

## **Künstliche Intelligenz 2 - FS20**

*Pascal Brunner - brunnpa7*

# **Inhaltsverzeichnis**

# Kapitel 1

## Unsupervised Learning

### 1.1 Clustering

Clustering bezeichnet das Einordnen von ungelabelte Daten in ähnliche Gruppen → ist eine Sammlung von Daten, welche ähnliche Eigenschaften aufweisen oder im Vergleich zu den anderen Clustern unähnlich aussehen.

Für das Clustering kann man unterschiedliche Techniken anwenden, diese werde in die unterschiedlichen Eigenschaften eingeordnet:

- Hierarchisches Clustering → Divisive; **Agglomerative**
- Partionales Clustering → **Centroid**; Model Based; Graph Theoretic; Spectral
- Bayesian Clustering → Decision Based; Nonparametric

#### 1.1.1 Agglomerative Clustering

Hierbei handelt es sich um ein einfacher Clustering-Algorithmus, wobei man eine Distanz zwischen den Cluster definieren soll.

**Komplexität:**  $O(m^2 \log(m))$

**Vorgehen:**

1. **Initial:** Zu Beginn ist jeder Datensatz ein eigenes Cluster
2. **Iteration:** a) Berechne die Distanz zwischen allen Cluster und speichere diese; b) Vereinige die beiden Cluster, welche am nächsten zueinander sind
3. Speichere beide Cluster und die Sequenz der Cluster-Operation
4. Als Resultat entsteht ein *Dendrogram*

#### Methoden für die Distanzberechnung

- Nearest Neighbour
- Furthers Neighbour
- Average Distance
- Centroid Distance

### 1.1.2 K-Means

Beim k-Means Clustering-Algorithmus handelt es sich um eine partionale Clustering Methode. Wobei der Algorithmus die Einteilung der gegebenen Daten in  $k$  Clusters vornimmt.

- Jedes Cluster hat ein Cluster-Center (*Centroid* genannt)
- $k$  ist durch den User spezifiziert

**Vorgehen:**

1. Wähle zufällig viele  $k$  Daten-Punkte (seeds) für den initialen Centroid (das Zentrum)
2. Weise jedem Datenpunkt den am nächsten gelegenden centroid zu
3. Berechne den centroid mittels den neuen cluster-Beziehungen aus
4. Wiederhole Schritt 2 und 3, bis das Konvergenzkriterium erreicht ist

⇒ Der K-Means Algorithmus ist ein EM-Algorithmus, wobei

- *E-Step*: weise die Punkte zum nächsten Cluster Center zu
- *M-Step*: Setze das Cluster-Center neu

⇒ K-Means  $\neq$  K-Nearest Neighbour

**mögliche Konvergenz-Kriterien**

- Keine neu Zuweisung von Datenpunkten in ein neues Cluster
- Kein Wechsel von den Centroids
- Keine Zunahme der Sum of Squared Error (SSE)  $\sum_{j=1}^k \sum_{x \in C_j} d(x, m_j)^2$

### 1.1.3 Elbow-Method

Es ist meistens die Frage wie hoch der SSE sein soll im Verhältnis zu der Anzahl von Cluster. Hierzu eignet sich die Elbow-methode. An dieser Stelle im Grafen wo es einen ellbogenartigen Knick hat, soll es entsprechend gewählt werden.

1. Berechne mittels des Clustering-Algorithmus für die verschiedenen Werte von  $k$
2. Für jedes  $k$ , berechne die SSE
3. Plot die SSE Kurve (y-Achse) und die Anzahl Clusters  $k$  (x-Achse)
4. Beim Knick im Plot ist der Indikator

### 1.1.4 Hard vs. Soft Clustering

*Hard-Clustering*

- Gibt es keine Überlappung
- Jedes Element gehört zu einem Cluster oder nicht
- Bspw.: Hierarchisches Clustering, K-Means

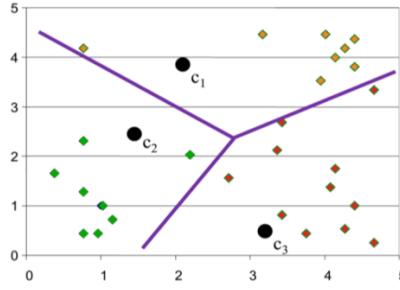


Abbildung 1.1: Grafik bei einem Hard Clustering

### Soft Clustering

- Clusters können überlappen
- Stärke der Assoziation zwischen Clustern und Elementen (Wahrscheinlichkeit)

## 1.2 Expectation Maximization (EM)

## 1.3 One-Class Classification

Die One-Class Classification wird auch als Ausreissererkennung, Neuheitserkennung oder Konzeptlernen bezeichnet. Dabei können unterschiedliche Algorithmen zum Zuge kommen:

- One-Class Support Vector Machine
- Clustering
- Decision Trees
- Neural Networks
- Bayesian Classifiers

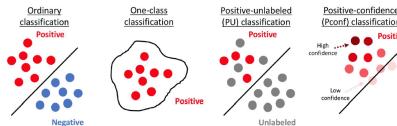


Abbildung 1.2: Überblick über die verschiedenen Classification Tasks

### 1.3.1 Simple One Class Classification Algorithmus

Eine neues Objekt  $z$  gehört zur entsprechenden Class, wenn folgendes gilt:

$$dist(z, NN^{train}(z)) < dist(NN^{train}(z), NN^{train}(NN^{train}(z))) \quad (1.1)$$

Wobei  $NN^{train}(x)$  der nearest neighbor von  $x$  in den Trainingsdaten ist.

### 1.3.2 Support Vector Data Description

Idee. Konstruiere eine 'hyper-sphere' um die Trainingsdaten, welche *nahezu alle* Datenpunkte im Datenset umschliesst

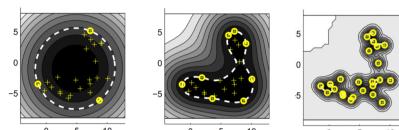


Abbildung 1.3: Support Vector Data Description

## 1.4 Principal Component Analysis

⇒ Mit PCA verfolgen wir das Ziel eine reduzierte Form der Daten zu finden, welche die Daten möglichst gut erklären

- Es sind  $n$  Samples gegeben (werden auch *observations* genannt)
- Für jedes sample haben wir  $m$  Messungen (werden auch *Dimensionen, Werte oder Variable* genannt)
- Dabei ist jede Messung eine reale Nummer

### 1.4.1 PCA Plots

#### Score Value

Der Score-Wert für eine Beobachtung, ist der Abstand vom Ursprung entlang der Richtung (loading vector) der ersten Komponenten bis zu dem Punkt, an dem diese Beobachtung auf den Richtungsvektor projiziert. Für jede Beobachtung (Zeile) im Datensatz gibt es einen Score-Wert, so dass es  $n$  Score-Werte für die erste Komponenten, weitere  $n$  Score-Werte für die zweite etc. gibt

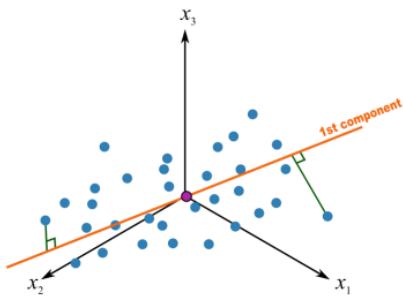


Abbildung 1.4: Beispiel eines ScoreValues

#### Score Plot

In einem Score-Plot stellt man zwei Komponenten gegenüber um diese in einer zweidimensionalen Grafik darzustellen.

#### Interpretation:

- Ähnliche Samples sind nahe beieinander
- Durchschnittliche Samples werden nahe beim Ursprung liegen
- Outliers werden weit ausserhalb sein

#### Loading

- Loading misst die Korrelation zwischen einer Komponenten und einer Variable
- Die Loading-Range ist von -1 bis 1
- Nahe bei -1 oder 1 ⇒ Variable hat einen sehr starken Einfluss auf den Komponenten (+ oder - hat hier keine grosse Bedeutung)
- Nahe bei 0 ⇒ Variable hat einen schwachen Einfluss auf die Komponenten
- Quadratische Belastungen geben den Anteil der Varianz der Variablen an, der durch die Komponenten erklärt wird

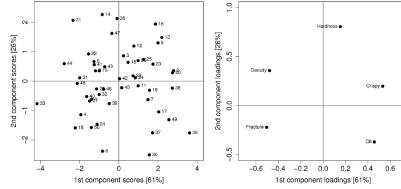


Abbildung 1.5: Kombination eines Scores Plot mit einem Loading Plot

### 1.4.2 Preprocessing for PCA

- Entferne Outliers:** bspw. wenn ein Sensor nicht funktioniert und offensichtlich falsche Resultate liefert
- Skalierung:** Stelle sicher, dass alle Achsen in etwa die gleichen Werte haben
- Centering:** Daten an den Ursprung des Koordinatensystems verschieben, indem der Mittelwert der Daten subtrahiert wird

#### Skalierung

**Problem:** Wenn Variablen sehr unterschiedliche Werte haben (z.B. Gewichte zwischen 70-100kg vs. Höhe zwischen 1.6-2.0m), hängt ihr Einfluss auf die PCA von diesen Werten ab

**Ziel:** Alle Variablen (auf der X-Achse) haben in etwa die gleiche Grösse

**Lösung:** Die am häufigsten verwendete Methode ist die Unit Variance Scaling , bei der jede Spalte durch ihre Standardabweichung geteilt wird. Dann wird jede Spalte eine Varianz von 1 haben.

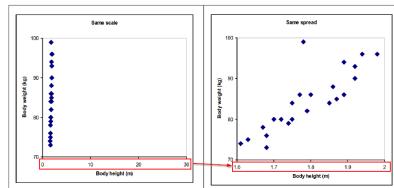


Abbildung 1.6: Unterschied nach der Skalierung

#### Centering

**Ziel:** Das Koordinatensystem vom Ursprung zu einem neuen Referenzpunkt verschieben, so dass die Beobachtungen um den neuen Punkt herum verstreut sind

**Lösung:** den Mittelwert aller Datenpunkte von jedem Datenpunkt subtrahieren

**Alternativ:** Mediane Zentrierung, da diese robuster gegenüber Ausreisern ist

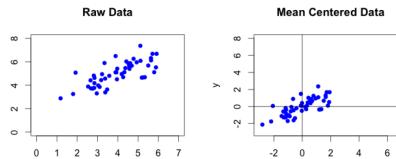


Abbildung 1.7: Unterschied der beiden Möglichkeiten nach der Zentrierung

### 1.4.3 Auswahl der Principal Components

- Daumenregel:** 2-5 Principal Components sind üblicherweise genügend
- Kaiser's Rule:** Wähle alle Components wo der Eigenvalue grösser als 1 ist
- Prozentsatz der Cumulative Varianz** siehe Unterkapitel
- scree Plot:** Nehmen Sie die Komponenten, die sich vor dem Abflachen der Neigung befinden  $\Rightarrow$  Ellbow

## Cumulative Varianz

Die kumulative Varianz ( $k$ ) ist die Varianz, welche durch den ersten  $k$  PC erklärt wird *Berechnung*:

$$\sum_{i=1}^k \frac{\lambda_i}{\sum_{j=1}^m \lambda_j} \quad (1.2)$$

## 1.5 Keyphrase Extraction

Die Hauptaufgabe ist *Finde möglichst gute Schlüsselwörter für ein Text-Dokument*, diese Aufgabe kann man in unterschiedlichen Varianten lösen:

- Keyphrase Extraction: nur einzelne Wörter aus dem Text
- Keyphrase Extractino: ein oder mehrere angrenzende Wörter aus dem Text
- Main Topic Identification: Erlaubt sogar Begriffe, welche nicht im Text stehen

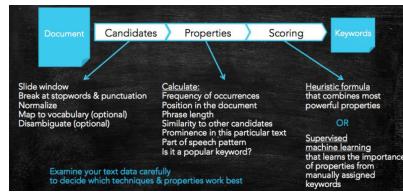


Abbildung 1.8: Hauptprinzip der Keyphrase Extraction

### Rapid Automatic Keyword Extraction (RAKE)

RAKE ist ein Algorithmus um Schlüsselwörter in einem Text zu identifzieren.

- Kandidat, wenn Wort zwischen zwei Stopwörter oder Satzzeichen stehen
- Parameters: 1. Stopword list; 2. Minimale Wortlänge (bspw. 4 char); 3. maximale Phrase-Länge (bspw. 3 Wörter)
- Output ⇒ eine Liste mit Phrasen, sortiert nach dem Phrase-Score

**Scoring: Durchschnittswerte:**

- $\text{degree}(\text{phrase}) \rightarrow$  Anzahl Wörter im Satz
- $\text{frequency}(\text{word}) \rightarrow$  Anzahl der Wort-Häufigkeit im gesamten Text
- $\text{degree}(\text{word}) \rightarrow$  Summe von  $\text{degree}(\text{phrase})$  über alle Sätze wo das entsprechende Wort beinhaltet
- $\text{score}(\text{word})$

**RAKE Score:**

- $\text{score}(\text{phrase}) \rightarrow$  Summe von allen  $\text{score}(\text{word})$  für jedes Wort im Satz

## 1.6 TF-IDF

Die Problematik ist, dass Wörter die sehr häufig vorkommen, nicht zwingend sehr relevant sind.

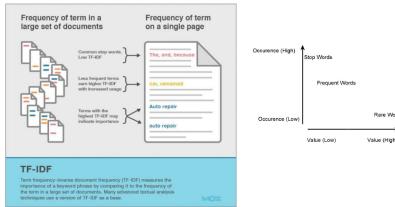


Abbildung 1.9: Häufigkeit der Wörter in Bezug auf ihre Relevanz

Dabei kann TF-IDF (Term Frequency - Inverse Document Frequency) abhelfen.

$$w_{x,y} = tf_{x,y} * \log\left(\frac{N}{df_x}\right) \quad (1.3)$$

Dies beschreibt wie häufig der Begriff  $x$  im Dokument  $y$  vorkommt.

$tf_{x,y}$  = Häufigkeit von  $x$  in  $y$

$df_x$  = Anzahl von Dokumenten die  $x$  enthalten

$N$  = Total Anzahl Dokumente

⇒ Daraus resultiert dann eine TF-IDF-Liste, welche bereits den Korrekturfaktor miteinberechnet.

DC-9 WITH 55 ABOARD CRASHES; AT LEAST 16 DEAD			
CHARLOTTE, NC (Reuter)			
A USAir DC-9 with 55 people on board crashed and burst into flames during a thunderstorm after missing an approach to Charlotte's international airport Saturday, killing at least 16 people. The flight, which originated in Columbia, South Carolina and was on its final approach, hit a house near the airport runway and caught fire, said Jerry Orr, aviation director at Charlotte Douglas International Airport. Orr said 16 people were dead, six were missing and presumed dead and 33 were taken to local hospitals. USAir reported 18 dead. Rescue teams fought to save the injured who were victims of the crash, which split into three sections on impact at about 6:50 p.m. EDT as the plane was trying to land at Charlotte during heavy storms.			
<b>top 15 terms ranked by</b>			
frequency	highest idf	$f * idf$	
32 the	1.00	0.000077	3.28 orr
16 were	1.00	0.000077	2.81 charlotte
14 said	0.93	0.000077	2.61 Payne
12 and	0.93	0.000077	2.48 dc
12 to	0.86	0.000077	2.24 usair
11 on	0.80	0.000077	2.09 plane
9 of	0.70	0.000016	1.93 crash
9 at	0.76	0.000077	1.74 bones
9 was	0.75	0.000077	1.61 survivors
7 in	0.73	0.000077	1.50 dripping
6 on	0.72	0.0149	1.49 wreckage
6 they	0.69	0.000077	1.36 dead
6 dripping	0.66	0.000077	1.29 hospitals
6 had	0.66	0.000077	1.29 airport
6 plane	0.66	0.000077	1.21 ss
...			

Abbildung 1.10: Beispiel einer TF-IDF-Liste

## 1.7 Topic Modeling (LDA)

Das Hauptziel ist, Dokumente im Kontext eines grossen Korpus (Textbestandes) zu erforschen.

'Verstehe' ein einziges Dokument → Finde 'ähnliche' Dokumente → Liefere besseres Suchresultat

### 1.7.1 Was ist ein Topic?

- Eine Gruppe von Wörtern, bei denen es eine sehr hohe Wahrscheinlichkeit gibt, dass sie im gleichen Kontext erscheinen
- Syntax wird nicht betrachtet
- Kann nur auf Nomen eingeschränkt werden, ist aber nicht notwendig
- Eine verborgene Struktur, mit deren Hilfe bestimmt werden kann, welche Wörter wahrscheinlich in einem Text vorkommen

### 1.7.2 Anwendungen

- Bessere Suchresultate → Suche nach 'genome' und liefere Dokumente die 'DNA', 'genes' oder 'RNA' zurück-liefert (mehr wie nur Synonyme)
- Erhalte einen Überblick über grosse Dokumentenmengen

- Ein einzelnes Dokument mit einer riesigen Sammlung von Dokumenten in Zusammenhang bringen

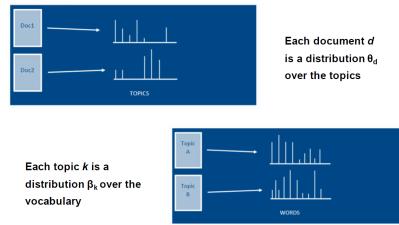


Abbildung 1.11: Probabilistic Topic Model

### 1.7.3 Probabilistic Generative Process

Grundidee:

- **Gegeben** eines festen Vokabulars und einer Reihe von  $K$ -Themen. Jedes Thema ist eine Wahrscheinlichkeitsverteilung über das Vokabular
- Um ein Dokument zu **erstellen**, wählt man zunächst die Textlänge, die Anzahl der Wörter und eine Verteilung auf die Themen (z.B. Thema1 60-Prozent, Thema2 40-Prozent)
- Für jedes Wort im Dokument **wählt man zufällig** das Thema entsprechend der Themenverteilung (z.B. Thema1), dann **wählt man zufällig** ein Wort aus dem Vokabular entsprechend der Themenverteilung im Vokabular (z.B. Affe)

⇒ Es kann verwendet werden um (theoretisch) Texte zu erstellen anhand der Topic-Verteilung, dies war für die LDA fundamental wurde in der Realität jedoch nie verwendet.

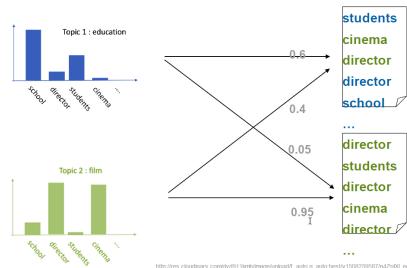


Abbildung 1.12: Ein Beispiel für ein Generative Process

### 1.7.4 LDA

LDA = Latent Dirichlet Allocation, mit dem Hauptziel *wir beobachten nur die Dokumente und ihre Worte, und unser Ziel ist es, auf die zugrunde liegende Themenstruktur zu schließen*

- **Latent** → Die zugrunde liegende Struktur ist unbekannt
- **Dirichlet** → sind die angenommenen Wahrscheinlichkeitsverteilungen des Themas und der Wörter
- **Allocation** → weil es die wahrscheinlichste Lösung auf der Grundlage der Daten zuweist

#### Grundidee des LDA-Algorithmus

Dabei handelt es sich um ein Expectation-Maximization-Algorithmus.

Gegeben sei ein Set  $D$  von Dokumenten.

**Ablauf:**

1. Fix the number of topics  $K$
2. Randomly initialize estimates of the hidden parameters  $\theta_d$  and  $\beta_k$
3. Repeat → 1. Use estimated probabilities to assign topics to each document → 2. Use estimated probabilities and topic assignments to assign topics to each single word → 3. Update estimates based on the topic-to-word assignments
4.  $\Rightarrow$  Bis Prozess konvergiert

### **anwenden von LDA**

1. Entferne irrelevanten Text
2. Sammle alle Wörter und sortiere dies nach der Frequenz
3. Reduziere die Wortliste (bspw. wenn ein Wort mehr als in 10
4. Lemmatization or stemming
5. Behalte die Top-Wörter (bspw. 50000 - 100000)
6. Verwende bag-of-words oder TF-IDF als input für LDA
7. Variiere mit den Hyperparameters von LDA (bspw. Anzahl Topics oder Dirichlet parameter)
8. manuelle Qualitätsanalyse der Themen und Themenzuweisung

# Kapitel 2

## Supervised Learning

### 2.1 Linear Regression

Die Linear Regression wird verwendet um eine Beziehung zwischen zwei fortlaufenden Variablen aufzuzeigen.  
Beispiele:

- Grösse und Gewicht → Wenn man grösser ist, erwartet man, dass man auch schwerer ist
- Alkoholkonsum und Alkoholgehalt im Blut
- Wie schnell man fährt und Benzinverbrauch

⇒ Wenn es keine Beziehung zwischen den Variablen gibt oder es sich um eine deterministische Beziehung handelt, ist dies nicht geeignete für lineare Regression

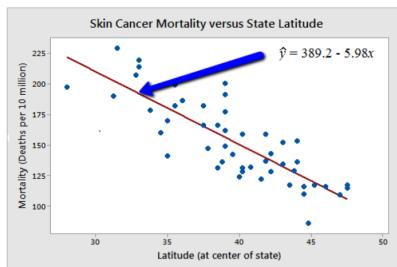


Abbildung 2.1: Ein Beispiel für die lineare Regression

#### 2.1.1 Bivariate vs. Multivariate Modelle

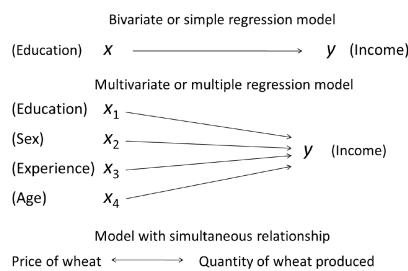


Abbildung 2.2: Ein Beispiel für den Unterschied von Bivariate und Multivariate Modelle

## 2.1.2 Simple Linear Regression (SLR)

Das Hauptziel von SLR ist die Parameter  $a$  und  $b$  der linearen Funktion zu finden, sodass sich der *Mean Squared Error (MSE)* minimiert

- **Variable X-Achse:** Wird als *explanatory* oder *predictor* oder *unabhängige Variable* bezeichnet
- **Variable Y-Achse:** Wird als *response* oder *criterion* oder *abhängige Variable* bezeichnet

**Model:**

$$y^i = ax^i + b\epsilon^i \quad (2.1)$$

- $i$  sind die Beispiele / Samples
- $a$  ist die Steigung der linearen Regression
- $b$  wird als *y-interceptor* betitelt
- $\epsilon^i$  ist ein zufälliger Fehler (anhand der Normalverteilung)

### Mean Squared Error

Für gegebenes  $a$  und  $b$ , sei  $\hat{y}^{(i)} = ax^{(i)} + b$  der vorhergesagte Wert

$$MSE = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 \quad (2.2)$$

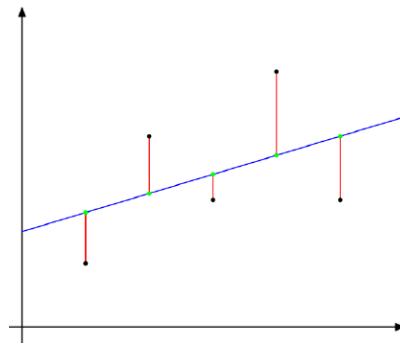


Abbildung 2.3: Ein Beispiel für den Unterschied von Bivariate und Multivariate Modelle

## Korrelation

Der *Pearson Correlation Coefficient* misst die Stärke der Beziehung zwischen den zwei quantitativen Variablen.

$$r = \frac{1}{m-1} \sum_{i=1}^m \left( \frac{x^{(i)} - \bar{x}}{s_x} \right) \left( \frac{y^{(i)} - \bar{y}}{s_y} \right) \quad (2.3)$$

Wobei

- $\bar{x}$  und  $\bar{y}$  die Stichprobenmittelwerte der x- und y-Variablen sind
- $s_x$  und  $s_y$  die Standardabweichung von x und y sind

## Berechnung von SLR

Die Variablen a und b minimieren den Mean Square Error

$$a = \frac{\sum_{i=1}^m x^{(i)} y^{(i)} - \bar{y} m \bar{x}}{\sum_{i=1}^m x^{(i)2} - \bar{x} \sum_{i=1}^m x^{(i)}} = r \frac{s_y}{s_x}; b = \bar{y} - a \bar{x} \quad (2.4)$$

**Idee:**

- Berechnen Sie partielle Ableitungen von MSE in Richtung a und b
- Setzen Sie diese auf Null, verwenden Sie Kalkül, um das Gleichungssystem zu lösen
- Zweite Gleichheit für a: verwendet die Standardabweichung s<sub>x</sub> und s<sub>y</sub> und den Pearson-Korrelationskoeffizienten r zwischen x und y siehe Definition auf dem nächsten Bild )

## Grundlegende Annahmen in SLR

1. Ein lineares Modell passt zu dem Problem
2. Die Beobachtungen sind voneinander unabhängig
3. Alle notwendigen unabhängigen Variablen sind im Modell enthalten
4. Die Fehlerbegriffe  $\epsilon_i$  are normal verbreitet
5. Fehler zwischen verschiedenen Beobachtungen haben eine konstante Varianz Homoskedastizität

## Residual

Das Residuendiagramm ist ein Streudiagramm von vorhergesagten Werten vs. Residuen.

Das Residual für die Beobachtung  $i$  ist gegeben durch  $e^{(1)} = y^{(i)} - \hat{y}^{(i)}$

## Multivariate Linear Regression

Zuvor haben wir uns mit der Formel  $y = ax + b$  bekannt gemacht.

Allgemeiner ausgedruckt wird die Formel wie folgt:  $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

- n steht für die Anzahl von Variablen (=measurements)
- m steht für die Anzahl von Samples (hier nicht aufgeführt)
- $h_\theta$  ist die Hypothese mit den Parametern  $\theta_0$  etc.

## Kostenfunktion

**Ziel:** Wir wollen  $\theta$  so wählen, dass wir  $J(\theta)$  minimieren  $\rightarrow$  möglichst wenig Kosten generieren.

**Idee:**

1. Initialisiere  $\theta$  (bspw. zufällig oder durch raten)
2. Wiederholend  $\theta$  anpassen um  $J(\theta)$  zu verkleinern
3. Durchführen bis Konvergenz erreicht wurde

## Formel:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \quad (2.5)$$

## 2.2 Gradient Descent

**Idee:** Verwendung von Gradienten von  $\theta$  um seinen Wert bei jedem Schritt zu adaptieren

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (2.6)$$

Wobei hier  $\alpha$  die Learning-rate ist. Die Learning rate ist durchaus wichtig im gesamten Prozess. Ist sie zu klein, so konvergiert es nur sehr langsam. Ist sie zu gross, so überschiesst sie allenfalls das Ziel und findet entsprechend die eigentliche Lösung nicht.

### 2.2.1 High-Order Linear Regression

Ein weiteres Problem haben wir, wenn wir nicht linearen Daten uns annähern wollen.

**Vorgehen:** Man nimmt den eindimensionalen input-wert  $z$  und projiziert diesen auf eine höhere Dimension, so dass die lineare Funktion auch kompliziertere Beziehungen zwischen  $x$  und  $y$  anpassen kann. **Beispiel:**

- Der Input ist  $z$  (ein-dimensional)
- Definiere eine Transformation  $x_k := f_k(z) = z^{2*k}$  für  $k = 1, 2, 3, 4$
- lineares Modell:  $h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 = \theta_0 + \theta_1 z^2 + \theta_2 z^4 + \theta_3 z^6 + \theta_4 z^8$
- $\Rightarrow$  Das Modell ist polynomial in  $z$ , aber linear in  $x$

## 2.3 Logistic Regression

Die Logistic Regression löst einen Classification Task.

Gegeben:

- Given are  $m$  training examples  $(x^{(i)}, y^{(i)})$
- Jedes  $x^{(i)}$  ist ein  $n$ -dimensionaler Feature-vector
- $y^{(i)} \in \{0, 1\}$

## 2.4 Support Vector Machines

**Ziel:** Eine Hyperebene finden, die die beiden Klassen von Eingabepunkten mit maximalem Spielraum trennt

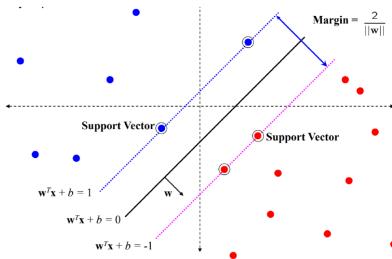


Abbildung 2.4: Support Vector Machine

### 2.4.1 Regulierung

- Was ist der beste Abstand?  $\rightarrow$  Meisten gibt es einen Kompromiss zwischen dem Abstand und der Anzahl Fehler auf den Trainingsdaten
- Soft-Margin  $\rightarrow$  evtl genauer anschauen

## 2.5 Corpus Construction

### Inter-Annotator Agreement

Misst inwiefern sich zwei Annotatoren einigen sind bezg. dem Data-Label:

$$K = \frac{n_{shared} - n_{random}}{1 - n_{random}} \quad (2.7)$$

- grösser als 0.7 ist ein guter Wert, kleiner als 0.4 ist ein schlechter Wert

## 2.6 Word Embeddings

### 2.7 Algorithm Selection

#### 2.7.1 Word Vectors

- Jeder Wort wird durch einen Vektor repräsentiert
- die Repräsentation wird durch den gegebenen Text und das Wort selbst, berechnet

#### Bag of Words

**Bag of Words** ist eine traditionelle Form wie man die Wörter repäsentieren kann.

- Jedes Wort hat eine ID, die Position im Wörterbuch von allen Wörtern
- Vektorlänge ist gleich der Anzahl Wörter im Wörterbuch
- Trägt eine 1 an, wenn es sich um das Wort handelt, sonst 0

$$\begin{aligned} & \text{call calm car cat} \\ V(\text{car}) &= [\dots, 0, 0, 1, 0, 0, \dots] \\ V(\text{cat}) &= [\dots, 0, 0, 0, 1, 0, \dots] \end{aligned}$$

Abbildung 2.5: Beispiel einer Bag of Words Repräsentation

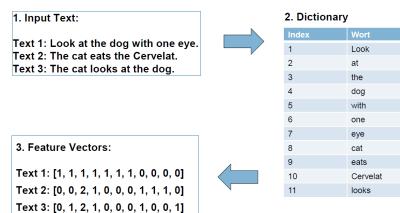


Abbildung 2.6: Ablauf einer Bag of Words Repräsentation

#### Neural Word Embeddings

Einfache String als Input funktioniert wirklich gut, daher muss man sich für Text-Analyse etwas mehr gedanken machen.

- Multidimensionale Repräsentationen von Wörtern
- 50-300 Dimensionen sind häufig normal
- werden auf Basis eines neuronalen Netzwerkes berechnet
- Haben eine dichte Repräsentation (wenig Nullen)

- Encoden eine semantische Ähnlichkeit (bspw. Synonyme)
- Encoden Analogien (bspw. Distanz zwischen King und Queen ist identisch wie die Distanz zwischen Frau und Mann)

### 2.7.2 Welcher Algorithmus soll nun gewählt werden

Kriterien:

- Genauigkeit
- Trainingszeit
- Parameter

⇒ Es gibt nicht den einen perfekten Algorithmus, sondern es muss je nach Anwendungsfall unterschieden werden → No Free Lunch Theorem

### 2.7.3 Daten

Typischerweise splitet man sein Trainingsset in 70% Train, 20% Validation und 10% Test

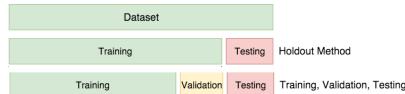


Abbildung 2.7: Aufsplittung der Daten

### 2.7.4 Cross-Validation

- Die Trainingsdaten werden in disjunkte Fold eingeteilt
- Trainiere auf einem Fold, evaluiere mit den anderen
- Durchschnittlicher Wert nach allen Experimenten
- Typische Varianten: *Leave One Out Cross Validation* oder *k-fold Cross Validation*

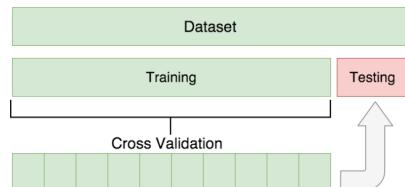


Abbildung 2.8: Schema Cross Validation

#### Leave One Out Cross Validation

- Jedes Trainingsbeispiel ist ein einzelner Fold
- Für jeden Fold F, wird das Modell auf allen (ausser F) trainiert und anschliessend auf F getestet
- Durchschnittswert gilt
- → Berechnung ist sehr teuer, nur mit kleinen Datensets verwenden

## k-fold Cross Validation

- Verwende ein zufälliges  $k$  an Subsets der Trainingsdaten
- bspw.  $k=10$

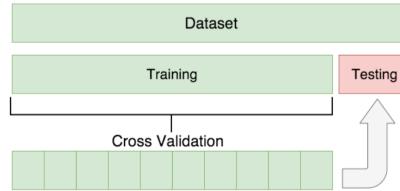


Abbildung 2.9: Schema k-Fold Cross Validation

## 2.8 Quality Measurements

### Genauigkeit

$$Accuracy = \frac{\text{correct Documents}}{\text{all Documents}} \quad (2.8)$$

### Confusion Matrix

		Actual Value		
		Positive	Negative	Neutral
Predicted Value	Positive	12	3	4
	Negative	2	8	1
	Neutral	17	20	143

Abbildung 2.10: Abbildung einer Confusion Matrix

Präzision für positive Dokumente:  $P_{pos}$  = true positives / (true positivies + false positives)

Recall für positive Dokumente:  $R_{pos}$  = true positives / all positive document

### 2.8.1 F-Score

A balanced combination of Precision and Recall

General F-Score:  $F_\beta = (1 + \beta^2) * \frac{\text{precision} * \text{recall}}{(\beta^2 * \text{precision}) + \text{recall}}$

F1 Scores uses  $\beta = 1$ :  $F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$

For sentiment analysis, the F1-Score of the resulting classes (positive and negative) is usually averaged:  $F = \frac{F_{pos} + F_{neg}}{2}$

## 2.9 Learning Curve Analysis

Bewerten Sie Tendenzen des Trainings und Testkurven für genügend Beispiele. **Grundregeln:**

- Wenn beide Kurven "nahe beieinander" liegen und beide eine niedrige Punktzahl haben -> potentielle **underfitting** (High Bias)
- Wenn die Trainingskurve ein viel besseres Ergebnis als die Testkurve hat -> mögliche overfitting (hohe Varianz)

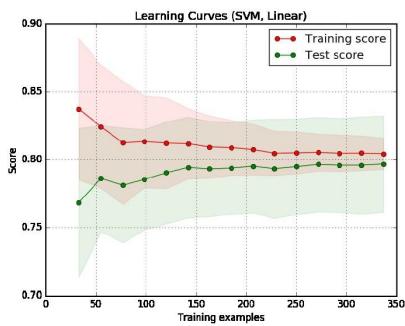


Abbildung 2.11: Abbildung einer Learning Curve

# Kapitel 3

## Neural Networks

### 3.1 Neural Networks in a Nutshell

Die ersten Ansätze von Neuronalen Netzwerk gehen auf die 40er Jahre zurück. Wobei man mehrere Inputs hatte, diese Gewichtet und verknüpft hat und anschliessend einen Output erhalten hat.

anschliessend kam *The Perceptron* welcher die Grundidee aufnahm adaptive (trainierbare) Gewichtungen miteinflussen zu lassen.

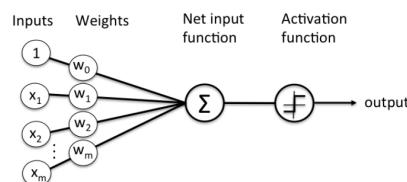


Abbildung 3.1: Abbildung vom Perceptron

Als Weiterentwicklung kam der Multilayer Perceptron in den 60er Jahren, war jedoch noch sehr ineffizient zum Trainieren

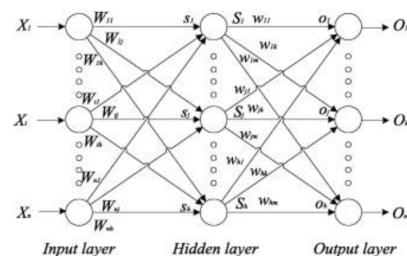


Abbildung 3.2: Abbildung vom Multi-Layer Perceptron

### 3.2 Perecptrons

#### Neuron aus biologischer Sicht

- Ein Neuron ist mit anderen Neuronen über ca. 10000 Synapsen verbunden
- Sobald der Input einen kritischen Wert überschreitet, entlädt das Neuron eine Spur eines elektrischen Impulses, der vom Körper durch das Axon hinunter zum nächsten Neuron (zu den nächsten Neuronen) wandert.
- Die Axonenden berühren fast die Dendriten oder den Zellkörper des nächsten Neurons.

#### Die technische Sicht - weshalb künstliche Intelligenz

- **Technischer Aspekt** → Einige Probleme wie die Zeichenerkennung oder die Vorhersage zukünftiger Zustände eines Systems erfordern eine massiv parallele und adaptive Verarbeitung.
- **Biologischer Aspekt** → ANNs können verwendet werden, um Komponenten des menschlichen (oder tierischen) Gehirns zu replizieren und zu simulieren, wodurch wir Einblick in die natürliche Informationsverarbeitung erhalten.



Abbildung 3.3: Abbildung zum Vergleich der Informatik zur Biologie

### 3.3 Backpropagation

Berechne die Gradienten Funktion und propagiere die Änderungen zurück zu den Gewichten, dadurch haben wir eine laufende Anpassung der Gewichtung

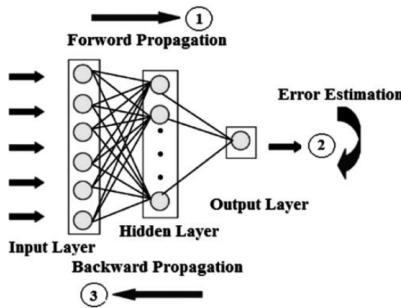


Abbildung 3.4: Abbildung des Backpropagation-Prozedere

Bei einem Multilayer Netzwerk berechnen wir für *for each node:* (1) *Compute weighted sum of inputs* (2) *Decide if node is activated*

#### Nachteile von Backpropagation

- Erfordert eine große Menge an beschrifteten Trainingsdaten
- Bei mehreren verborgenen Schichten ist die Lernzeit langsam
- Kann in lokalen Optima stecken bleiben
- Problem des verschwindenden Gradienten

### 3.4 Universality Theorem

Das Theorem sagt aus, dass *Ein neuronales Netz mit einer verborgenen Schicht und einer beliebigen Anzahl von Neuronen kann jede beliebige kontinuierliche Funktion approximieren*

*Idee für den Beweis:* Schneiden Sie eine gegebene Funktion  $g$  in eine ausreichende Menge kleiner Stücke, dann verwenden Sie 2 Neuronen, die eine partielle lineare Funktion modellieren, um jedes dieser Stücke zu approximieren

#### 3.4.1 Schlüsselerfolg: Differenzierbarkeit

⇒ Wenn alle Komponenten in einem neuronalen Netz differenzierbar sind, können wir den Gradienten der Leistungsfunktion effizient berechnen

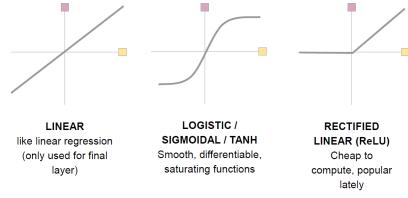


Abbildung 3.5: Abbildung der verschiedenen Aktivierungsfunktionen

**Wichtig!** ReLU ist bei 0 nicht differenzierbar, aber wir können einen Wert für die Ableitung 0 oder 1 definieren und dann damit arbeiten.

## 3.5 Feed-Forward Neural Networks

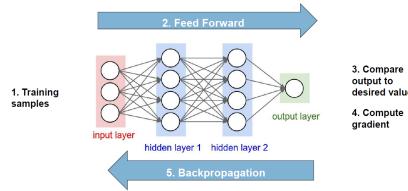


Abbildung 3.6: Abbildung des FeedForward-Prozedere

### 3.5.1 Vanishing Gradient Problem

### 3.5.2 Softmax

**Ziel:** Für den Klassifizierungstask wollen wir eindeutige Klassen, in welchen wir im Outpunkt Layer 1.0 für die korrekte Klasse und 0 für alle anderen Klassen erhalten

**Idee:** Ein *Max-Layer* setzt 1 für den maximalen Wert vom vorherigen Layer und 0 bei allen anderen Outputs

**Problem:** Dies ist nicht differenzierbar

**Lösung:** Eine Softmax-Funktion

**Bemerkungen:**

- skaliert alle Werte zwischen 0 und 1
- Die Summe von allen Werten ist 1, ähnlich wie bei einer Wahrscheinlichkeitsverteilung
- Der Maximumswert vom vorherigen Layer wird sehr gross sein im Vergleich zu den anderen Werten

$$\begin{array}{|c|} \hline 1.3 \\ \hline -1 \\ \hline 3.2 \\ \hline 0.5 \\ \hline \end{array} \rightarrow \sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}} \rightarrow \begin{array}{|c|} \hline 0.121 \\ \hline 0.012 \\ \hline 0.812 \\ \hline 0.055 \\ \hline \end{array}$$

Abbildung 3.7: Vor und nach dem Dropout

### 3.5.3 Cross-Entropy

#### Cross-Entropy Loss

**Entropy** Für eine Wahrscheinlichkeitsverteilung  $y = (y_1, \dots, y_r)$  ist die Entropy  $H(y) = \sum_{i=1}^r y_i \log(\frac{1}{y_i})$

#### Cross Entropy Loss for one Sample:

Für ein einziges Samples mit

*ground truth* Wahrscheinlichkeitsverteilung  $y = (y_1, \dots, y_r)$  und

*vorhersagenden* Wahrscheinlichkeitsverteilung  $\hat{y} = (\hat{y}_1, \dots, \hat{y}_r)$  ist die

**Cross Entropy Loss**  $H(y, \hat{y}) = \sum_{i=1}^r y_i \log \frac{1}{\hat{y}_i}$   
 ⇒ Die Cross Entropy ist assymetrisch

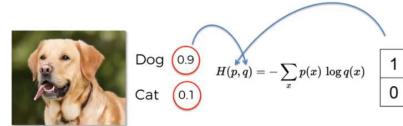


Abbildung 3.8: Abbildung Cross Entropy Loss

**Cross-Entropy Cost** Für ein set von  $n$  Sampels, bei dem für jedes gilt  
*ground truth Wahrscheinlichkeitsverteilung*  $y^{(k)}$  und  
*vorhersagenden Wahrscheinlichkeitsverteilung*  $\hat{y}^{(k)}$  ist die

**Cross Entropy Cost Funktion**  $H(\{y^{(k)}\}, \{\hat{y}^{(k)}\}) = \sum_{k=1}^n H(y^{(k)}, \hat{y}^{(k)})$

⇒ Daumenregel: Verwende Mean-Squared-Error (MSE) für die Regression und Cross-Entropy für Multiclass-Classification Probleme

### 3.5.4 Activation Functions

### 3.5.5 Dropout

**Ziel:** Vorbeugen von overfitting durch eine bessere Generalisierung

**Idee:** einen zufälligen Bruchteil von Knoten und entsprechenden Aktivierungen während des Trainings zu ignorieren (Nullung)

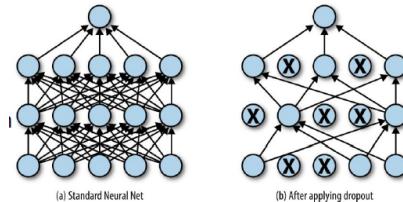


Abbildung 3.9: Vor und nach dem Dropout

## 3.6 Keras

## 3.7 Convolutional Neural Networks

Bei den CNNs vergleicht man nicht Pixel um Pixel, sondern man hat eine Art Fenster, bei welchem auch die benachbarten Pixels miteinbezogen werden. Danach berechnet man die Summe aller Pixel und schreibt diesen in die Feature-Map

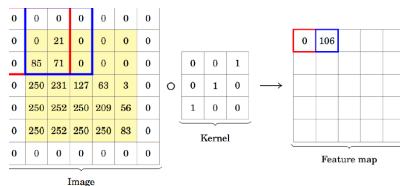


Abbildung 3.10: Abbildung eines CNNs

Man trainiert die Werte in einem Filter und verwendet hierzu eine *activationFunctions* für die Erstellung einer Features-Map. Dazu kann man eine oder mehrere Feature-Maps (mit unterschiedlichen Gewichtungen) verwenden. Der Standort der Invarianz wird verwendet um bspw. die Edge-Detection abzubilden.

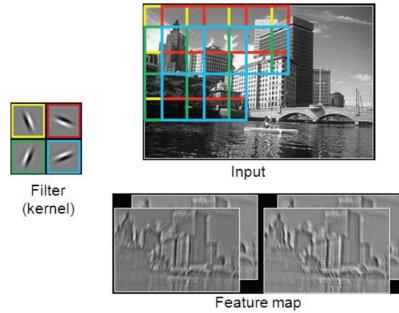


Abbildung 3.11: Abbildung eines CNNs mit Edge Detection

## Max Pooling

Man reduziert die Bildgrösste durch ein gleitendes Fenster (Sliding Window) ohne Überlappung. Dabei behält man jeweils den maxValue der Pixel. Dies wird auch *sub-sampling* genannt.

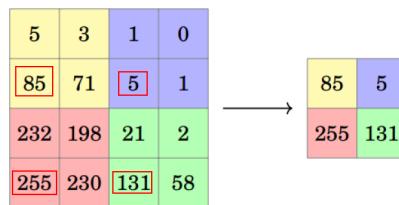


Abbildung 3.12: Abbildung eines CNNs mittels Max Pooling

### 3.7.1 CNN für Textanwendungen

#### Word Embeddings

Einfache String als Input funktioniert wirklich gut, daher muss man sich für Text-Analyse etwas mehr gedanken machen.

- Multidimensionale Repräsentationen von Wörter
- 50-300 Dimensionen sind häufig normal
- werden auf Basis eines neuronalen Netzwerkes berechnet
- Haben eine dichte Repräsentation (wenig Nullen)
- Encoden eine semantische Ähnlichkeit (bspw. Synonyme)
- Encoden Analogien (bspw. Distanz zwischen King und Queen ist identisch wie die Distanz zwischen Frau und Mann)

### 3.7.2 CNN für Textanalyse

Gegeben 1 Text und die Word Embeddings

Für jedes Wort haben wir einen Vektor, danach erfolgt ein 1D-Convolution bzw. 2D Convolution  $\Rightarrow$  Man erhält einen Vektor und nimmt dabei das Maximum  $\Rightarrow$  Mehr als 2 Layer bei einem CNN ist nicht sinnvoll bzw. bringt keinen zusätzlichen Wert

## 3.8 Recurrent Neural Networks (RNNs)

RNN's haben eine Art Gedächtnis, wo sie speicher, was sie früher berechnet haben.

- Man verwendet die sequenziellen Informationen
- Man führt den Task für jedes Element einer Sequenz durch (*Recurrent* in RNN)
- Der Output hängt von den vorherigen Berechnungen ab

### 3.8.1 Beispiel Text-Character by Character

- **Goal** Lerne ein Sprachmodell, welches die Wahrscheinlichkeit von jedem Character vorhersagt, gegeben des vorherigen Characters
- Beispiel am Wort *hello*
- Um ein Text zu erstellen, starte bei einem zufälligen Character und lass das Netzwerk den nächsten Charakter vorhersagen, dies ist dann der Input für den nächsten Schritt im Netzwerk (RNN)

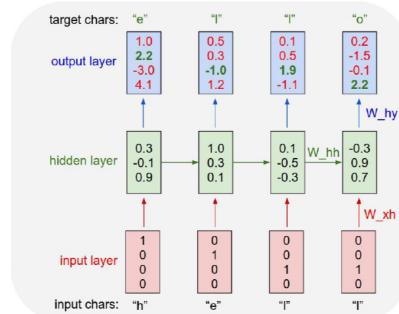


Abbildung 3.13: Beispiel eines RNN mit dem Wort Hello

### 3.8.2 Beispiel Sentiment Analyse

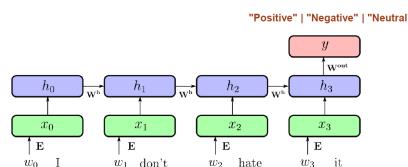


Abbildung 3.14: Beispiel eines RNN für die Sentiment Analyse

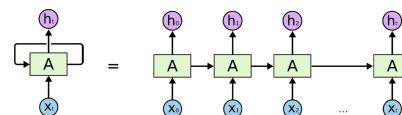


Abbildung 3.15: Abbildung der Struktur eines RNNs

- $x_t$  ist der Input zum Zeitpunkt  $t$ ,  $h_t$  ist der Output zu diesem Zeitpunkt
- ist das Model, welches repetiert wird. In jedem A wird das gleiche Neural Network verwendet.

### 3.8.3 Langzeit Abhangigkeit

Dies beschreibt die Abhangigkeit in einem langen Satz. Wenn wir innerhalb eines Satzes, das nachste Wort vorhersagen mochten.

Ein Problem dabei ist, dass das Gedachtnis sehr "kurzfristig" ist. Jeder Wert, der in einem Zeitschritt ausgegeben wird, wird im nachsten Schritt eingegeben, aber wenn derselbe Wert nicht wieder ausgegeben wird, geht er beim nachsten Schritt verloren.

⇒ Theoretisch ware dies moglich, in der Praxis ist dies aber nicht moglich!

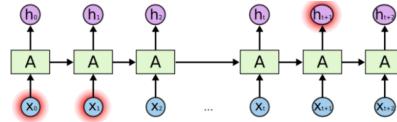


Abbildung 3.16: Abbildung der Struktur eines longterm RNNs

## 3.9 RNNs with Long Short Term Memory Networks (LSTM)

- Spezielle Form von RNNs
- Explizit fur das Problem der langzeit-Abhangigkeit entwickelt
- erinnert sich an Informationen fr eine lange Zeit
- Lost das Vanishing Gradient Problem von RNNs
- RNNs mit einem Speicher und einem *Gating Mechanismum* fr Input, forget, output
- funktioniert fr viele Anwendungsfalle und ist in weitem Gebraucht
- Konnen tausende von Zeitschritten unterhalten
- gibt verschiedene Variante (Peephole Connections, Gated Recurrent Units (GRU), Bi-directional LSTMs etc.)

### 3.9.1 Aufbau von LSTM

#### Notation



Abbildung 3.17: Notation eines LSTM

#### Kernidee

- Schlssel fr den Erfolg ist der Cell State  $C_t$ , welches einen Vektor darstellt
- LSTM kann Informationen diesem Cell State hinzufugen oder loschen
- Dies geschieht uber die regulierte Struktur der Gates
- LSTM haben drei Gates fr den Schutz und Kontrolle des Cell State

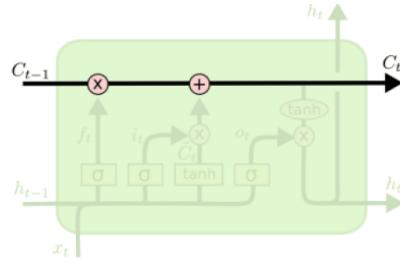


Abbildung 3.18: Abbildung des Cell State in LSTM

### Forget Gate

- Das Forget-Gate entscheidet welche Information verworfen werden
- Schaut bei  $h_{t-1}$  und  $x_t$  und generiert einen Output zwischen 0 und 1
- 1 repräsentiert, dass die Information behalten werden soll
- 0 repräsentiert, dass die Information verworfen werden kann
- Falls  $f_t$  0 ist an einem Eingang, dann wird der Eingang von  $C_{t-1}$  vollständig gelöscht

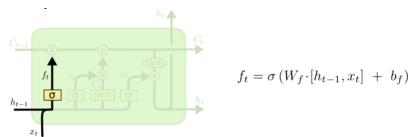


Abbildung 3.19: Abbildung des Forget Gates in LSTM

### Input Gate

- das Input-Gate entscheidet, welche neuen Informationen im Cell State gespeichert werden sollen
- 1. Part: Ein sigmoid-Layer (Input gate layer): Entscheidet welcher Wert  $i_t$  wir update sollen
- 2. Part: ein tanh-Layer: Erstellt einen Vektor, welcher ein neuer candidate value ist  $\rightarrow \tilde{C}_t$

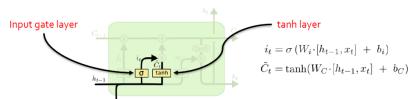


Abbildung 3.20: Abbildung des Input Gates in LSTM

### Update Cell State

- updatet der Alte-Status von  $C_{t-1}$  in den neuen Cell state  $C_t$
- 1. Operation - Multiply: Multipliziert den alten Status von  $f_t$  (Vergisst die Dinge, welche mir früher entschieden haben zu vergessen)
- 2. Operation - Add: addiert  $i_t * \tilde{C}_t$

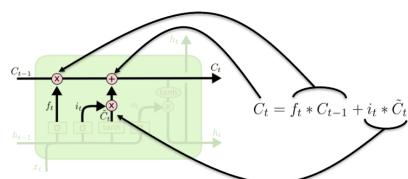


Abbildung 3.21: Abbildung des Update Gates in LSTM

## Output Generation

- Das Output-Gate entscheidet was wir als Output liefern
- 1. Step: Wir lassen den Sigmoid-Layer laufen, welcher entscheidet welche Abschnitte des Cell-States als Output ( $o_t$ ) geliefert werden sollen
- 2. Step: Wir lassen den Cell-State durch den tanh-Layer laufen und multiplizieren diesen mit dem Output des sigmoid-Layer
- 3. Step: Output and Cell-State are passed on as new input for the next step

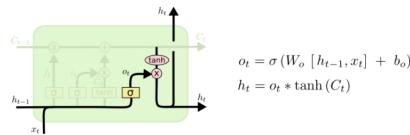


Abbildung 3.22: Abbildung der Output Generatino in LSTM

### 3.9.2 RNN vs. LSTM

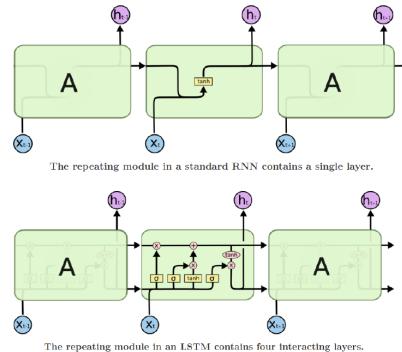


Abbildung 3.23: Strukturvergleich zwischen RNN und LSTM

## 3.10 Neural Network Architecture Quiz

### 3.11 Dialogue Systems

**Definition Conversational Dialogue Systems (Chatbots) :=** Allow users to chat in a natural way about arbitrary topics

#### 3.11.1 Recap Generative Adversarial Networks (GANs)

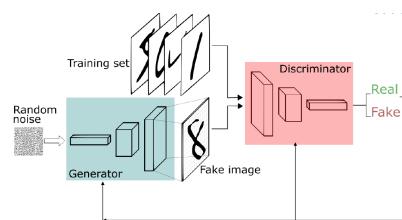


Abbildung 3.24: Abbildung eines GANs

### 3.11.2 Stereotypes of Dialogue Systems

- Question-Answering (bspw. Who was the director of Titanic?)
- Pro-Active Dialogue Systems (bspw. Hast du Fieber?, ..., Diagnose)
- Conversational Agents (Chatbot) (bspw. Was ist dein Name, etc.)

### 3.11.3 Chatbots

Hauptgründe wieso Chatbots fehlschlagen

- Vereinfacht nicht den Task
- Keine Interaktion mit anderen Systemen
- Schwacher Use-Case
- Zu ambitioniert

### 3.11.4 Komponenten eines Dialog Systems

#### Speech to Text

- Intent Classification
- Entity Recognition
- Context Tracking

#### Text to Speech

- Answer Selection
- Slot-Filling
- Natural Language Generation

Betrachtet man die Entwicklung eines Conversational Systems, dann kann man folgende Daumenregel anwenden:

- Training Data → Millionen von real-world Dialoge
- Technologie → Deep Neural Network ('Sequence2Sequence')
- Training Time → einige Wochen

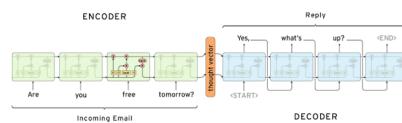


Abbildung 3.25: Abbildung des Vorgehens eines Conversational Systems

# Kapitel 4

## Reinforcement Learning

→ Ganz andere Denkweise zu Deep Learning

⇒ Wählt diesen Status aus, welchen den Reward in der GESAMTHEIT zu maximieren

Aus Erfahrungen mit der Interaktion der Umwelt, lernt wie man Probleme lösen kann → Das Ziel ist mit möglichst wenig Daten bereits ein sehr mächtiges Model erstellen können

Analogie zum Velo fahren lernen: Man kann es nicht durch Youtube-Videos oder Erklärungen erlernen. Sondern es ist ein Trial-Error-Verfahren. Dabei sind die viele verschiedenen Faktoren relevant.

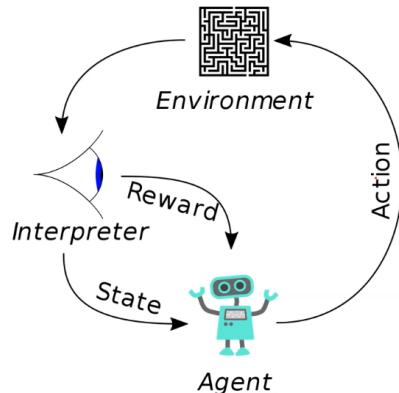


Abbildung 4.1: Abbildung zum Ablauf von Reinforcement Learning

Das Model erhält durch die Handlungen positive bzw. negative Rewards. Wenn es ein sehr negativer Reward ist, dann strömt der Spieler. Dadurch lernt das System von mal zu mal immer besser wie das Spiel funktioniert.

Bei den Atari Spielen war der Durchbruch, dass der Computer keine Informationen zu den Spielen erhalten hat, sondern sich alles selbstständig beigebracht hat - durch Beobachtung der Umwelt.

### 4.1 Schritte für das Erstellen von Reinforcement Systeme

1. Formuliere das Problem: goal, environment, states, actions, rewards
2. Bestimme das environment: real-world oder simuliert
3. Wähle oder implementiere den Trainings-Algorithmus und die value-function → Erhält nur die notwendigsten Informationen Bspw. Goal und environment, aber sicherlich nicht die rewards
4. Episodenweise Durchführung → Erkunden des environment um das Modell zu trainieren

#### 4.1.1 Goal

Gegeben dem **aktuellen Status**, eine **Aktion zu wählen** um in Zukunft alle **totalen Rewards zu maximieren** → Manchmal macht es Sinn einen Umweg zu gehen und auf gewisse Rewards zu verzichten, um in Zukunft die Rewards

zu maximieren

## 4.2 Reinforcement Agent

Was steuert das Verhalten meines Agenten?

1. policy  $\pi \rightarrow$  Algorithmus  $\Rightarrow$  Ist das grundsätzliche Ziel, jedoch hilft hierzu das Model bzw. value-Function
2. value-Funktion  $v$  oder  $q \rightarrow$  Wie gut ist der aktuelle state bzw. Action  $\Rightarrow$  Mit einer guten value-Funktion gibt es eine gute policy
3. Model  $\rightarrow$  Agenten-Repräsentation des Environments  $\Rightarrow$  Gibt den Wert von bspw. jedem Feld und gibt eine gute Wert-Funktion

## 4.3 Markov Decision Processes

A Markov Decision Process is a tuple  $\langle S, A, P, R, \gamma \rangle$

- $S$  is a finite set of states
- $A$  is a finite set of actions
- $P$  is a state transition probability matrix,  
 $P_{ss'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $R$  is a reward function,  $R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\gamma$  is a discount factor  $\gamma \in [0, 1]$ .

Abbildung 4.2: Abbildung zur Beschreibung zu Markov Decision Process

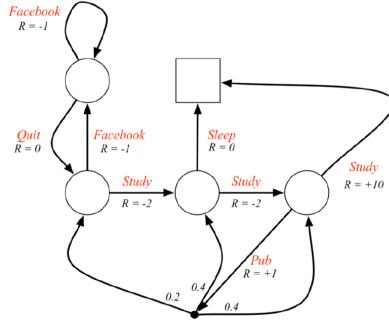


Abbildung 4.3: Abbildung zu einem beispielhaften MDP mit dem Verhalten von Studierenden

### 4.3.1 policy

die Policy  $\pi$  ist eine Verteilung über die Aktionen von gegebenen States.

- Eine Policy definiert das Verhalten eines Agenten vollständig
- MDP-Policies hängen vom aktuellen State ab (nicht von der History)

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \quad (4.1)$$

### Transition Probability and Rewards under a policy

Gegeben einer Policy  $\pi$ , können wir die Übergangswahrscheinlichkeit und den Reward berechnen

$$\begin{aligned} P_{s,s'}^\pi &= \sum_{a \in A} \pi(a|s) P_{ss'}^a \\ R_s^\pi &= \sum_{a \in A} \pi(a|s) R_s^a \end{aligned} \quad (4.2)$$

### 4.3.2 Return

**return**  $G_t$  ist der discounted Reward für den Zeitschritt  $t$

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\inf} \gamma^k R_{t+k+1} \quad (4.3)$$

- Der Discount  $\gamma \in$

$$0, 1$$

ist der present-value of future rewards

- der Wert der Belohnung nach  $k+1$  Zeitschritten beträgt  $\gamma^k R$
  - Dieser Wert der sofortige Belohnung ist höher als die verzögerte Belohnung  $\rightarrow \gamma$  nahe bei 0 führt zu 'myopic' evaluation  $\rightarrow \gamma$  nahe bei 1 führt zu 'far-sighted' evaluation
- $\rightarrow$  Finde eine policy  $\pi^*$  welche den **return**  $G_0$  maximiert.

#### Rekursiver Ausdruck von $G_t$

Der Return zum Zeitpunkt  $t$  ist

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \quad (4.4)$$

### 4.3.3 Discount Factor $\gamma$

- $0 \leq \gamma \leq 1$
- discounts future rewards
- typischer Wert ist  $\gamma = 0.9$  oder  $\gamma = 0.99$

### 4.3.4 Discounted Rewards

Wieso werden viele Markov-Entscheidungen diskontiert?

- Mensch und Tier haben eine Präferenz für sofortige Belohnungen
- Wenn es sich um eine finanzielle Belohnung handelt, können sofortige Belohnungen mehr Zinsen einbringen als verspätete Belohnungen
- Es ist mathematisch gesehen, angenehmen Rabatte (Discounts) zu gewähren
- Vermeide unendlich returns in zyklischen Markov Prozessen

## 4.4 Value Functions

### 4.4.1 action-Value Functions

**Definition:** Die *action-value Function*  $q_\pi(s, a)$  ist der erwartende Return, beginnend beim State  $s$ , mit der Aktion  $a$  und der Policy  $\pi$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \quad (4.5)$$

*Hinweis:* In der action-value function wird die Aktion  $a$  nicht durch die Richtlinie  $\pi$  gewählt, sondern kann eine beliebige Aktion sein

#### 4.4.2 State-Value Function

$$v_\pi(s) = E_\pi[G_t | S_t = s] \quad (4.6)$$

### 4.5 Policy Iteration

Define a partial ordering over policies:  $\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$

**Theorem:**

Für jeden Markov Decision Process:

- existiert eine optimale Policy  $\pi_*$ , welche besser oder gleich zu allen anderen Policies ist,  $\pi_* \geq \pi, \forall \pi$
- Alle optimalen Policies erreichen die optimale action-value Function  $q_{\pi_*}(s, a) = q_*(s, a)$

#### 4.5.1 Finden der optimalen Policy

Die optimale Policy kann gefunden werden, in dem man eine Maximierung von  $q_*(s, a)$  findet

$$pi(a|s) \begin{cases} 1, & \text{wenn } a = argmax_{a \in A} q_*(s, a) \\ 0, & \text{sonst} \end{cases} \quad (4.7)$$

$$(4.8)$$

Es gibt immer eine deterministische optimale Policy für jeden MDP (Markov Decision Process)  
 $\Rightarrow$  Wenn wir  $q_*(s, a)$  kennen, haben wir sofort auch die optimale Policy!

### 4.6 $\epsilon$ -Greedy Exploration

- Ist die einfachste Idee die kontinuierliche Exploration zu gewährleisten
  - alle  $m$  Aktionen werden mit einer non-zero Wahrscheinlichkeit versucht
  - Mit der Wahrscheinlichkeit  $1 - \epsilon$  wird die greedy-Action ausgewählt
  - Mit der Wahrscheinlichkeit  $\epsilon$  wird eine Aktion nach dem Zufallsprinzip ausgewählt
- $\Rightarrow$  In Schritt k einer Episode können wir  $\epsilon = \frac{1}{k}$  verwenden, um mit der Zeit von der Exploration zur Ausbeutung überzugehen

#### 4.6.1 Exploration vs. Exploitation

Exploration  $\Rightarrow$  Taking a change to learn new things

Exploitation  $\Rightarrow$  What you have learned

- Die nächste action auszuwählen ist ein balancierter Akt zwischen *exploration* und *exploitation*
- Wenn man immer exploration auswählt, rennt man allenfalls sehr häufig in Probleme rein
- Wenn man immer exploitation auswählt, erreicht man allenfalls nie 'high-value rewards'
- Zu Beginn, kann der Agent etwas mehr untersuchen (explore) für eine zufällige Anzahl episoden ( $\Rightarrow$  heatmap phase) und danach fokussierte greedy-Entscheide treffen

### 4.7 Model-Based Learning

Beim **Model-based Learning** kennen wir die Zustandsübergangswahrscheinlichkeiten **P** und die Belohnungsfunktion (Reward Function) **R** eines MDP.

In diesem Fall können wir eine gegebene Policy  $\pi$  durch eine *Policy Iteration* verbessern:

1. gegeben P,R,  $\pi \rightarrow$  berechne v,q
2. gegeben  $\pi$  und v,q  $\rightarrow$  erstelle eine bessere Policy  $\pi'$

### 4.7.1 Bellman Equations for Value Functions

$$\begin{aligned} v_\pi(s) &= \sum_{a \in A} \pi(a|s)(R_s^a) + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \\ q_\pi(s, a) &= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s', a') \end{aligned} \quad (4.9)$$

### 4.7.2 Greedy Policy Improvement

Gegeben einer Policy  $\pi$  und seiner state-value function  $v_\pi$ , können wir eine neue (nicht zwingend optimale) Policy  $\pi'$  mit dem *greedy policy improvement* kreieren:

$$\pi'(s) = \operatorname{argmax}_{a \in A} R_s^a + P_{ss'}^a V(s') \quad (4.10)$$

⇒ Die neue Policy  $\pi'$  ist mindestens so gut wie  $\pi$

- **Policy evaluation** estimate  $v_\pi$  iterative policy evaluation
- **Policy Improvement** Generate  $\pi' \geq \pi$  Greedy policy improvement

**Wichtig!** Der Prozess des *greedy policy iteration* mit bekanntem **R** und **P** konvergiert **immer** zu einer optimalen policy  $\pi^*$

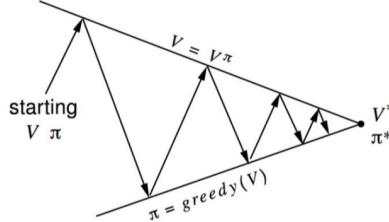


Abbildung 4.4: Abbildung der Policy Iteration

## 4.8 Model-Free Prediction

Beim **Model-free Learning**, kennen wir die MDP-Paramter **nicht** (wie bspw. die Übergangswahrscheinlichkeit **P** und die reward-Function **R**).

→ in diesem Fall, können wir die value-function  $v$ , sowie das  $q$  einer Policy  $\pi$  nur schätzen, in dem man eine oder mehrere Epsioden des MDP mit  $\pi$  durchläuft.

### 4.8.1 Monte-Carlo (MC)

- MC Methoden lernen direkt aus den Erfahrungen der Episoden
- MC ist ein model-free Learning → Es gibt keine Kenntnisse von MDP-Übergang/Belohnungen
- MC lernt aus *kompletten* Episoden
- MC verwendet die einfachstmögliche Idee
- **Vorbehalt:** Monte-Carlo Learning kann nur auf episodische MDPs angewendet werden und alle Episoden müssen beendet werden

#### 4.8.2 Temporal Difference Learning (TD)

- TD Methoden lernen direkt aus den Erfahrungen der Episoden
- TD ist ein model-free LEarning → Es gibt keine Kenntnisse von MDP-Übergang/Belohnung
- TD lernt aus *unvollständigen* Episoden → durch Bootstrapping
- TD aktualisiert eine Vermutung in Richtung einer Vermutung
- Einfachster Zeitdifferenz-Lernalgorithmus: TD(0)
- Wert  $V(S_t)$  wird zum geschätzten Return  $R_{t+1} + \gamma V(S_{t+1})$  aktualisiert:  $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$

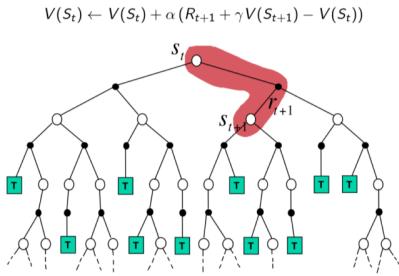


Abbildung 4.5: Abbildung des TD Learning

**Hinweis:** Q-Learning ist ein TD-Learning, da man einen Schritt vorausschaut und nimmt den aktuellen Zustand für das Update zu machen.

#### 4.9 Q-Learning

Q-Learning verfolgt das Ziel die Policy zu maximieren

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal

```

Abbildung 4.6: Abbildung zum Q-Learning Algorithmus

- Q-Funktion  $Q(s, a)$  mit Zustand (State) und Aktion → Sagt uns wie gut jede Aktion im aktuellen Zustand ist (bspw. Greedy)
- terminal-States da wird das Spiel beendet
- episode: Spiel einmal durchlaufen lassen



Abbildung 4.7: Ablauf zum Q-Learning

⇒ Bei genügend Durchläufe konvergiert Q zu einer optimalen Q-Funktion → mit maximiertem Reward

#### 4.9.1 Learning Q

- Ziel von Q-Learning ist zu Q zu lernen mit der 'Action-value Function'
- Q ratet wie viel Reward erwartet werden kann, wenn eine bestimmte Aktion durchgeführt wird
- Gegeben eines guten Q, ist es sehr einfach eine gute policy für den Agenten zu konstruieren
- Q-Learning konvergiert zu einem optimalen Q

### 4.10 SARSA

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A)) \quad (4.11)$$

**Theorem:** SARSA konvergiert zur optimalen Aktionswertfunktion  $Q(s, a) \rightarrow q^*(s, a)$ , unter bestimmten vorausgesetzten Bedingungen

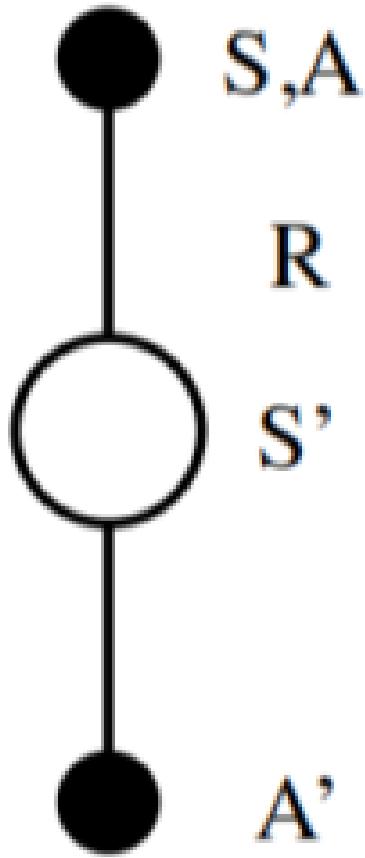


Abbildung 4.8: Abbildung des SARSA-Schemas

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
    Initialize  $S$ 
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
    Repeat (for each step of episode):
        Take action  $A$ , observe  $R, S'$ 
        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
         $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$ 
         $S \leftarrow S'; A \leftarrow A'$ ;
    until  $S$  is terminal

```

Abbildung 4.9: Abbildung des SARSA PseudoCodes

#### 4.10.1 SARSA vs. Q-Learning

SARSA folgt einfach der aktuellen Policy, die nächste Aktion aus  $S'$  auszuwählen, während bei Q-Learning das Maximum von  $Q$  über alle möglichen Aktionen aus Zustand  $S'$  verwendet wird.