FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Rekurrente Neuronale Netze

Malte Deckers, Marvin Winkens, Tobias Faßbender

Übersicht

- Grundlagen
- Rekurrente Neuronen
- Strukturen von RNNs
- Trainieren von RNNs
- LSTMs / GRUs
- Beispiele für die Verwendung von RNNs

Übersicht

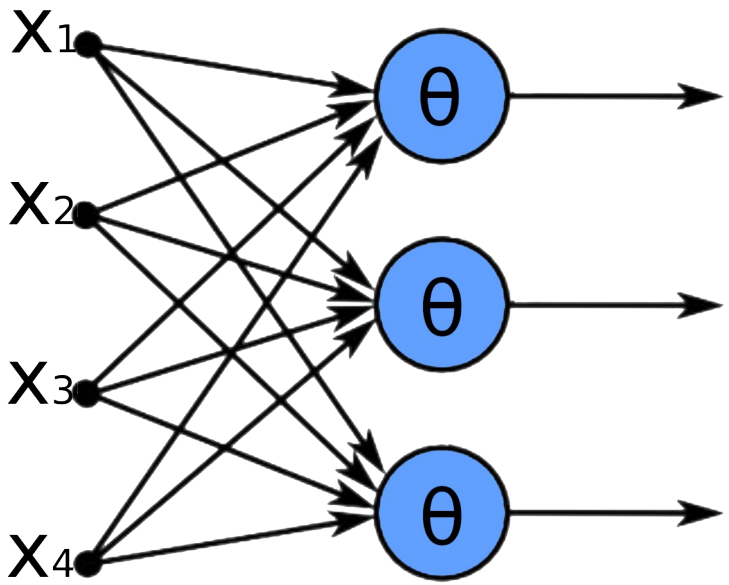
- Grundlagen
- Rekurrente Neuronen
- Strukturen von RNNs
- Trainieren von RNNs
- LSTMs / GRUs
- Beispiele für die Verwendung von RNNs

Abgrenzung von Feedforward-Netzen

- Feedforward-Netze:
  - Verschiedene Möglichkeiten Neuronen zu verbinden
    - Vollvermaschte Netze
    - Convolutional Neuronal Networks
  - Gemeinsamkeit: Alle vorwärts gerichtet

Abgrenzung von Feedforward-Netzen

- Feedforward-Netze:
  - Verschiedene Möglichkeiten Neuronen zu verbinden
    - Vollvermaschte Netze
    - Convolutional Neuronal Networks
  - Gemeinsamkeit: Alle vorwärts gerichtet

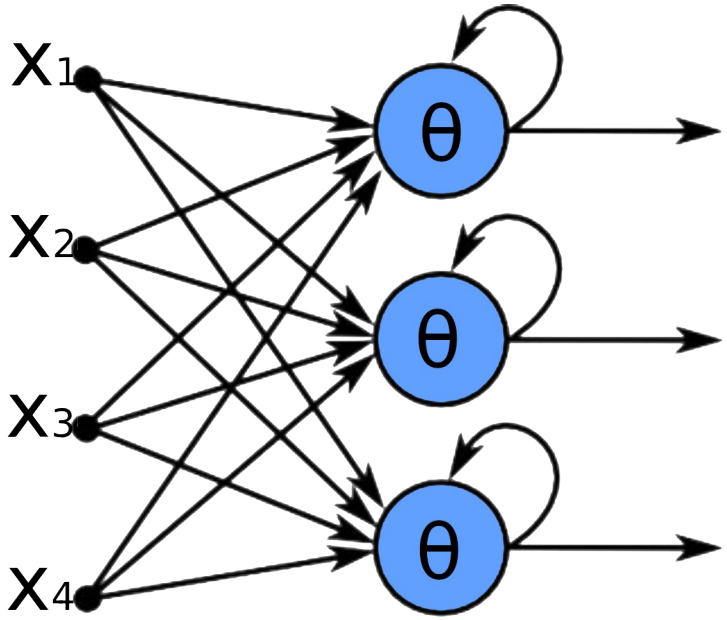


Abgrenzung von Feedforward-Netzen

- Rekurrent Neuronal Networks:
  - Enthalten rückwärts gerichtete Verbindungen
  - Ausgaben der Neuronen werden wieder als Eingaben (meist für die selben Neuronen) verwendet

Abgrenzung von Feedforward-Netzen

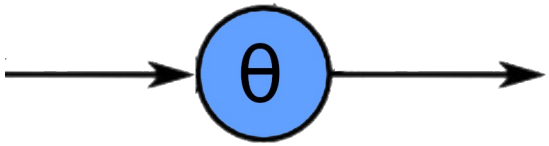
- Rekurrent Neuronal Networks:
  - Enthallten rückwärtsgerichtete Verbindungen
  - Ausgaben der Neuronen werden wieder als Eingaben (meist für die selben Neuronen) verwendet



Wiso rückwärtsgerichtete Verbindungen?

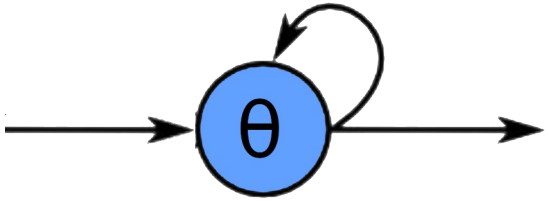
- Bevorzugte Verschaltungsweise neuronaler Netze in Gehirnen (von Menschen und Tieren)
- Erzeugen die Möglichkeit von Gedächnisswerten
  - Verarbeiten von Informationen durch verwenden vorher bekannter Informationen
  - Neue Informationen werden nicht von Grund auf neu interpretiert
  - Ermöglichen Verarbeitung von Zeitreihen, ...

Was können RNNs?



Feedforward Netze:

- Wiedererkennen von bekannten Strukturen (z.B. in Bildern)
- Verarbeiten Eingaben vorgegebener Länge (z.B. MNIST: 28x28 Pixel)
- Bewerten und Interpretieren gegebene Informationen



Rückgekoppelte Netze:

- Analyse und Vorhersage von Zeitreihen (z.B. in Aktienkursen)
- Verarbeiten Eingaben Variabler Länge (nützlich z.B. in Sprachverarbeitung, automatische Übersetzung)
- Sagen eine wahrscheinliche Zukunft vorher

Verwendungszwecke von RNNs

- Verarbeitung von Zeitreihen
  - Bewerten von Aktienkursen
  - Analyse von Audiodateien

- Sprachverarbeitung
  - Verarbeiten von Sprache (z.B. Google Assistant, Alexa, Siri, ...)
  - Automatische Übersetzung (z.B. Google Übersetzer)
  - Meinungsanalysen (z.B. Interpretieren von Filmbewertungen o.ä.)

- „Kreative“ Künstliche Intelligenz

Übersicht

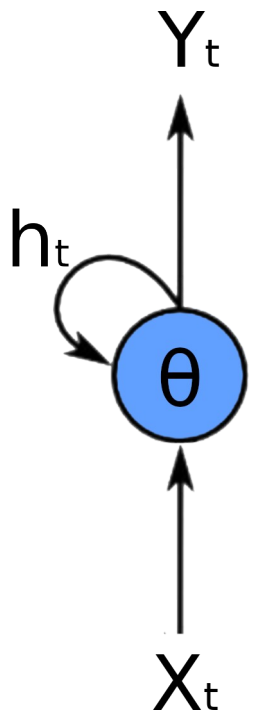
- Grundlagen
- Rekurrente Neuronen
- Strukturen von RNNs
- Trainieren von RNNs
- LSTMs / GRUs
- Beispiele für die Verwendung von RNNs

Rekurrente Neuronen

- Nicht nur vorwärts gerichtete Verbindungen
- Enthalten Schleifen

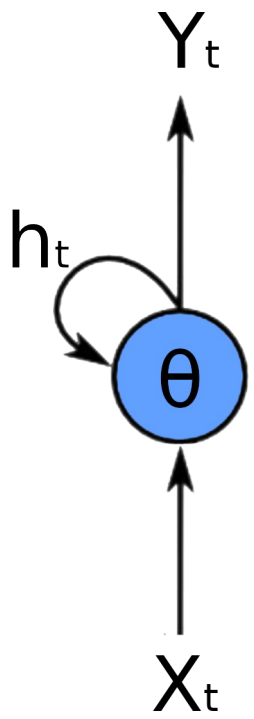
Rekurrente Neuronen

- Nicht nur vorwärts gerichtete Verbindungen
- Enthalten Schleifen



Rekurrente Neuronen

- Nicht nur vorwärts gerichtete Verbindungen
- Enthalten Schleifen



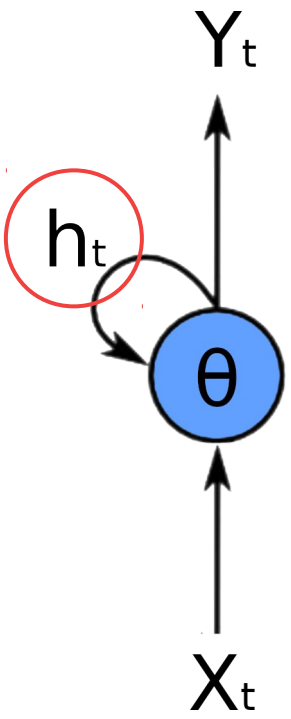
Berechnung:

$$y_t = \theta\left(x_t^T \cdot w_x + y_{t-1} \cdot w_y + b\right)$$

- y_t: Ausgabe zum Zeitpunkt t
- x_t: Eingabe zum Zeitpunkt t
- w_x: Gewichtsvektor der aktuellen Eingabe
- w_y: Gewichtsvektor der letzten Ausgabe
- b: Bias Term

Rekurrente Neuronen

- Nicht nur vorwärts gerichtete Verbindungen
- Enthalten Schleifen



Berechnung:

$$y_t = \theta\left(x_t^T \cdot w_x + y_{t-1} \cdot w_y + b\right)$$

- y_t: Ausgabe zum Zeitpunkt t
- x_t: Eingabe zum Zeitpunkt t
- w_x: Gewichtsvektor der aktuellen Eingabe
- w_y: Gewichtsvektor der letzten Ausgabe
- b: Bias Term

- h_t: Zustand der Zelle zum Zeitpunkt t

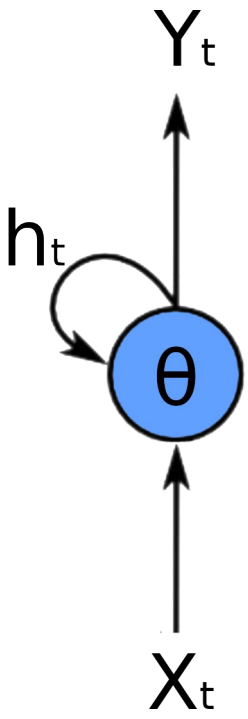
$h_t = f\left(h_{t-1}, x_t\right)$ ($h_t = y_{t-1}$ gilt allgemein **nicht**)

Das Netz entlang der Zeitachse aufrollen

Alternative Darstellungsform des Netzes:

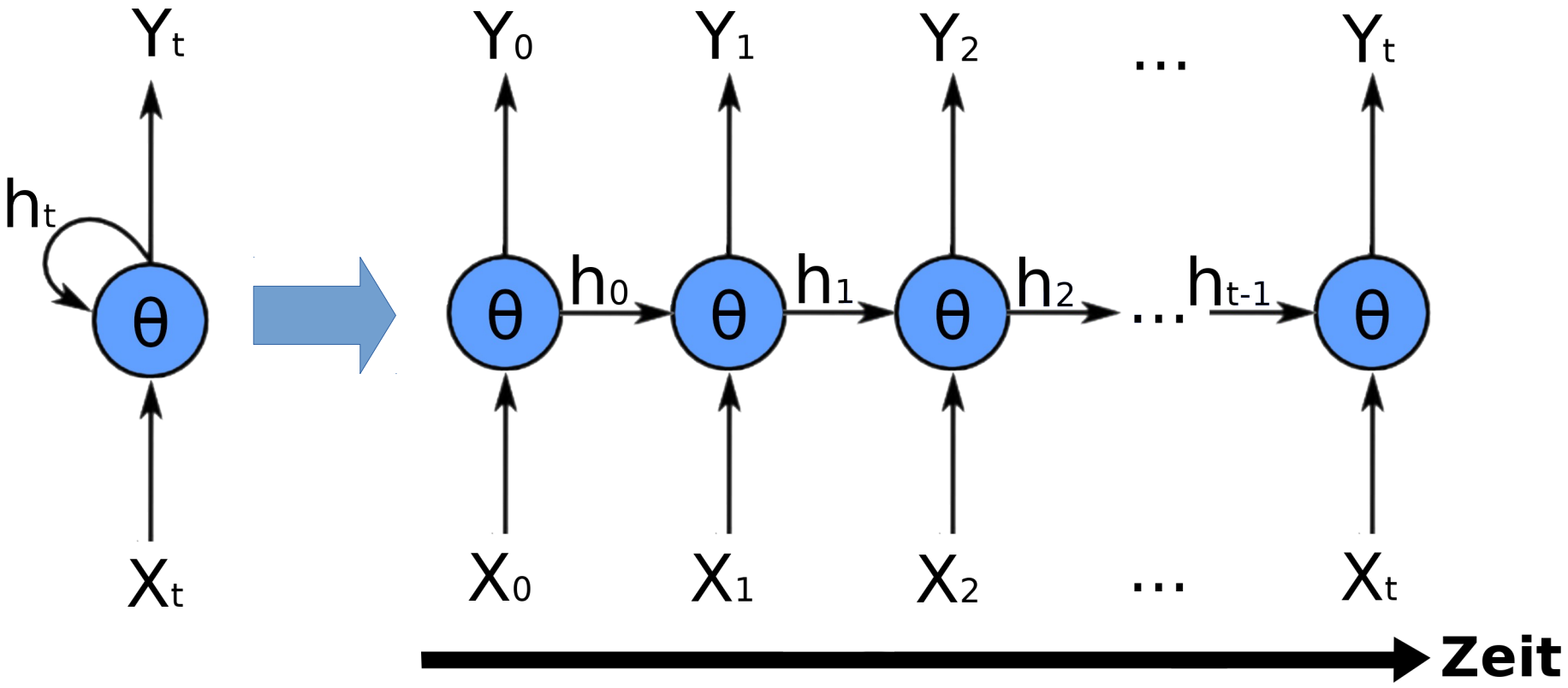
Das Netz entlang der Zeitachse aufrollen

Alternative Darstellungsform des Netzes:



Das Netz entlang der Zeitachse aufrollen

Alternative Darstellungsform des Netzes:

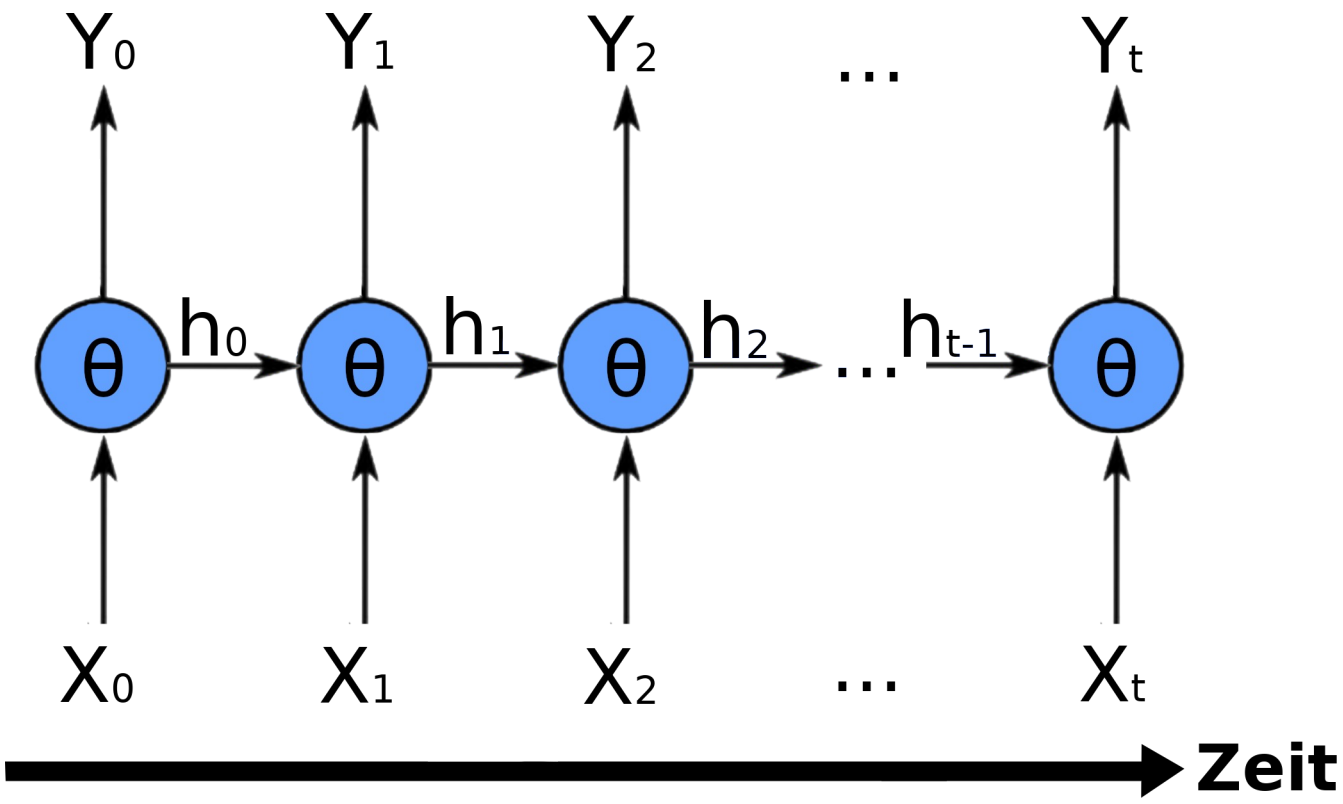


Das Netz entlang der Zeitachse aufrollen

Alternative Darstellungsform des Netzes:

Ein **einzelnes Neuron** und die **zeit-/iterations-abhängigen Werte** (X, Y und h) werden auf einer Zeitachse dargestellt

Wichtig für das **Trainieren des Netzes**



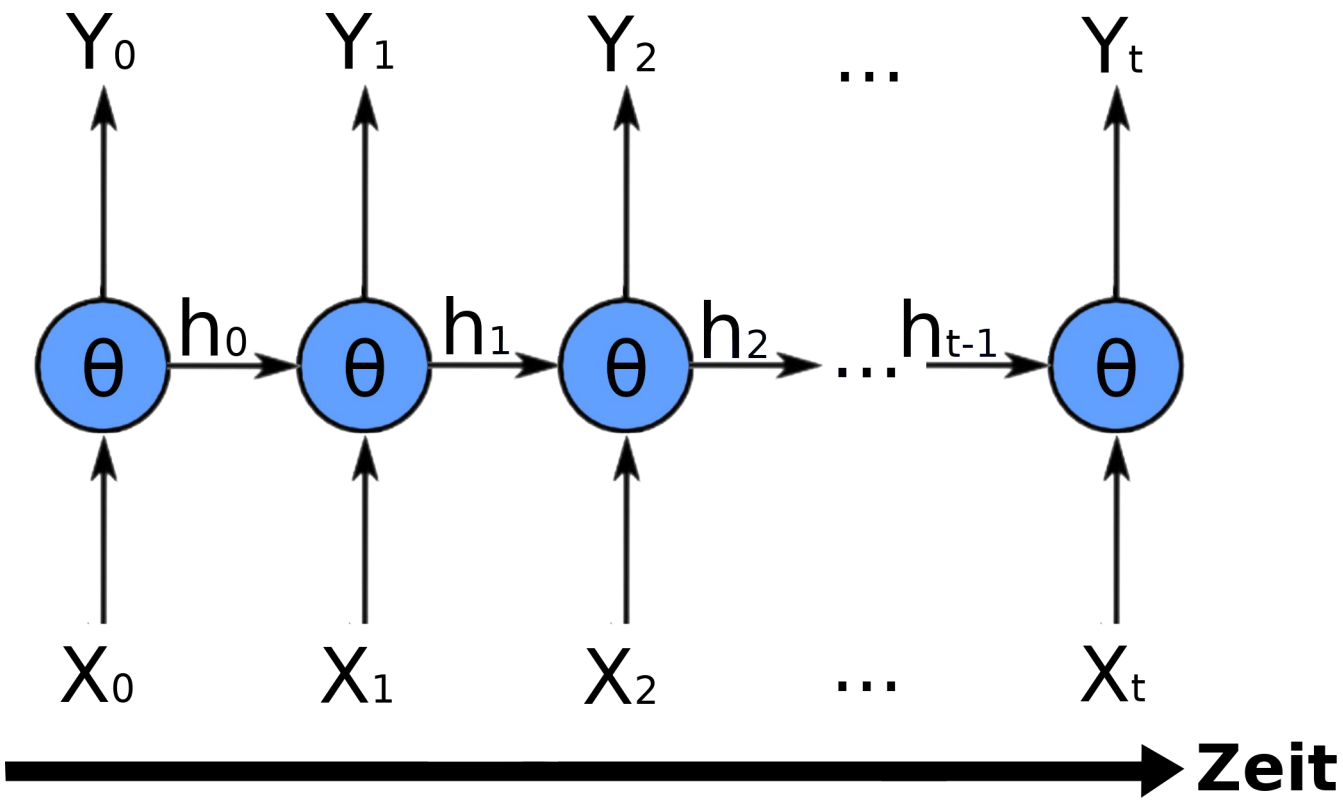
Das Netz entlang der Zeitachse aufrollen

Alternative Darstellungsform des Netzes:

Ein **einzelnes Neuron** und die **zeit-/iterations-abhängigen Werte** (X, Y und h) werden auf einer Zeitachse dargestellt

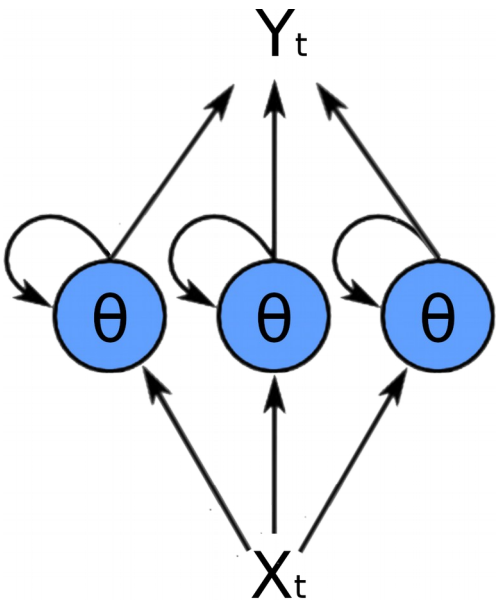
Wichtig für das **Trainieren des Netzes**

Ein- und Ausgaben können dabei Skalare oder Vektoren sein

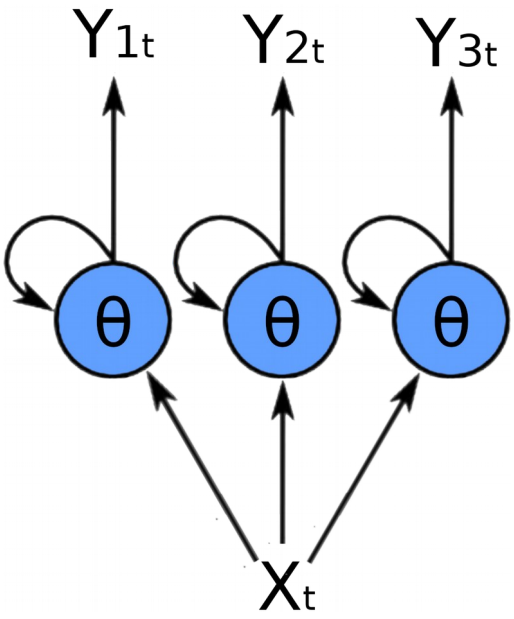


Das Netz entlang der Zeitachse aufrollen

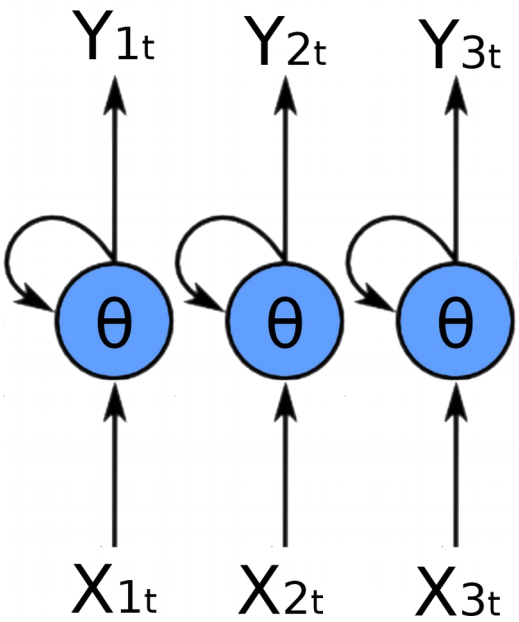
Skalar - Skalar



Skalar - Vektor



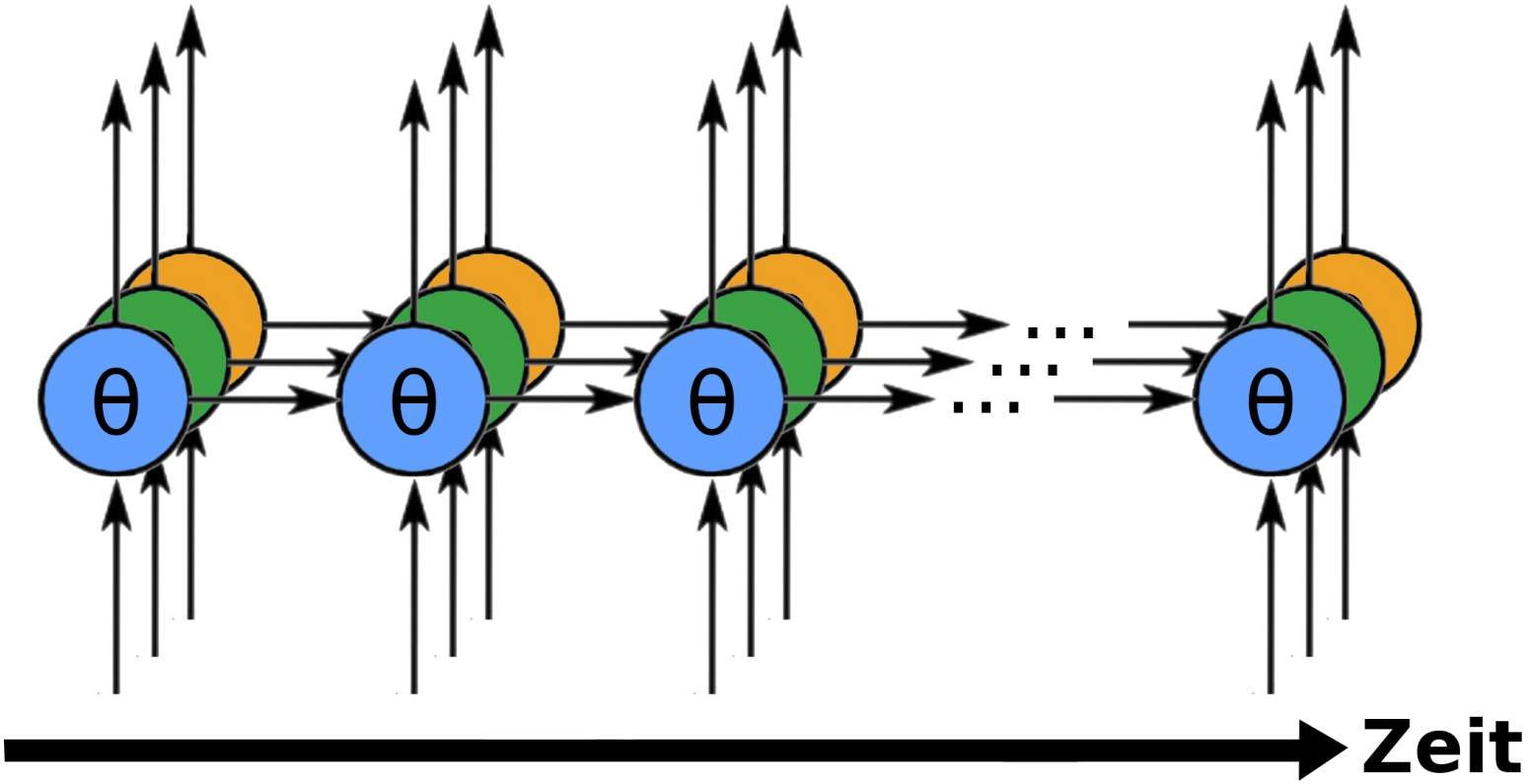
Vektor - Vektor



Ein- und Ausgaben können dabei Skalare oder Vektoren sein

Das Netz entlang der Zeitachse aufrollen

Aufgerollte Netze mit mehreren (parallelen) rekurrenten Neuronen können als 2-Dimensionales Gitter dargestellt werden

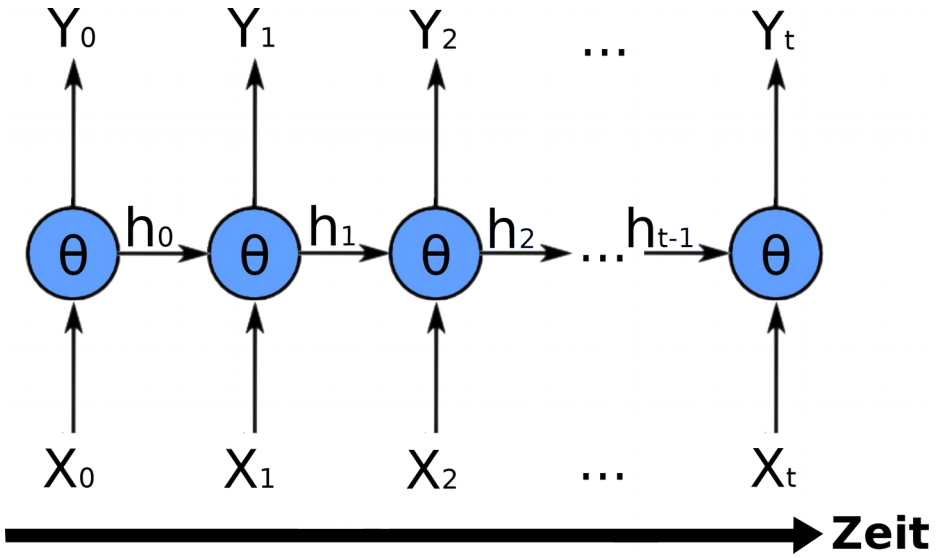


Gedächtnis der Neuronen

Ausgabe eines Neurons zur Zeit t:

$$Y_t = f(X_t, h_{t-1})$$

$$h_t = g(X_t, h_{t-1})$$



Gedächtnis der Neuronen

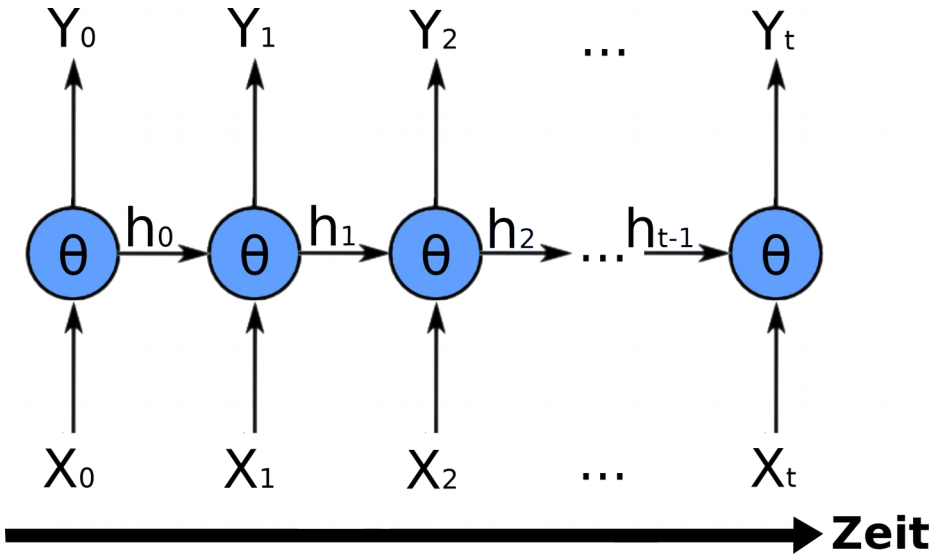
Ausgabe eines Neurons zur Zeit t:

$$Y_t = f(X_t, h_{t-1})$$

$$h_t = g(X_t, h_{t-1})$$

Rekursionsgleichung:

$$Y_t = f(X_t, g(X_{t-1}, g(X_{t-2}, \ldots)))$$



Gedächtnis der Neuronen

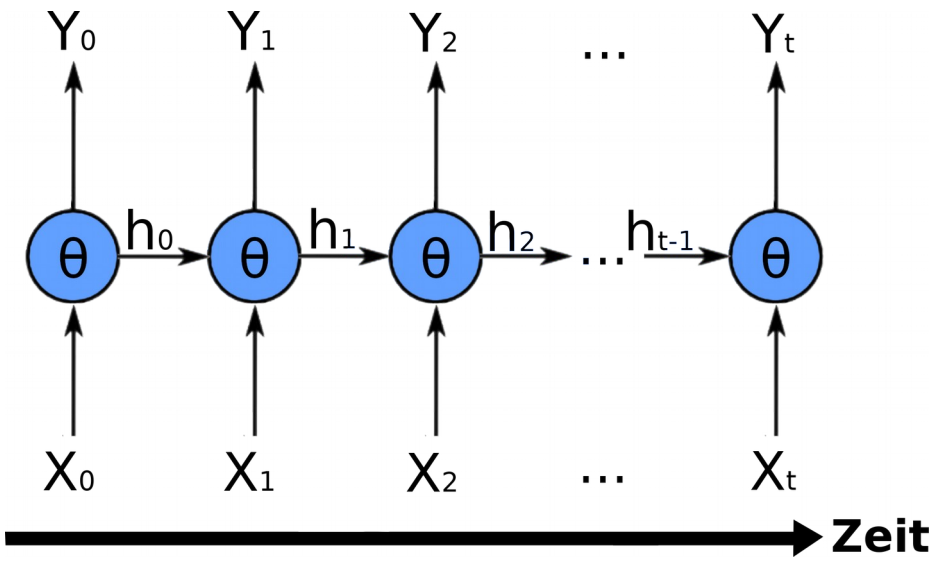
Ausgabe eines Neurons zur Zeit t:

$$Y_t = f(X_t, h_{t-1})$$

$$h_t = g(X_t, h_{t-1})$$

Rekursionsgleichung:

$$Y_t = f(X_t, g(X_{t-1}, g(X_{t-2}, \ldots)))$$



=> Die Ausgabe eines Neurons ist eine Funktion in Abhängigkeit **aller bisherigen Eingaben** des Neurons

=> Das Neuron hat eine Art **Gedächtnis**

Übersicht

- Grundlagen
- Rekurrente Neuronen
- Strukturen von RNNs
- Trainieren von RNNs
- LSTMs / GRUs
- Beispiele für die Verwendung von RNNs

Ein- und Ausgabesequenzen

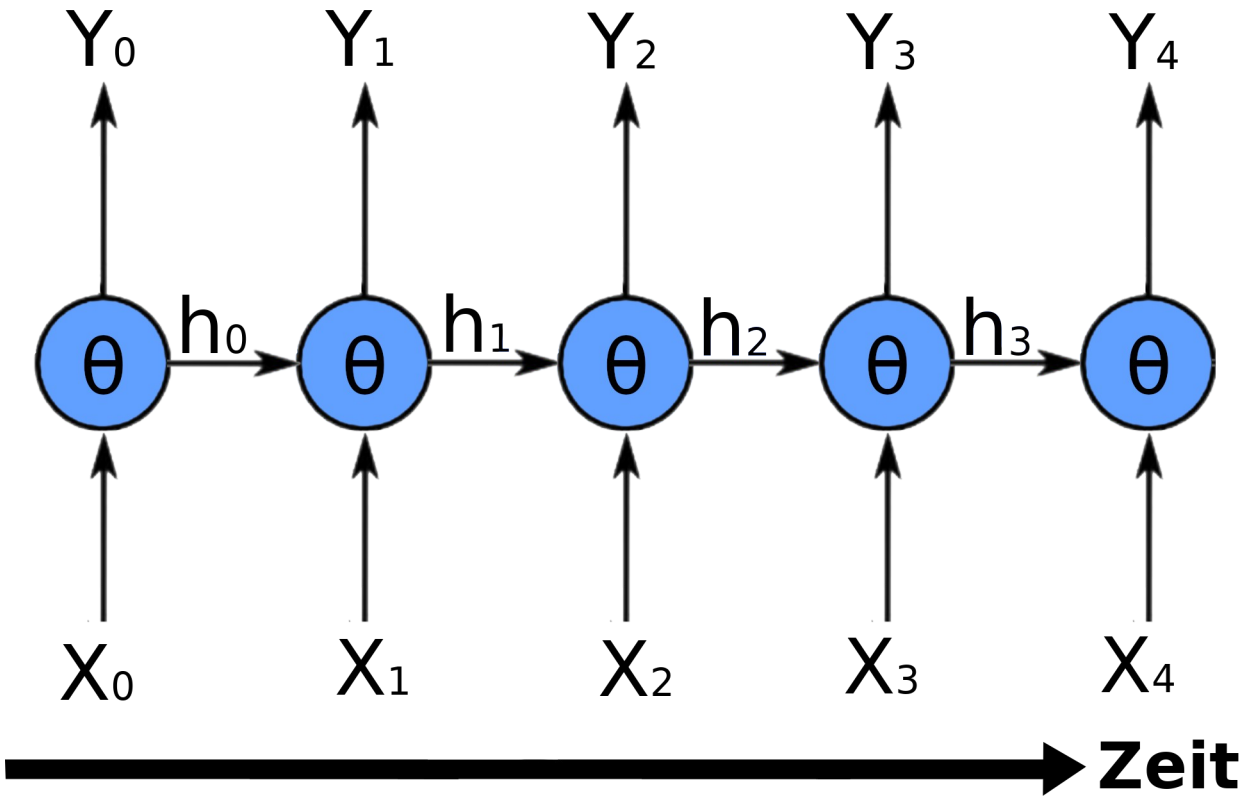
- Feedforward-Netze verarbeiten nur Eingaben vorgegebener Länge
- Rekurrente Neuronale Netze haben keine Vorgegebenen Längen (weder in der Eingabe, noch in der Ausgabe)

=> RNNs können für verschiedene Zwecke verwendet werden, bei denen sie verschiedene Längen von Ein- und Ausgaben erhalten / generieren

=> Sequenz zu Sequenz, Sequenz zu Vektor, Vektor zu Sequenz, verzögerte Sequenz zu Sequenz

Ein- und Ausgabesequenzen

Sequenz zu Sequenz

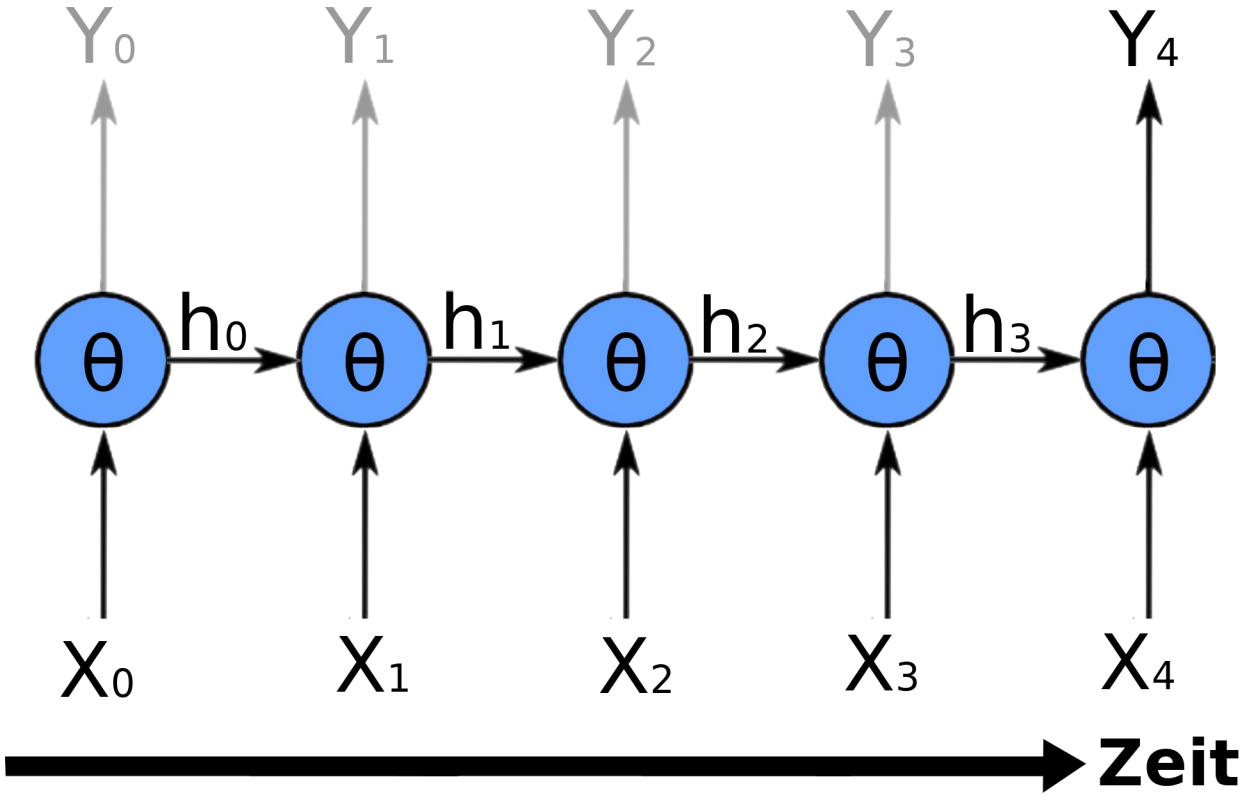


Verwendungsbeispiel:

Vorhersage von Zeitreihen (wie z.B. Aktienkursen)

Ein- und Ausgabesequenzen

Sequenz zu Vektor

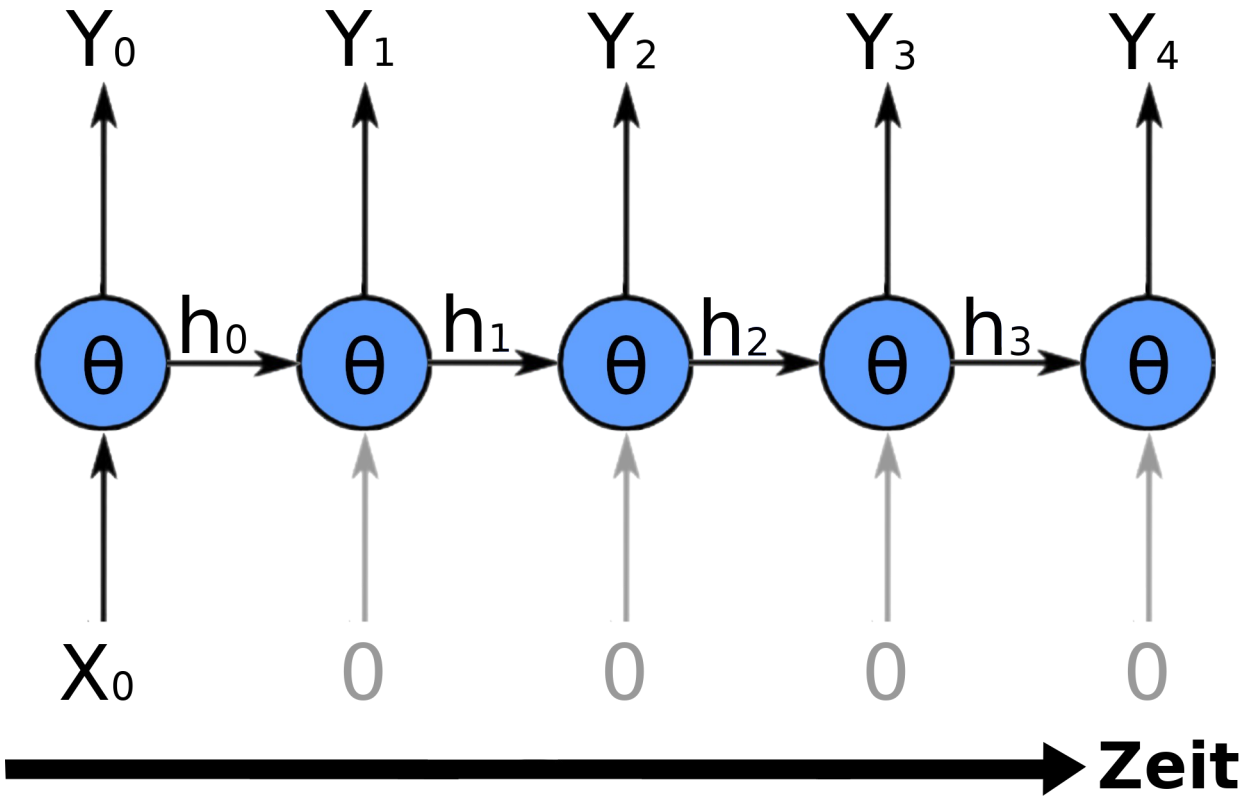


Verwendungsbeispiel:

Analyse von Texten (z.B. Filmbewertungen: Eingabe ist der Text der Filmbewertung)

Ein- und Ausgabesequenzen

Vektor zu Sequenz

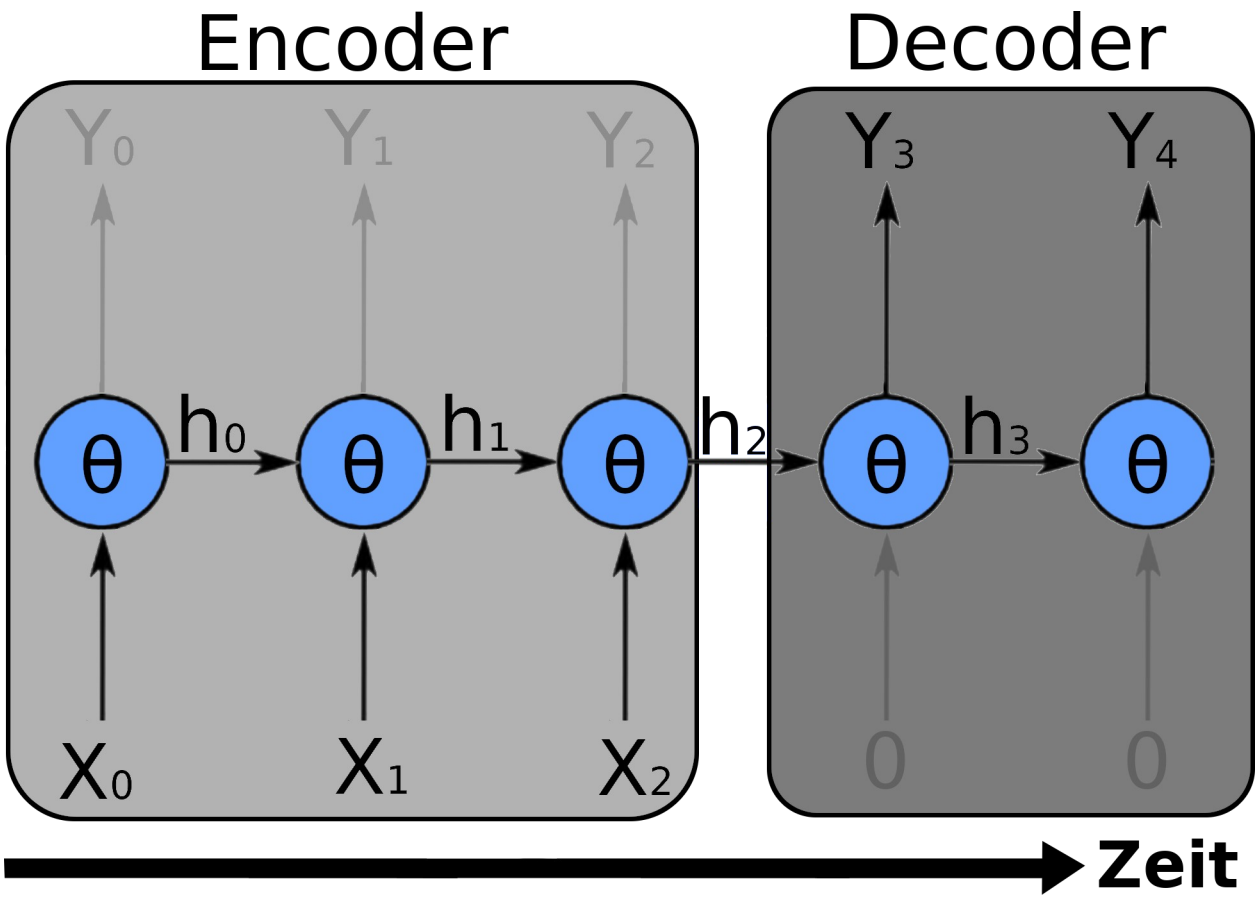


Verwendungsbeispiel:

Bildbeschreibung
(Eingabe ist ein Bild,
Ausgabe ist die
Beschreibung des Bilds)

Ein- und Ausgabesequenzen

Encoder - Decoder

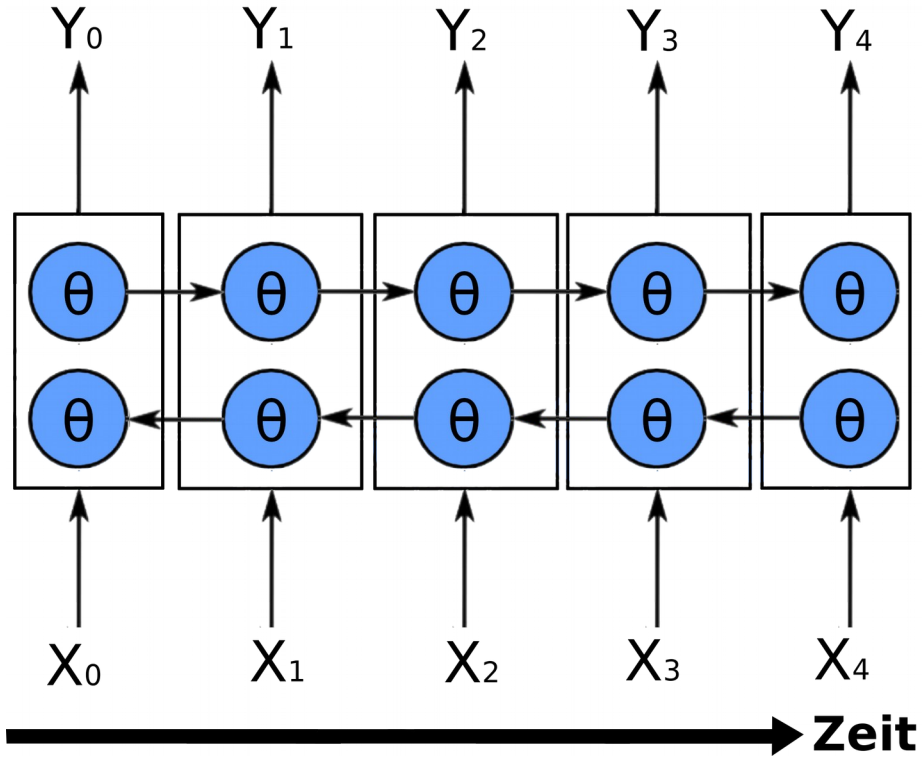


Verwendungsbeispiel:

Automatische Übersetzung von Text (die Eingabelänge muss nicht gleich er Ausgabelänge sein)

Bidirektionale RNNs

- Informationen aus Vergangenheit und Zukunft
- Jeweils ein Hidden-Layer für Vorwärts- und Rückwärtsrichtung



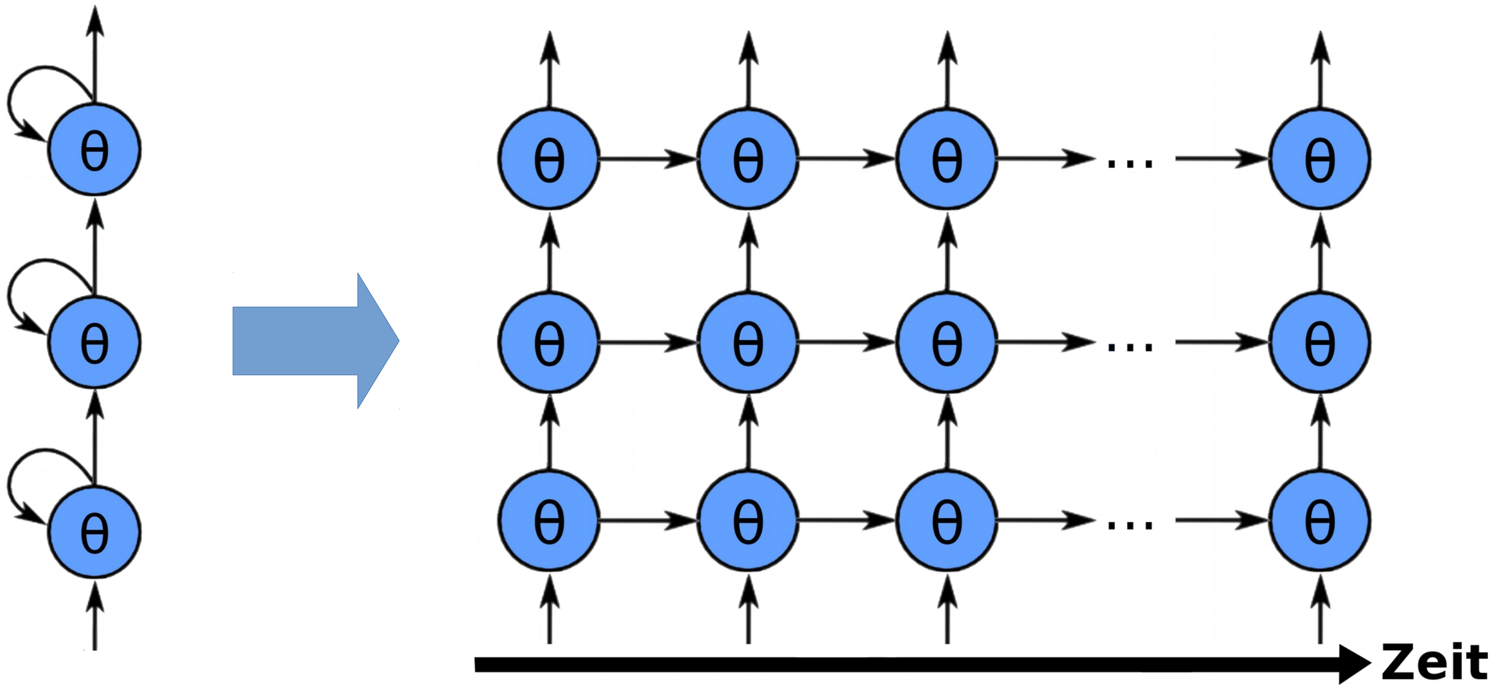
- Nützlich wenn bereits alle Informationen bekannt sind
- Erhöht den Anteil von Input-Informationen im Netz

Verwendungsbeispiel:

Erkennen von handgeschriebenen Ziffern (MNIST), da alle Daten bekannt sind und verwendet werden können

Deep RNNs

- RNNs lassen sich auch mit mehreren Hidden-Recurrent-Layern bilden
- Meistens werden dafür (vergleichsweise) wenige Hidden-Layers verwendet (2-4 Layer)



Übersicht

- Grundlagen
- Rekurrente Neuronen
- Strukturen von RNNs
- Trainieren von RNNs
- LSTMs / GRUs
- Beispiele für die Verwendung von RNNs

Training von RNNs

RNNs werden mit Backpropagation Through Time (BPTT) trainiert.

Die gleichen Parameter werden über die Zeit beibehalten (Parameter Sharing)

Es folgen:

- Grundlagen BPTT
- Unterschiede zur traditionellen BP
- Vanishing Gradient Problem
- Dropout
- Layer-Normalization

Wdh | BP

- Ermöglicht effiziente Berechnung lokaler Gradienten
- Lokale Gradienten werden vom Ende des Netzes her durch Rekursionvorschrift bestimmt

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

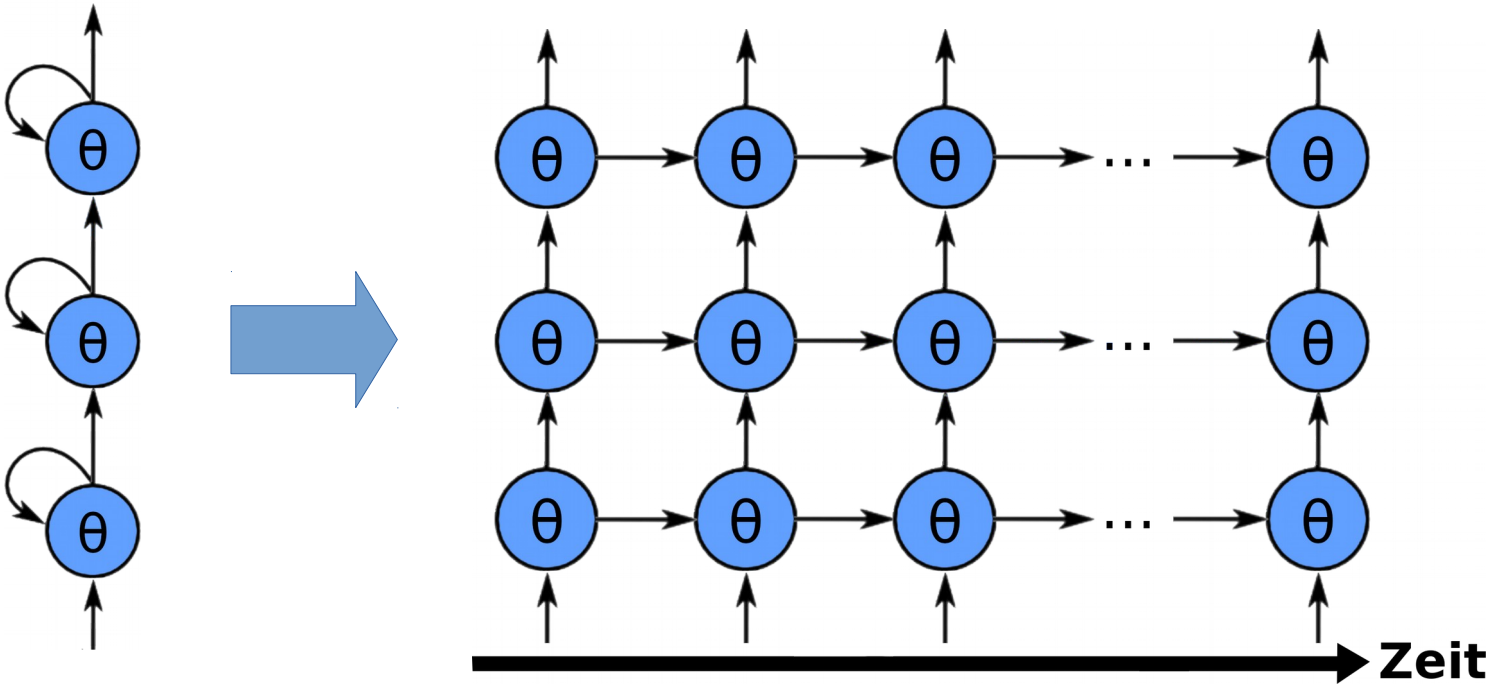
- Stochastic Gradient Descent
- Loss durch Kreuzentropie

→ RNNs Äquivalent

Lösung für RNNs: **Backpropagation Through Time**

BPTT | Backpropagation Through Time

- Beginnt damit, das Netz aufzufalten
- Gewichte Rekursiv über zeitliche Layer berechnen
- Gleicher Algorithmus, wie bei Feedforward NNs



BPTT

- standard RNNs sind hard zu trainieren
- Sequenz kann sehr lang sein
- Back-propagation durch viele Layer

→ Oft wird der BP-Algorithmus abgekürzt

→ Truncated Backpropagation Through Time

BPTT

- standard RNNs sind hard zu trainieren
- Sequenz kann sehr lang sein
- Back-propagation durch viele Layer

→ Oft wird der BP-Algorithmus abgekürzt

→ Truncated Backpropagation Through Time

Fragen:

1.) Welchen Vorteil hat ein Abkürzen der Backpropagation?

TBPTT | Truncated BPTT

- Anstelle Berechnen aller vorherigen Gewichte

→ Nur letzten K Gewichte

→ Aufrollen nur K mal

→ Weniger Kosten

→ Wirkt gegen Vanishing Gradients für kleines K

Fragen:

1.) Wie wählt man K?
2.) Wann macht man das Gewichtsupdate beim Betrachten einer Sequenz?

TBPTT

- **TBPTT(n, n)**
  - Klassisch: Update am Ende der Sequenz über alle Zeitschritte
- **TBPTT(1, n)**
  - Nach einem Sequenzschritt Update über alle Zeitschritte
- **TBPTT(k1, 1)**
  - Nach k1 Sequenzschritten Update für einen Zeitschritt
- **TBPTT(k1, k2)** k1 < k2 < n
  - Viele Updates pro Sequenz

    → Beschleunigte Lernrate
- **TBPTT(k1, k2)** k1=k2=h
  - Gebräuchlich: TF und Keras implementierung

Vanishing Gradient Problem

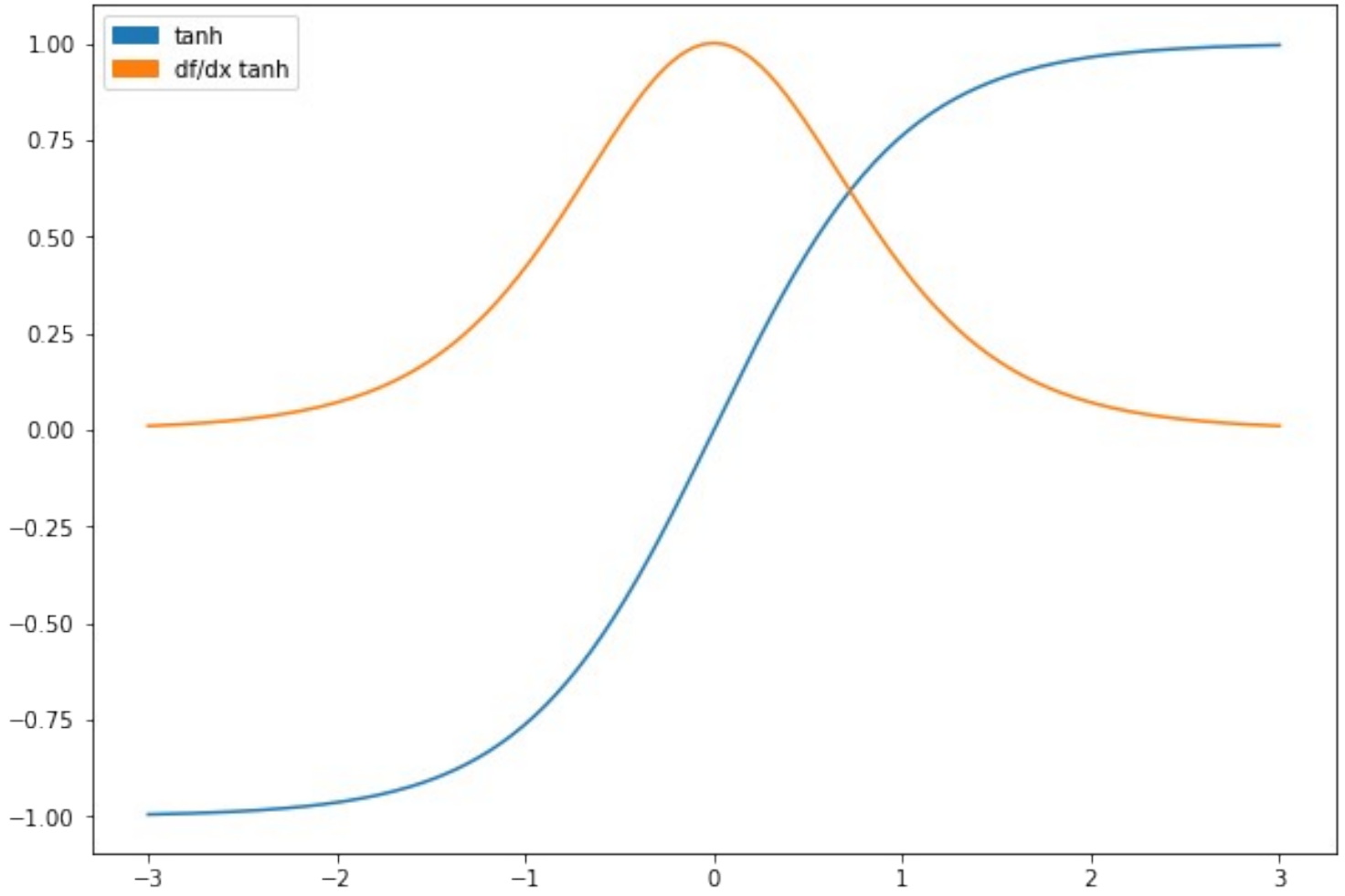
- Kettenregel geht über Zeit
- $\frac{df}{dx}$ tanh (oder sigmoid)-Funktion geht gegen 0 an beiden Enden
- Kleine Werte werden multipliziert

→ Gradient geht gegen 0

→ Lernen über längere Zeit nicht möglich

→ Vergleichbar mit unglaublich tiefen Netzen

BPTT



Lösung Vanishing Gradient Problem

- Initialisierung von W reduziert Effekt
- Regularisierung
  - Dropout
  - Layer-Normalization
  - Etc.
- ReLU anstelle von tanh oder sigmoid
- LSTMs oder GRUs

Exploding Gradient Problem

Gradient steigt exponentiell

→ „explodiert“

Lösungen

→ Clipping:

- Gradientenlänge wird verkürzt btw. „geclippt“ (Norm based clipping)
- Größte Teilkomponente wird „geclippt“ (Value based clipping)

→ Andere Regularisierer

- Dropout
- Layer-Normalization

RNN Dropout

Mehrere Arten von Dropout in RNNs

- Input Dropout
- Recurrent Dropout
- weitere (Variational, Fraternal, Curriculum, etc.)

RNN Dropout

Mehrere Arten von Dropout in RNNs

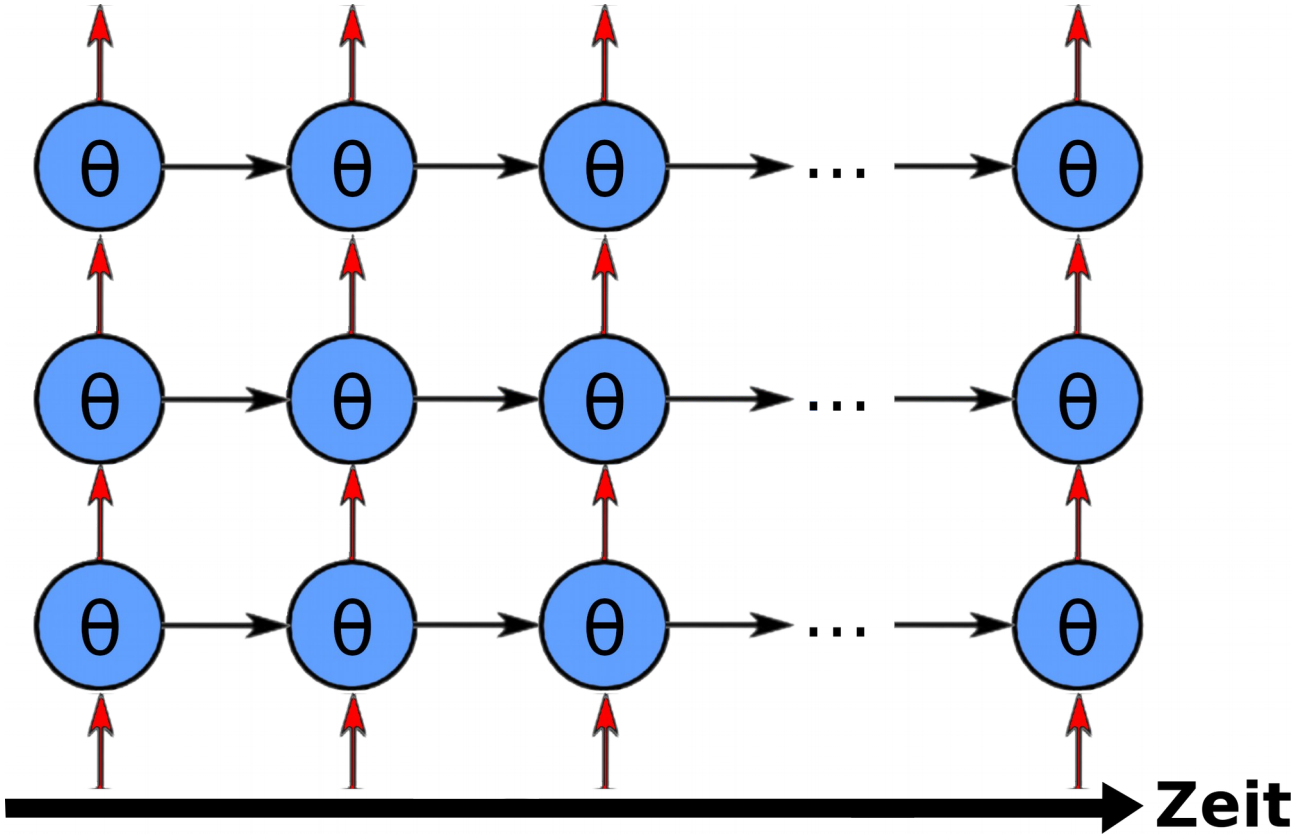
- Input Dropout
- Recurrent Dropout
- weitere (Variational, Fraternal, Curriculum, etc.)

Fragen:

1.) Warum mehrere Arten?
2.) Wie könnten diese funktionieren?

„normaler“ Dropout in RNNs

Die Roten Pfeile markieren die betroffenen Verbindungen



Input Dropout

- Daten an einer RNN-Input-Node werden ausgeschlossen
- Kein Gewichtsupdate für ausgeschlossene Nodes

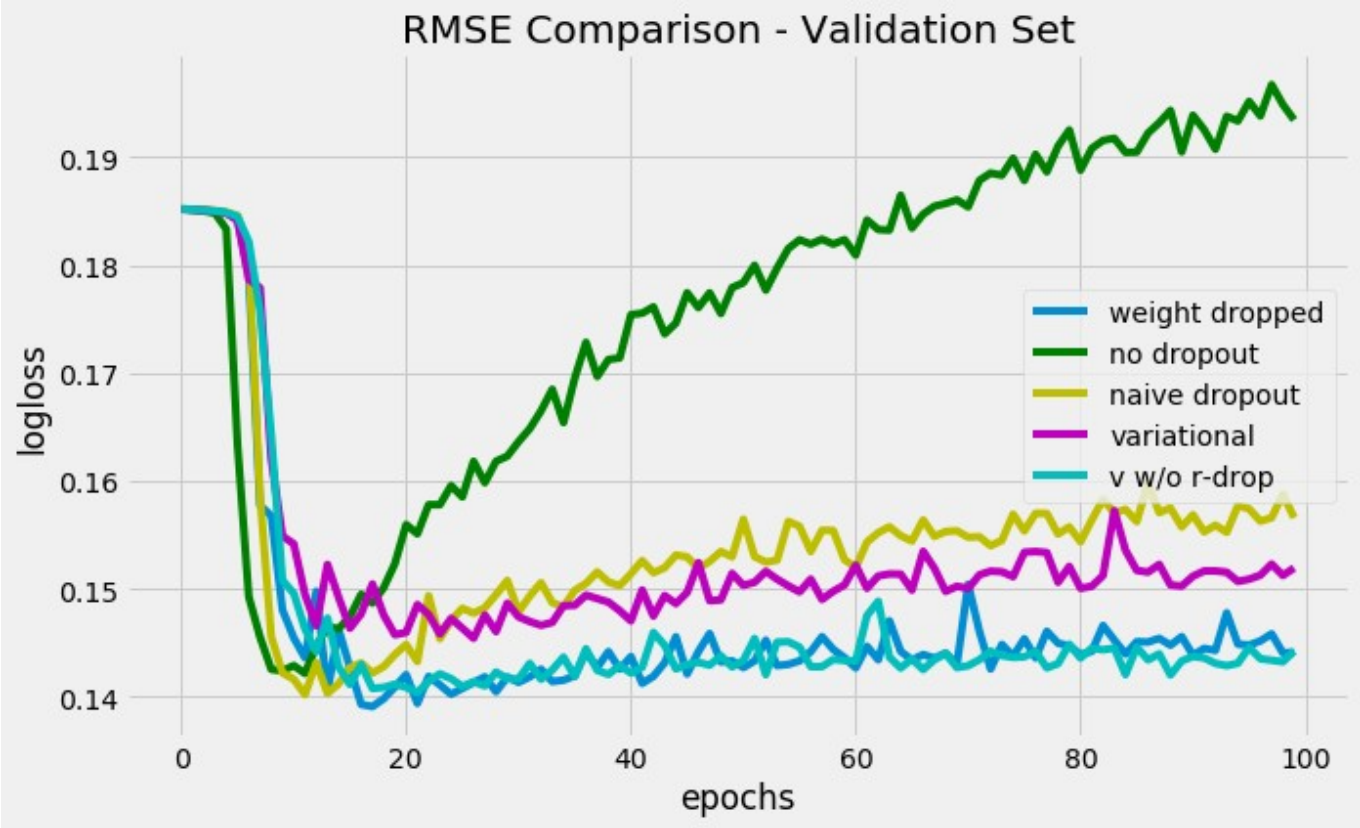
→ Entspricht „normalen“ Dropout

Recurrent Dropout

- Dropout am „Gedächtnis-Speicher“ von RNN Nodes
- Nur für LSTMs:
  - Dropout für Subnetzwerk
  - Nur Update des „Gedächtnis-Speichers“ auf den Input der zeitlich nächsten Node betroffen

Dropout Loss Beispiel

Beispiel mit Pytorch auf Cornell Film Review Dataset



Von *https://towardsdatascience.com/learning-note-dropout-in-recurrent-networks-part-3-1b161d030cd4*

Batch Normalization

Klassisch:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad \mu = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

- Unklar wie man Batch-Normalization für RNNs einsetzt
- Man benötigt sehr große mini-Batches um Statistiken genau vorherzusagen

→ Layer Normalization

Layer Normalization

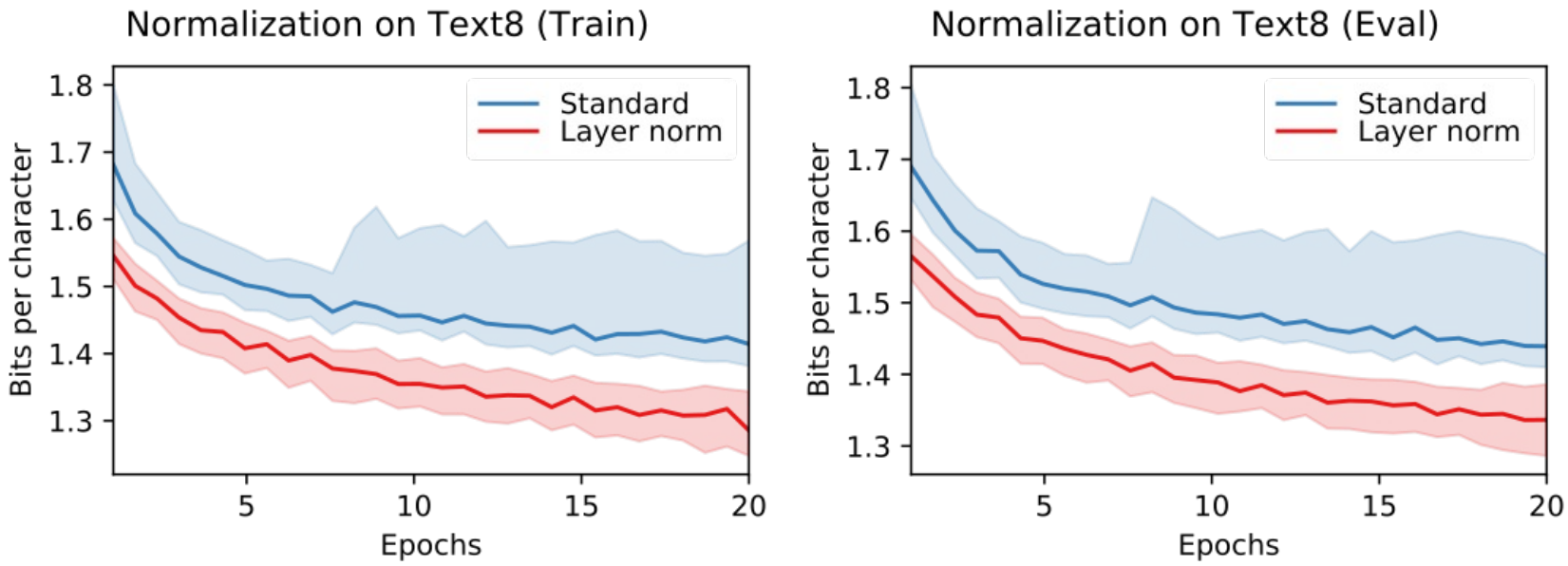
- Features werden normalisiert

$$\mu = \frac{1}{D}\sum_{d=1}^{D} x_i^d$$

$$\sigma^2 = \frac{1}{D}\sum_{d=1}^{D}(x_i^d - \mu)^2$$

$$\hat{x}_i^d = \frac{x_i^d - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Layer Normalization



Von *https://danijar.com/language-modeling-with-layer-norm-and-gru/*

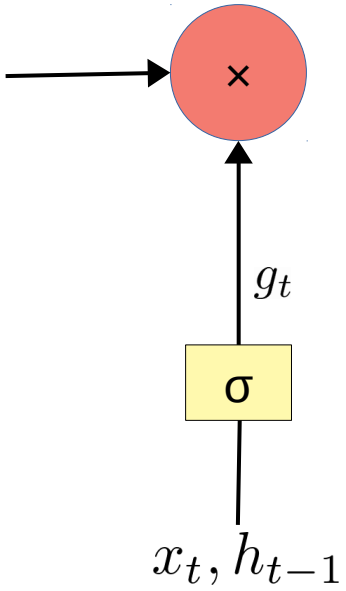
Übersicht

- Grundlagen
- Rekurrente Neuronen
- Strukturen von RNNs
- Trainieren von RNNs
- LSTMs / GRUs
- Beispiele für die Verwendung von RNNs

Long Short Term Memory

- Probleme von Vanilla-RNNs:
    - Abhängigkeiten und Informationen gehen über lange Sequenzen hinweg verloren
    - *Vanishing Gradient*
- Einführung eines Zellzustandes / Kontext C_t
    - „Langzeitgedächtnis“
    - zusätzlich zum *hidden state* h_t

Long Short Term Memory
Gates



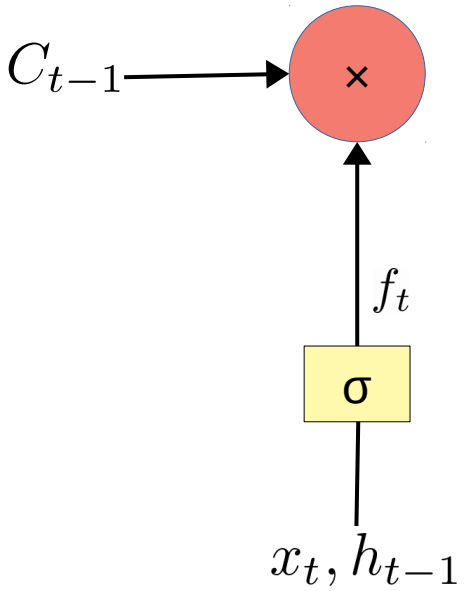
- Gates bestimmen welche Informationen „durchgelassen“ werden
- Bestehen aus Sigmoid-Schicht und einer elementweisen Multiplikation ○ oder ×
- Eingabe: Netzeingabe und *hidden state*
- Sigmoid-Layer erzeugt Werte zwischen 0 und 1
  - 1 bedeutet Information wird durchgelassen
  - 0 bedeutet Information wird nicht durchgelassen

$$g_t = \sigma(W_x x_t + b_x + W_h h_{t-1} + b_h)$$

Kürzere Notation: $g_t = \sigma\left(W_g \left[h_{t-1}, x_t\right] + b_g\right)$

Long Short Term Memory

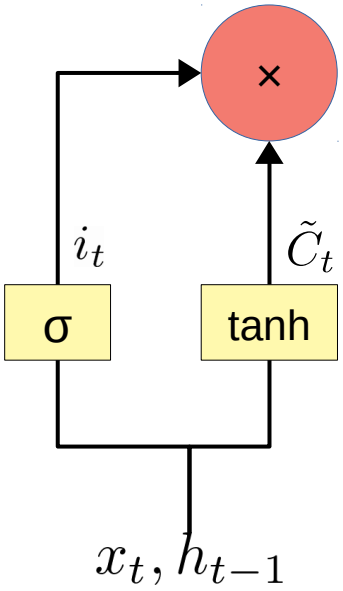
Forget Gate



Bestimmt welche Informationen aus dem Kontext „vergessen“ werden

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

Long Short Term Memory
Input Gate



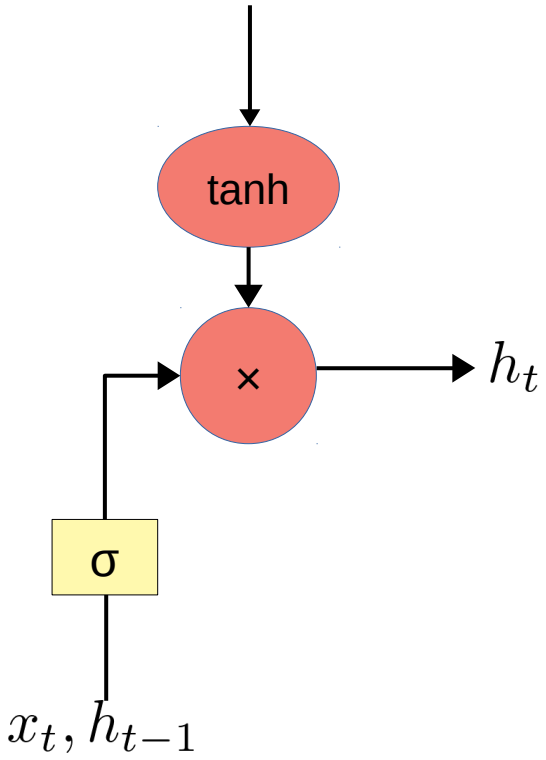
Bestimmt welche Informationen aus dem alten Kontext und dem aus der Eingabe ermittelten Kandidaten $\tilde{C}_t$ in den neuen Zustand übernommen werden

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

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

Long Short Term Memory

Output Gate

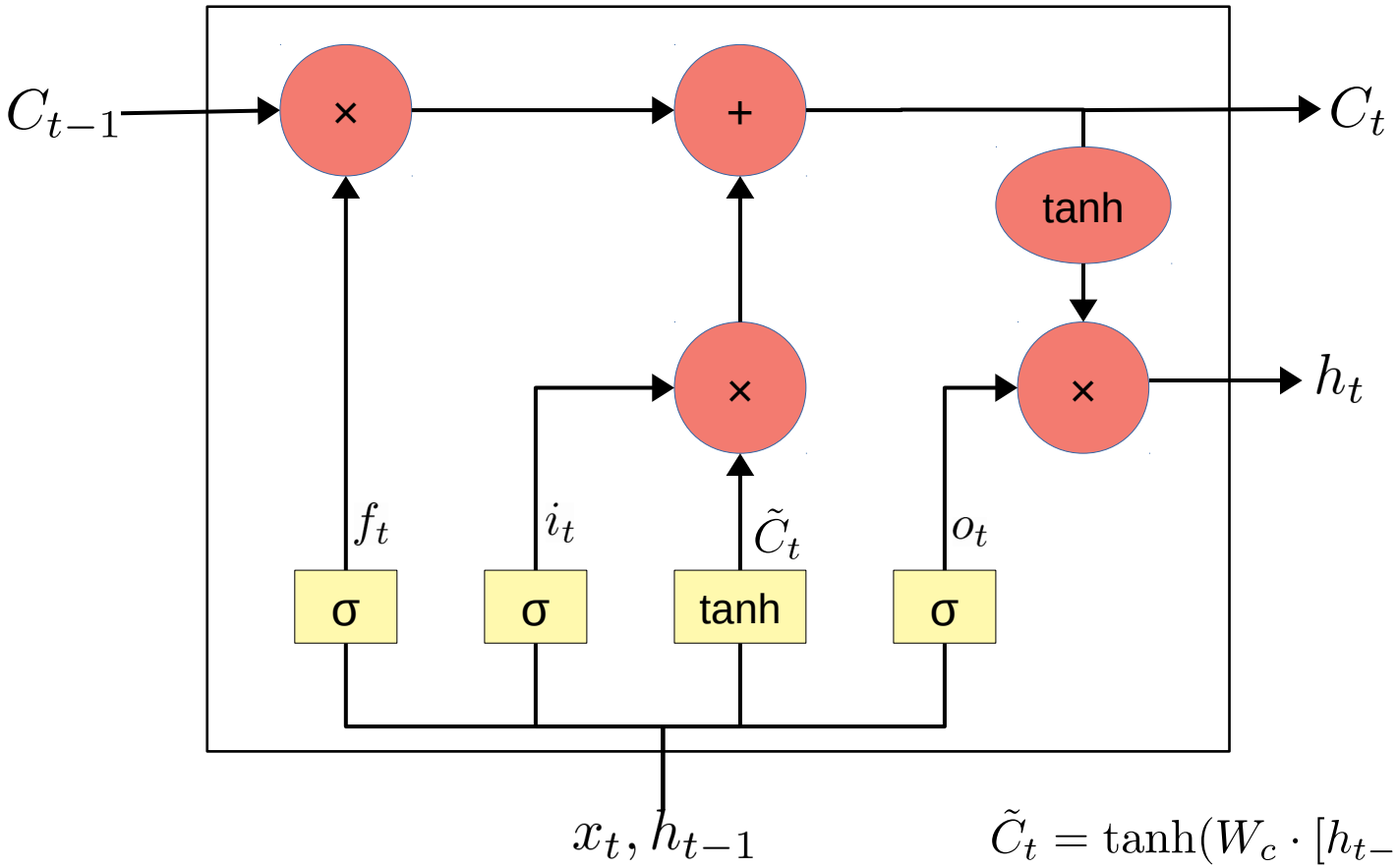


Bestimmt welche Informationen aus dem Zellzustand in die Ausgabe übernommen werden

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

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

Long Short Term Memory

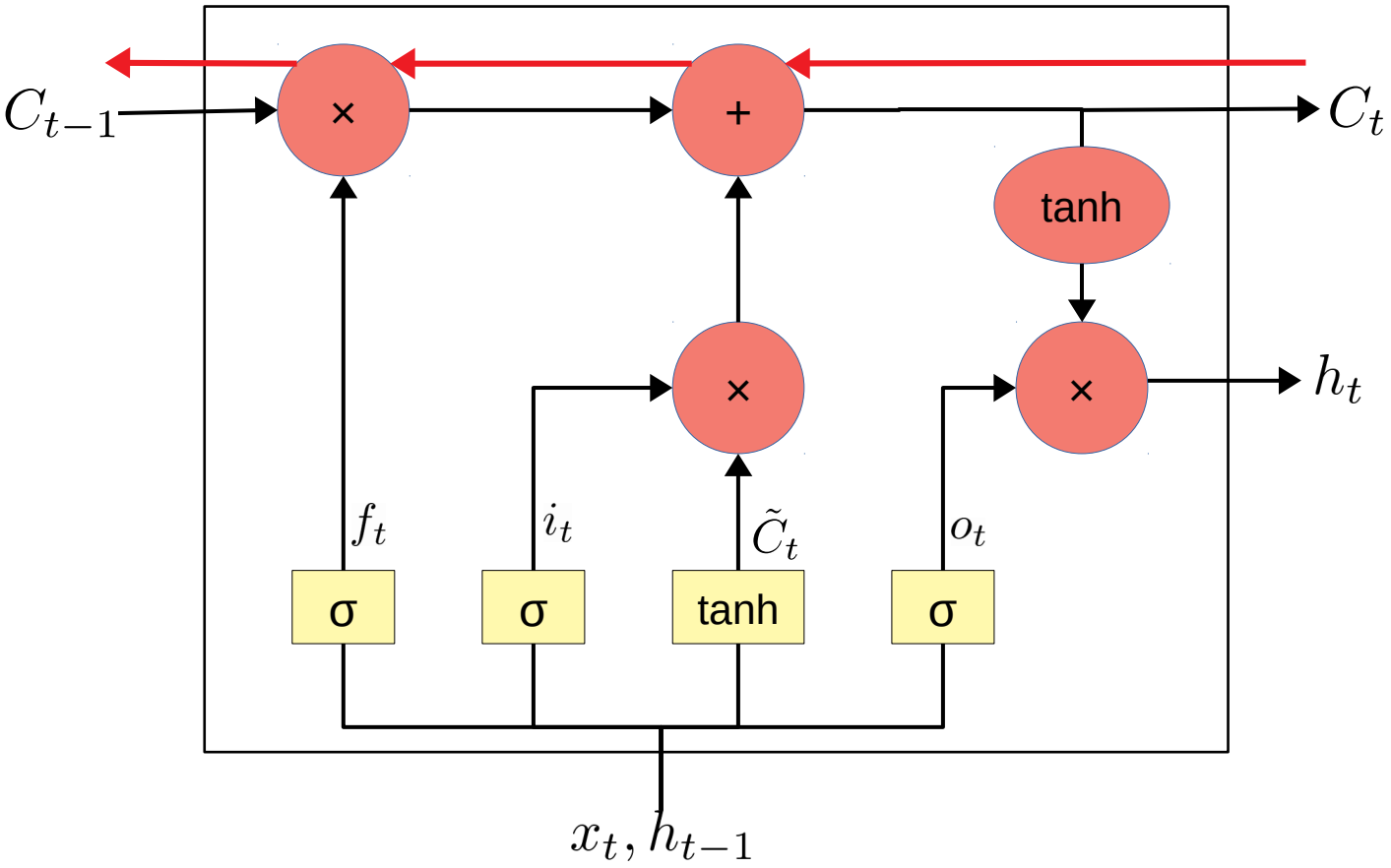


$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

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

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

Long Short Time Memory
Gradientenfluss



Long Short Time Memory

Gradientenfluss

- LSTMs schwächen das Problem verschwindender Gradienten ab
- Problem im Vanilla-RNN: häufige Multiplikation mit der selben Gewichtsmatrix
    - Verstärkung verschwindender / explodierender Gradienten
- Vorteil LSTM: „Ungehinderter“ Gradientenfluss für den Zellzustand
    - Lediglich elementweise Operationen, keine Matrixmultiplikation
    - *Forget gate* kann über die Schritte hinweg unterschiedlich aussehen, es handelt sich nicht immer um die selbe Matrix

LSTM Varianten

- Peephole Connections: Gates „schauen“ auf den Zellenkontext

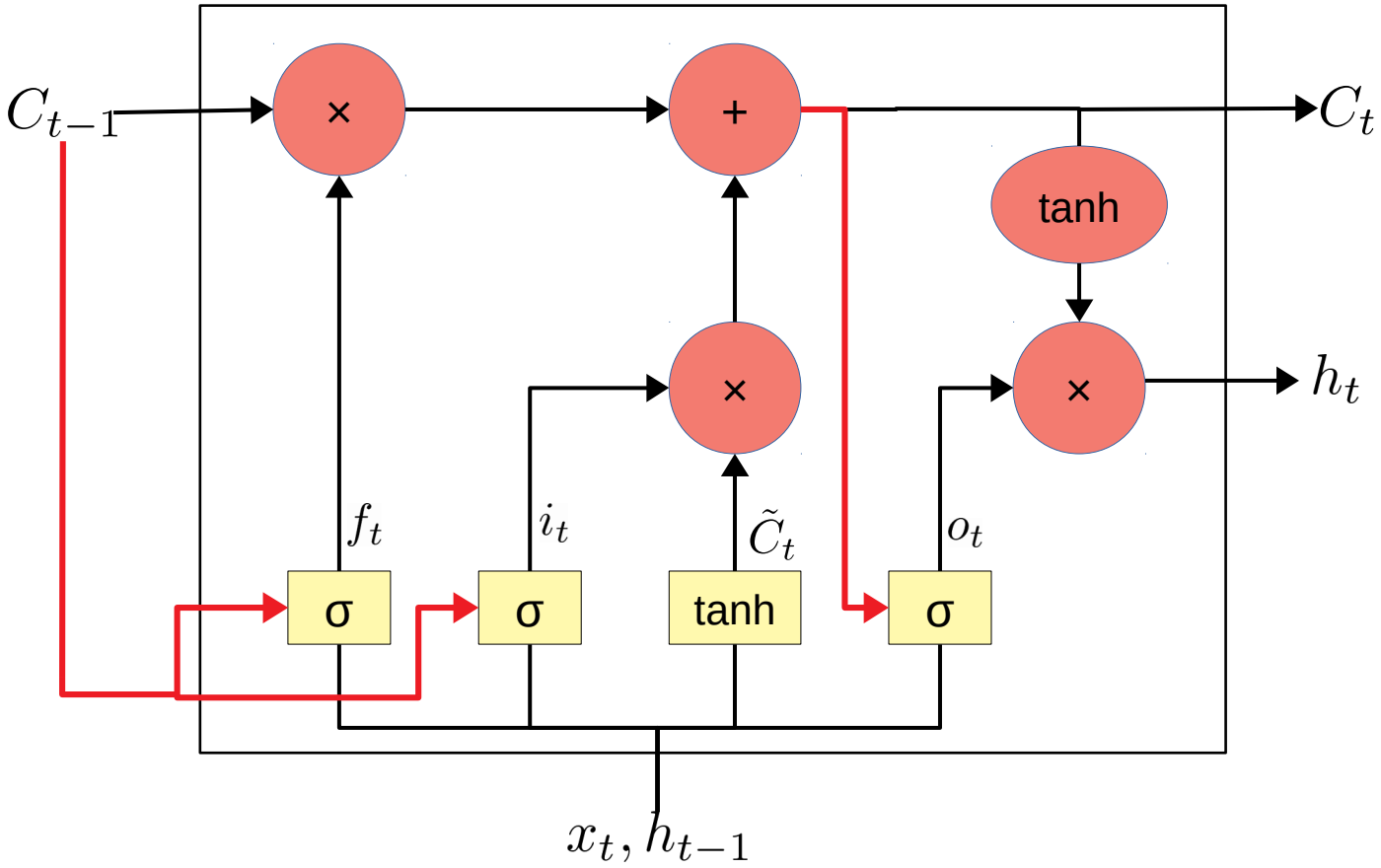
$$
\begin{aligned}
f_t &= \sigma\left(W_f\left[\boldsymbol{C_{t-1}}, h_{t-1}, x_t\right] + b_f\right) \\
i_t &= \sigma\left(W_i\left[\boldsymbol{C_{t-1}}, h_{t-1}, x_t\right] + b_i\right) \\
o_t &= \sigma\left(W_o\left[\boldsymbol{C_t}, h_{t-1}, x_t\right] + b_o\right)
\end{aligned}
$$

- Gekoppeltes *forget* und *input* gate

$$
\mathrm{C}_t = f_t \circ C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) \circ \tilde{C}_t
$$

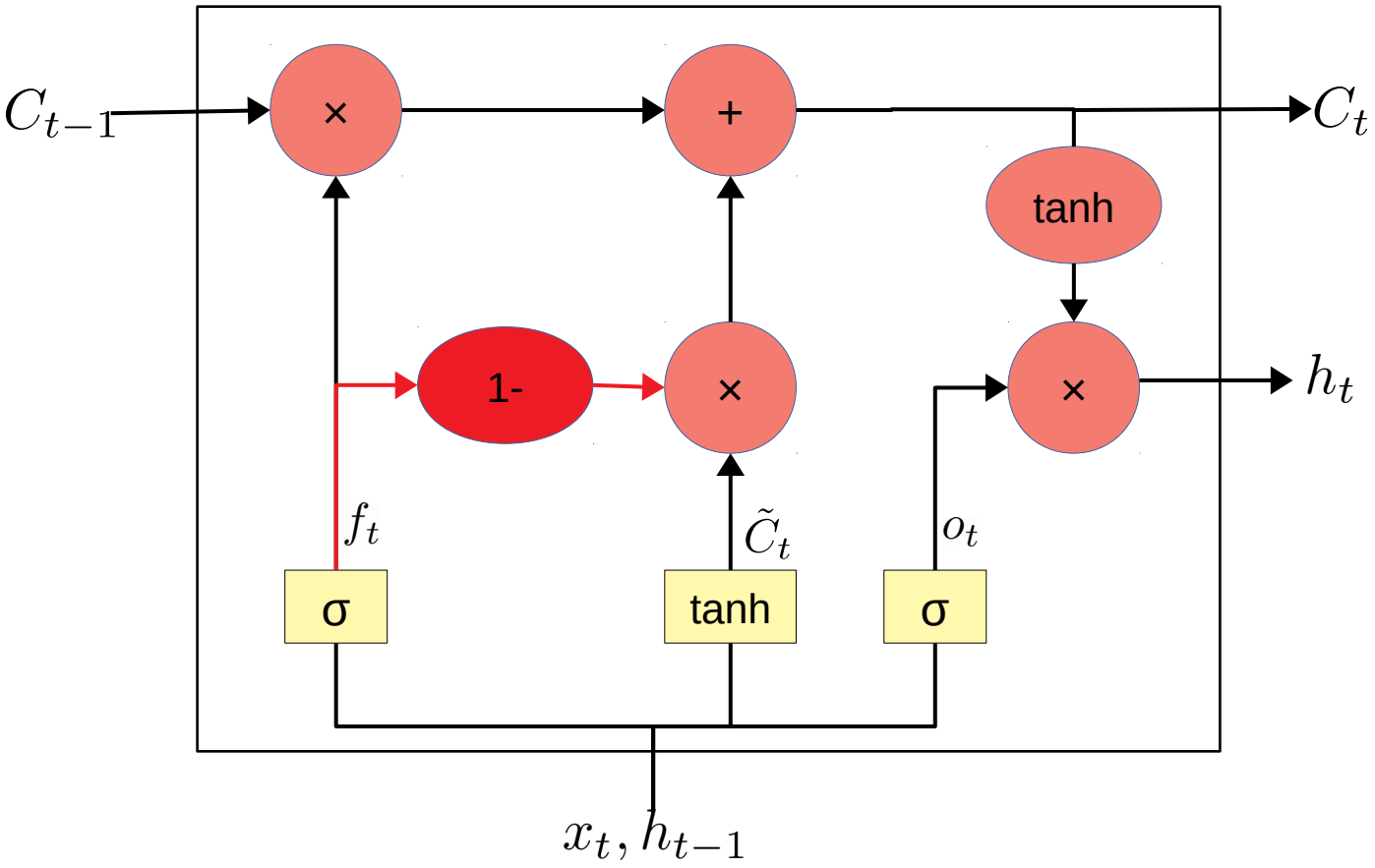
LSTM Varianten

Peephole Connections



LSTM Varianten

Gekoppeltes *forget* und *input gate*



Gated Recurrent Units

- Vereinfachtes LSTM, verwendet keinen Zellzustand C_t
- GRU nutzt ein *reset gate* an Stelle von *forget* und *output gate*, so wie ein *update gate*

Reset Gate:

Bestimmt wie viel aus dem vorherigen *hidden state* in den Neuen eingeht

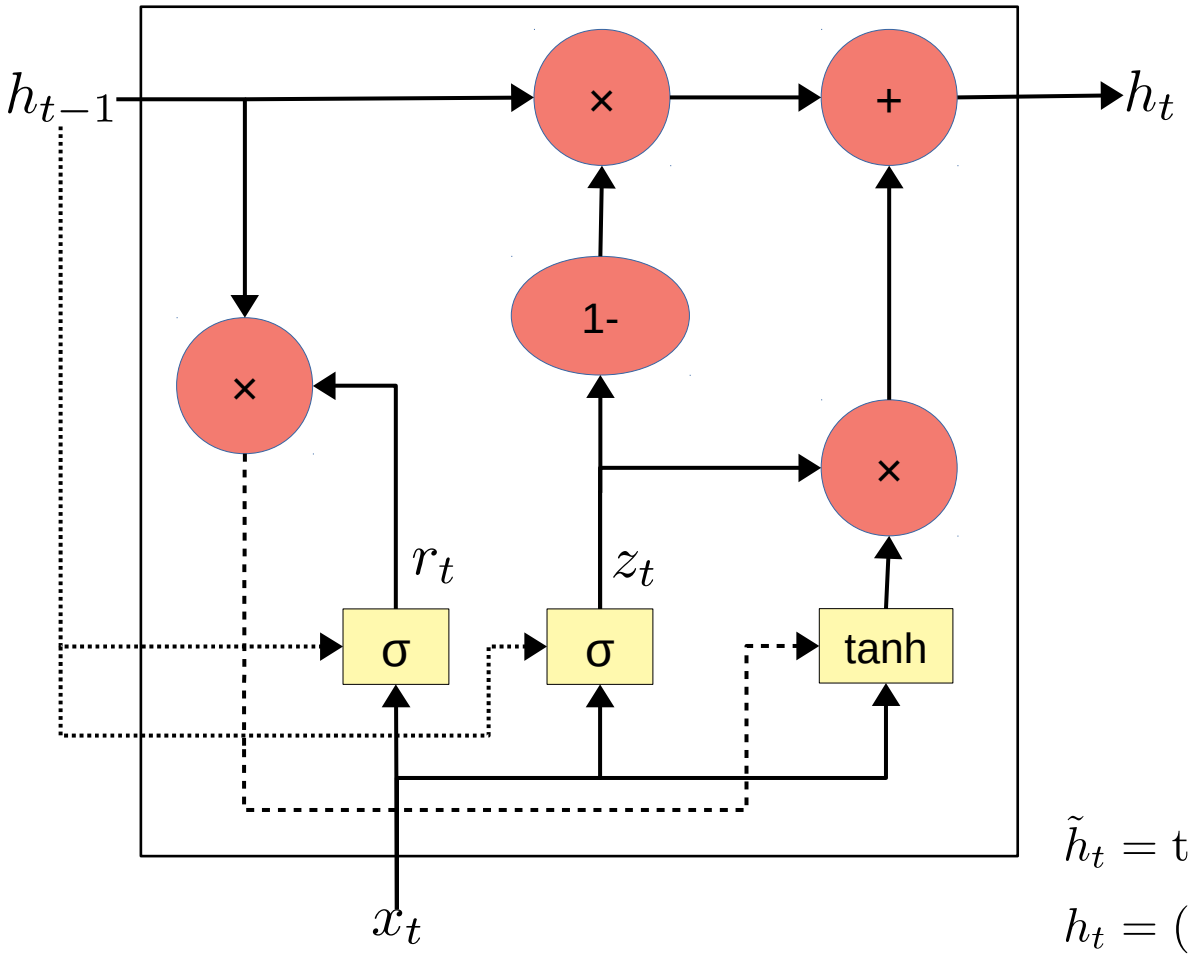
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

Update Gate:

Bestimmt zu welchen Anteilen neuer Input und alter *hidden state* in den neuen *hidden state* eingehen

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

Gated Recurrent Unit



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

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

LSTM und GRU

- LSTM und GRU können auch mehrschichtig verwendet werden
  - Eingabe ist hier der *hidden state* der vorhergehenden Schicht
- Welche LSTM-Alternative sich am besten eignet ist abhängig von der Problemstellung
- Einträge des Zellkontexts C_t können Sachverhalte direkt abbilden und Speichern
  - Bsp.: Vorzeichenwechsel eines Eintrags an öffnendem Anführungszeichen, Wechsel zurück erst am Ende der wörtlichen Rede

One-Hot-Kodierung

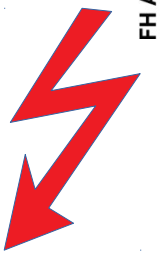
– Kategorien, Wörter und Zeichen müssen zur Eingabe in NN numerisch kodiert werden

– „Durchnummerierung“ ist oft nicht geeignet: Wörter erhalten so eine Ordnung bzw. Gewichtung

– Erzeugung eines Vektors, welcher 1 an der Stelle ist, die das Wort / Kategorie beschreibt, 0 sonst

– Erfordert begrenztes, bekanntes Alphabet oder Wörterbuch

Rot	Grün	Blau
1	2	3



$\Rightarrow Rot < Grün < Blau?$

Rot	Grün	Blau
1	0	0
0	1	0
0	0	1

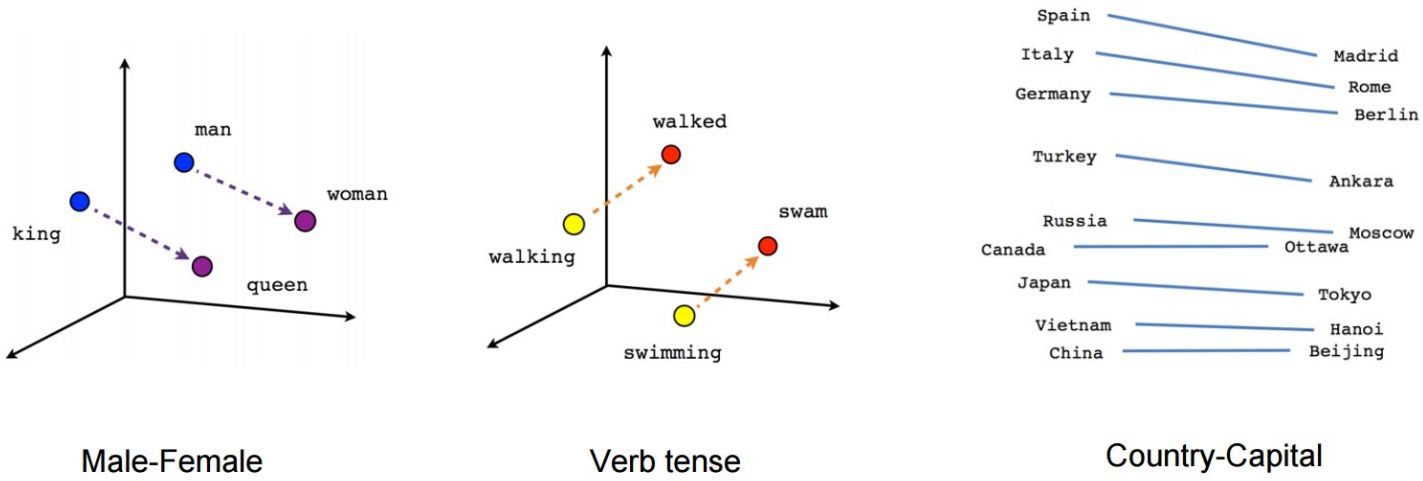
Word2Vec

– One-Hot-Kodierung ist ineffizient auf großem Wörterbuch (großer, dünnbesetzter Vektor)

– Alternative: Einbettungen wie *Word2Vec*

  – Einbettung wird erlernt, Vektor beschreibt Position im Raum

  – Distanzen und Position der Wörter beschreiben Beziehungen und semantische Bedeutungen



Male-Female | Verb tense | Country-Capital

Bildquelle: https://www.tensorflow.org/images/linear-relationships.png, CC BY 4

Tensorflow Tutorial „Vector Representations of Words“: https://www.tensorflow.org/tutorials/representation/word2vec

Übersicht

- Grundlagen
- Rekurrente Neuronen
- Strukturen von RNNs
- Trainieren von RNNs
- LSTMs / GRUs
- Beispiele für die Verwendung von RNNs

Beispiele für die Verwendung von RNNs

- MNIST Classifier mit (verschiedenen) RNNs
- Zeitreihenvorhersage mit RNNs
- Erstellen „kreativer“ Zeitreihen mit RNNs

Die Beispiele zeigen wir auf dem Jupyter Hub

(Ihr findet sie auch im Ilias)

Vielen Dank für die Aufmerksamkeit!

Fragen?