

ML2 – Deep Learning

Recurrent neural network (RNN)

Christian Gilles, Paul Krüger, Alexander Vollmer,
Melanie Ecker, Jan Michels

Agenda Vorlesung

1. Motivation
2. Rekurrente Neuronen
3. Struktur von RNNs
4. Training von RNNs
5. Einführung in Gated RNNs
6. LSTMs
7. GRUs
8. Anwendungsbeispiele
9. Hinweise zur Übung



Lernziele

1. Sie wissen unter welchen Umständen Sie ein RNN einsetzen sollten.
2. Sie wissen wie das Training eines RNN abläuft und was für spezielle Probleme dabei entstehen können.
3. Sie kennen verschiedene Varianten von RNNs und welche Vorteile/Nachteile diese haben.
4. Sie können ein eigenes RNN Modell trainieren und für Vorhersagen von Buchstaben nutzen.





Motivation

Wo kommen die CNNs an ihre Grenzen?

„Ich komme aus **Frankreich**... In meinem letzten Urlaub konnte ich endlich nochmal meine **Muttersprache** sprechen.“

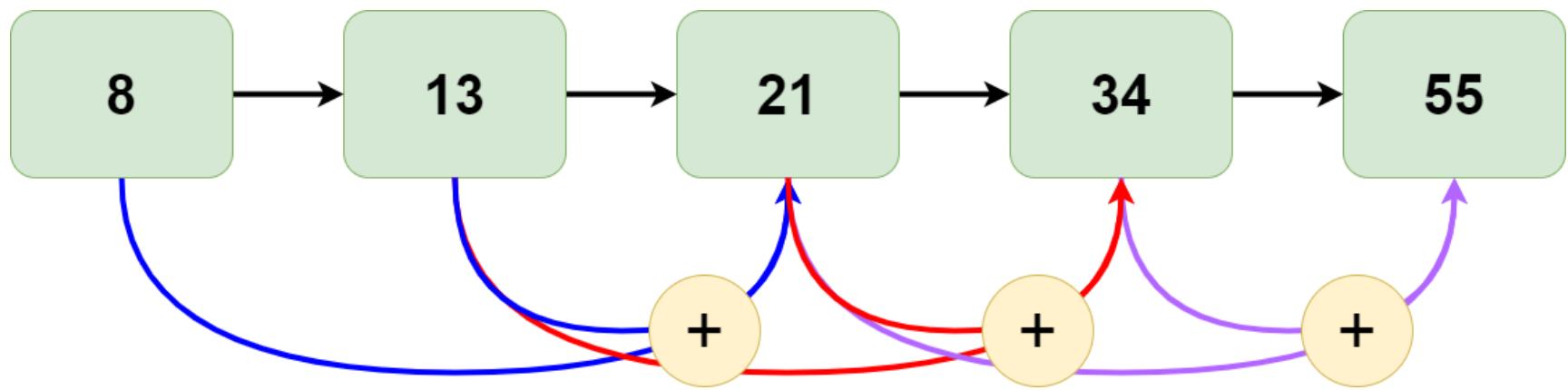


Wie würde man eine solche Beziehung mit einem CNN auflösen?



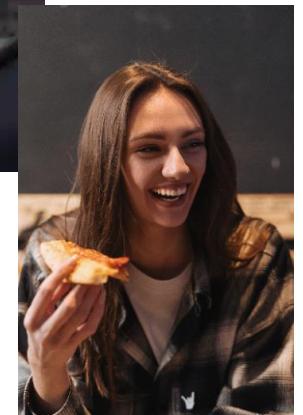
Um einen Text beliebiger Länge analysieren zu können, muss dieser Sequentiell verarbeitet werden.
Für die Zuordnung der Muttersprache ist ein Gedächtnis notwendig.
→ CNN sind ungeeignet.

Was sind eigentlich Sequenzen?



Sequenzen treten in vielen verschiedenen Bereichen.

- Sprache
- Videos
- Fotos
- Biomedizinische Domäne (z.B. Gensequenzen)



Was sind die Besonderheiten bei handgeschriebenen Nummern gegenüber etwa Hausnummern.

- Handgeschriebene Zahlen? → CNN ist dein Netz!
- Hausnummern auf Bildern? → Sequenz von Zahlen



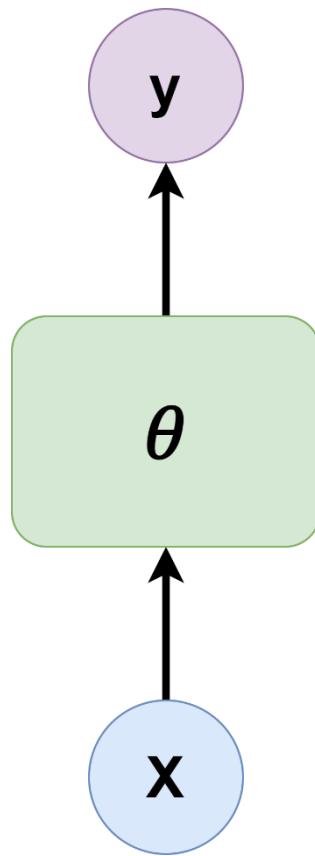
Quelle: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>



Rekurrente Neuronen

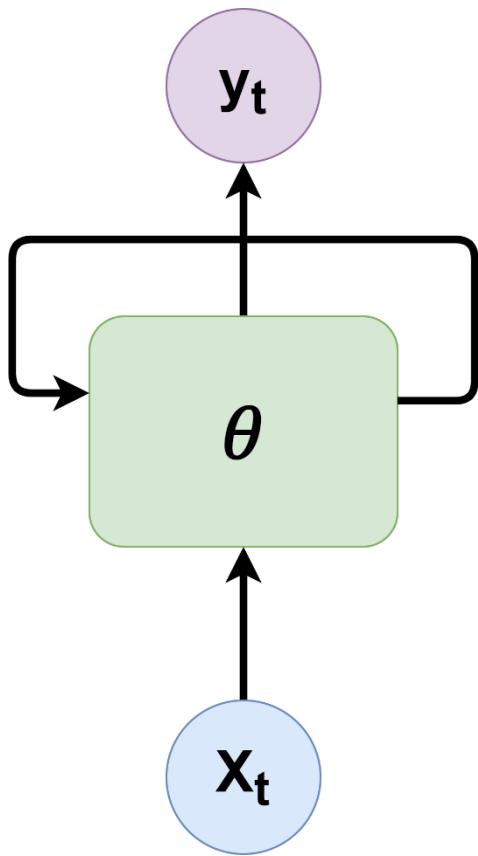
Bildquelle: <https://www.alexanderthamm.com/wp-content/uploads/Neuronale-Netze-Artikel-1-e1539848085792.jpg>

CNNs eignen sich hervorragend für Bildverarbeitung. Jedoch können diese keine Sequenzen mit beliebiger Länge verarbeiten.



- Input hat feste Länge
- Output hat feste Länge
- Verwendung einer festen Anzahl an Layern
- Position im Input relevant

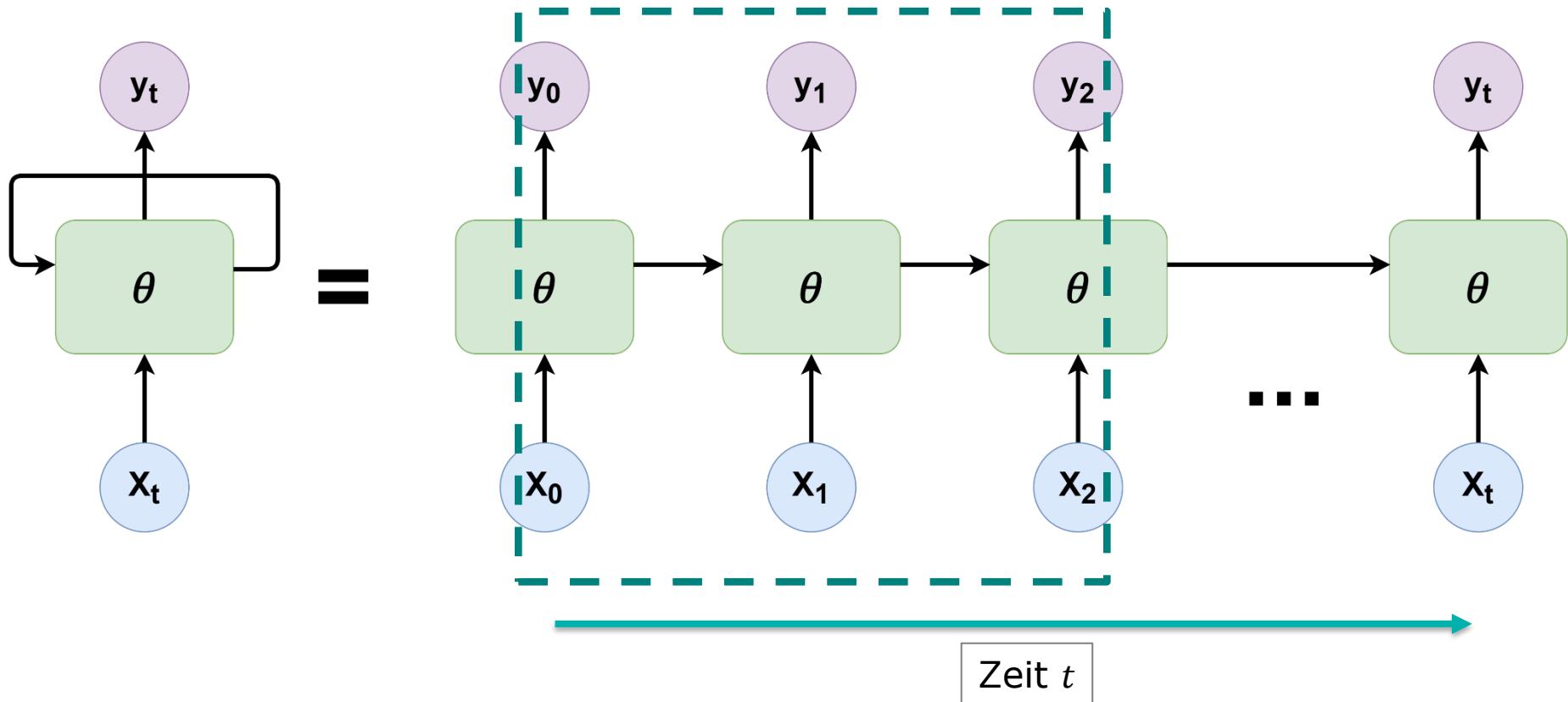
RNNs verarbeiten den Input zyklisch.
Deshalb kann dieser eine variable Länge haben.



- Input als Sequenz mit variabler Länge
- Output als Sequenz mit variabler Länge
- Zyklische Verarbeitung
- Rückwärtsgerichtete Verbindungen
(Output wird als Input wieder verwendet)
- **Hidden State**
 - Ergebnis von x_t hat Einfluss auf x_{t+1}
 - Neuron hat „Gedächtnis“

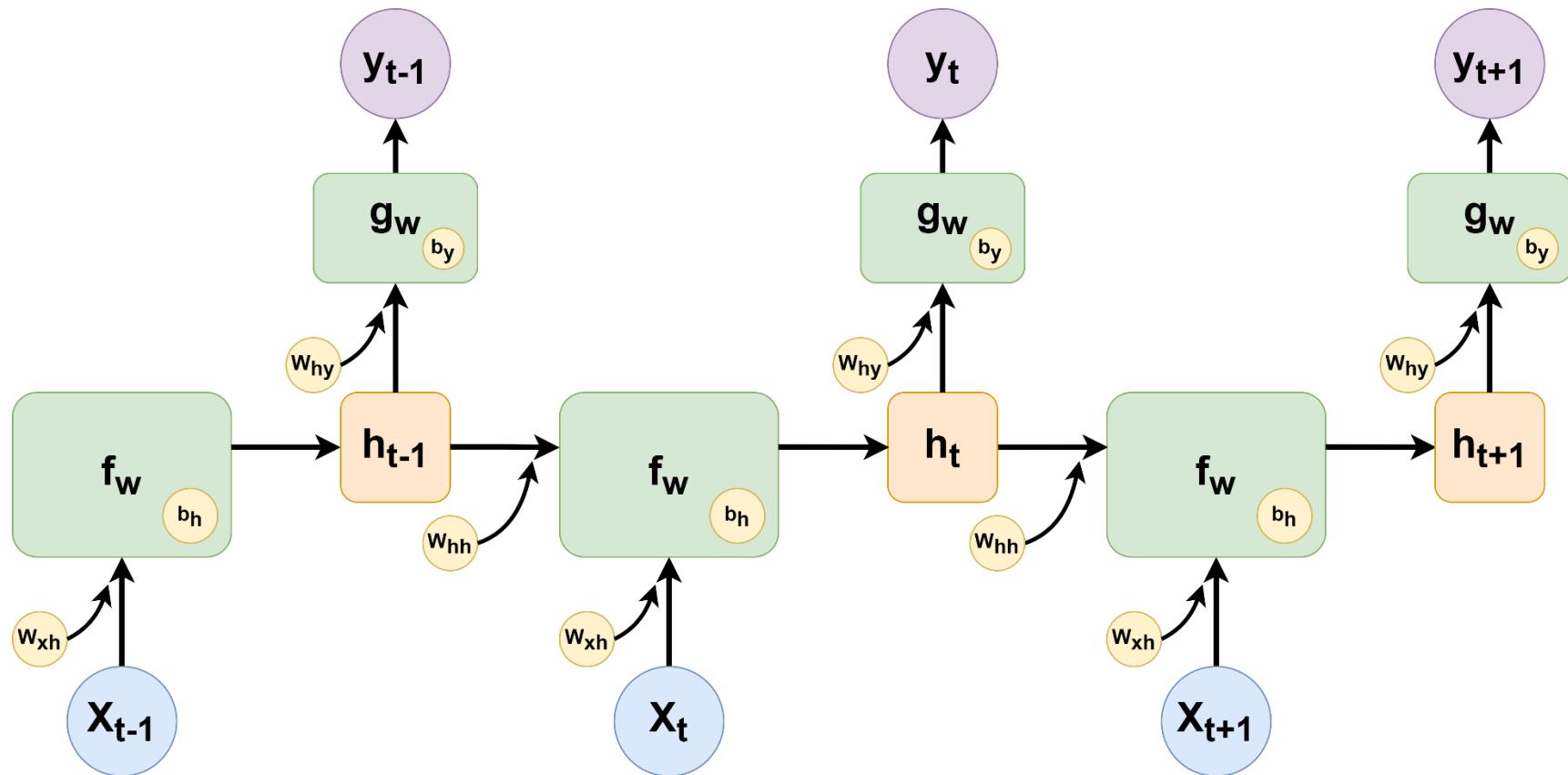
Rekurrente Neuronen – Aufbau eines RNNs

Für ein besseres Verständnis wird das zyklische Netz oftmals ausgerollt dargestellt.



Rekurrente Neuronen – Aufbau eines RNNs

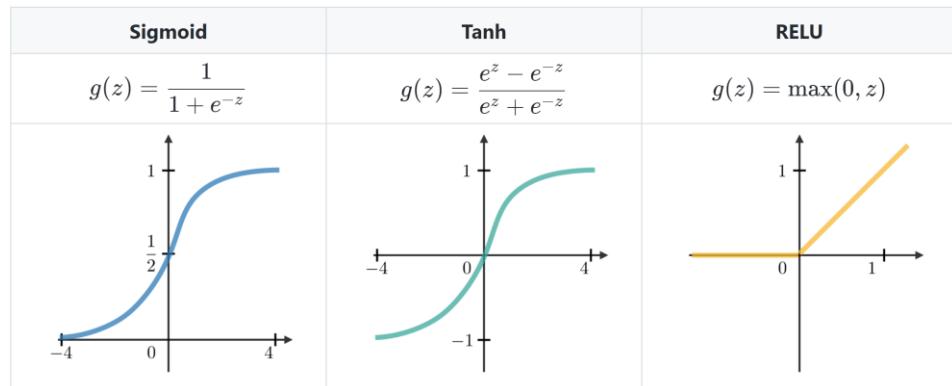
Bei einem klassischen RNN werden zu jedem Zeitpunkt t der gleiche Bias b , die gleichen Gewichte W und die gleichen Aktivierungsfunktionen f, g verwendet.



Vanilla RNN

- Parameter

- **Eingabe** X_t
- **Hidden State** h_{t-1}



- Konstanten (in jedem Zeitschritt t gleich)

- **Gewichtsvektoren** W_{hx}, W_{hh}, W_{yh}
- **Bias** b_h, b_y

- Aktivierungsfunktion f_w berechnet **Hidden State** h_t

$$h_t = f_w(x_t, h_{t-1})$$

$$h_t = \tanh(W_{hh} * h_{t-1} + W_{hx} * x_t + b_h)$$

- Aktivierungsfunktion g_w berechnet **Ausgabe** y_t

$$y_t = g_w(h_t)$$

$$y_t = \text{softmax}(0, W_{yh} * h_t + b_y)$$

Bei der One Hot Kodierung (oder 1-aus-n-Code) wird jeder Zustand durch ein eigenes Bit abgebildet.

- Genau ein Bit 1, restlichen $n-1$ Bits 0
- Darstellung von n Zuständen benötigt n Bits
- Binärkodierung bräuchte nur \sqrt{n} Bits

■ Anwendung

- Konvertierung kategorialer Daten in Numerische
- Vermeidung von unerwünschten Beziehungen

Wert	is_Unknown	is_Char_A	is_Char_B	is_Char_C	is_Char_D	...
A	0	1	0	0	0	...
B	0	0	1	0	0	...
C	0	0	0	1	0	...
D	0	0	0	0	1	...
...	0	0	0	0	0	...

Dezimal	1-aus-n-Code (n=10)	Binär
0	0000000001	0000
1	0000000010	0001
2	0000000100	0010
3	0000001000	0011
4	0000010000	0100
5	0000100000	0101
6	0001000000	0110
7	0010000000	0111
8	0100000000	1000
9	1000000000	1001



Struktur von RNNs

Bildquelle: <https://havitsteelstructure.com/steel-structure-house-building/>

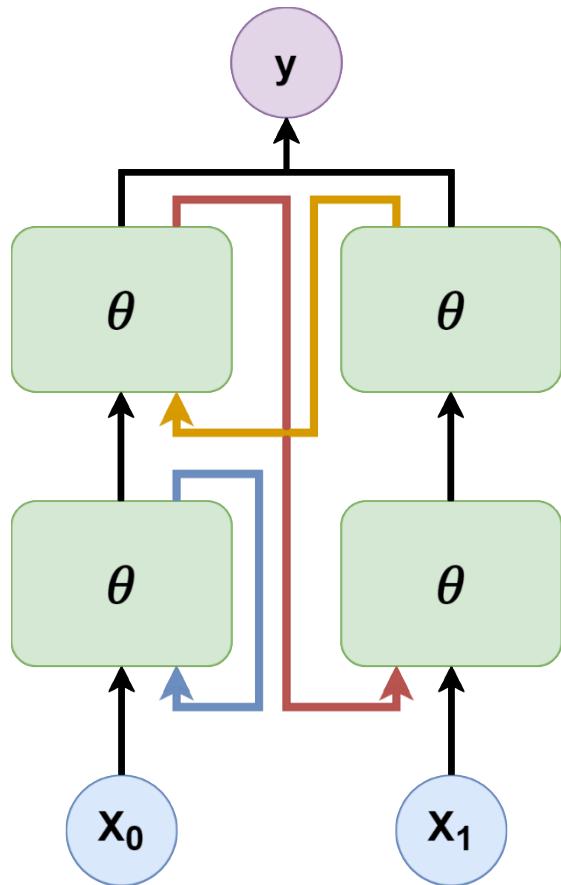
28.05.2021

Paul Krüger | Machine Learning 2 - RNNs

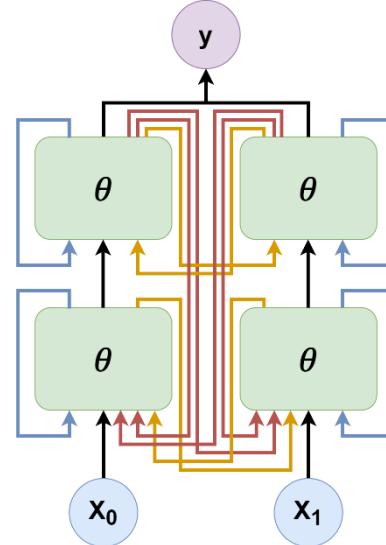
16

Struktur von RNNs – Verbindungsarten

Im Allgemeinen gibt es vier Arten von Verbindungen bei RNNs.

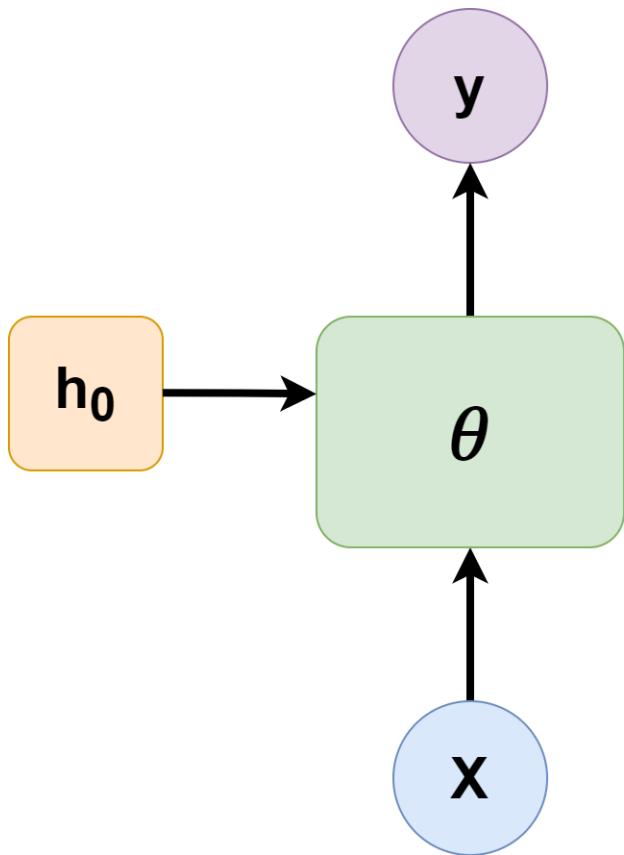


- Direkte Rückkopplung
- Indirekte Rückkopplung
- Seitliche Rückkopplung
- Vollständig verbundenes Netz



One to One

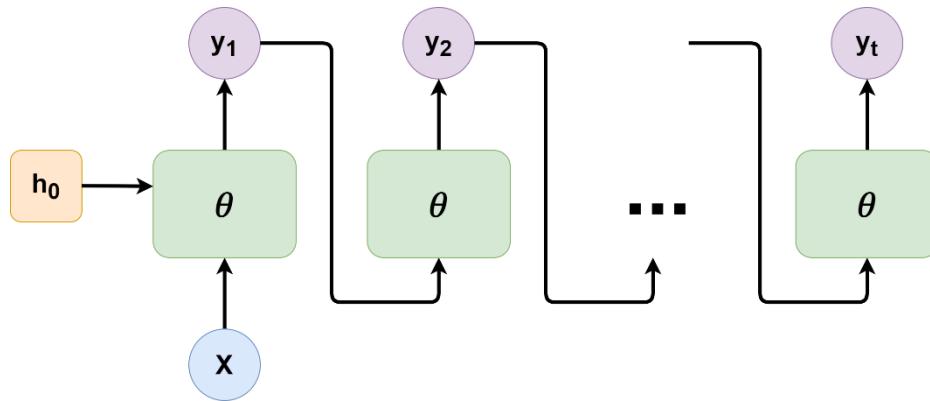
$$T_x = T_y = 1$$



- Klassisches neuronales Netz
 - Jeder Input erzeugt ein Output
 - Sequentielle Verarbeitung des Inputs
 - Berücksichtigung des Hidden States
-
- **Anwendungsbeispiel:**
 - Erzeugung von Hausnummern

One to Many

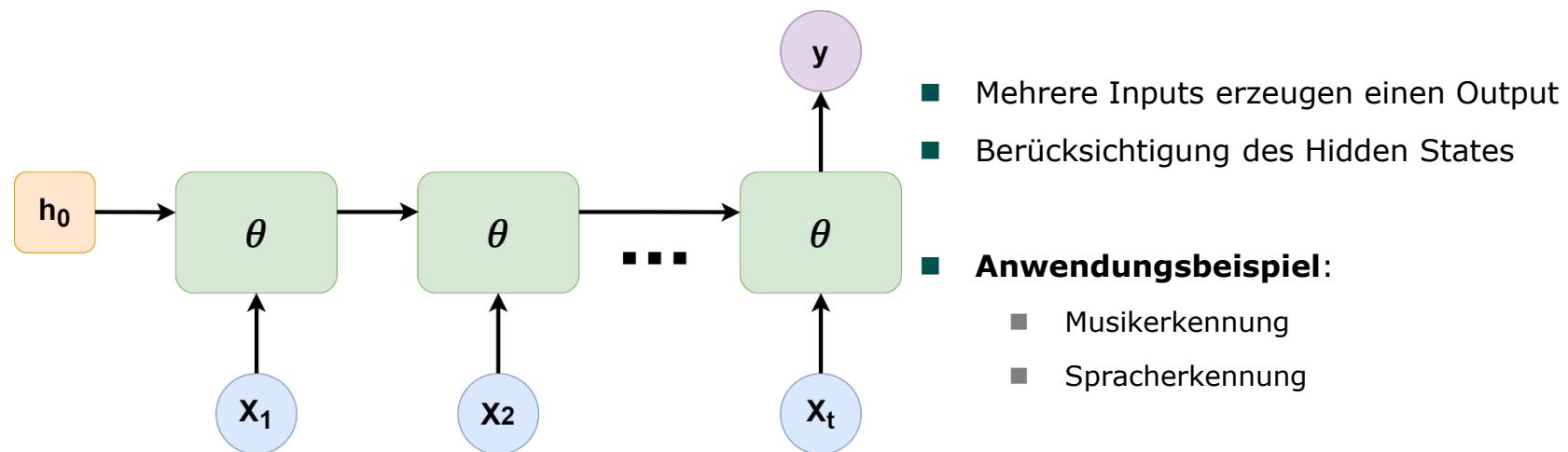
$$T_x = 1, T_y > 1$$



- Ein Input erzeugt mehrere Outputs
- Berücksichtigung des Hidden States
- **Anwendungsbeispiel:**
 - Erzeugung von Musik
 - (mehrfache) Objekterkennung in Bildern
 - Generierung von Texten

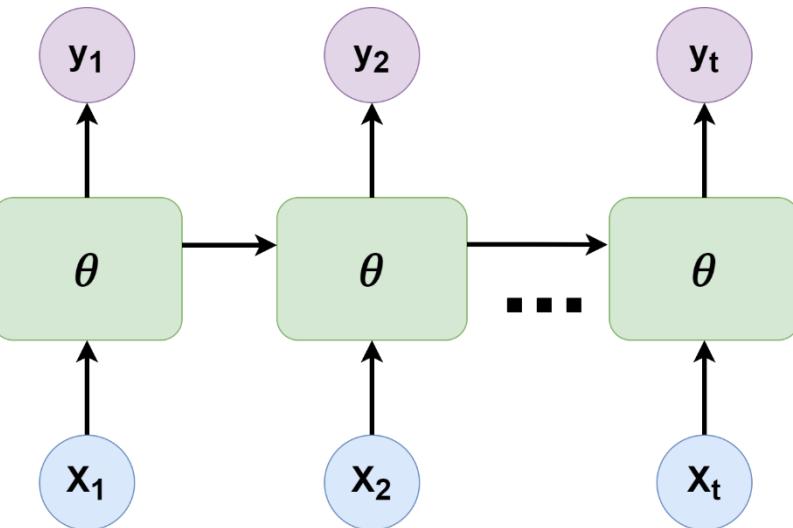
Many to One

$$T_x > 1, T_y = 1$$



Many to Many

$$T_x = T_y, T_x > 1$$



- Mehrere Inputs erzeugen mehrere Outputs
- Berücksichtigung des Hidden States
- **Anwendungsbeispiel:**
 - Eigennamenerkennung (Named Entity Recognition)

Ousted WeWork founder Adam Neumann lists his Manhattan penthouse for \$37.5 million

[organization]

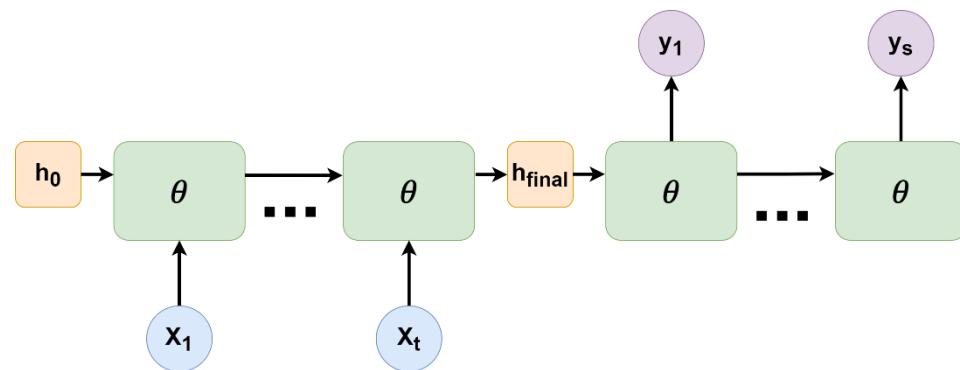
[person]

[location]

[monetary value]

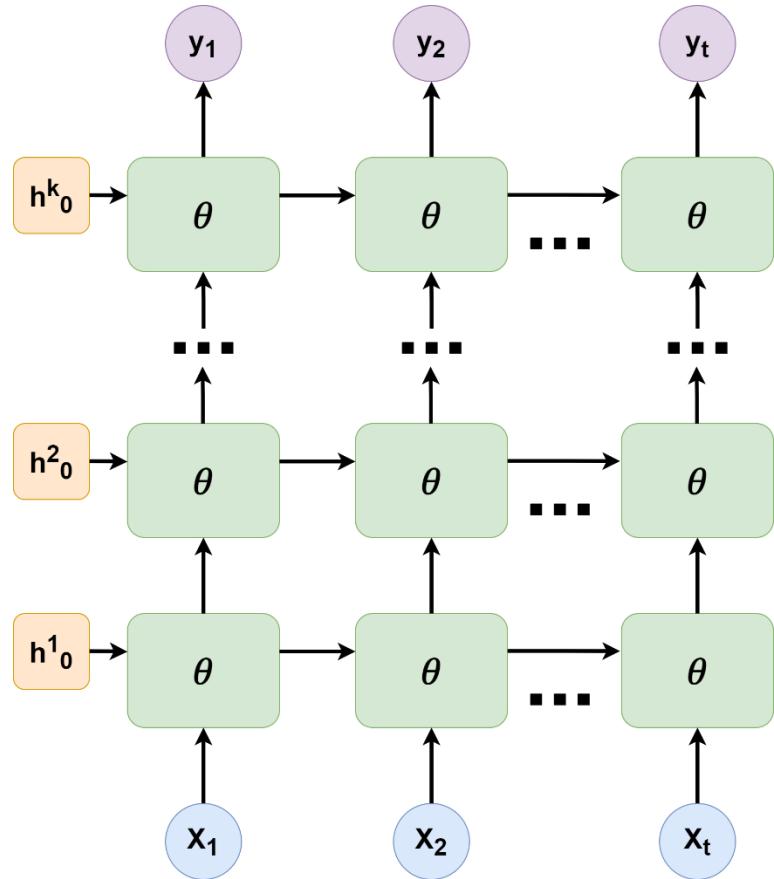
Encoder Decoder

$$T_x \geq 1, T_y \geq 1$$



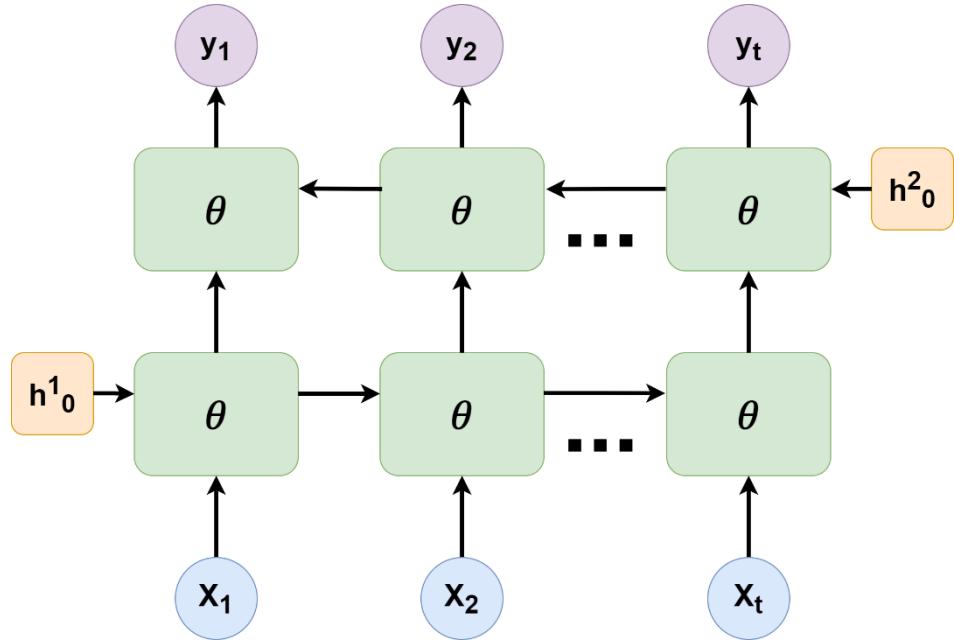
- Kombination Many-To-One und One-To-Many
 - Mehrere Inputs erzeugen mehrere Outputs
 - Output ist i.d.R. zeitversetzt
 - Anzahl der Inputs und Outputs kann unterschiedlich sein
 - Finaler Hidden State wird zur Generierung der Outputs genutzt
-
- **Anwendungsbeispiel:**
 - Übersetzung von Text
 - Handschrifterkennung

Deep Recurrent Neural Networks



- Stack aus RNNs
- Mehrere Layer mit eigenen Hidden States
- **Anwendungsbeispiel:**
 - Komplexe Zusammenhänge

Bidirectional Recurrent Neural Networks

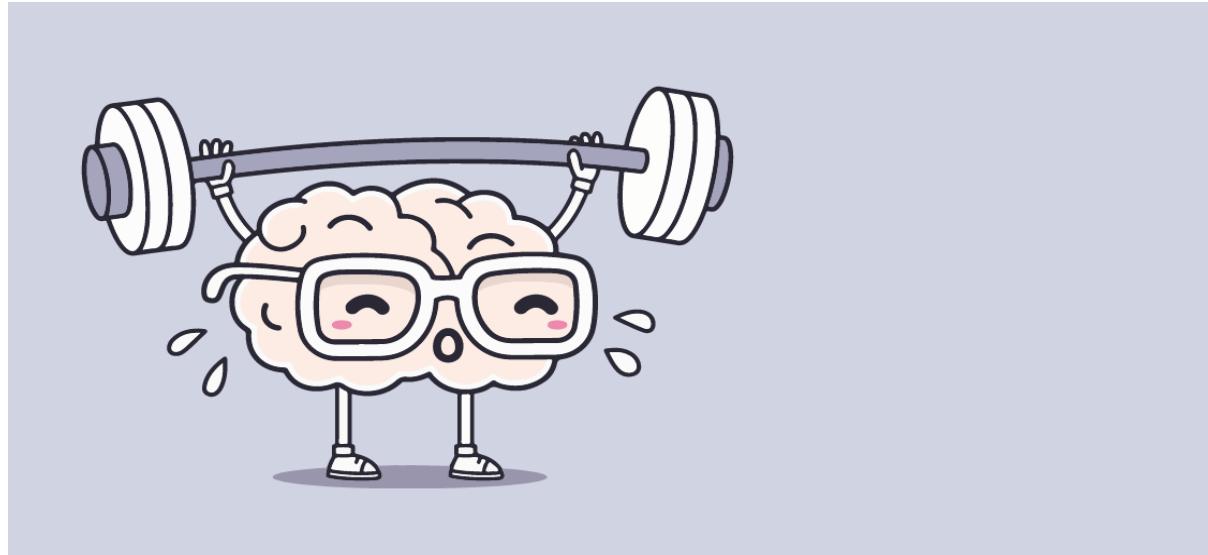


- Hidden State bidirektional weitergeben
- Abhängigkeit zu **nachfolgenden** Inputs
- **Anwendungsbeispiel:**
 - Auffüllen von Textsequenzen

Ich habe _____. Hunger. Ich könnte jetzt zwei Burger essen!

Zusammenfassung des bisherigen Wissens über RNNs.

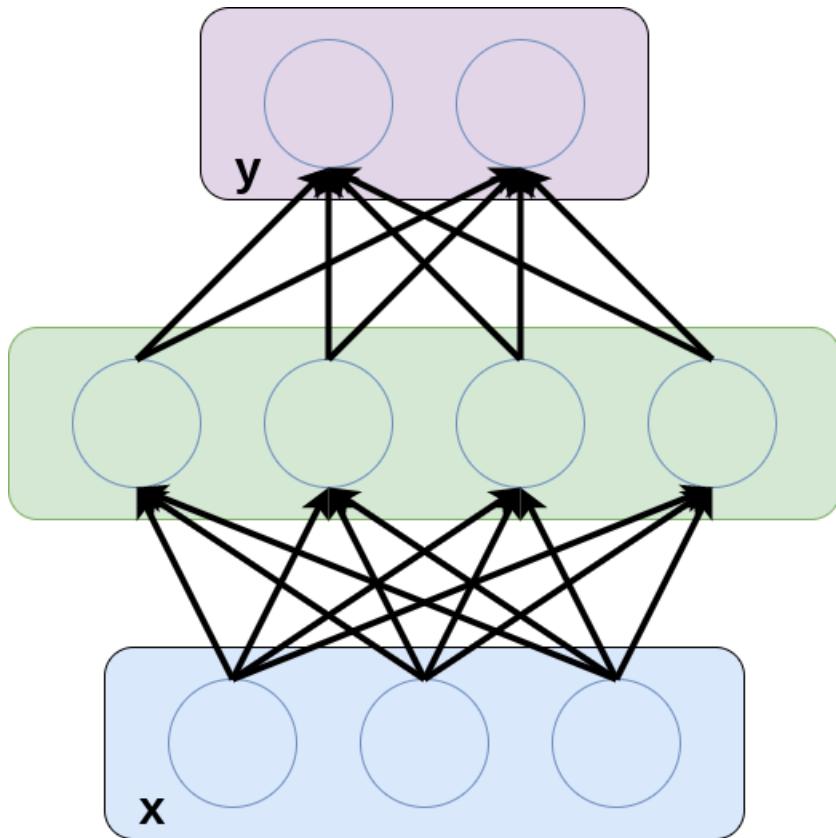
- Was sind klassische Einsatzgebiete von RNNs?
 - Wenn Länge von Input oder Output variiert → Sequentielle Verarbeitung
 - Wenn Wissen aus dem vorherigen Schritt benötigt wird
- Wozu ist der Hidden State bei einem RNN da?
 - Das „Gedächtnis“
- Wie viele Gewichte gibt es bei einem klassischen RNN?
 - **Gewichtsvektoren** W_{hx}, W_{hh}, W_{yh} , **Bias** b_h, b_y
- Wozu benötigt das RNN zwei Aktivierungsfunktionen?
 - Zur Berechnung des neuen Hidden State
 - Zur Berechnung des Outputs auf Basis des Hidden States



Training von RNNs

Bildquelle: <https://www.hammer.de/fitnesswissen/gesundheit/bewegung-haelt-gehirn-fit>

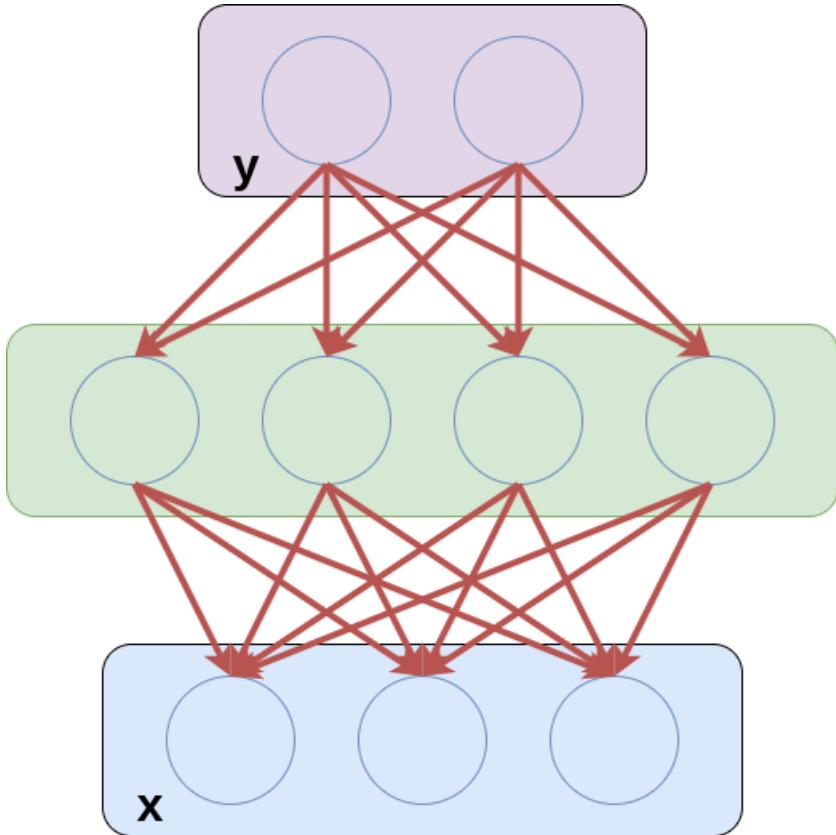
Forwardpass



Auf gegebenem Datenpunkt

- Forward Propagation der Daten durch das Netzwerk
- Bestimme Output aller Schichten
- Bestimme Loss der Ausgabe

Backpropagation



- Setze

$$\delta(o, o) = \frac{\partial L}{\partial s_o} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial s_o} = \frac{\partial L}{\partial o} * \theta'(s_o)$$

- Nutze Rekursion

$$\delta(h_r, o) = \theta'(s_r) * \sum_{i \in A(r)} w_{r+1} * \delta(h_i, o)$$

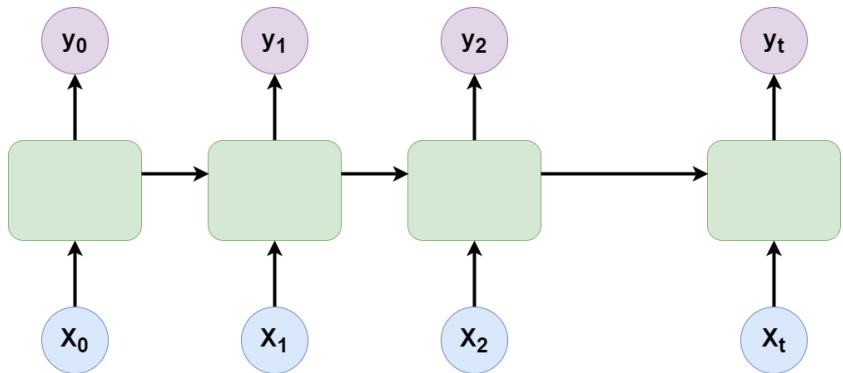
- Berechne lokale Gradienten

$$\begin{aligned}\frac{\delta L}{\delta w_r} &= \delta(h_r, o) * h_{r-1} \\ \frac{\delta L}{\delta b} &= \delta(h_r, o) * 1\end{aligned}$$

- Nutze lokale Gradienten für Gewichtsupdates

$$\begin{aligned}w_r &\leftarrow w_r - \frac{\delta L}{\delta w_r} \\ b_r &\leftarrow b_r - \frac{\delta L}{\delta b_r}\end{aligned}$$

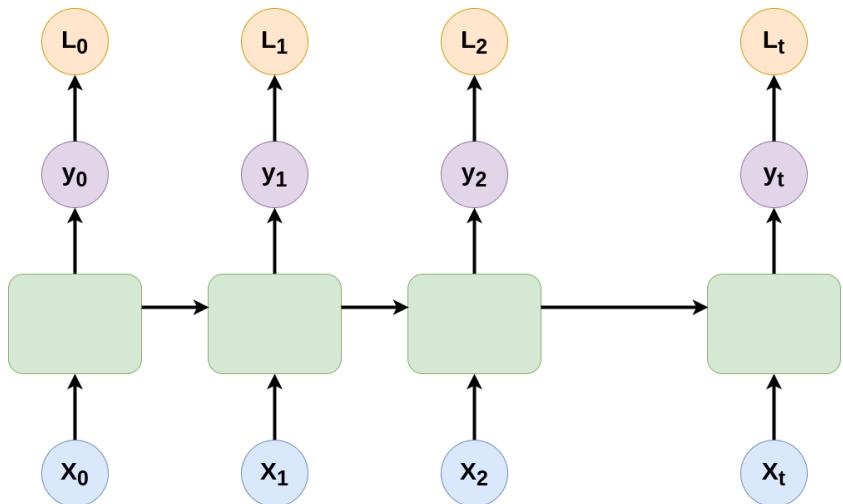
Forwardpass im RNN



Auf gegebener Sequenz von Daten

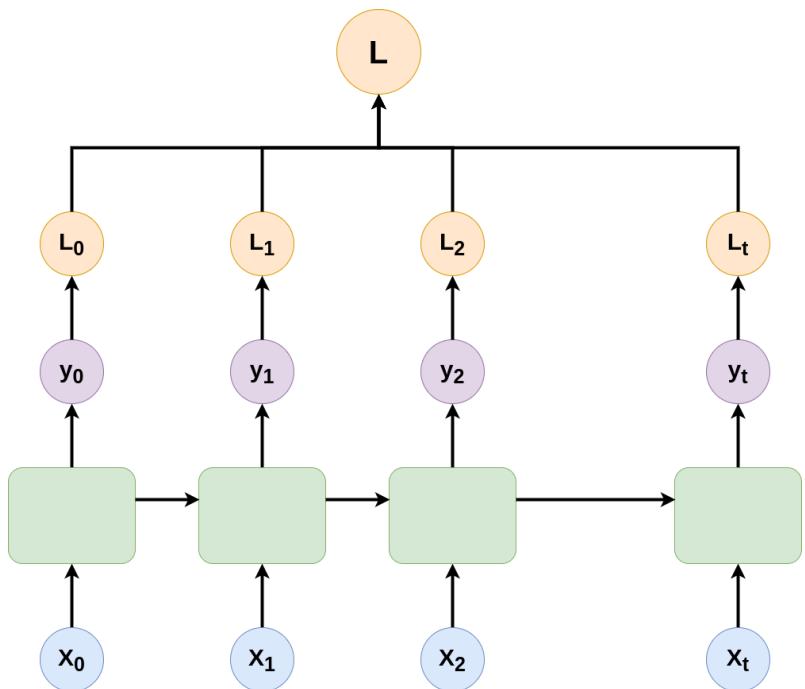
- Forward Propagation der jedes Datenpunktes der Sequenz durch das RNN
- Bestimme output jedes Zeitschritts
- Bestimme Loss – aber wie?

Loss auf RNNs



- Bestimme Loss der einzelnen RNN-Zellen

Loss auf RNNs

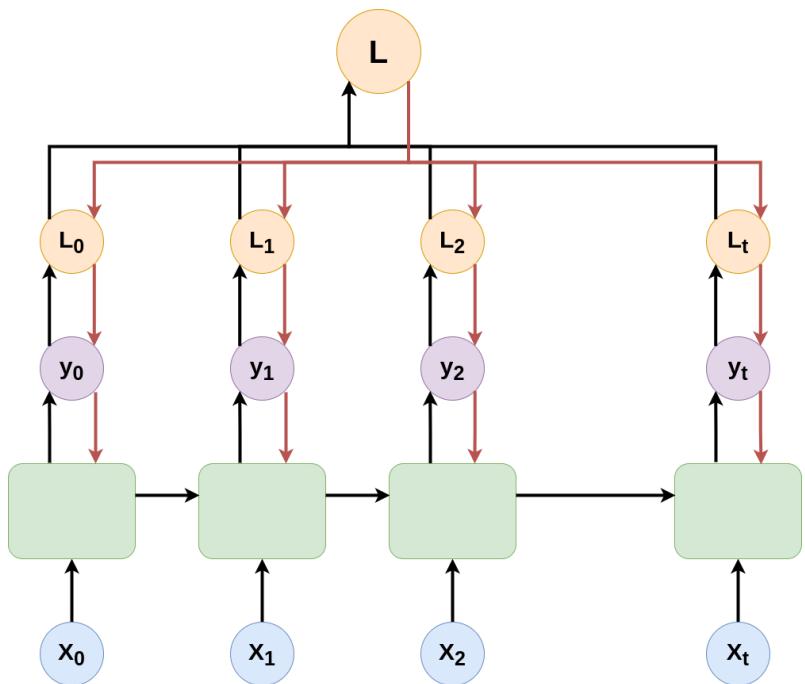


- Bestimme gesamten Loss als Summe

$$L = \frac{1}{T} \sum_{t=0}^T L_t$$

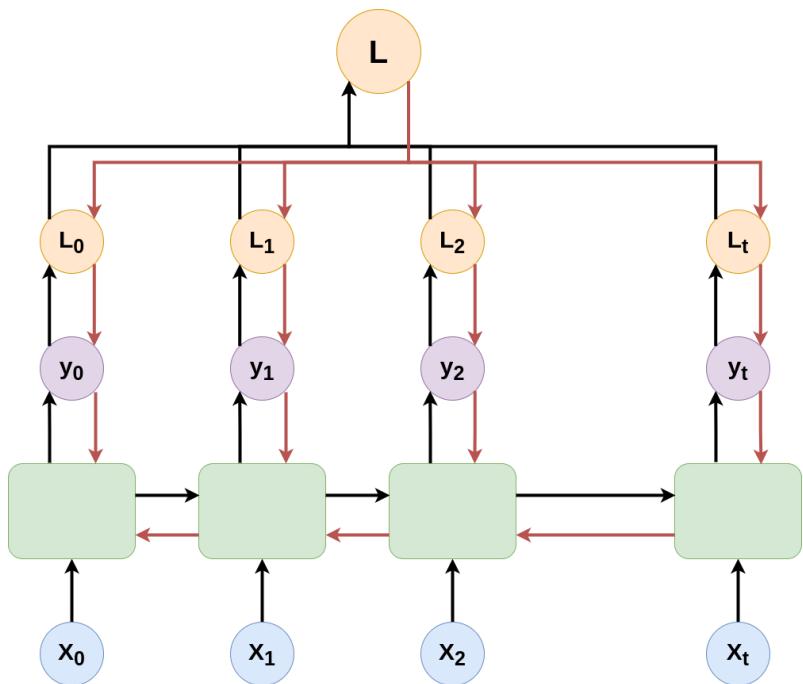
Training eines RNN

Backpropagation



- Triviale Idee:
 - Betrachte jeden Zeitschritt wie ein klassisches Feedforward Netz
- Wir verlieren Einfluss von h_{t-1}

Backpropagation Through Time



2 Schritte bei Backpropagation

1. "Klassische" Backpropagation bei Berechnung des Outputs y
2. Backpropagation through time im rekurrenten Teil (Hidden State h_t)

Backpropagation Through Time

- Gradientenberechnung für ein Gewicht W_h

$$\frac{\partial L}{\partial W_h} = \frac{1}{T} \sum_t \frac{\partial L_t}{\partial W_h}$$
$$\frac{\partial L}{\partial W_h} = \frac{1}{T} \sum_t \frac{\partial L_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial W_h}$$

- Erinnerung:

$$h_t = f(x_t, h_{t-1}, W_h)$$

- Deshalb benötigt $\frac{\partial h_t}{\partial W_h}$ rekursive Betrachtung

Backpropagation Through Time - $\frac{\partial h_t}{\partial W_h}$

- Kettenregel

$$\frac{\partial h_t}{\partial W_h} = \frac{\partial h_t}{\partial W_h} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_h}$$

- Es lässt sich zeigen, dass

$$\frac{\partial h_t}{\partial W_h} = \frac{\partial h_t}{\partial W_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_i}{\partial W_h}$$

- Viele Gradienten-Berechnungen bei vielen Zeitschritten => Sehr teuer!!!

$$\text{Probleme durch } \frac{\partial h_t}{\partial W_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^t \frac{\partial h_t}{\partial h_{j-1}} \right) \frac{\partial h_t}{\partial W_h}$$

Viele Gradientenberechnungen

- Sehr teuer
- Exploding Gradients
 - durch Gradient-clipping zum größten Teil gelöst
- Vanishing Gradients

Vanishing Gradient Problem

- Gradienten gehen für Updates von frühen Zeitschritten gegen null.
 - Lange Abhängigkeiten werden nicht berücksichtigt.
 - Netz bekommt eine Art „Bias“ auf kurze Abhängigkeiten.
- „Ich komme aus **Frankreich**... In meinem letzten Urlaub konnte ich endlich nochmal meine **Muttersprache** sprechen.“

Truncated Backpropagation Through Time

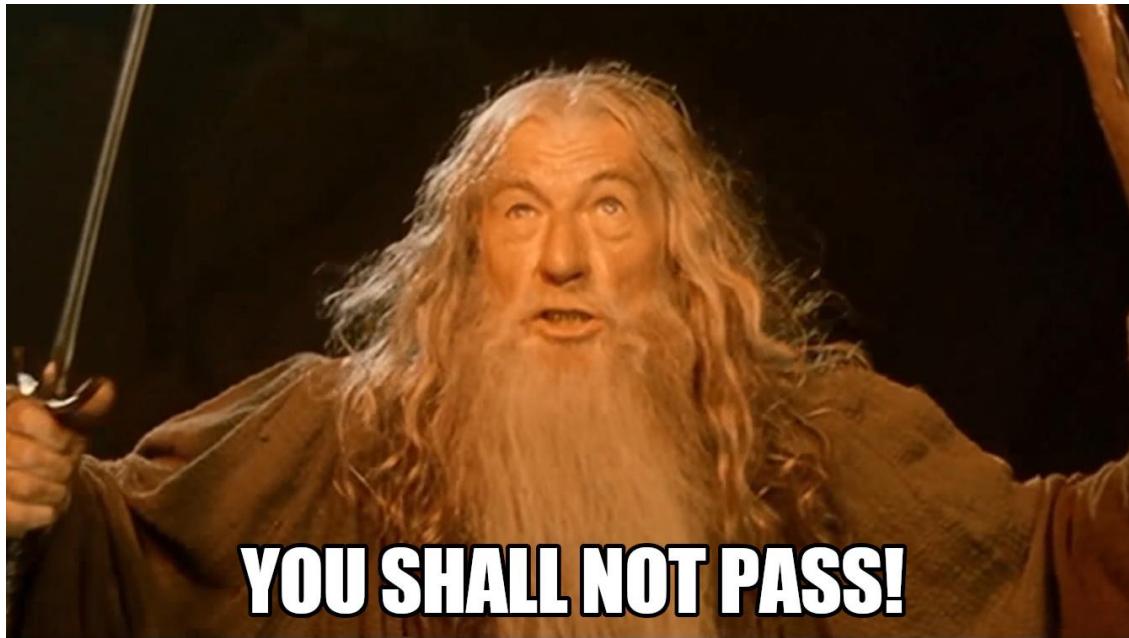
- Aufteilung der Berechnung in Teile
 - Wähle $k_1, k_2 \leq S$ mit S als Länge der Sequenz
-
1. Führe Forward-Propagation auf k_1 Zeitschritten aus
 2. Führe Backpropagation auf k_2 Zeitschritten aus
 3. Führe Gewichtsupdate aus
 4. Wiederhole 1 – 3 bis gesamte Sequenz durchlaufen ist
-
- Welche Probleme löst das?

Vanishing Gradient Problem - Lösungs Ansätze

1. Angepasste Initialisierung der Gewichtsmatrizen
 - Zum Beispiel Identität
2. Nutzung von ReLU anstatt $tanh$ oder σ
 - Ableitung der ReLU ist immer 0 oder 1
3. Ändern der Architektur



Wie könnte eine so geänderte Architektur aussehen?

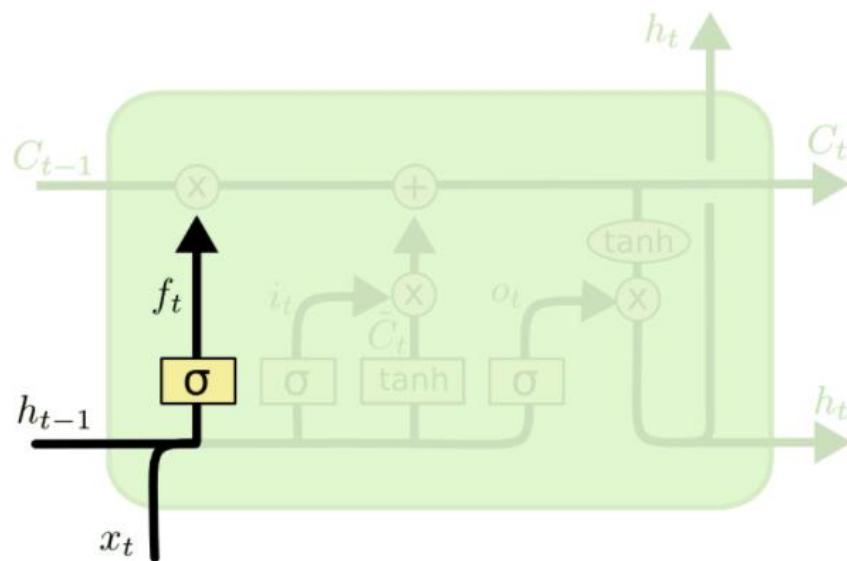


Einführung in Gated RNNs

Bildquelle: <https://knowyourmeme.com/memes/you-shall-not-pass>

Einführung in Gated RNNs – Forget Gate

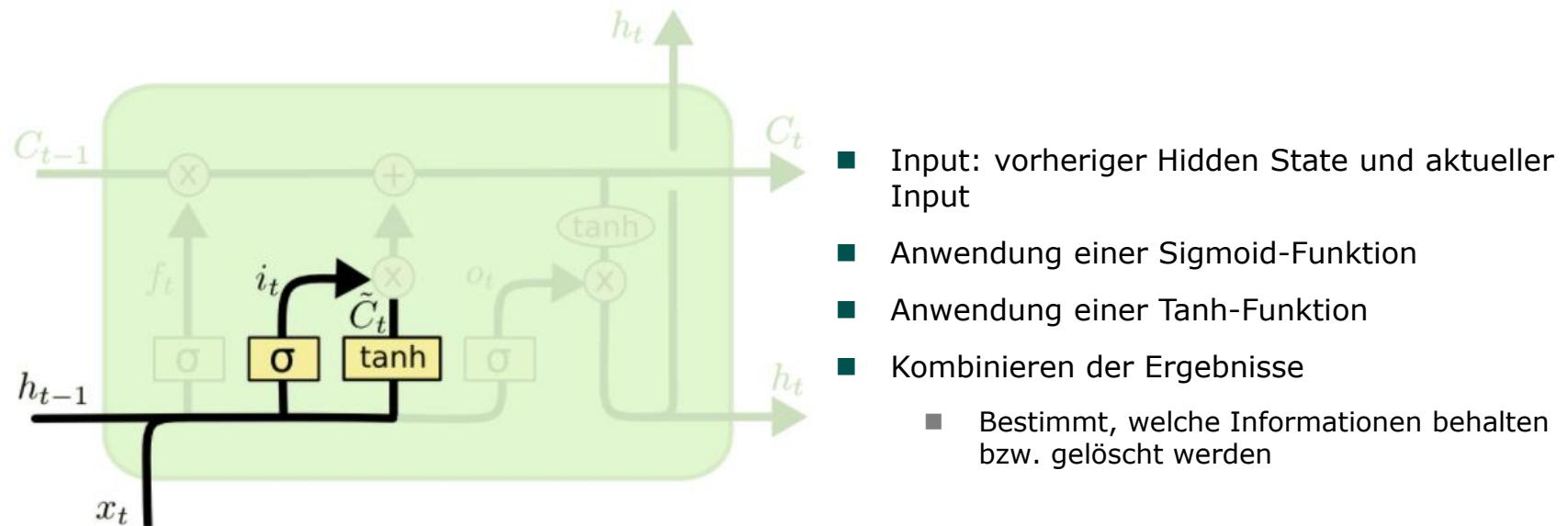
Mit dem Forget Gate wird gesteuert, welche Informationen von vorherigen Schritten vergessen werden.



- Input: vorheriger Hidden State und aktueller Input
- Anwendung einer Sigmoid-Funktion
- Output: Werte zwischen 0 und 1
 - 0: alles vergessen
 - 1: alles behalten

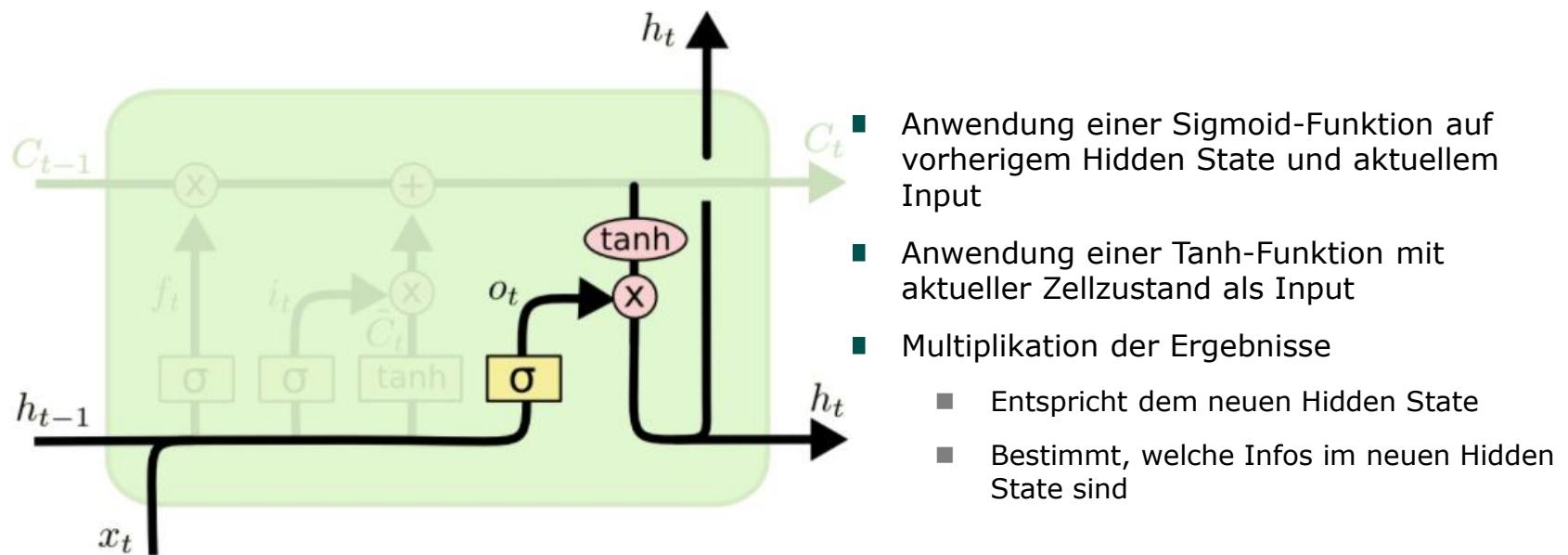
Einführung in Gated RNNs – Input Gate

Das Input Gate bestimmt, welche Informationen des aktuellen Schrittes in dem Cell State gespeichert bzw. geupdated werden.



Einführung in Gated RNNs – Output Gate

Basierend auf dem Cell State filtert das Output Gate, was ausgegeben werden soll.

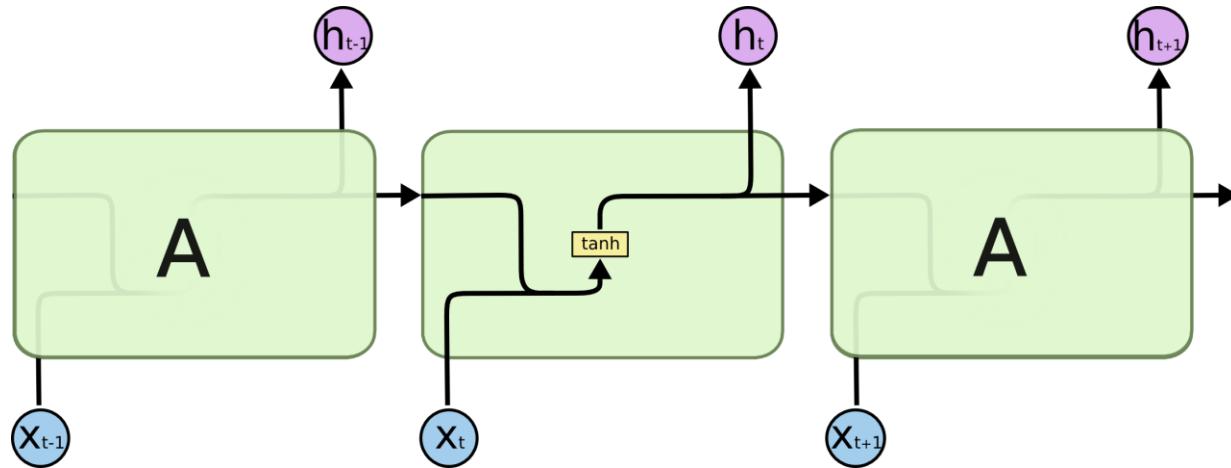




Long Short-Term Memory **(LSTM)**

Bildquelle: <https://specials-images.forbesimg.com/imageserve/5ed68862570be80006474c04/960x0.jpg?fit=scale>

Standard RNNs haben Probleme mit "Long-Term Dependencies".



- Vanishing Gradient Problem bei BPTT
- Gradientenabstieg führt zu fast keiner Änderung

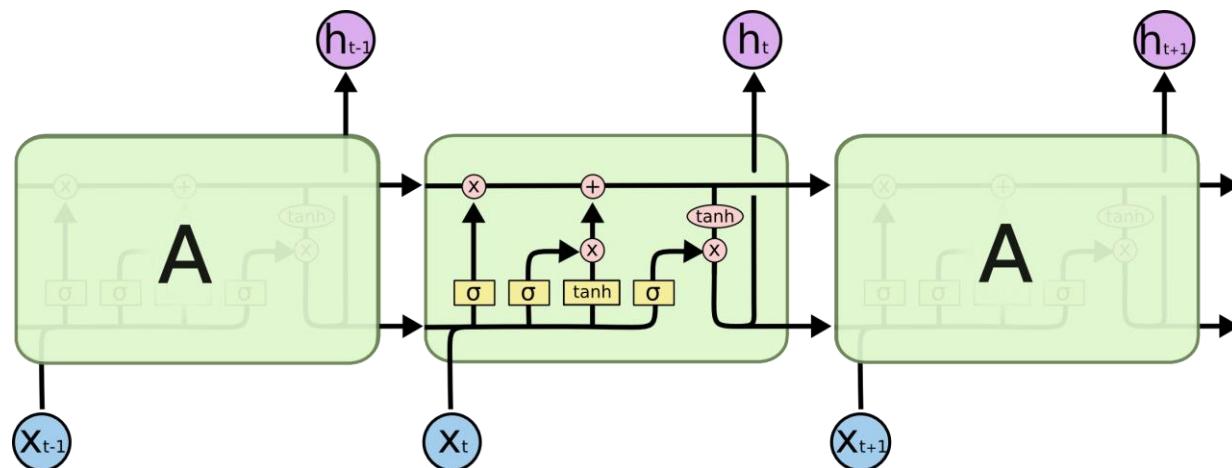


Zusammenhänge über einen längeren Zeitraum hinweg können nicht gelernt werden.

„Ich komme aus **Frankreich**... In meinem letzten Urlaub konnte ich endlich nochmal meine **Muttersprache** sprechen.“

LSTM – Austausch der RNN-Zelle durch eine LSTM-Zelle

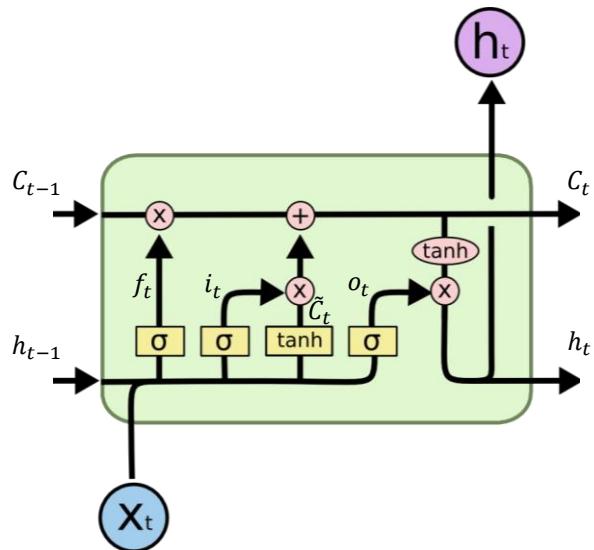
Hinzunahme eines Cell States, ermöglicht Informationen längerfristig zu speichern.



- Cell State kann als „Langzeitgedächtnis“ gesehen werden
- Gates können den Cell State zu jedem Zeitpunkt überschreiben
- Input zu jedem Zeitpunkt: aktueller Datenpunkt, vorheriger Hidden State, vorheriger Cell State

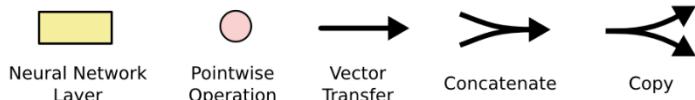
LSTM – Forward Propagation

Gates enthalten Single-Layer-NN.



$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{c}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_0) \end{aligned}$$

$$\begin{aligned} C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\ h_t &= o_t * \tanh(C_t) \end{aligned}$$



Wie viele und welche Parameter müssen trainiert werden?

Gewichte und Bias der Single-Layer-NN:

W_f und b_f

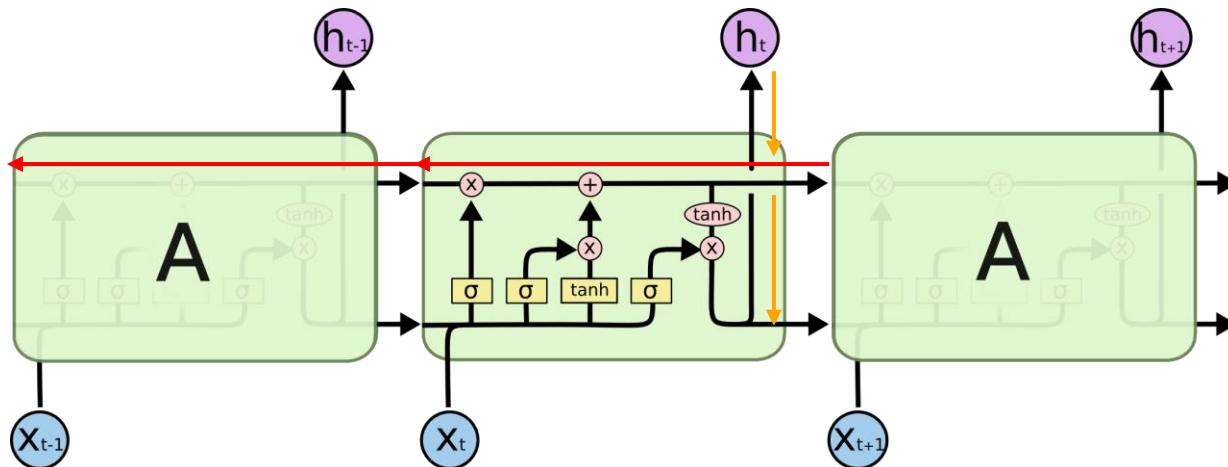
W_i und b_i

W_C und b_C

W_o und b_o

LSTM – Backpropagation through time

Das Vorgehen können wir von den Standard RNNs übernehmen.



$$\frac{\partial L}{\partial W} = \frac{1}{T} \sum_T \frac{\partial L_T}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial C_T} \frac{\partial C_T}{\partial C_{T-1}} \dots \frac{\partial C_2}{\partial C_1} \frac{\partial C_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \underbrace{\frac{\partial h_T}{\partial C_T} \left(\prod_{t=2}^T \frac{\partial C_t}{\partial C_{t-1}} \right)}_{\text{orange bracket}} \frac{\partial C_1}{\partial W}$$

$$\lim_{t \rightarrow \infty} \left(\prod_1^t 0.5 \right) = 0$$

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

LSTM – Zurück zum Vanishing Gradient Problem

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial C_T} \left(\prod_{t=2}^T \frac{\partial C_t}{\partial C_{t-1}} \right) \frac{\partial C_1}{\partial W}$$

$$\begin{aligned}\frac{\partial C_t}{\partial C_{t-1}} &= \frac{\partial}{\partial C_{t-1}} [C_{t-1} * f_t + \tilde{C}_t * i_t] \\ &= \frac{\partial}{\partial C_{t-1}} [C_{t-1} * f_t] + \frac{\partial}{\partial C_{t-1}} [\tilde{C}_t * i_t] \\ &= \frac{\partial f_t}{\partial C_{t-1}} \cdot C_{t-1} + \frac{\partial C_{t-1}}{\partial C_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial C_{t-1}} \cdot \tilde{C}_t + \frac{\partial \tilde{C}_t}{\partial C_{t-1}} \cdot i_t\end{aligned}$$

$$\begin{aligned}\frac{\partial C_t}{\partial C_{t-1}} &= \sigma'(\mathbf{W}_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} * \tanh'(C_{t-1}) \cdot C_{t-1} \\ &\quad + \mathbf{f}_t \\ &\quad + \sigma'(\mathbf{W}_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} * \tanh'(C_{t-1}) \cdot \tilde{C}_t \\ &\quad + \sigma'(\mathbf{W}_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} * \tanh'(C_{t-1}) \cdot i_t\end{aligned}$$

i

Für mathematisch orientierte Hörer: Eine Herleitung finden Sie im Anhang

$$\prod_{t=2}^T \frac{\partial C_t}{\partial c_{t-1}} = \prod_{t=2}^T A_t(\mathbf{W}_f) + \mathbf{f}_t + B_t(\mathbf{W}_i) + D_t(\mathbf{W}_c) \quad (1)$$

Wie löst/mindert (1) dieses Problem

- Gradient als additive Funktion
 - Bessere Balancierung der Gradienten
 - Zeitabhängigkeit, sodass Gradient unterschiedliche Werte annehmen kann
 - f_t als ein Summand des Gradienten
 - Direkter Einfluss sorgt für einen effektiven Einstellparameter.
- Die Wahrscheinlichkeit, dass (1) verschwindet wird somit geringer.

LSTM – Vergleich mit RNN

LSTM hat 4x mehr trainierbare Parameter als RNN.

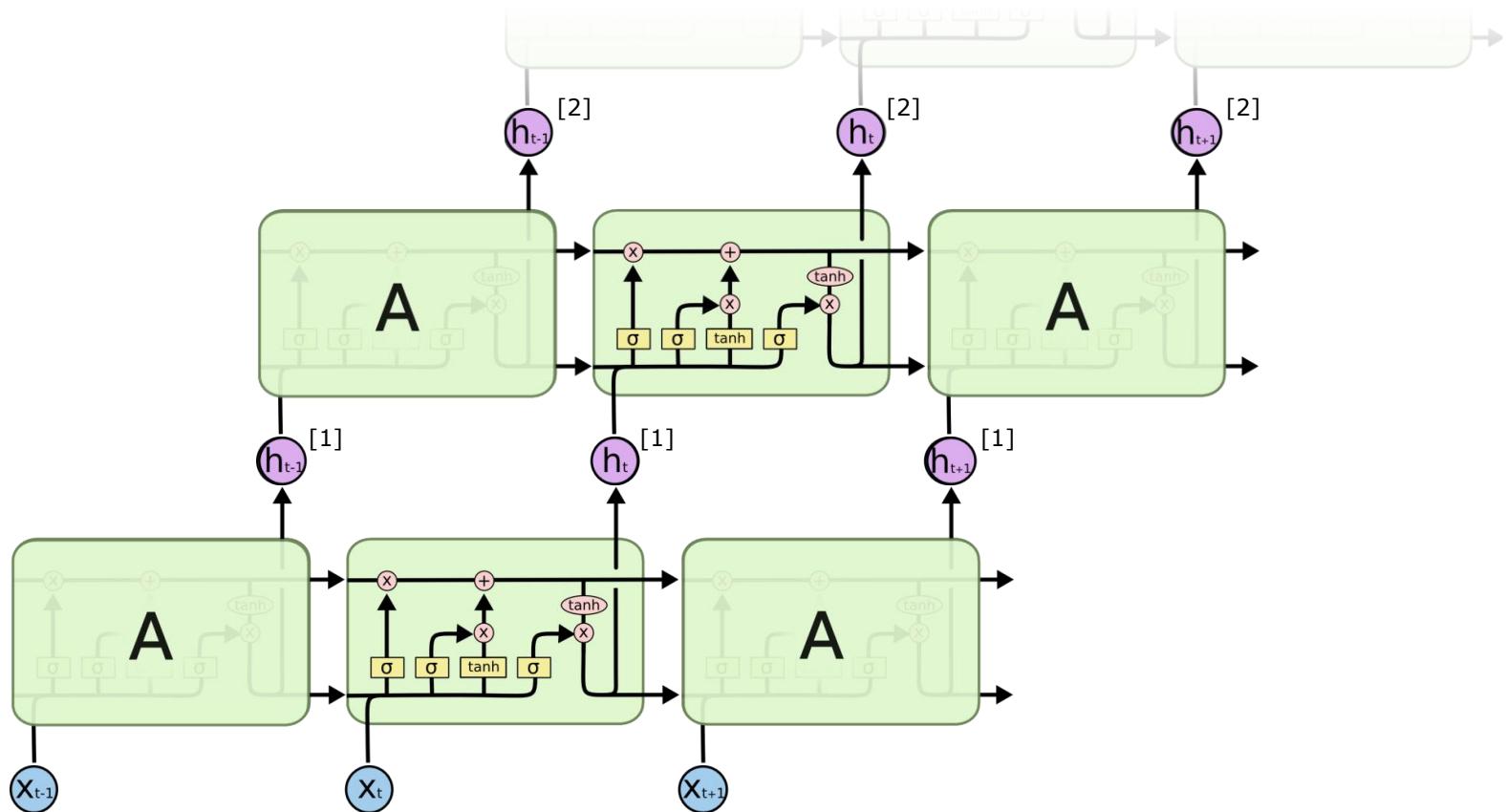
	RNN	LSTM
Trainierbare Parameter	$mn + n^2 + n = (m + n)n + n$	$4(mn + n^2 + n)$
Long-Term Dependencies	Nein	Ja
Deep RNN/LSTM	Ja	Ja

m:= Dimension des Inputs

n:= Hidden Units = Dimension des Hidden States

LSTM Varianten – Deep LSTM

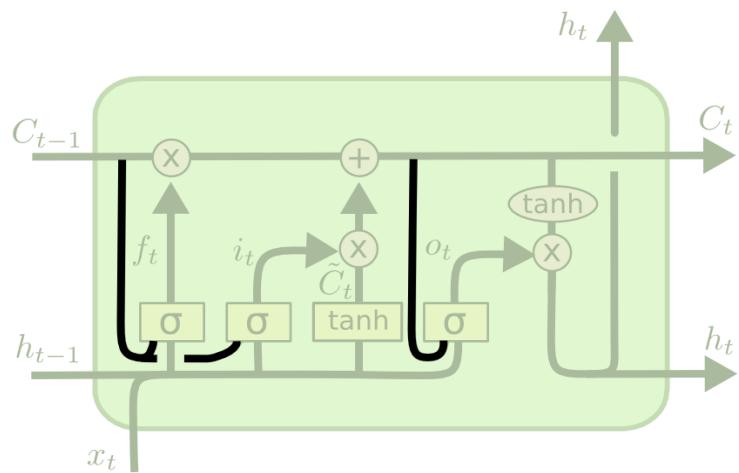
LSTMs können wie RNNs mehrere Schichten besitzen.



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

LSTM Variationen – Peephole Connections

Gates haben Zugriff auf den aktuellen Zellstatus.



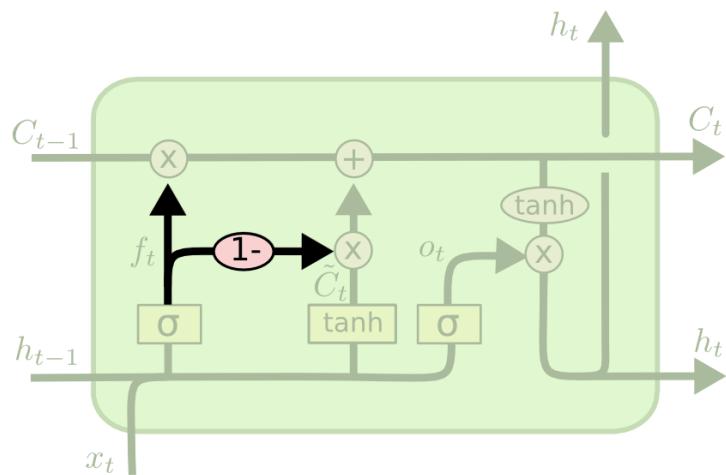
- Intuition: Bevor wir etwas ersetzen, wollen wir auch wissen, was wir ersetzen
- Auch weniger Peephole Connections möglich

$$\begin{aligned} f_t &= \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o) \end{aligned}$$

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

LSTM Variationen – Gekoppeltes Forget- und Input-Gate

Input-Gate verwendet das "Komplement" des Forget-Gates.



- Anteil der Informationen, welche vergessen wurden, werden durch das Input Gate wieder aufgefüllt
- Weniger Parameter führen zu effizienterem Training

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

$$i_t = 1 - f_t$$

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

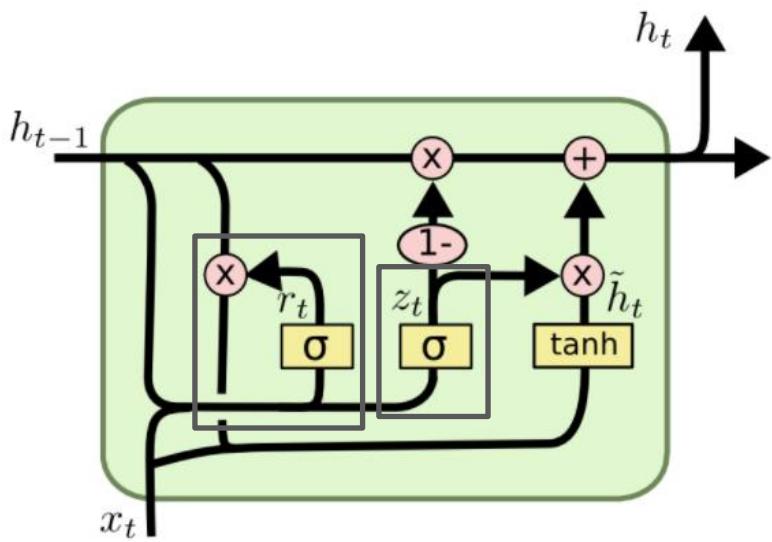


Gated Recurrent Unit **(GRU)**

Bildquelle: <https://www.elektroniknet.de/elektronik-neo/deep-learning-kann-viel-mehr-als-bildklassifikation.176537.html>

Gated Recurrent Unit

Die neue Generation der RNNs ist die Gated Recurrent Unit (GRU) und ähnlich zu der LSTM.



- Kein Cell State → nur Hidden State
- Input: vorheriger Hidden State und aktueller Input
- Reset Gate: bestimmt, welche Informationen der vergangenen Schritte wichtig sind bzw. vergessen werden können
 - Anwendung einer Sigmoid-Funktion
- Update Gate: ähnelt einer Kombination aus dem Forget und Input Gate
 - Bestimmt, welche Informationen behalten werden
 - Anwendung einer Sigmoid-Funktion

$$\begin{aligned} r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\ z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\ \tilde{h}_t &= \tanh(W_h \cdot [r_t * h_{t-1}, x_t]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \end{aligned}$$

Beide Modelle sind sich ähnlich und in der Praxis werden meist beide genutzt, um für den konkreten Fall zu entscheiden, welches besser geeignet ist.

LSTM

- 1997
- Cell State und Hidden State
- Transport der Informationen über den Cell State
- 3 Gates (Forget, Input, Output)
- Erzielt bei mehr Daten bessere Ergebnisse

GRU

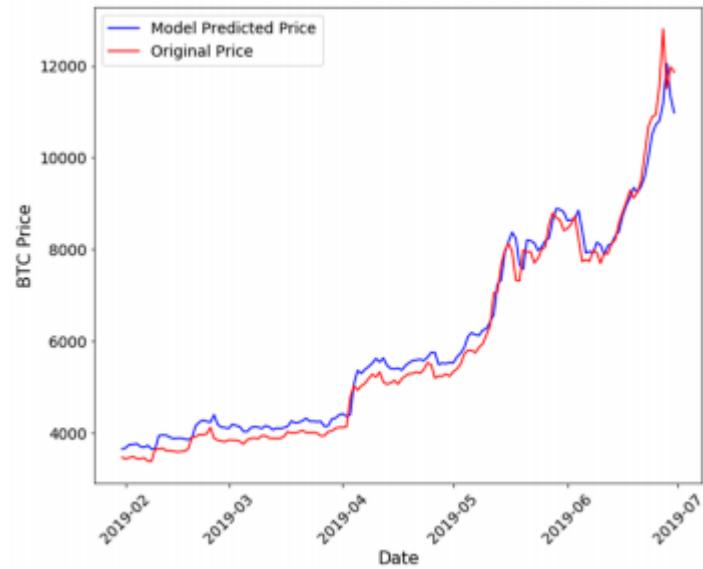
- 2014
- Keinen Cell State, nur Hidden State
- Transport der Informationen über den Hidden State
- 2 Gates (Update, Reset)
- Geringere Anzahl an Parametern → schneller zu trainieren
- Benötigt geringer Anzahl an Daten

Vergleich von LSTM und GRU

Die Preisentwicklung von Bitcoins kann mit RNNs, besonders mit GRUs, besser modelliert werden als mit ARIMA-Modellen.

- Vorher über ARIMA-Modell
- Mit RNNs höhere Accuracy und bessere Ergebnisse
- Vergleich von Simple RNNs, LSTM und GRU
- Ergebnis: als finales Modell wurde GRU gewählt

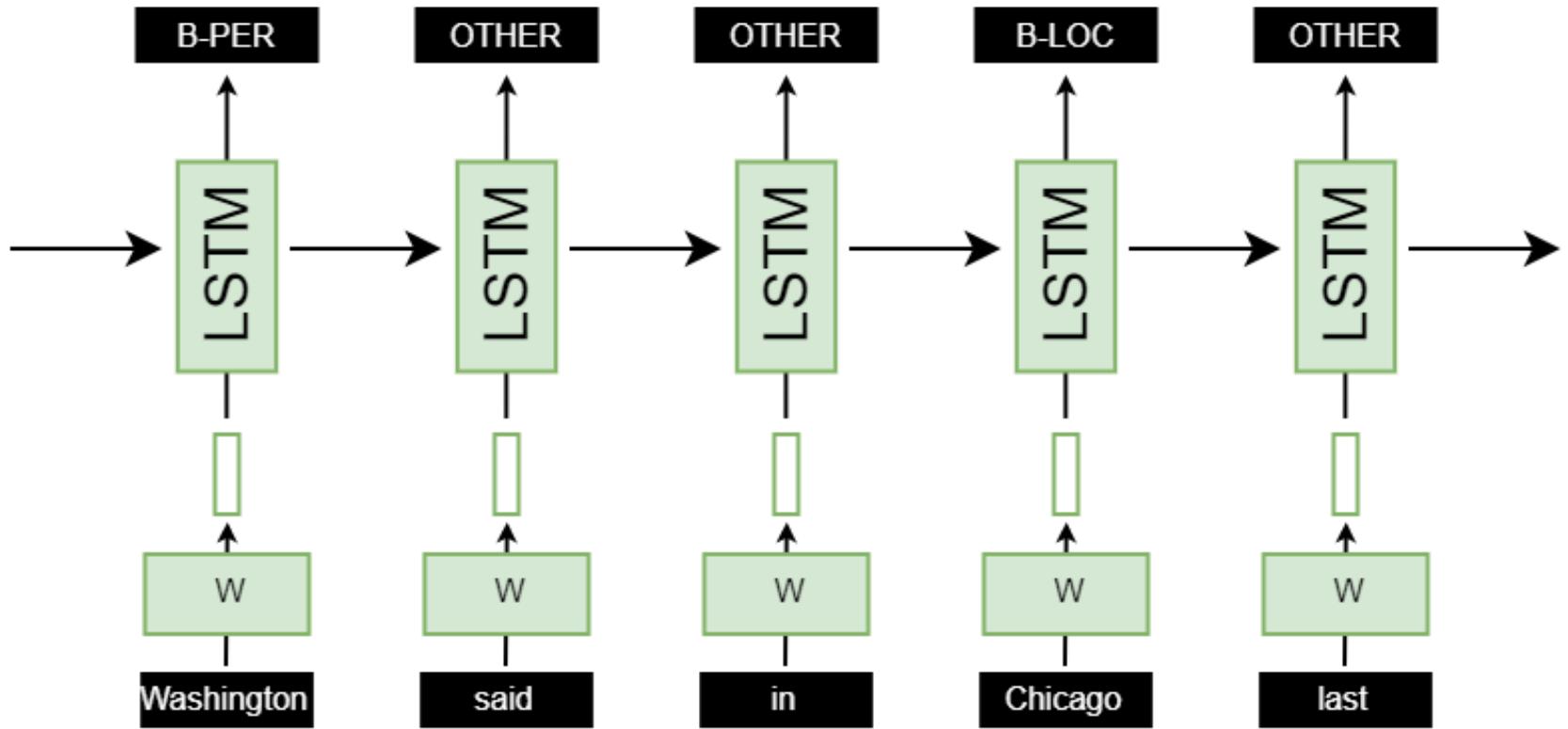
Modell	RMSE Train	RMSE Test
Simple RNN	0.020	0.031
LSTM	0.010	0.024
GRU	0.010	0.019





Anwendungen

Im Rahmen der natürlichen Sprachverarbeitung werden RNNs für z.B. Sequence Labeling eingesetzt.



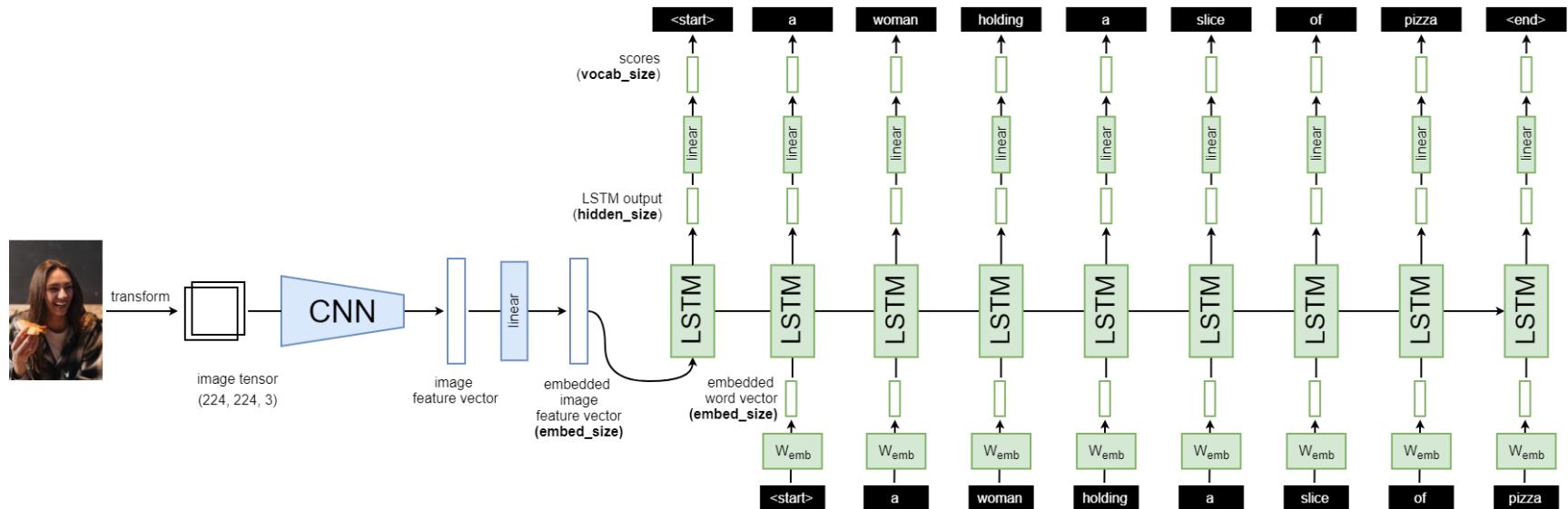
Textgenerierung mithilfe von RNNs kann die Struktur der gelernten Werke sehr gut reproduzieren. Semantisch sind sie allerdings in der Regel schwach.

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
    rw->name = "Getjbbregs";
    bprm_self_clearl(&iv->version);
    regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
    return seitable;
}
```

Quelle: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Anwendungen

Mithilfe aus einer Kombination von CNN und RNN können Bildbeschriftung generiert werden.



Quelle: <https://thinkautonomous.medium.com/rnns-in-computer-vision-image-captioning-597d5e1321d1>

Tesla setzt bei seinem Autopiloten auf RNNs, welche Bilder aus den Kameras auswertet.



- Tesla nutzt RNNs mit Kameras für seinen Autopiloten

Quelle: <https://heartbeat.fritz.ai/computer-vision-at-tesla-cd5e88074376>

RNNs finden auch Anwendung in der Call Center Analyse.



- Analyse des Anrufs wird durchgeführt
- Die Analyse ergibt warum der Support Mitarbeiter erfolgreich war oder auch nicht
- Dient eher zur Quantitativen Analyse
(Qualitative Analysen können auch Sachbearbeiter durchführen)

Quelle: <https://towardsdatascience.com/a-machine-learning-approach-to-automated-customer-satisfaction-surveys-946d2604e309>

Für die Call Center Analyse Bedarf es eines Dreischritts.



Quelle: <https://towardsdatascience.com/a-machine-learning-approach-to-automated-customer-satisfaction-surveys-946d2604e309>

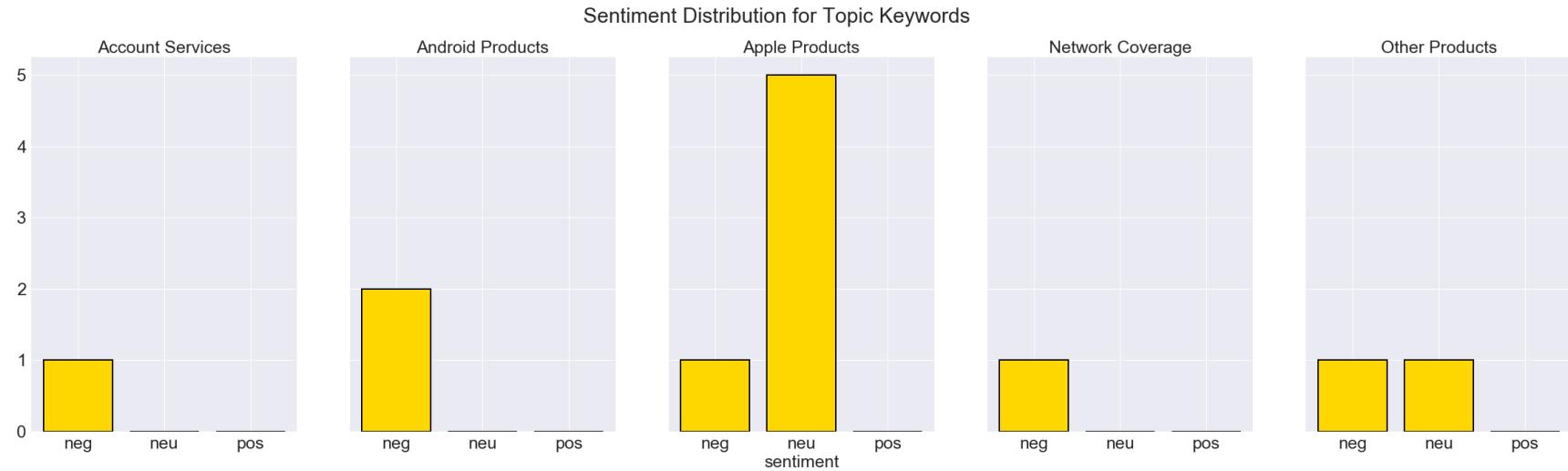
Das Ergebnis einer solchen Call Center Analyse hier einmal visuell aufbereitet.



Quelle: <https://towardsdatascience.com/a-machine-learning-approach-to-automated-customer-satisfaction-surveys-946d2604e309>

Anwendungen

Die Sentiment Verteilung pro Stichwort kann für weitere Strategien hilfreich sein.



Quelle: <https://towardsdatascience.com/a-machine-learning-approach-to-automated-customer-satisfaction-surveys-946d2604e309>

Was haben wir heute gelernt?

1. Rekurrente Neuronen

2. Struktur von RNNs

- a. Many-to-many, many-to-one, one-to-many
- b. Deep RNNs
- c. Uni-, Bidirektional

3. Training von RNNs

- a. Backpropagation Through Time
- b. Vanishing Gradient Problem

4. Gated RNNs

- a. Input / Output
- b. Forget Gates

4. LSTM

- a. Struktur (Kombination der Gates)
- b. Gradientenfluss (+ Auswirkung auf Vanishing Gradient Problem)
- c. Varianten (Peephole Connections, gekoppelte forget and input gates)

5. GRU

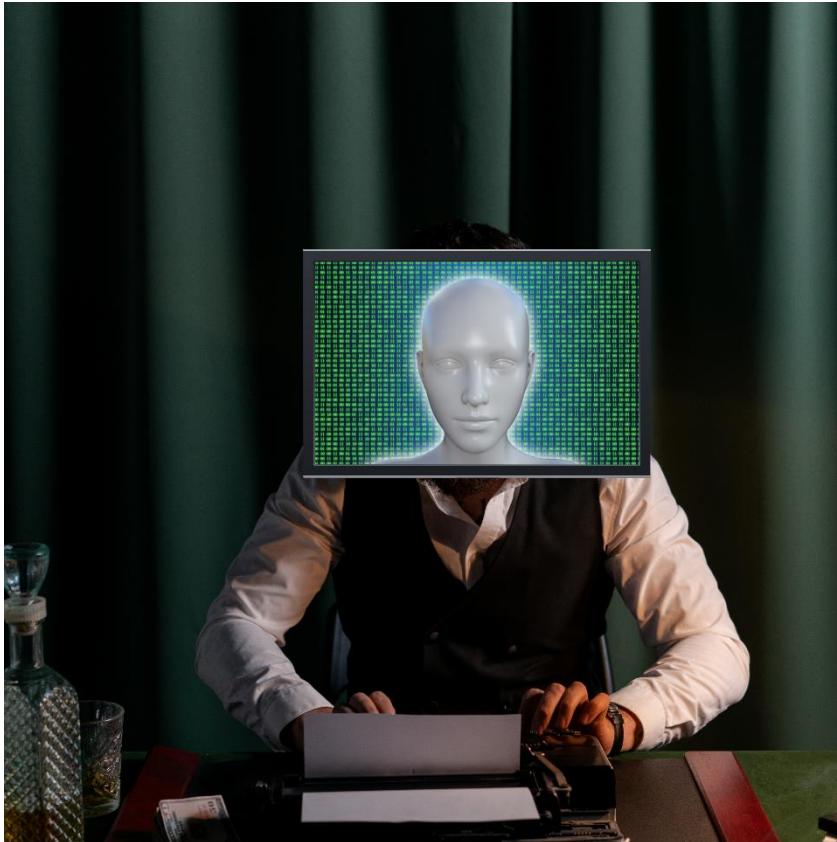
- a. Reset und Update Gate
- b. Vergleich LSTM und GRU

6. Anwendungsbeispiele



Hinweis zur Übung

Im Rahmen der Übung werden Sie sich der Text Prediction zuwenden.

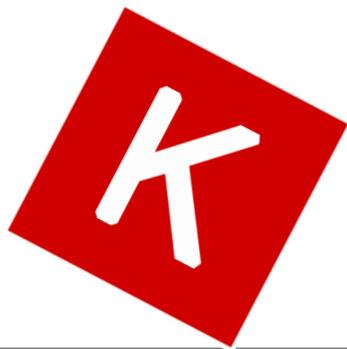


- Ziel: Vorhersage von einem Satz zu einer gegebenen Startsequenz. ("ROMEO:")
 - Erstellung eines Trainingdatensatzes aus einem Buch mithilfe der OneHot Kodierung
 - Erstellung und Training eines Modells für die Vorhersage des nächsten Zeichens zu einer gegebenen Startsequenz
 - Wettkampf (7.5.2)
Wer reicht den besten Satz ein?
- In den optionalen Aufgaben beschäftigen Sie sich mit den beiden RNN Varianten (LSTM und GRU) und können danach zum Schriftsteller werden. (Weiteres Modell, nach eigener Vorstellung aufbauen)
- Übung im Ilias oder über diesen [Link](#) verfügbar

Technische Hinweise zur Übung



**GPU
SUPPORT
AKTIVIEREN**





Vielen Dank für Ihre
Aufmerksamkeit!

Anhang - Gradientenberechnungen LSTM

$$\begin{aligned}\frac{\partial f_t}{\partial C_{t-1}} \cdot C_{t-1} &= \frac{\partial}{\partial C_{t-1}} [\sigma(W_f \cdot [h_{t-1}, x_t])] \cdot C_{t-1} \\ &= \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot \frac{\partial h_t}{\partial C_{t-1}} \cdot C_{t-1} \\ &= \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} * \tanh'(C_{t-1}) \cdot C_{t-1}\end{aligned}$$

$$\begin{aligned}\frac{\partial i_t}{\partial C_{t-1}} \cdot \tilde{C}_t &= \frac{\partial}{\partial C_{t-1}} [\sigma(W_i \cdot [h_{t-1}, x_t])] \cdot \tilde{C}_t \\ &= \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot \frac{\partial h_t}{\partial C_{t-1}} \cdot \tilde{C}_t \\ &= \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} * \tanh'(C_{t-1}) \cdot \tilde{C}_t\end{aligned}$$

$$\begin{aligned}\frac{\partial \tilde{C}_t}{\partial C_{t-1}} \cdot i_t &= \frac{\partial}{\partial C_{t-1}} [\sigma(W_C \cdot [h_{t-1}, x_t])] \cdot i_t \\ &= \sigma'(W_C \cdot [h_{t-1}, x_t]) \cdot W_C \cdot \frac{\partial h_t}{\partial C_{t-1}} \cdot i_t \\ &= \sigma'(W_C \cdot [h_{t-1}, x_t]) \cdot W_C \cdot o_{t-1} * \tanh'(C_{t-1}) \cdot i_t\end{aligned}$$