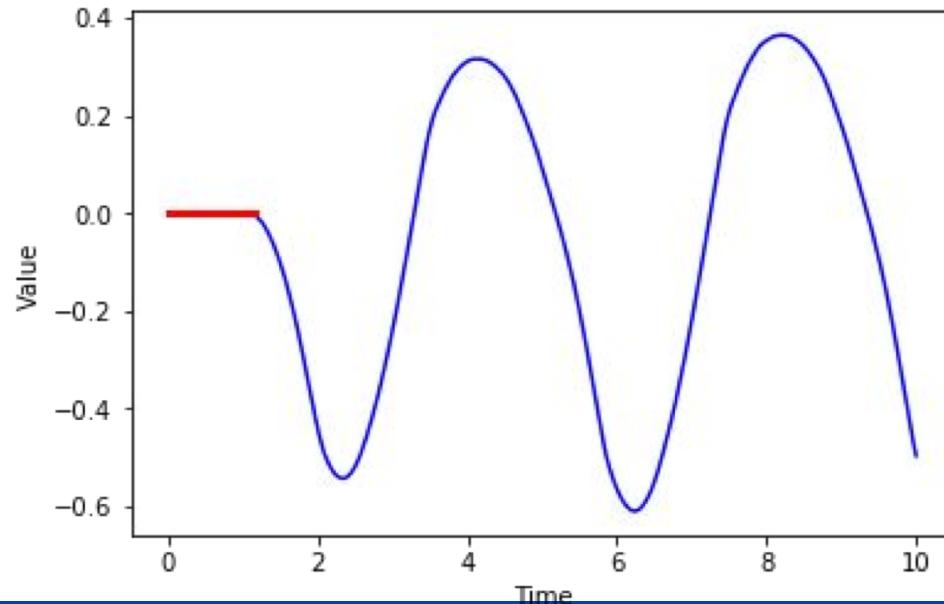
RNN mit TensorFlow API

- Dann erhält das trainierte Modell eine Zeitreihe und versucht, eine Zeitreihe vorherzusagen, die um einen Zeitschritt nach vorne verschoben wurde.



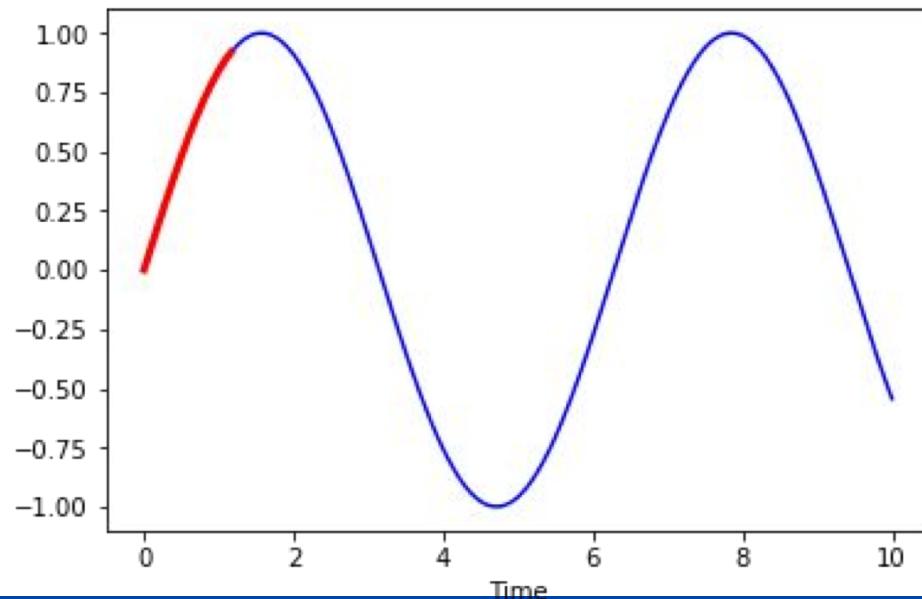
RNN mit TensorFlow API

- Danach werden wir das gleiche Modell verwenden, um viel längere Zeitreihen zu erzeugen, wenn wir eine Seed-Serie haben.



RNN mit TensorFlow API

- Danach werden wir das gleiche Modell verwenden, um viel längere Zeitreihen zu erzeugen, wenn wir eine Seed-Serie haben.



RNN mit der TensorFlow API

Anwendung Teil 1

RNN mit der TensorFlow API

Anwendung Teil 2

RNN mit der TensorFlow API

Anwendung Teil 3

RNN mit der TensorFlow API

Anwendung Teil 4

Zeitreihen-Übung

Zeitreihen-Lösungen

Teil 1

Zeitreihen-Lösungen

Teil 2

Kurzer Hinweis zu Word2Vec

Kurzer Hinweis zu Word2Vec



- Optionale Lektionen zum Thema Word2Vec mit TensorFlow.
- Die gensim-Library ist empfehlenswert, wenn du dich für Word2Vec interessierst.

Word2Vec

Natural Language Processing

Natural Language Processing



- Nachdem wir jetzt verstehen wie man mit Zeitreihen von Daten arbeitet, werfen wir nun einen Blick auf eine andere wichtige Datenquelle der Sequenzen: den Wörter.
- Zum Beispiel der Satz:
["Hi", "wie", "geht's", "dir"]

Natural Language Processing



- Im "klassischen" Natural Language Processing (NLP) werden Wörter typischerweise durch Zahlen ersetzt, die eine gewisse Häufigkeitsbeziehung zu ihren Dokumenten angeben.
- Dadurch verlieren wir Informationen über die Beziehung zwischen den Wörtern selbst.

Natural Language Processing



- Zählbasiert (Count-Based)
 - Häufigkeit der Wörter im Korpus
- Vorhersagebasiert (Predictive-Based)
 - Nachbarwörter werden anhand eines Vektorraumes vorhergesagt.

Word2Vec-Modell von Mikolov et al.



- Lass uns nun einen der bekanntesten Anwendungsfälle von Neuralen Netzwerken in der natürlichen Sprachverarbeitung erforschen:
 - Das Word2Vec-Modell von Mikolov et al.

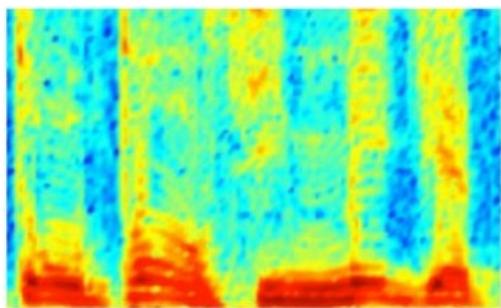
Word2Vec-Modell von Mikolov et al.



- Das Ziel des Word2Vec-Modells ist es, Worteinbettungen zu lernen, indem jedes Wort als Vektor im n-dimensionalen Raum modelliert wird.
- Aber warum verwenden wir die Wort-Einbettungen?

Darstellung der Daten

AUDIO



Audio Spectrogram

DENSE

IMAGES

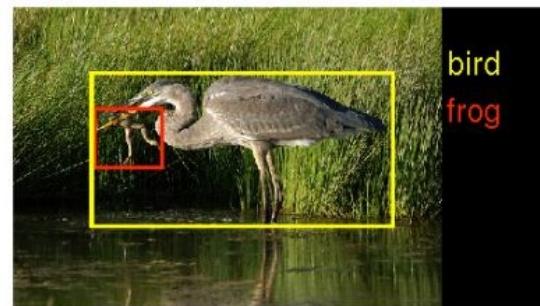
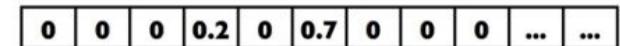


Image pixels

DENSE

TEXT



Word, context, or
document vectors

SPARSE

Word2Vec



- Word2Vec erstellt Vektormodelle, die Wörter in einem kontinuierlichen Vektorraum darstellen (einbetten).
- Mit Wörtern, die als Vektoren dargestellt werden, können wir Vektormathematik auf Wörtern durchführen (z.B. Ähnlichkeiten prüfen, Vektoren addieren/subtrahieren).

Word2Vec



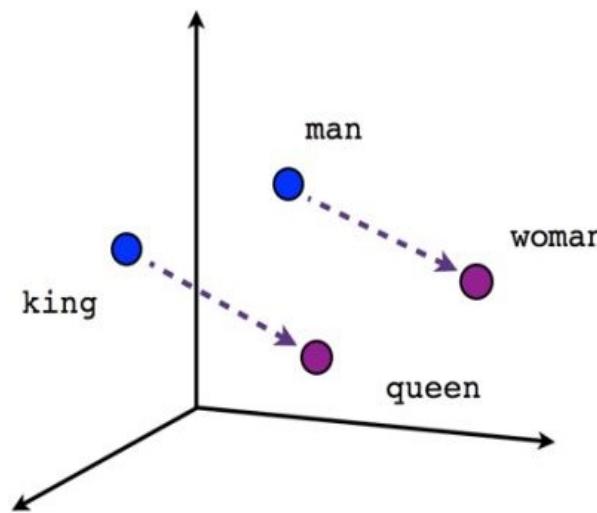
- Zu Beginn des Trainings ist jede Einbettung zufällig, aber durch die Rückführung wird das Modell den Wert jedes Wortvektors in der angegebenen Anzahl von Dimensionen anpassen.
- Mehr Dimensionen bedeuten mehr Trainingszeit, aber auch mehr "Informationen" pro Wort.

Word2Vec

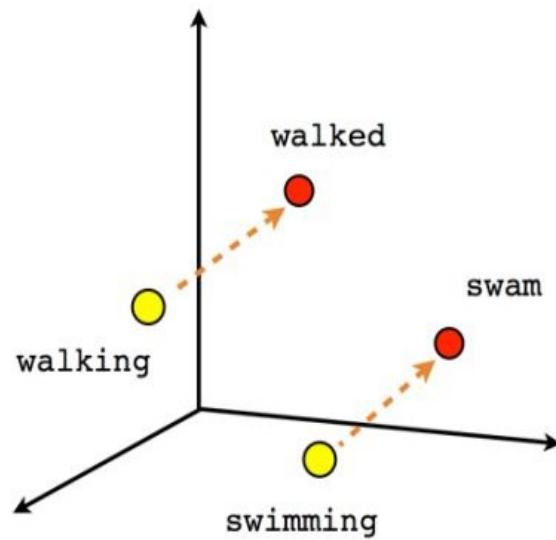


- Ähnliche Wörter haben ihre Vektoren näher zusammen.
- Noch eindrucksvoller ist, dass das Modell Achsen erzeugen kann, die Begriffe wie Geschlecht, Verben, Singular vs. Plural usw. darstellen.

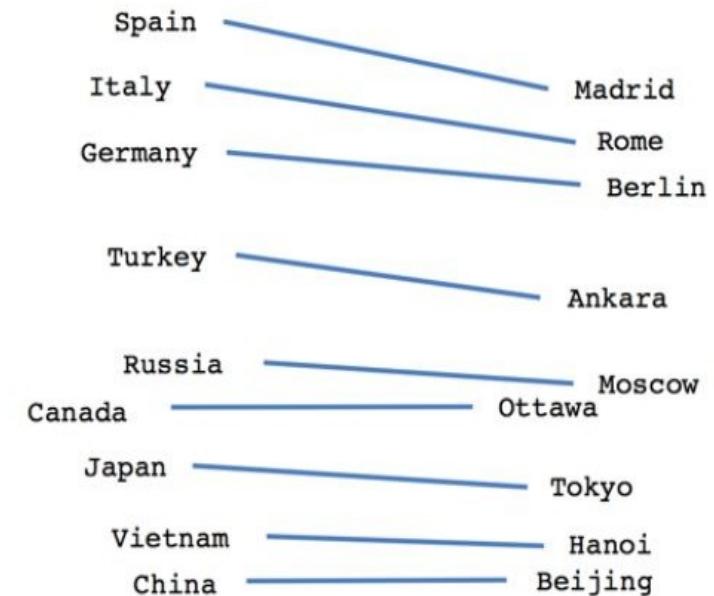
Word2Vec



Männlich - Weiblich



Verb - Zeitform



Land - Hauptstadt

Vorhersage Ziel

- Skip-Gram-Modell
 - Der Hund kaut den Knochen
 - Typischerweise besser für größere Datensätze
- CBOW (Continuous Bag of Words)
 - Der Hund kaut den Knochen
 - Typischerweise besser für kleinere Datensätze

Vorhersage Ziel

- Der Hund kaut den $w_t=?$

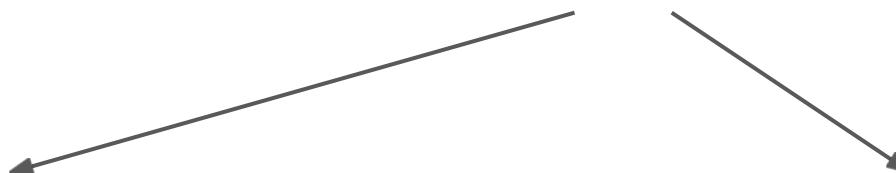
Knochen

Ziel Wort

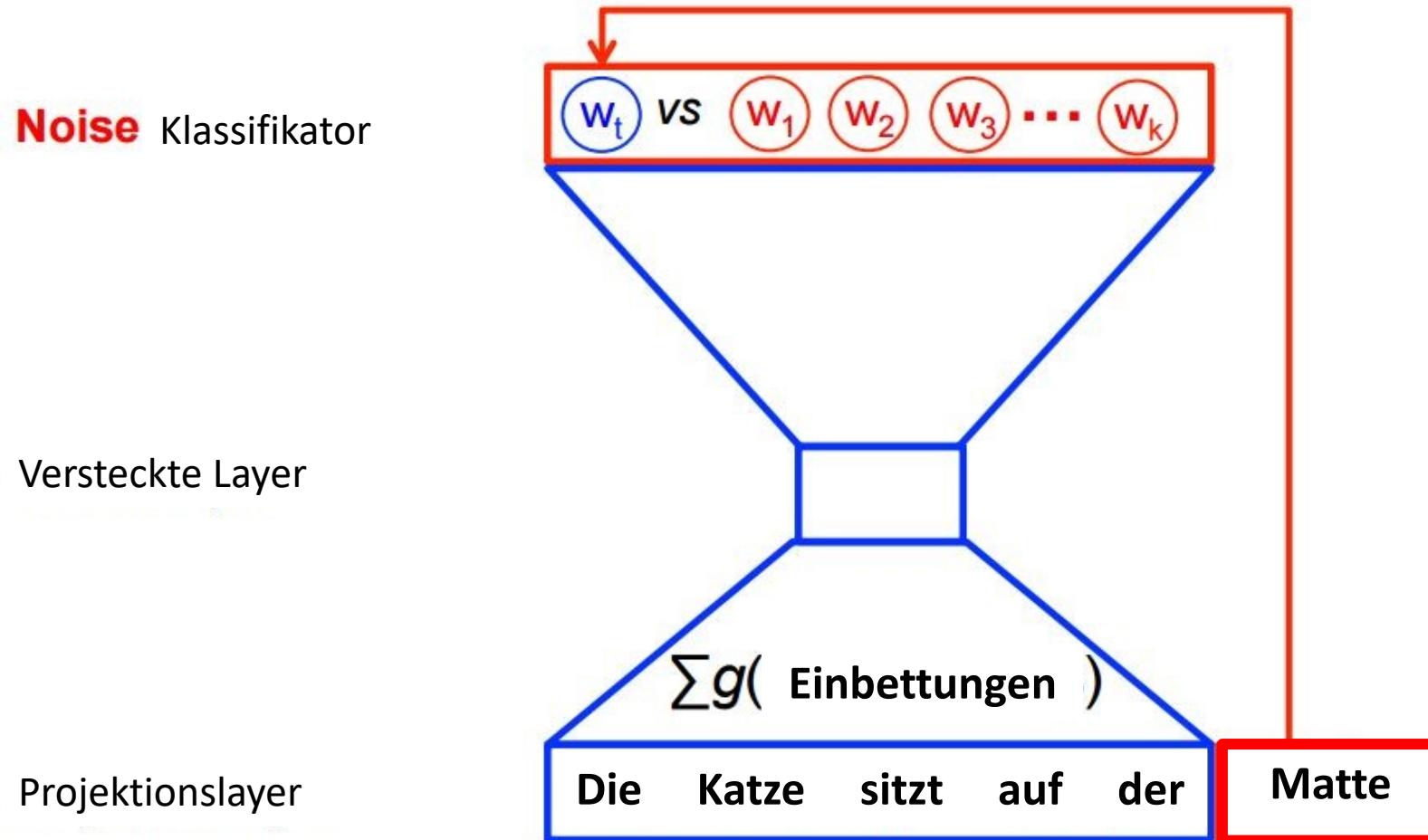
vs

[Buch, Auto ,Haus,Sonne,guitar]

Noise Wörter



Vorhersage Ziel mit NN



Noise-Contrastive Training

- Das Zielwort wird durch Maximierung vorhergesagt
 - $J_{NEG} = \log Q_\theta(D=1 | w_t, h) + k_{n \sim P_{noise}} E [\log Q_\theta(D=0 | w_n, h)]$
- $Q_\theta(D=1 | w_t, h)$ ist binäre logistische Regression, die Wahrscheinlichkeit, dass das Wort w_t im Kontext h des von θ parametrisierten Datensatzes D steht.

Noise-Contrastive Training



- Das Zielwort wird durch Maximierung des
 - $J_{NEG} = \log Q_\theta(D=1 | w_t, h) + k_{n \sim P_{noise}} E [\log Q_\theta(D=0 | w_n, h)]$
- w_n sind k Worte aus der Noise Distribution (Lärmverteilung).

Noise-Contrastive Training

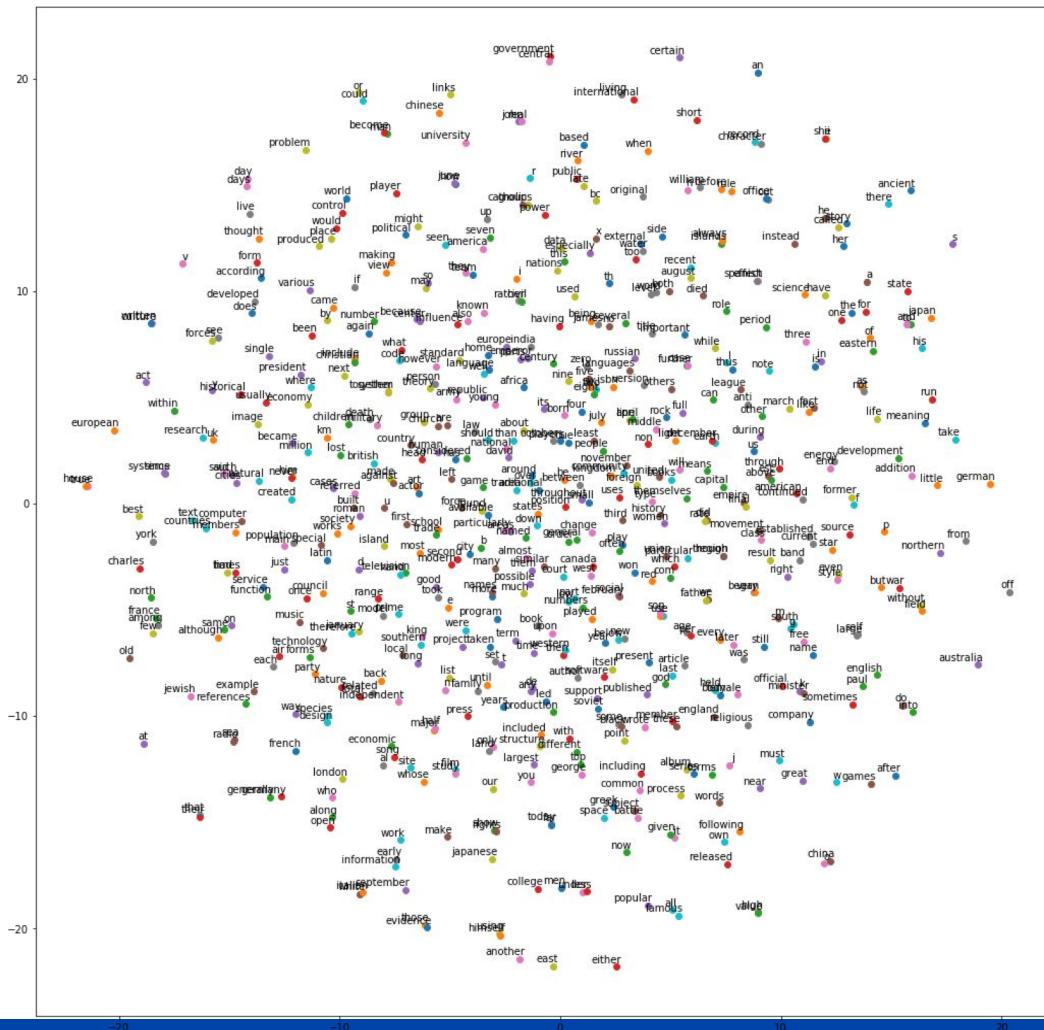
- Das Zielwort wird durch Maximierung des
 - $J_{NEG} = \log Q_\theta(D=1 | w_t, h) + k_{n \sim P_{noise}} E [\log Q_\theta(D=0 | w_n, h)]$
- w_n sind k Worte aus der Noise Distribution (Lärmverteilung).
- Ziel ist es, korrekten Wörtern eine hohe Wahrscheinlichkeit zuzuweisen und verrauschten (noise) Wörter eine geringe Wahrscheinlichkeit zuzuweisen.

t-Distributed Stochastic Neighbor Embedding



- Sobald wir Vektoren für jedes Wort haben, können wir Beziehungen visualisieren, indem wir die Dimensionen von 150 auf 2 reduzieren.
- t-Distributed Stochastic Neighbor Embedding
(t-verteiltes stochastisches Neighbor Embedding) .

t-Distributed Stochastic Neighbor Embedding



Los geht's!

Word2Vec Praxis-Lektion

Word2Vec Praxis-Lektion



- Wir werden die TensorFlow Dokumentation Beispiel-Implementierung von Word2Vec verwenden.
- Wir werden uns oft auf das mitgelieferte Jupyter Notebook für einzelne Codeblöcke beziehen!

Word2Vec



- Wenn Word2Vec etwas ist, dass dich interessiert, dann schau dir die gensim-Library für Python an, sie hat eine viel einfacher zu verwendende API für Word2Vec und zusätzliche Funktionen!

Word2Vec Praxis-Lektion

Teil 2

Exklusive Gutscheine



Verwende den Gutschein „**SLIDEShare2018**“ auf *Udemy* oder die Shortlinks und erhalte unsere Kurse für nur 10,99€ (95% Rabatt).

Deep Learning Grundlagen mit TensorFlow und Python

<https://goo.gl/FqNoAe>

Python für Data Science und Machine Learning:

<https://goo.gl/cE7TQ3>

Original Python Bootcamp - Von 0 auf 100:

<https://goo.gl/gjn7pX>

R für Data Science und Machine Learning:

<https://goo.gl/8h5tH7>