Maschinelles Lernen
- Informatik -

Rapp, DHBW Lörrach

26.01.2024

Inhaltsübersicht

Prüfungsaufgabe

Prüfungsaufgabe 3

Erstellung eines Wissensgraphen für Kunst (Aufgabe 1) und Abfrage mithilfe natürlicher Sprache(Aufgabe 2)

1. Optimieren Sie das Python-Programm Art_Knowledge_Graph.py auf die Verwendung mit dem Datensatz Art_Stories.csv (s. Moodle) für die Erzeugung einens sinnvollen Wissensgraphen.
2. Implementieren Sie eine Python-Funktion zur SPARQL-Abfrage des Wissensgraphen aus *Aufgabe 1* mithilfe natürlicher Sprache. Gehen Sie hierzu wie folgt vor:

   a) Analysieren Sie den generellen Aufbau des Programmcodes QAsparql.
   - Aktualisierter Code s. Moodle (Aus: https://github.com/Sylvia-Liang/QAsparql)
   - Aus: Querying knowledge graphs in natural language, Liang et. al, ETH Zürich, Springer Open, Journal of Big Data.
   - **Evtl. hilfreiche Installationshinweise:**
   - python -m spacy download en_core_web_lg
   - pip install fasttext-wheel
   - Glove-Download nach QAsparql-master\learning\treelstm\data\glove: https://github.com/stanfordnlp/GloVe?tab=readme-ov-file (z.B. Wikipedia 2014 + Gigaword 5)

   b) Implementieren Sie eine Python-Funktion zur Extraktion von RDF-Tripeln *(z.B. als .ttl-File, vgl. dbpedia 3Eng class.ttl)* aus Ihrem in *Aufgabe 1* erzeugten Wissensgraphen.

   c) Erstellen Sie für die 3 im Paper erwähnten Fragetypen List, Count und Boolean jeweils Beispiel-Trainingsdaten
   Hinweis: Die Beispiele können analog zu den Trainingsdaten in QAsparql-master/data/LC-QUAD/train-data.json aufgebaut sein.

   d) Implementieren Sie eine Python-Funktion, die eine Anfrage in natürlicher Sprache in die entsprechende SPARQL-Query basierend auf Ihren in c) generierten Beispiel-Trainingsdaten übersetzt:
   - **Funktionsname:** natural_language_to_sparql_query(natural_language_query_string)
   - **return:** sparql_query_for_knowledge_graph_string

Beispiele für Bewertungskriterien

- Aus dem Programmcode ist ersichtlich, dass die wesentlichen Konzepte aus der Vorlesung verstanden und auf die konkrete Problemstellung angewendet wurden.
- Der Fremdanteil am Programmcode ist mit entsprechenden Quellenangaben gekennzeichnet.
- ...

Vergleich menschliches und künstliches neuronales Netz

Neurologische Betrachtung menschlichen Lernens

Menschliche Großhirnrinde

- ca. 10 Milliarden Neuronen
- können parallel arbeiten
- stark vernetzt: jedes Neuron ist mit ca. 2.000 anderen Neuronen direkt verbunden

Speicherung von Informationen im menschlichen Gehirn

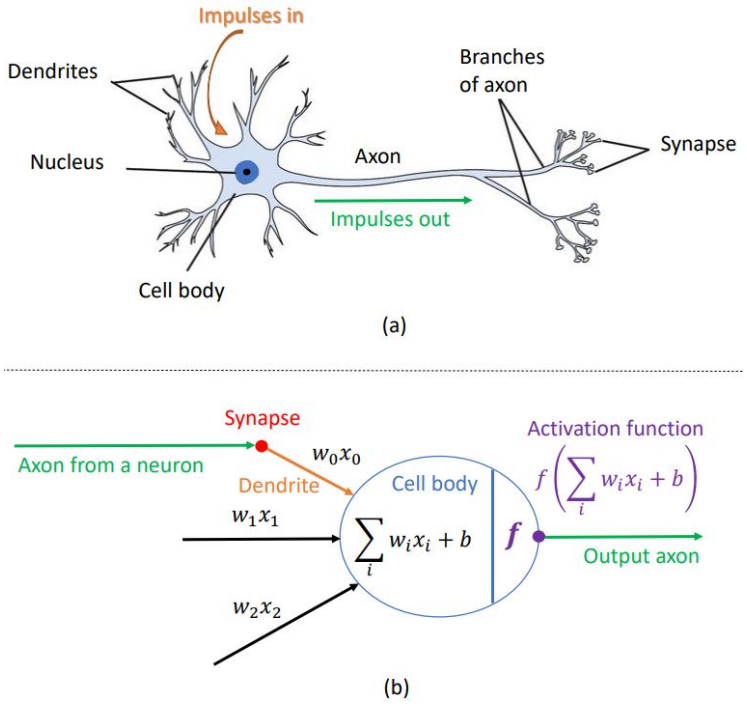
Informationen sind nicht in den einzelnen Neuronen gespeichert, sondern werden durch den **gesamten Zustand des Netzes** mit allen **Verbindungen** und **Bindungsstärken** repräsentiert.

Menschliches Lernen

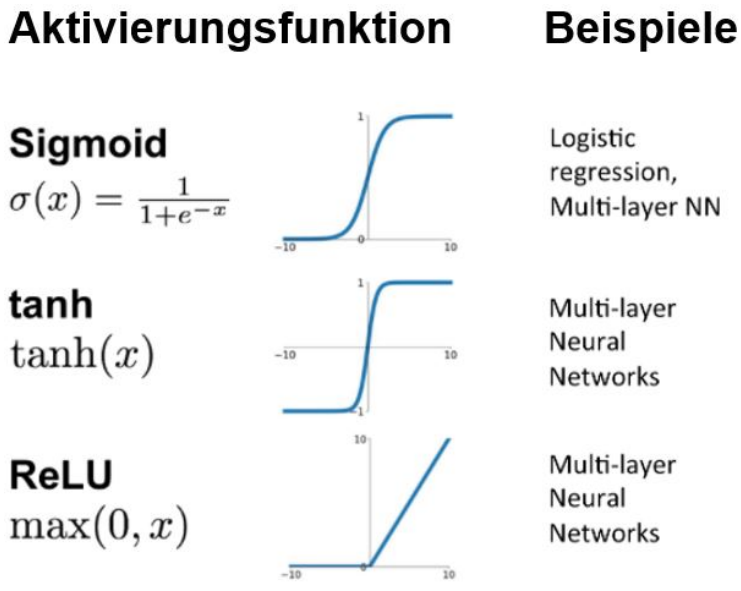
Kann zurückgeführt werden auf die Veränderung der Verbindungen und Bindungsstärken von Neuronen.

Menschliches vs. künstliches neuronales Netz

Menschliches Neuron (a) und Modellierung eines Neurons in einem künstlichen neuronalen Netz (b). Die **Synapse** entspricht dem Input x_i und der **Zellkörper** dem Sammeln von mit **Gewichten** w_i multiplizierten Inputs *(b: Bias)* und Filtern durch die **Aktivierungsfunktion** f.



Roffo, 2017



Beispiele für Aktivierungsfunktionen künstlicher neuronaler Netze

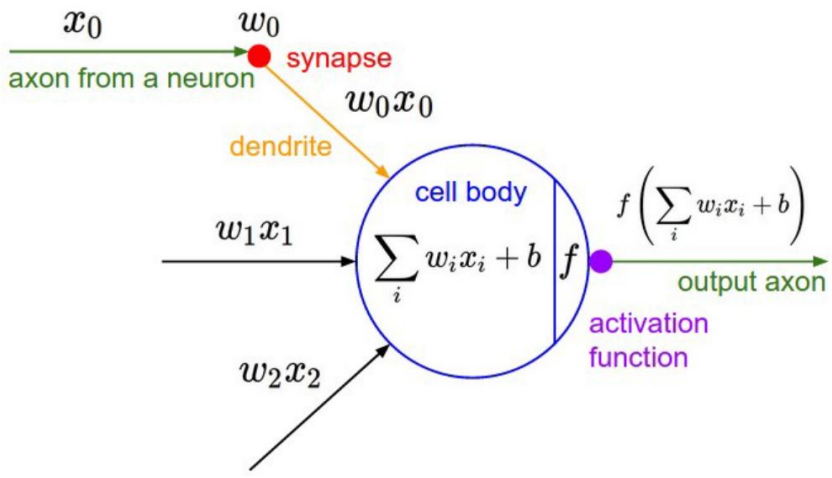
Beispiel Sigmoid: Sind die ankommenden Signale von anderen Nervenzellen stark genug, wird also ein bestimmter Schwellenwert der Erregung überschritten, **feuert das Neuron:** $\sigma(x) \approx 1$, d.h. ein elektrischer Impuls, das so genannte **Aktionspotenzial,** schießt am Axon entlang in Richtung Synapse. Reiz unterhalb Schwellenwert: **Zelle feuert keinen Impuls ab:** $\sigma(x) \approx 0$.

Vereinfachtes künstliches neuronales Netz

Perzeptron *(engl. perception: Wahrnehmung, Rosenblatt 1957)*

Einzelnes künstliches Neuron mit anpassbaren Gewichten und nichtlinearer Aktivierungsfunktion mit Schwellenwert.

Das Neuron **feuert,** wenn eine **lineare Kombination** seiner Eingaben einen bestimmten **Schwellenwert** überschreitet:



Klassifikation des Inputs x:

$$a_{out} = f(\sum_i w_i x_i + b)$$

$$= \begin{cases} 1 & \text{wenn } \sum_i w_i x_i + b > 0, \\ 0 & \text{sonst} \end{cases}$$

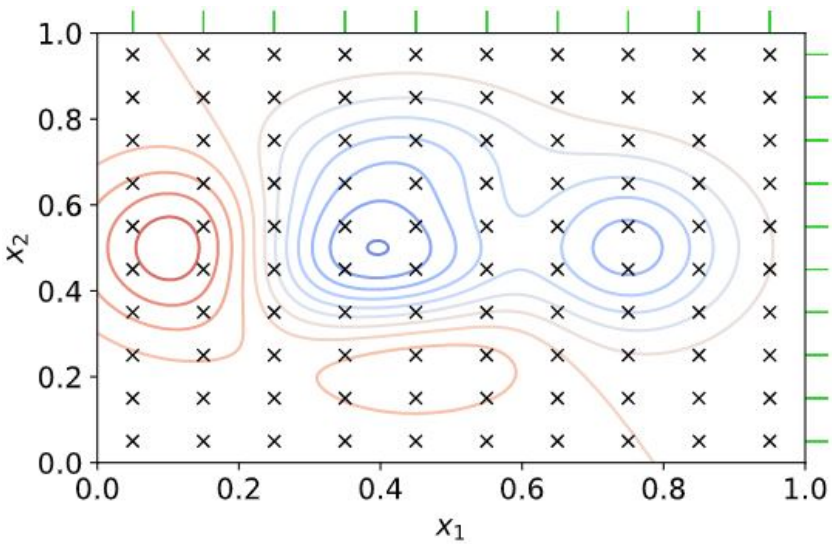
Perzeptron-Netze sind **vorwärtsgerichtet** *(engl. feedforward)* und wandeln einen **Eingabevektor** in einen (z.B. eindimensionalen) **Ausgabevektor** um ($\rightarrow$ *Assoziativspeicher*).

Hyperparameter

Hyperparameter künstlicher neuronaler Netze

Hyperparameter

Werden zur Steuerung des Lernprozesses verwendet und bereits vor dem Training des Modells festgelegt. Im Gegensatz dazu werden die Werte weiterer Parameter (z.B. Gewichte w_i) erst während des Trainings gelernt.



Optimierung via Rastersuche

- Im Beispiel links werden für zwei Hyperparameter jeweils 10 verschiedene Werte betrachtet
- Insgesamt 100 verschiedene Kombinationen
- blau: Regionen mit guten (rot: schlechten) Resultaten

Beispiele für Hyperparameter

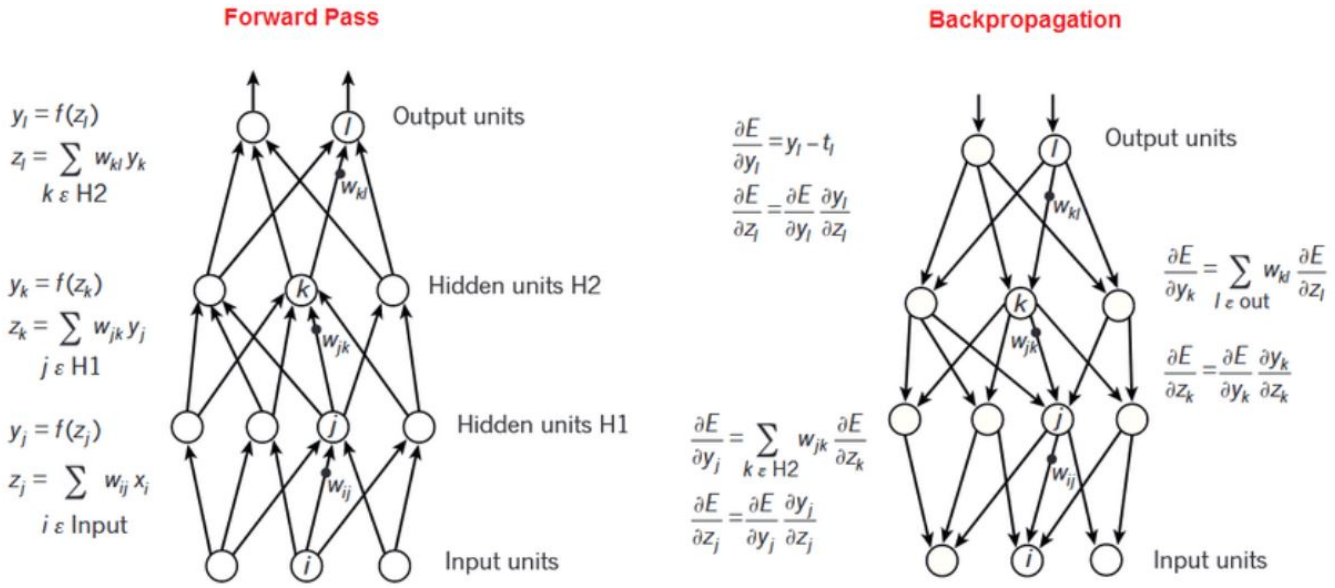
- Aktivierungsfunktion
- Anzahl verdeckter Schichten im neuronalen Netz

Lernprozess

Training mit Forward Pass und Backpropagation

Die **Anpassung der Gewichte** im Training erfolgt in **3 aufeinanderfolgenden Schritten:**

1. **Forward Pass** *(links):* Zuerst durchläuft der Input die Input-Neuronen und anschließend wird daraus der Output des neuronalen Netzes berechnet.
2. **Fehlerbestimmung:** Die gewünschten Output-Werte ($\rightarrow$ Labels im überwachten Lernen) werden mit denjenigen aus den tatsächlichen im Forward-Pass ermittelten Werten verglichen.

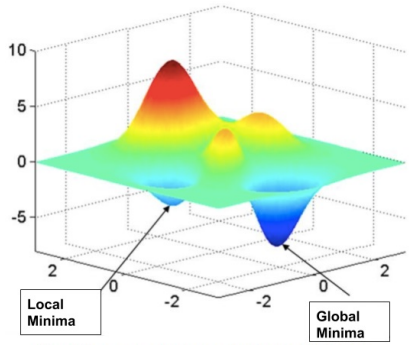


LeCun et al. Deep learning, Nature, 2015

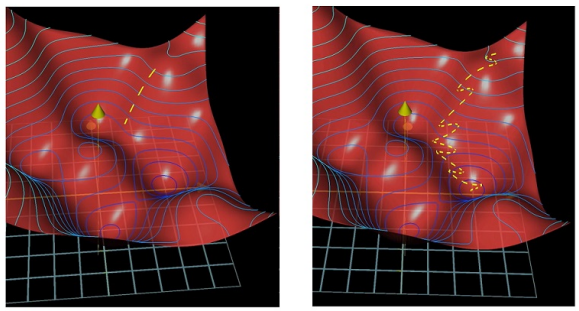
3. **Backpropagation** (*Lernprozess des künstlichen neuronalen Netzes, rechts*):
   Die Fehlerterme breiten sich in entgegengesetzter Richtung bis zur Input-Schicht aus. Mit Hilfe dieser Fehlerterme werden nun nach und nach (d.h. zunächst zwischen Output und letzter Hidden-Schicht, dann zwischen letzter und vorletzter Hidden-Schicht usw.) die Gewichte des Netzes modifiziert, so dass die Fehlerterme kleiner werden.

Zwei Optimierungsalgorithmen im Vergleich

Backpropagation verwendet einen **Optimierungsalgorithmus** für die Anpassung der Gewichte in den einzelnen Layern des künstlichen neuronalen Netzes, um iterativ die Fehler zu **minimieren.**



Visualisierung von lokalem und globalem Minimum; Data Analytics, vitalflux.com



Links: Gradient Descent, rechts: Stochastic Gradient Descent; 3blue1brown.com/lessons/gradient-descent

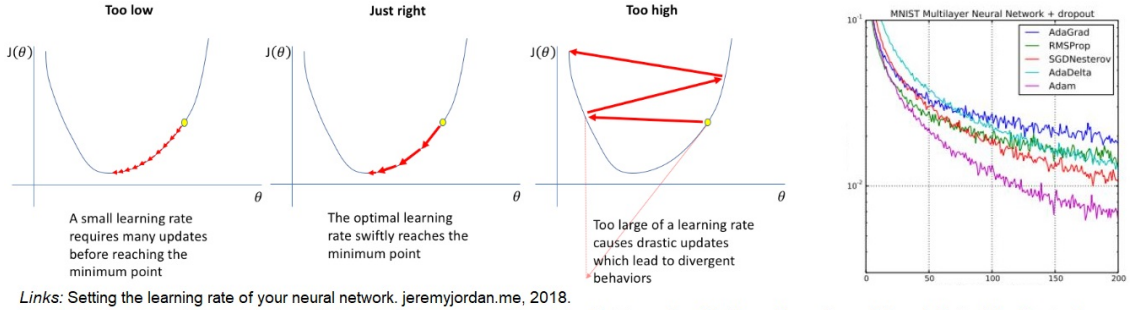
Gradient Descent vs. Stochastic Gradient Descent

- **GD:** Durchlaufen *aller Beispiele* eines Trainingssatzes notwendig für einzelnes Update innerhalb einer Iteration
- **SGD:** Verwendung von nur *einem* oder *Teilsatz* von Trainingsbeispielen aus dem Trainingssatz wird für die Updates verwendet
  - konvergiert oft viel schneller als GD
  - Genauigkeit kann in bestimmten Fällen schlechter sein, z.B. wenn die Werte um das Optimum oszillieren → Berücksichtigung von Momentum *(„Impuls“)* für stabilere und noch schnellere Konvergenz

Erweiterung von SGD durch adaptive Lernrate

Lernrate α

- Gibt die Schrittgröße des Optimierungsverfahrens in Richtung des Minimums der Fehlerfunktion an.
- α zu groß: Minima können übersprungen werden
- α zu klein: Geringe Konvergenzgeschwindigkeit bzw. Algorithmus bleibt in lokalem Minimum hängen



Links: Setting the learning rate of your neural network. jeremyjordan.me, 2018.
Rechts: Vergleich von Adam mit anderen Optimierungsalgorithmen während des Trainings eines Multilayer Perzeptrons. Adam: A Method for Stochastic Optimization; Kingma et al., 2015

Adam (Adaptive Moment Estimation)

- Verwendet **adaptive Lernrate** anstelle einer vordefinierten und fixen Lernrate, wie das bei Stochastic Gradient Descent der Fall ist.
- arbeitet rechentechnisch effizient und für große Datensätze geeignet

Standard-Optimierungsalgorithmus für die **iterative Anpassung der Netzwerkgewichte** im Deep Learning *(vgl. Keras Python Bibliothek).*

Recurrent Neural Networks

Recurrent Neural Network - RNN

Rekurrentes neuronales Netz

Rückgekoppeltes künstliches neuronales Netz, in dem die Neuronen einer Schicht im Gegensatz zu Feed Forward Netzen zusätzlich mit Neuronen derselben oder vorangegangenen Schicht verbunden sind.

Durch den Aufbewahrungsmechanismus *(„interner Speicher“)* für vergangene Informationen können zukünftige Werte vorhergesagt werden.
$\Rightarrow$ Aufdeckung zeitlich codierter Informationen in den Daten

Anwendungsbereiche

- Zeitreihen (z.B. Aktienkurse)
- Sequenzen (z.B. Kunden, die A kaufen, kaufen mit hoher Wahrscheinlichkeit auch B innerhalb der nächsten 4 Transaktionen)

Analogie menschliches Gehirn

- Bevorzugte Verschaltungsweise im Neocortex

Sequenzen und Zeitreihen

Sequenz

Zuordnung der Indexmenge $I_n = \{1, 2, ..., n\}$ auf eine Domäne O (z.B. Gensequenz $\{C, G, A, T\}$).

Zeitreihe

- Spezialfall einer Sequenz
- Werte werden über die Zeit aufgezeichnet
- Indexmenge I_n wird durch Timestamps repräsentiert

Beispiele für Zeitreihen

- **Univariat:** Aktienkurs $p(t)$ *(p : Preis)* oder Temperaturkurve $T(t)$
- **Multivariat:** Kombination aus Wettersensordaten Temperatur, Feuchtigkeit, Windstärke,...
  - Jede Variable hängt von der Zeit sowie jeweils den anderen Variablen ab

Vergleich zu Feed-Forward Netzwerk

In einem neuronalen **Feed-Forward Netzwerk** werden die Informationen **nur in eine Richtung,** d.h. vom Input Layer durch die Hidden Layer zum Output Layer, weitergegeben.

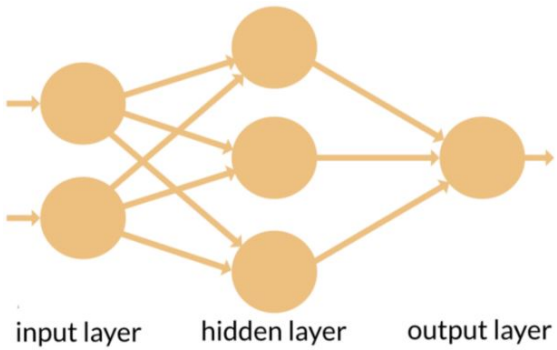


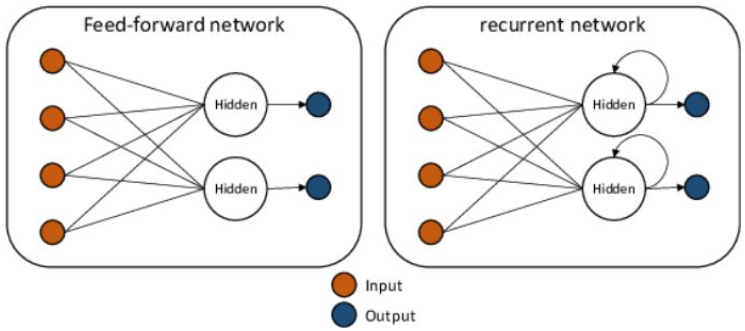
Abb.: Informationsfluss im Feed Forward Netz

Feed-Forward Netzwerk (z.B. CNN)

- Nur die aktuelle Eingabe wird berücksichtigt
  $\Rightarrow$ keine zeitliche Reihenfolge abbildbar
- Keine Erinnerung an vorherige Eingangssignale
  $\Rightarrow$ keine Vorhersage möglich, was als nächstes kommt

Gedächtnis des RNN

In einem **Recurrent Neural Network** hingegen durchlaufen die Informationen zusätzlich eine Schleife:



Vergleich Informationsfluss Feed-Forward vs. RNN, **Motaz et. al, ResearchGate**

Wenn ein RNN eine Entscheidung trifft, berücksichtigt es die **aktuelle Eingabe** und auch das, was es aus den **zuvor erhaltenen Eingaben** gelernt hat. *Alternative Formulierung:*

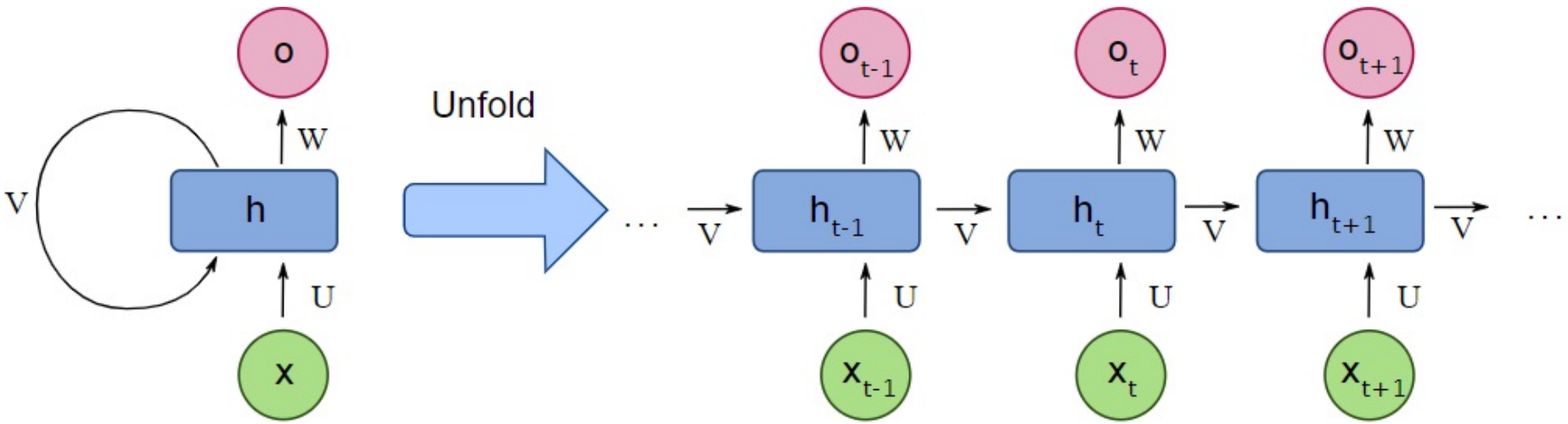
Reccurent Neural Networks fügen im Gegensatz zu Feed Forward Netzen der **Gegenwart** die **unmittelbare Vergangenheit** hinzu.

Ein Recurrent Neural Network hat daher **zwei Eingänge,** einen für die Informationen aus der **Gegenwart** und einen für diejenigen aus **Vergangenheit.**

Generisches RNN

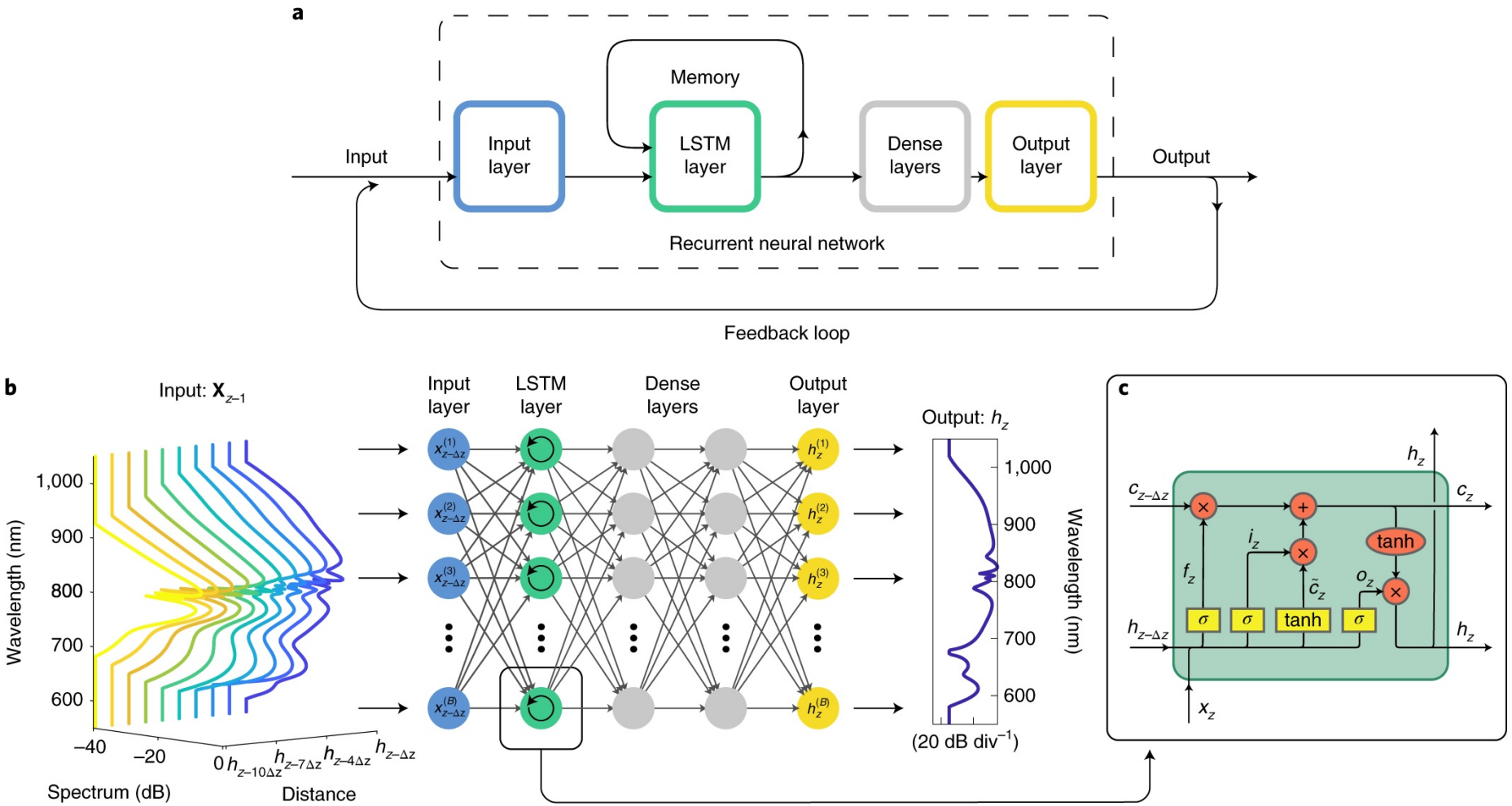
Fully Recurrent Neural Network

Die Outputs aller Neuronen sind mit den Inputs aller Neuronen verbunden. Durch Nullsetzen einzelner Gewichte können weitere Varianten aus dieser generischen Architekturvariante abgeleitet werden.



Links: kompakte Darstellung des Fully Recurrent Neural Network
Rechts: zeitlich ausgerollt mit x_t : Input , h_t : Hidden und o_t : Output Layer zum Zeitpunkt t.
Quelle: Wikipedia

Praxisbeispiel RNN/LSTM



a: LSTM Layer als Baustein eines künstlichen neuronalen Netzes, b: Rückkopplung im LSTM Layer, c: schematischer Aufbau LSTM

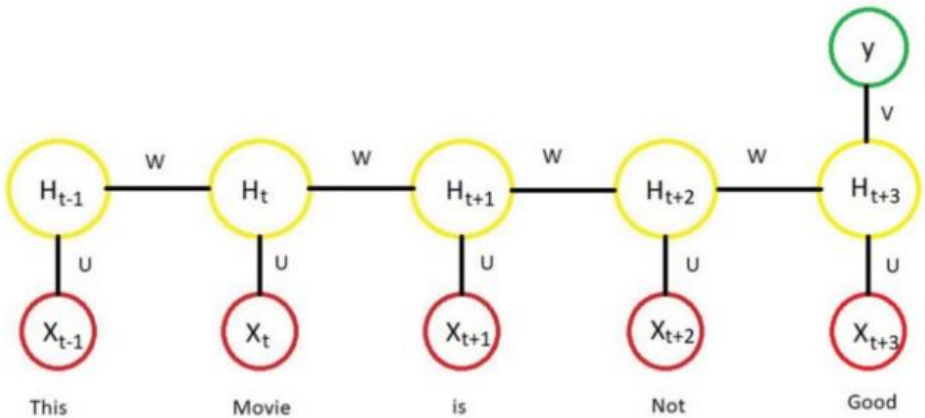
Aus: Predicting ultrafast nonlinear dynamics in fibre optics with a recurrent neural network, Nature https://www.nature.com/articles/s42256-021-00297-z

RNN Architekturtypen

Many-To-One

Many-To-One RNN

- **Input:** Sequenz (z.B. Text)
- **Output:** Einzelner Wert (z.B. Klassifikationslabel)



H_t: Output der Aktivierungsfunktion zum Zeitpunkt t
W: Gewichtematrix

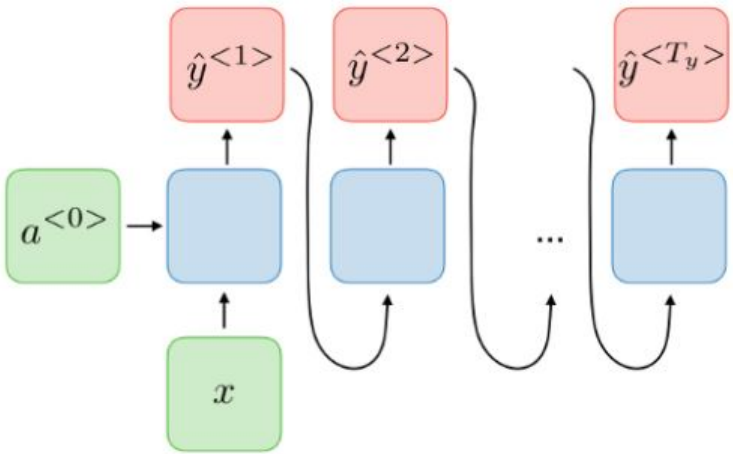
Anwendungsbeispiel

- Sentiment-Analyse für Tweets oder Facebook Posts (z.B. positiv/negativ konnotiert)

One-To-Many

One-To-Many RNN

- **Input:** Einzelner Sequenzwert (z.B. Bild)
- **Output:** Sequenz (z.B. Bildbeschreibung)



x: Input
y: Outputsequenz

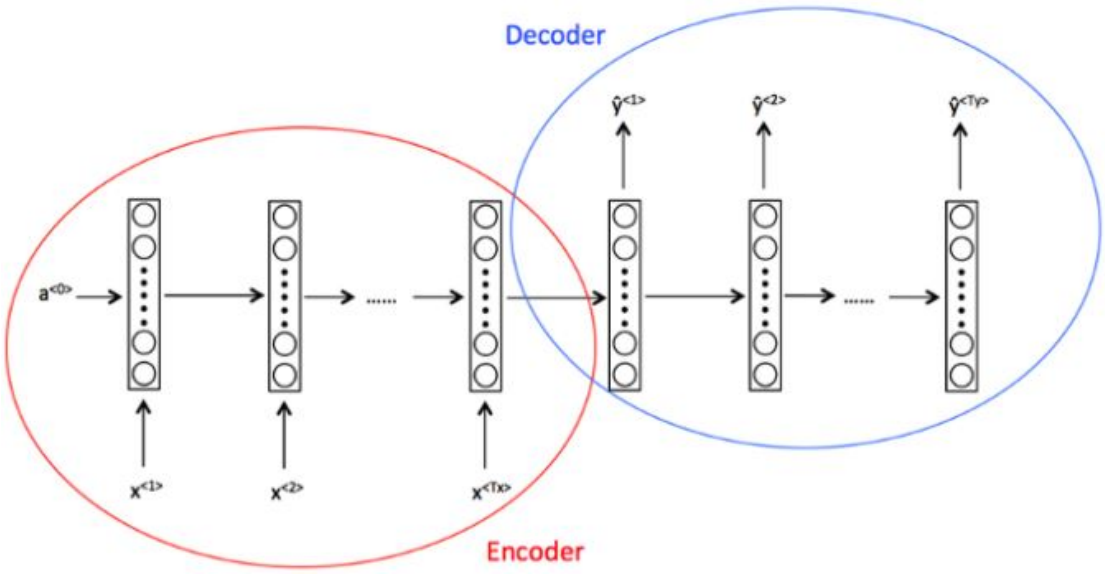
Anwendungsbeispiele

- Textergängzung mit Siri
- Erzeugung von Musik basierend auf erster Note

Many-To-Many

Many-To-Many RNN

- Input/Output: Sequenz (z.B. Satz auf Englisch/Deutsch)



Encoder: *Der Teil des Netzwerkes, der den zu übersetzenden Satz/die zu beantwortende Frage liest.*
Decoder: *Übersetzt den Satz in die gewünschte Sprache/beantwortet die Frage.*
Die Länge der Inputs kann sich von derjenigen der Outputs unterscheiden.

Anwendungsbeispiele

- Maschinelles Übersetzen
- Chatbots


WHY ARE WE NOT DONE YET?

Drei Probleme mit RNN

1. **Explodierender Gradient**
   In tiefen neuronalen Netzen kann der Fehlergradient während eines Updates durch wiederholtes Multiplizieren von Gradienten mit betragsmäßigen Werten größer 1 exponentiell vergrößern. Das führt zu großen Updates der Netzgewichte und somit zu einem unstabilen Netz durch z.B. Overflow oder NaN Werte.
   Lösungsansatz: „Abschneiden“ bzw. „Stauchen“ der Gradienten
2. **Verschwindender Gradient**
   Wir sprechen von „verschwindenden Gradienten“, wenn die Werte des Gradienten zu klein sind und das Modell nicht mehr lernt oder das Lernen aus diesem Grund zu lange dauert.
3. **Kurzzeitgedächtnis**
   Weit zurückliegende Informationen werden beim RNN „vergessen“.

Lösungsansatz: Gated Architekturen wie z.B. LSTM und GRU

Long-Short Term Memory

LSTM

Long Short-Term Memory

- ermöglicht *„Kurzzeitgedächtnis, das lange anhält“*
- spezieller Funktionsblock rekurrenter neuronaler Netze
- erfunden von Sepp Hochreiter und Jürgen Schmidhuber in 1997

Mit LSTMs versehene RNNs sind in der Lage, sich an **Langzeit-Abhängigkeiten** zu erinnern.

LSTM Anwendungen

LSTM-Meilensteine

- 2018: Bill Gates nannte es einen enormen KI Fortschritt, als OpenAI Bots mit einem 1024-Unit LSTM die besten **Dota 2**-Spieler besiegten
- 2019: Der Einsatz eines tiefen LSTM Kerns durch DeepMind's Programm AlphaStar im komplexen Videospiel **Starcraft II** stellt eineb bedeutenden Schritt in Richtung *Artificial General Intelligence* dar.

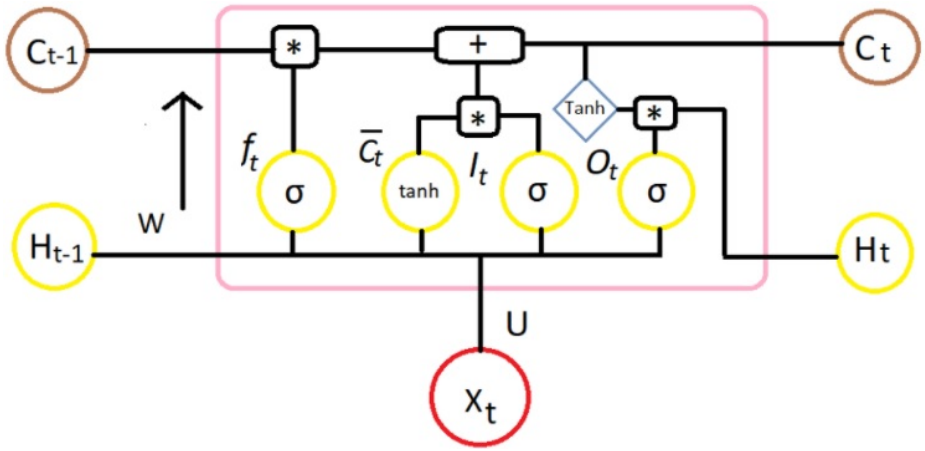
Weitere Anwendungsbeispiele

- Apple: Intelligente Quicktype-Tastaturfunktion und sprachgesteuerter Assistent Siri
- Amazon: Alexa-Assistent
- Google: Spracherkennung und Translate
- Google Deep Mind: AlphaGo gegen Lee Sedol

Long-Short Term Memory - Modell

Cell State c_t

- Kernkomponente der LSTM-Einheit
- transportiert relevante Informationen über die Zeitschritte hinweg
  $\Rightarrow$ stellt **Gedächtnis** dar



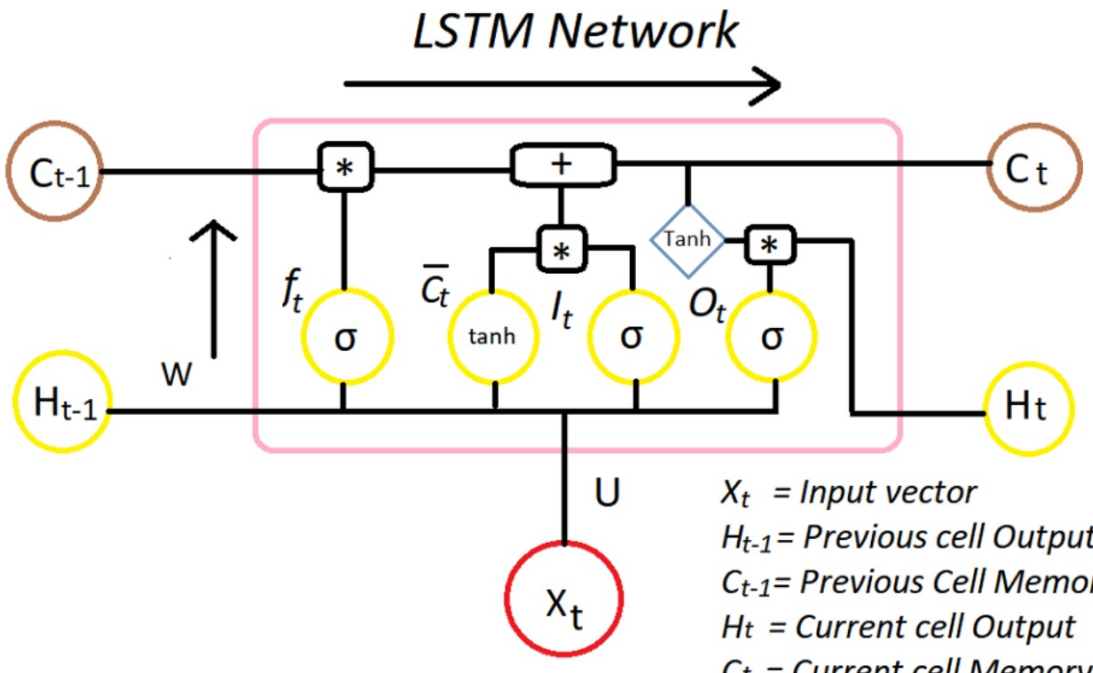
Schematischer Aufbau einer LSTM-Einheit (kettenartig hintereinandergeschaltet)
medium.com

Welche Informationen als relevant gelten, entscheiden **drei Gates,** die...

- **Input:** ...neue Informationen einlassen oder nicht
- **Forget:** ...Informationen löschen, weil sie nicht wichtig sind
- **Output:** ...den Ausgang im aktuellen Zeitschritt beeinflussen

Formeldarstellung

- f: Forget Gate (NN mit Sigmoid)
- $\bar{C}$: Candidate Layer (NN mit Tanh)
- I: Input Gate (NN mit Sigmoid)
- O: Output Gate (NN mit Sigmoid)
- H: Hidden State (Vektor)
- C: Memory (Cell) State (Vektor)



X_t = Input vector
H_{t-1} = Previous cell Output
C_{t-1} = Previous Cell Memory
H_t = Current cell Output
C_t = Current cell Memory

(*) = Element-wise multiplication
(+) = Element-wise addition

$$f_t = \sigma(X_t * U_f + H_{t-1} * W_f)$$
$$\bar{C}_t = \tanh(X_t * U_c + H_{t-1} * W_c)$$
$$I_t = \sigma(X_t * U_i + H_{t-1} * W_i)$$
$$O_t = \sigma(X_t * U_o + H_{t-1} * W_o)$$

$$C_t = f_t * C_{t-1} + I_t * \bar{C}_t$$
$$H_t = O_t * \tanh(C_t)$$

W, U = weight vectors for forget gate (f), candidate (c), i/p gate (I) and o/p gate (O)

Note : These are different weights for different gates, for simpicity's sake, I mentioned W and U

Betrachten Sie einen **Hidden LSTM-Layer** mit n Neuronen, d.h. $f_t \in (0,1)^n$:

$$(f_t)_k \approx 0 \rightarrow \text{Vergiss Informationen zu } (c_{t-1})_k$$
$$(f_t)_k \approx 1 \rightarrow \text{Behalte Informationen zu } (c_{t-1})_k$$

Bidirektionale RNN

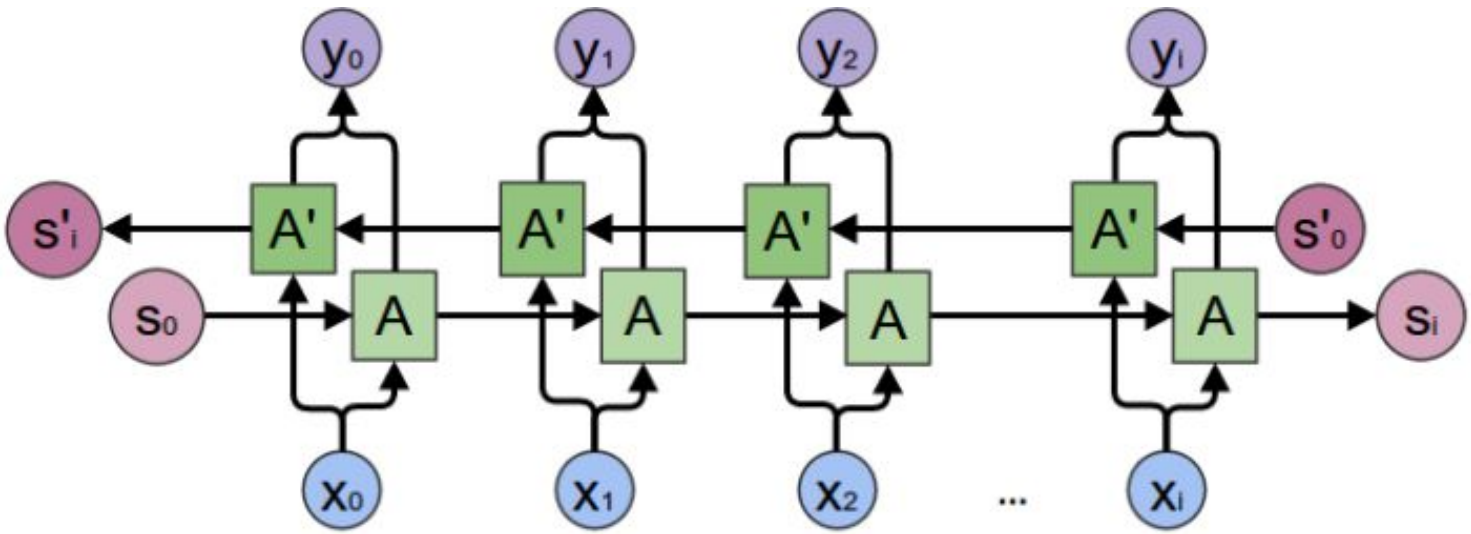
Motivation bidirektionales RNN

In den bisher vorgestellten RNN Architekturen können lediglich **vorherige Inputs** berücksichtigt werden.

In Bezug auf **Natural Language Processing** bedeutet das, dass nur der Effekt von Wörtern **vor dem aktuellen Wort** berücksichtigt werden kann.

Da dies für den Aufbau einer Sprachstruktur nicht ausreicht, kommen **bidirektionale Recurrent Neural Networks** ins Spiel.

Aufbau bidirektionales RNN



Beispiel bidirektionales RNN

Vorhersage des nächsten Wortes in einem Satz

Das **unidirektionale LSTM** sieht lediglich den folgenden Satzanfang: "The boys went to..." und versucht das nächste Wort auf Basis dieses Kontexts vorherzusagen.

Ein **bidirektionales RNN** besteht aus einem **Forward** und **Backward RNN**. Die Vorhersage setzt sich zu einem gegebenen Zeitpunkt t aus den **Ergebnissen beider Netzwerke** zusammen.
Das **bidirektionale LSTM** sieht beide Richtungen:

1. **Forward LSTM**
   "The boys went to..."
2. **Backward LSTM**
   "...and then they got out of the pool"

Mit der Information aus der Zukunft fällt es dem künstlichen neuronalen Netz leichter, das nächste Wort vorherzusagen.

Fazit

Deep Feed-Forward neuronale Netze

- vielseitig anwendbar
- nicht optimal für Zeitreihen-Probleme, da z.B. Gedächtnislosigkeit

Recurrent Neural Networks

- Eignen sich u.a. für Zeitreihen-Probleme (z.B. Spracherkennung, Sprachübersetzung, Vorhersage von Aktienkursen)
- Aber: Noch immer Vanishing/Exploding Gradient Problem

Gated Architekturen

- z.B. LSTM und GRU
- Reduzieren Vanishing/Exploding Gradient Problem
- Allerdings: Auch Gated Architekturen stoßen bei sehr weit zurückliegenden Informationen an ihre Grenzen $\rightarrow$ Transformer