

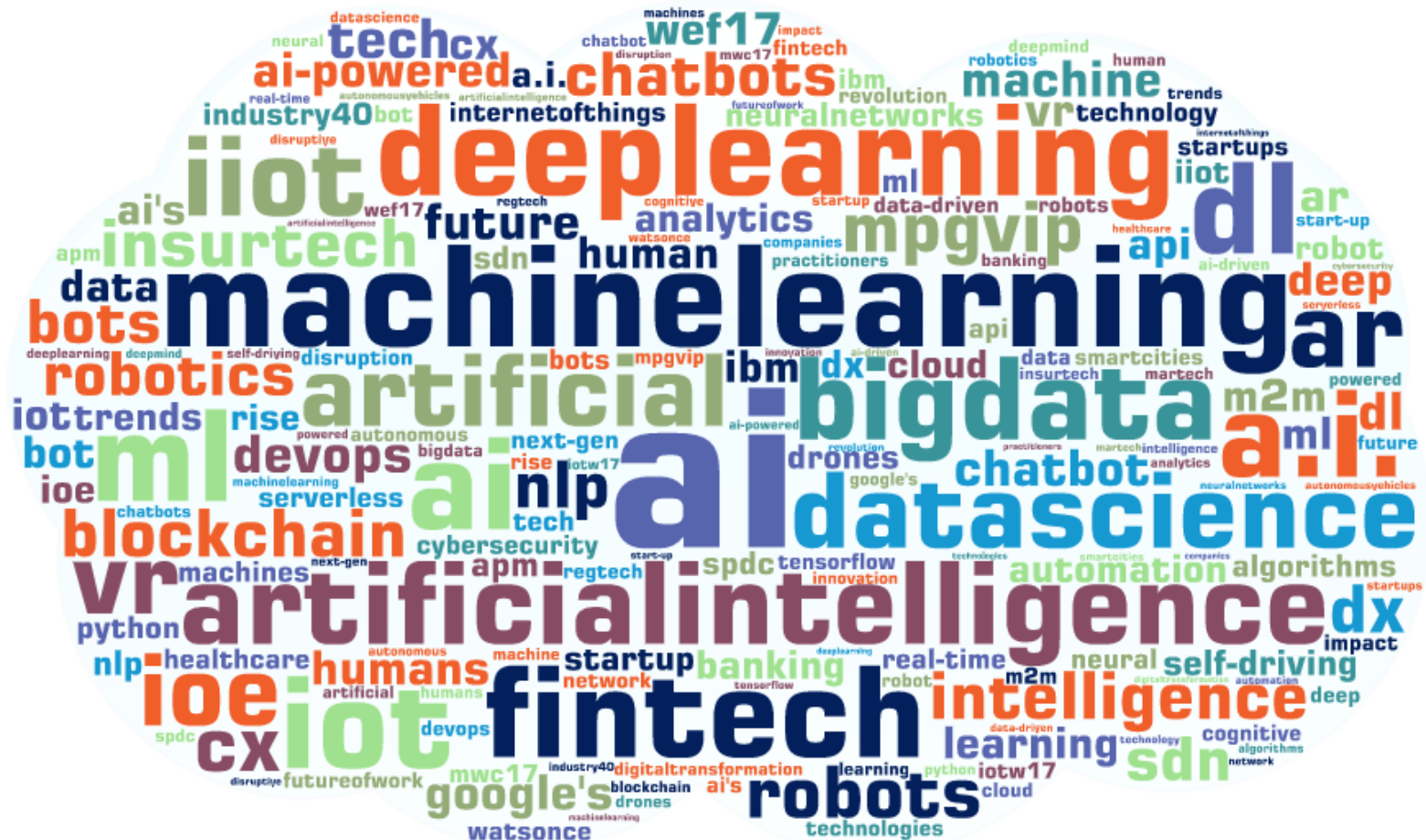


Software Anwendungen mit Künstlicher Intelligenz: Aufgabe 4 – Deep Q-Learning

Dr. Richard Müller, Dr. Christian Neuhaus

Friedrich-Alexander-Universität Erlangen-Nürnberg, 10. Juli 2019

Wie funktionieren eigentlich Machine Learning und neuronale Netze?

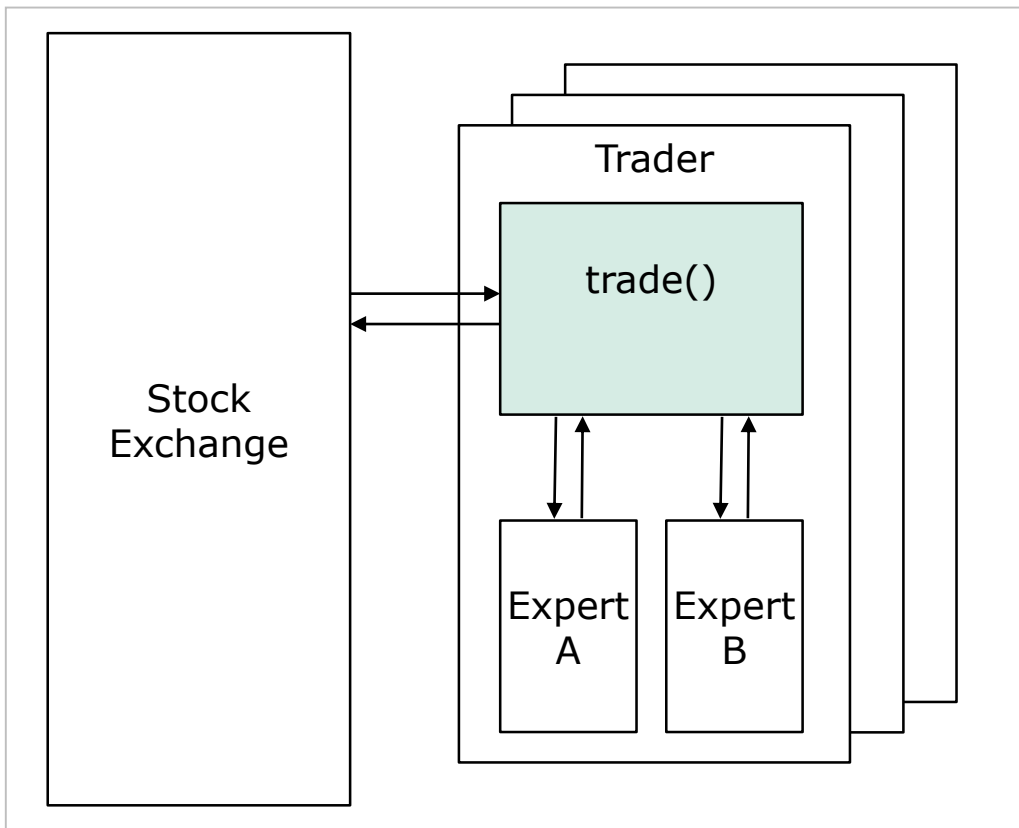


Machine Learning hat schon jetzt Relevanz in unterschiedlichsten Gebieten



Aufgabe: Baut einen Trader, welcher auf Basis der Eingaben der Börse (Wertpapierkurse) und Experten (Empfehlungen) selbstständig handelt

Setting

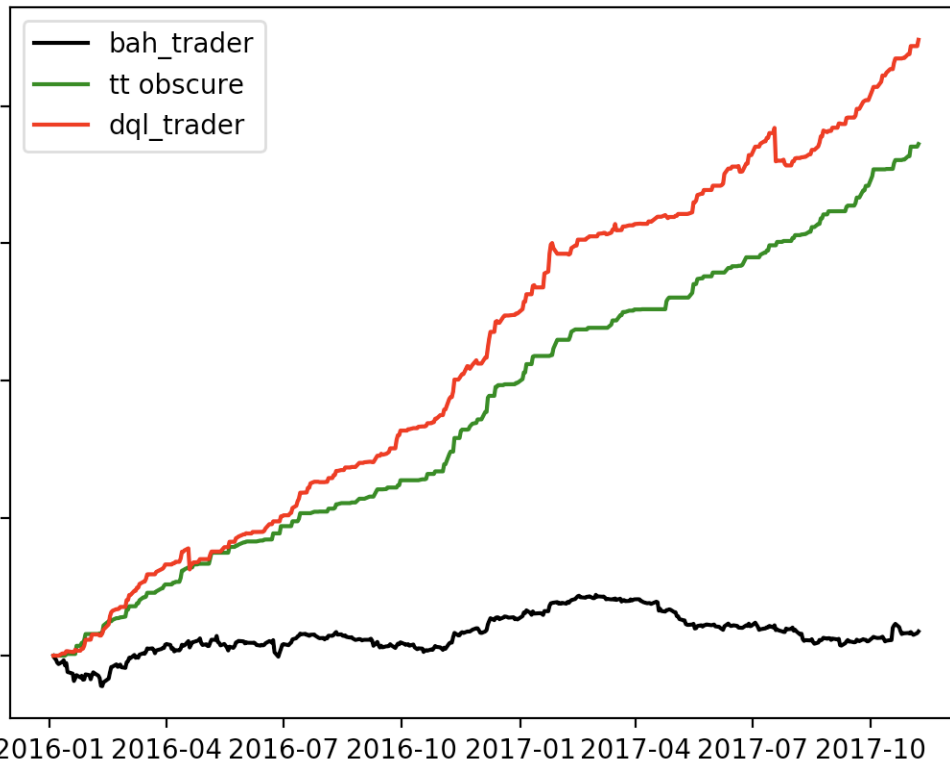


Beschreibung

- **Börse:** Handelt Wertpapiere A, B
- **Trader:** Gibt einmal täglich Handelsentscheidungen an Börse
- **Experten:** Geben einmal täglich Empfehlung für Wertpapiere A, B

Aufgabe: Baut einen Trader, welcher auf Basis der Eingaben der Börse (Wertpapierkurse) und Experten (Empfehlungen) selbstständig handelt

Setting



Beschreibung

- **Börse:** Handelt Wertpapiere A, B
- **Trader:** Gibt einmal täglich Handelsentscheidungen an Börse
- **Experten:** Geben einmal täglich Empfehlung für Wertpapiere A, B
 - Empfehlung kann **falsch** sein

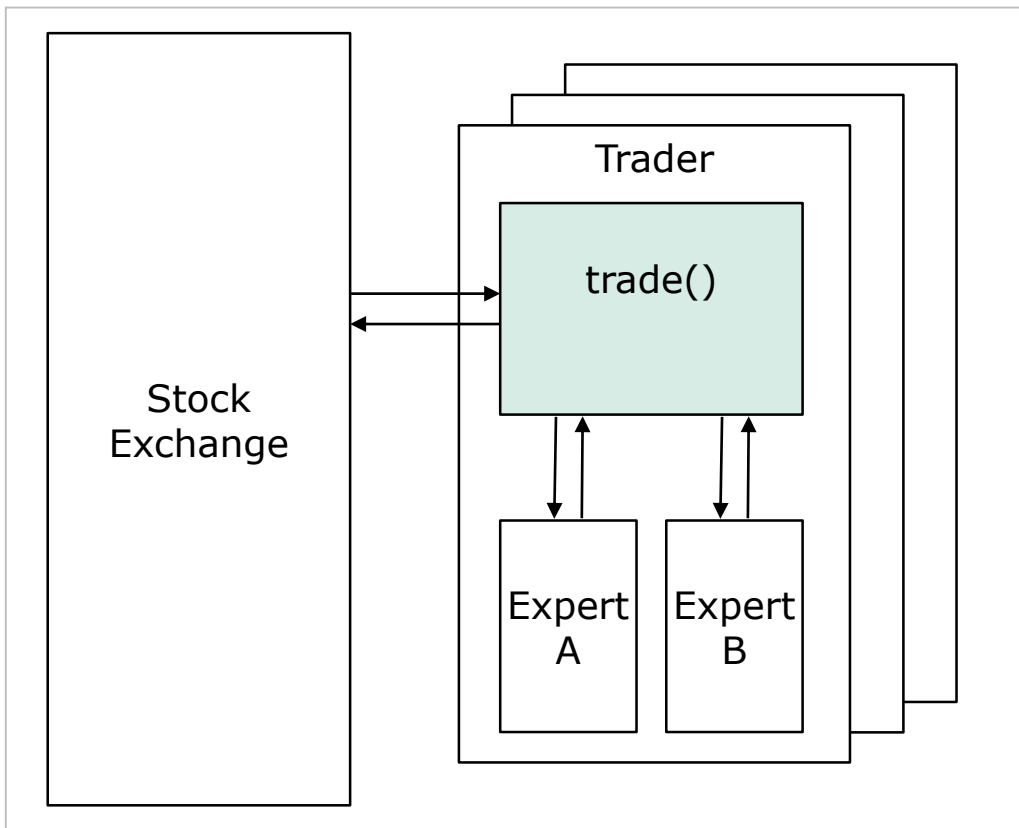
Wertentwicklung:

- BAH = Buy and Hold Trader¹⁾
- TT = Trusting Trader¹⁾
- DQL = Deep Q-Learning Trader

¹⁾bekommt ihr gestellt

Aufgabe: Baut einen Trader, welcher auf Basis der Eingaben der Börse (Wertpapierkurse) und Experten (Empfehlungen) selbstständig handelt

Setting



Beschreibung

- **Börse:** Handelt Wertpapiere A, B
- **Trader:** Gibt einmal täglich Handelsentscheidungen an Börse
- **Experten:** Geben einmal täglich Empfehlung für Wertpapiere A, B
 - Empfehlung kann **falsch** sein

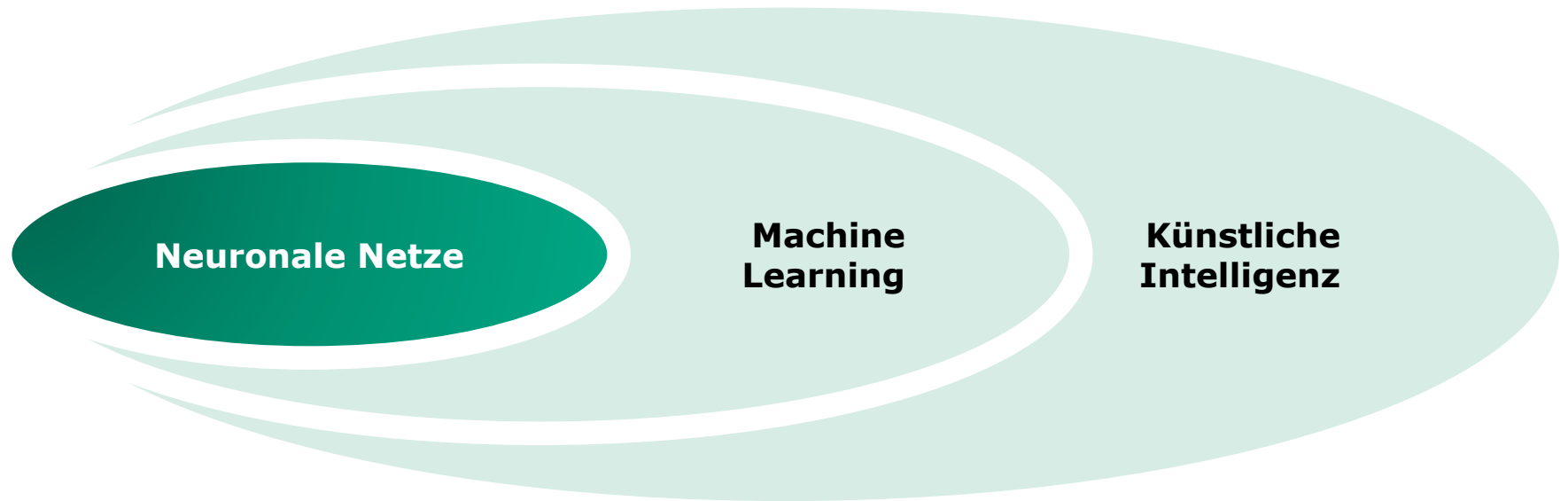
Anforderungen:

1. Trader handelt vollautomatisch
2. Trader lernt aus Handlungen
3. Lernen des Trades soll mittels Deep Q-Learning realisiert werden

- **Neuronale Netze**

- Deep Q-Learning
 - Framework
-

Neuronale Netze sind eine Technik von Machine Learning, und Machine Learning ist ein Teil von künstlicher Intelligenz



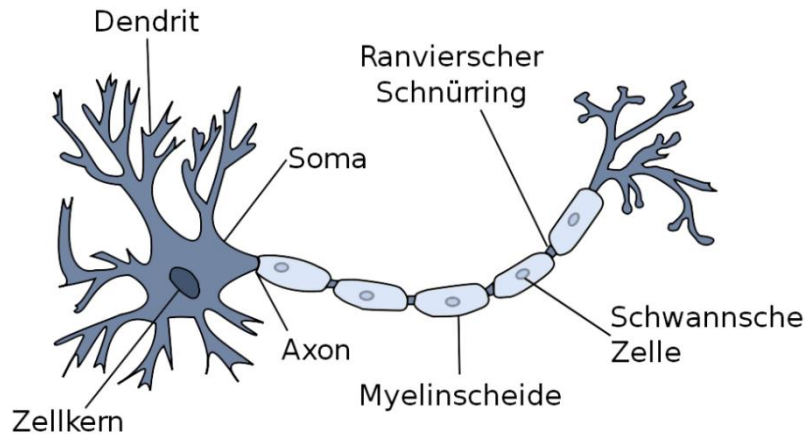
- Ziel: Netze aus künstlichen Neuronen (als Modell) zur Informationsverarbeitung untersuchen

- Ziel: Maschinen die Fähigkeit zur Generierung von Wissen aus Erfahrung geben

- Ziel: Maschinen zur Erledigung intelligenter Aufgaben bauen
- Wahrnehmung, Robotik, Planung, Lernen, ...

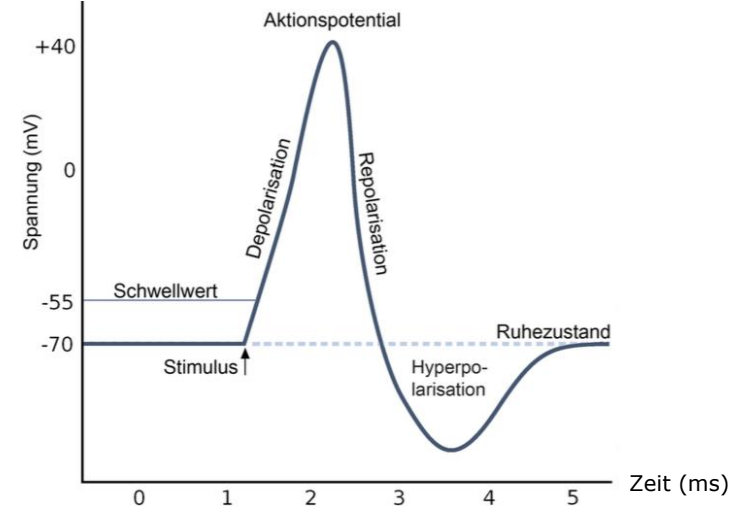
Vorbild für neuronale Netze ist die Funktionsweise biologischer Neuronen

Biologisches Neuron



- „Schalter mit Informationseingang/-ausgang“
- Aufnahme eingehender Signale von anderen Zellen über Dendriten
- „Verarbeitung“ im Soma und evtl. Weiterleitung ausgehender Signale über Axon

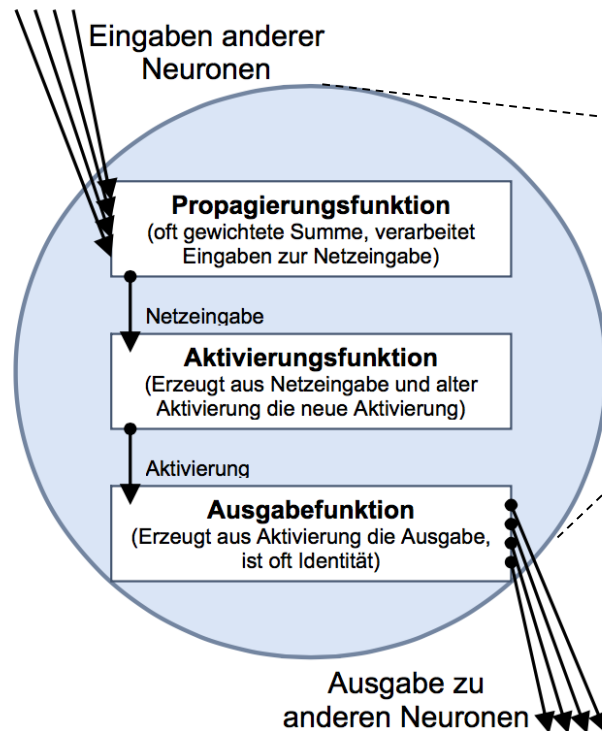
Funktionsweise



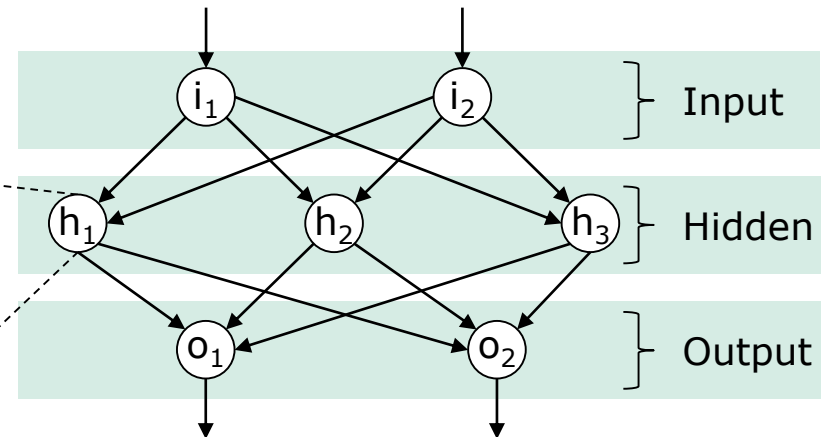
- Bei Schwellenwertüberschreitung wird Neuron aktiviert und elektrisches Signal ausgelöst
- Rückkehr in den Ruhezustand nach „Zwangspause“ von 1-2 ms

Künstliche neuronale Netze bestehen aus künstlichen Neuronen, welche die Informationsverarbeitung biologischer Neuronen nachahmen

Künstliches Neuron



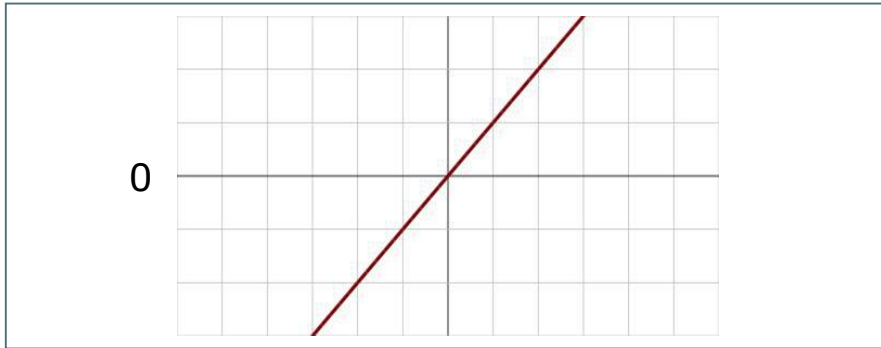
Künstliches neuronales Netz



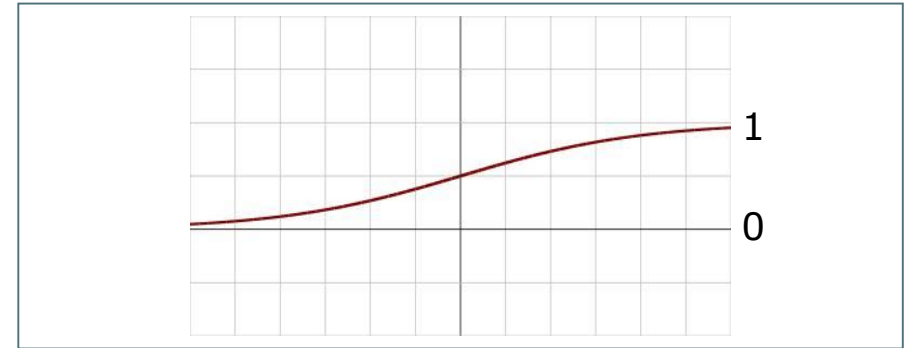
- Gerichtetes, gewichtetes Netzwerk aus künstlichen Neuronen
- Typischerweise Anordnung der Neuronen in Schichten (In-/Output, Hidden Layers)
- Tiefes Netzwerk: üblicherweise mehr als einen Hidden Layer

Exkurs: Übliche Aktivierungsfunktionen in neuronalen Netzen sind Linearfunktion, ReLU, Sigmoidfunktion oder tanh

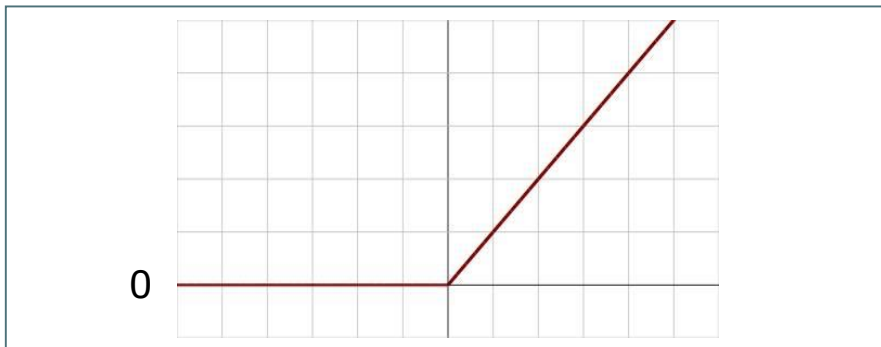
Identität / Linearfunktion



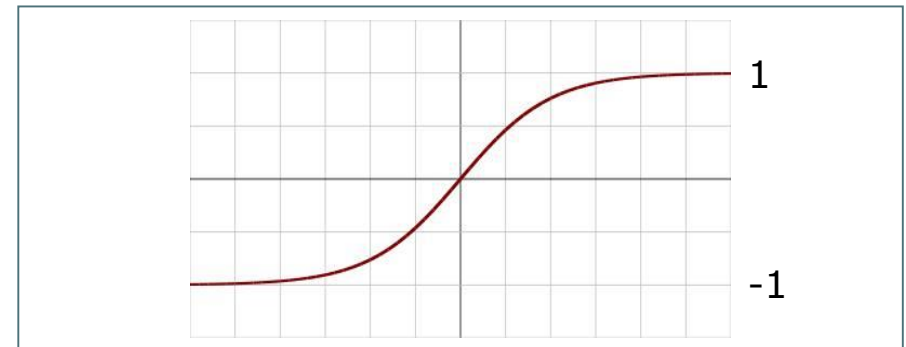
Sigmoidfunktion



Rectified linear unit (ReLU)



Tangens hyperbolicus (tanh)



Keras (Python Bibliothek) ermöglicht Beschreibung und Training eines neuronalen Netzes in wenigen Code-Zeilen

Struktur

- ① Aufbau, Größe und Verbindungen der Layers: Sequential = linearer Stack an Layers
- ② Dense = Layer mit allen Verbindungen

Aktivierungsfunktionen

- ③ Berechnet Aktivierung aus Eingaben und vorheriger Aktivierung (z.B. Linear, Rectifier/Relu, Sigmoid, ...)

Training (Lernen)

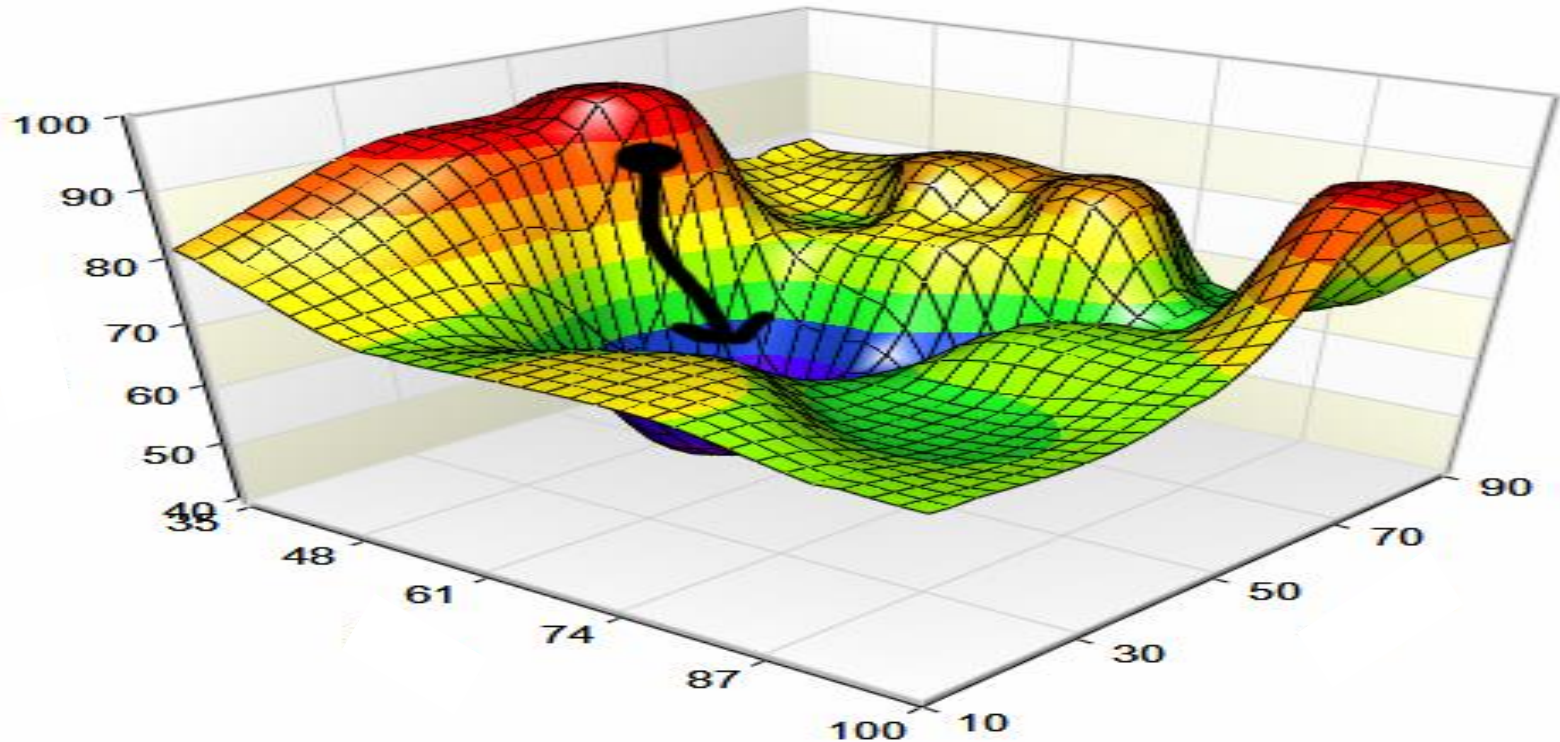
- ④ Kostenfunktion vergleicht aktuellen und erwarteten Output (Abstandsmaß)
- ⑤ Minimierung der Kostenfunktion durch Gradientenabstieg

```
① model = Sequential()
② { model.add(Dense(HIDDEN_SIZE, input_dim=INPUT_SIZE, activation='relu'))
③ { model.add(Dense(HIDDEN_SIZE, activation='relu'))
    model.add(Dense(OUTPUT_SIZE, activation='linear'))

④ model.compile(loss='mean_squared_error', optimizer='sgd')
⑤ model.fit(X_TRAIN, Y_TRAIN, epochs=EPOCHS, batch_size=BATCH_SIZE)
```

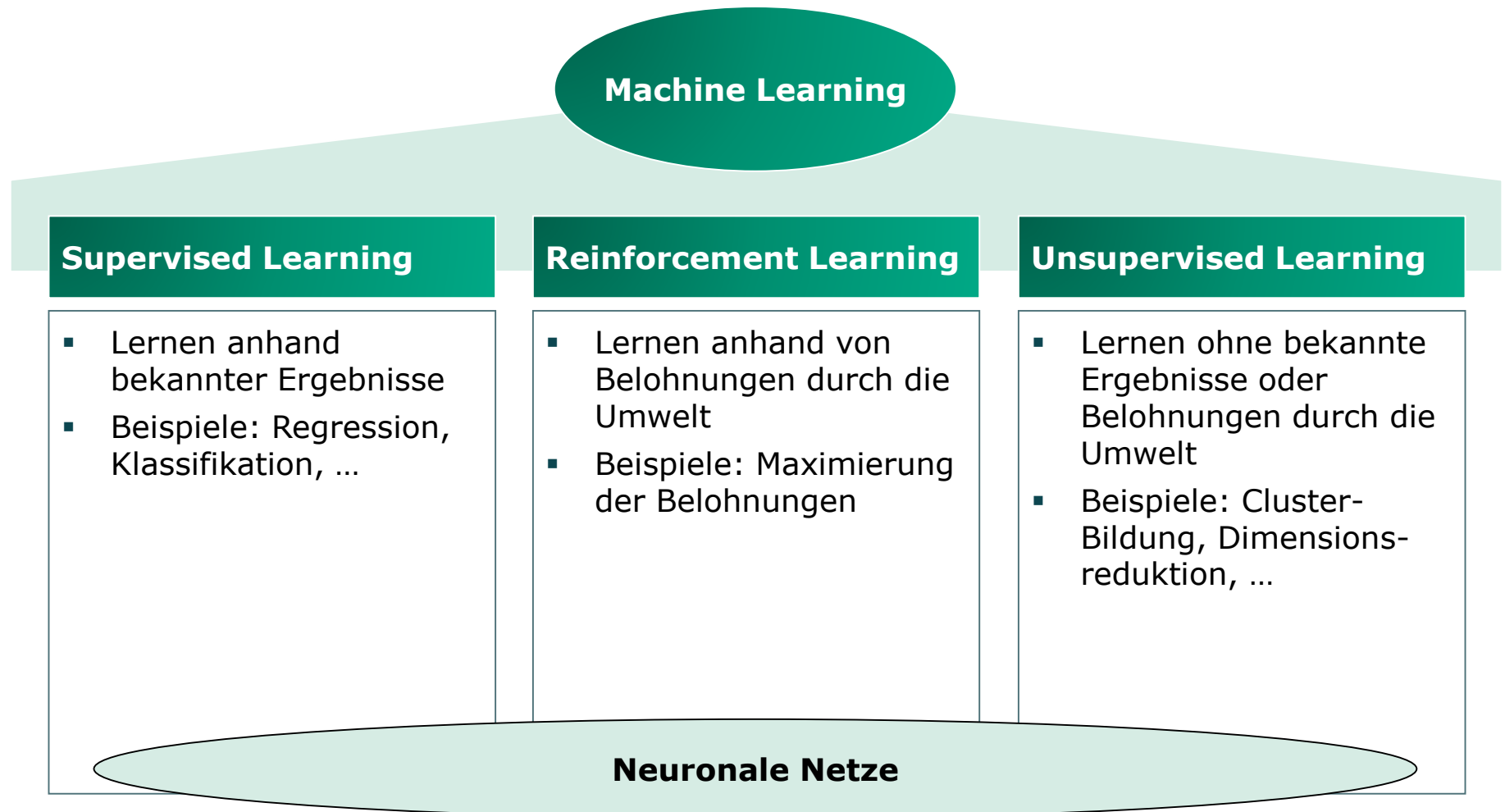
Exkurs: Ein neuronales Netz lernt durch das (lokale) Minimieren der Kostenfunktion mittels Gradientenabstieg

Gradientenabstieg bei zweidimensionaler Fehlerfunktion

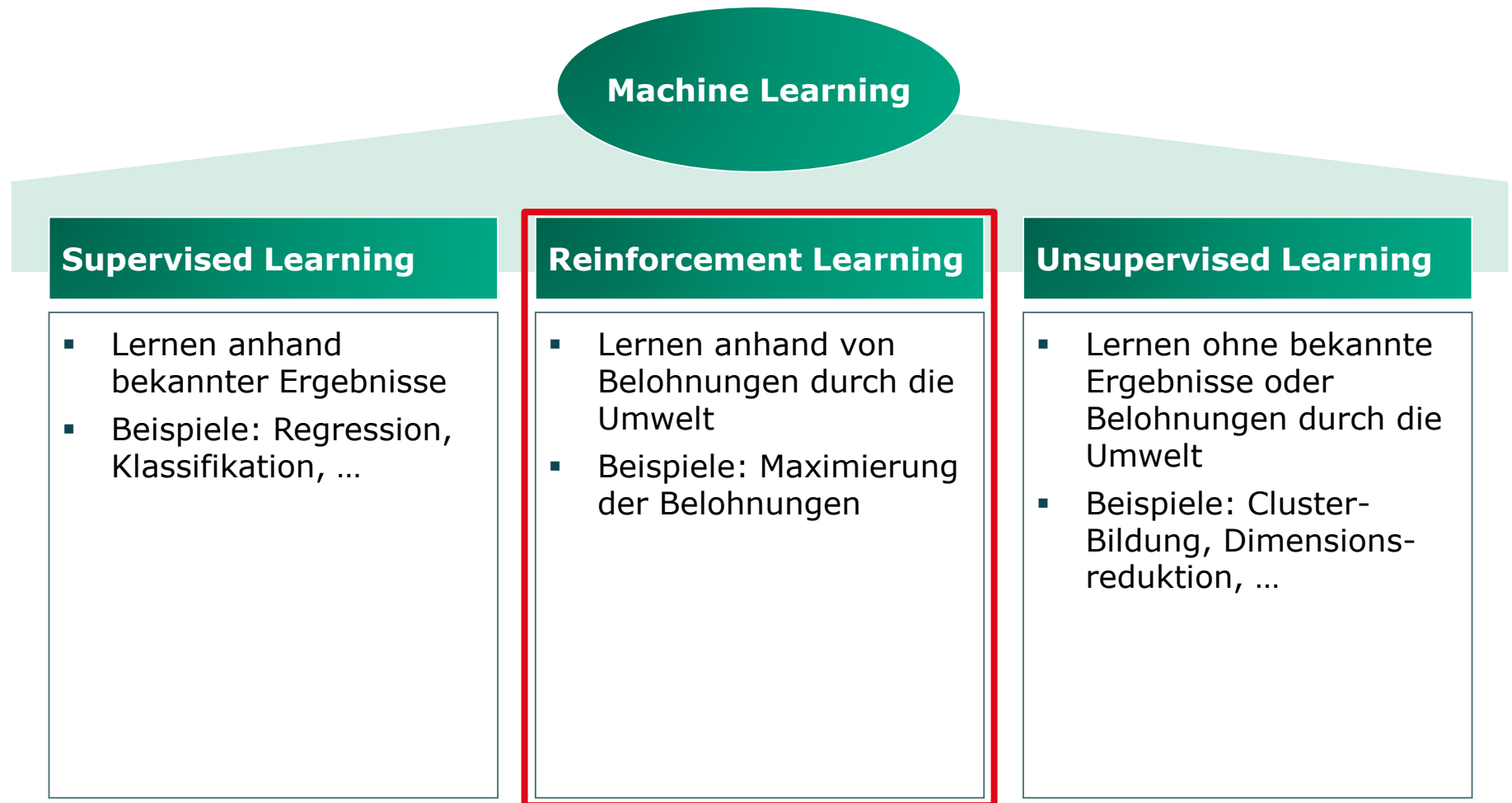


-
- Neuronale Netze
 - **Deep Q-Learning**
 - Framework
-

Arten von Machine Learning unterscheiden sich anhand der Eingaben

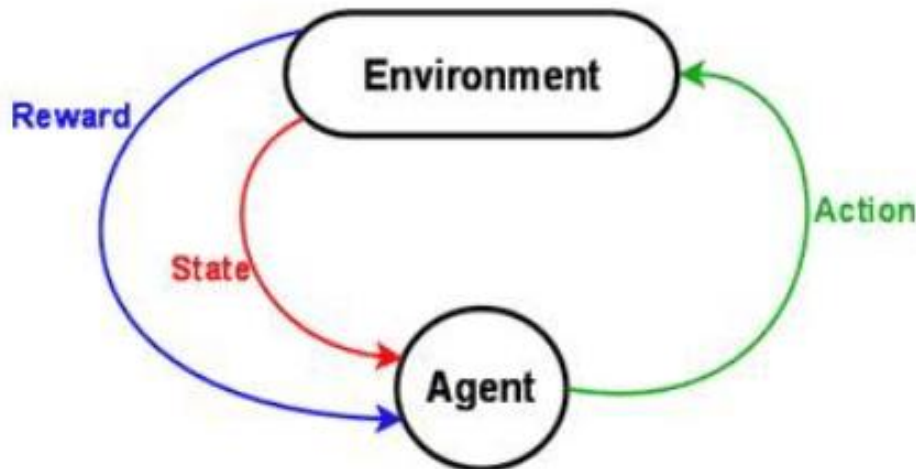


Reinforcement Learning ist Kompromiss aus Supervised Learning und Unsupervised Learning



Voraussetzung: Lernproblem wird durch die Interaktion eines Agenten mit seiner Umgebung modelliert

Illustration

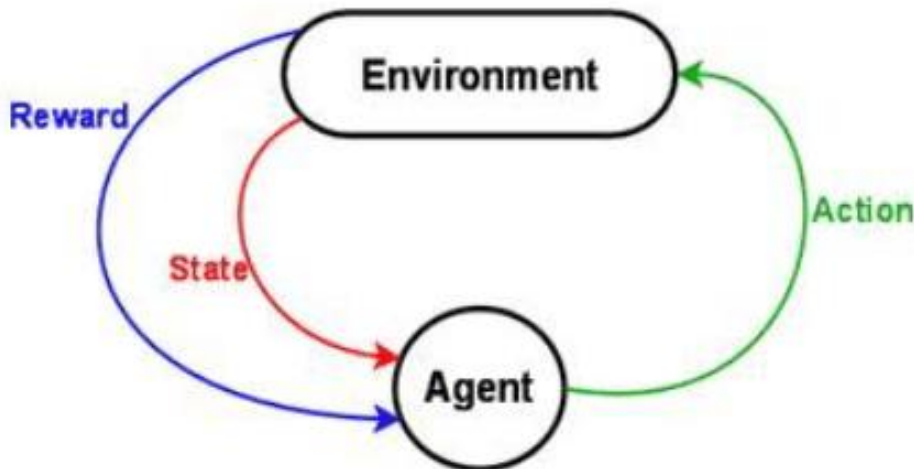


Grundbegriffe

- **Agent:** Handelndes System, welches Aktionen in einer Umgebung ausführen kann
 - Ziel: Maximierung der Rewards bis Spielende
- **Environment:** Umgebung, welche durch Agent beeinflusst werden kann
- **State:** Aktueller Zustand von Agent und Umgebung
- **Action:** Aktion, die in der Umgebung ausgeführt wird und zu einem neuen State führt
- **Reward:** Positives oder negatives Ergebnis nach einer Aktion

Idee: Maximierung der Rewards durch Nutzung einer Qualitätsfunktion Q („Komplettlösung“) für alle Actions

Anwendung von Q



$$\text{Action} = \operatorname{argmax}_a Q(\text{State}, a)$$

Definition von Q

- **Qualitätsfunktion Q**
 - Input: State s und Action a
 - Output: Qualitätsmaß, d.h. wie gut Action a in State s für den Agenten ist
- Q muss aktuelle und zukünftige Rewards¹ berücksichtigen:
 - $R_t = r_t + r_{t+1} + r_{t+2} + \dots + r_n$
 - $Q(s_t, a_t) = \max R_{t+1}$

Woher bekommen wir Q?

Q-Learning: Erlernen von Q durch wiederholtes Ausprobieren von Actions

Q-Learning

Initialisiere Q zufällig

Beobachte Anfangszustand s

wiederhole

 wähle Action a und führe a in Umgebung aus
 beobachte Reward r und neuen State s'

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

 s = s'

bis Spielende

Beschreibung

- Iteratives Update von Qualitätsfunktion Q durch neu beobachtete Rewards
 - Berechnung des neuen $Q(s, a)$ über die Actions a' des Folgezustands s'
- Diskontierungsfaktor γ steuert die Gewichtung zukünftiger Rewards
 - $\gamma = 1$ bedeutet: Zukünftige Rewards werden maximal einberechnet
 - $\gamma = 0$ bedeutet: Zukünftige Rewards werden ignoriert

Deep Q-Learning: Speichern von Q als trainiertes neuronales Netz

Deep Q-Learning

Initialisiere Q als neuronales Netz zufällig
Beobachte Anfangszustand s

wiederhole

wähle Action a und führe a in Umgebung aus
beobachte Reward r und neuen State s'

trainiere Q mit Supervised Learning:

Input: s, a

Erwartet: $r + \gamma \max_{a'} Q(s', a')$

$s = s'$

bis Spielende

Beschreibung

- Implementierung von Qualitätsfunktion Q als neuronales Netz anstelle einer Tabelle o.Ä.
 - Einmaliges Training von Q mit jedem neu erhaltenen Reward durch Supervised Learning

Trick 1/3: Effizienzgewinn durch Berechnung von $Q(s)$ statt $Q(s, a)$

Deep Q-Learning

Initialisiere Q als neuronales Netz zufällig
Beobachte Anfangszustand s

wiederhole

wähle Action a und führe a in Umgebung aus
beobachte Reward r und neuen State s'
trainiere Q mit Supervised Learning:

Input: s

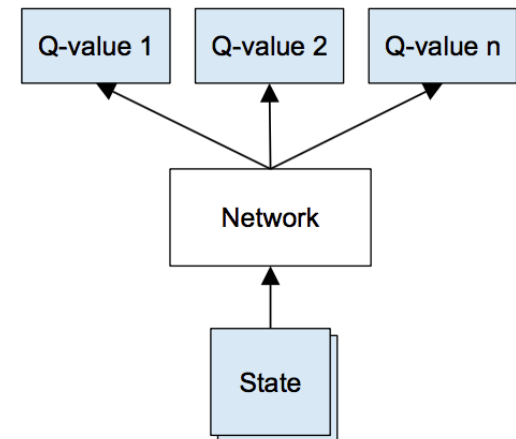
Erwartet: **Vektor mit** $r + \gamma \max_{a'} Q(s', a')$ **bei** a

$s = s'$

bis Spielende

Beschreibung

- Problem: Um $\max_{a'} Q(s', a')$ zu finden sind viele Aufrufe von Q notwendig
- Lösung: Implementierung von Q mit Input s und Ausgabe eines Vektors von Q -Werten zu s



Trick 2/3: Regelmäßig zufällige Actions ermöglichen neues Lernen trotz bekannter funktionierender Strategien (Exploration vs. Exploitation)

Deep Q-Learning

Initialisiere Q als neuronales Netz zufällig
Beobachte Anfangszustand s

wiederhole

 wähle Action a

 mit Wahrscheinlichkeit ϵ zufällig

 mit Wahrscheinlichkeit $1-\epsilon$ nach $\max Q(s, a)$

 führe a in Umgebung aus und reduziere ϵ

 beobachte Reward r und neuen State s'

 trainiere Q mit Supervised Learning:

 Input: s

 Erwartet: Vektor mit $r + \gamma \max_{a'} Q(s', a')$ bei a

$s = s'$

bis Spielende

Beschreibung

- Problem: Sollte man bei einer bekannten Strategie bleiben oder neue, vielleicht bessere Strategien erkunden?
- Lösung: Einführung eines Parameters ϵ
 - ϵ ist Wahrscheinlichkeit, mit der eine zufällige Action (Exploration) anstelle einer bekannten Action (Exploitation) gewählt wird
 - Beispiel DeepMind: Minderung von ϵ über die Zeit von 1 bis minimal 0.1

Trick 3/3: Schnelleres Lernen durch Experience Replay

Deep Q-Learning

Initialisiere Q als neuronales Netz zufällig
Beobachte Anfangszustand s

wiederhole

 wähle Action a

 mit Wahrscheinlichkeit ε zufällig

 mit Wahrscheinlichkeit $1-\varepsilon$ nach $\max_a Q(s, a)$

 führe a in Umgebung aus und reduziere ε

 beobachte Reward r und neuen State s'

 speichere Erfahrung $\langle s, a, r, s' \rangle$

 nimm zufällige Menge an Erfahrungen $\langle s, a, r, s' \rangle$

 berechne erwarteten Output $r + \gamma \max_{a'} Q(s', a')$

 pro ausgewählter Erfahrung

 trainiere Q mit Supervised Learning für die

 ausgewählten Erfahrungen

$s = s'$

bis Spielende

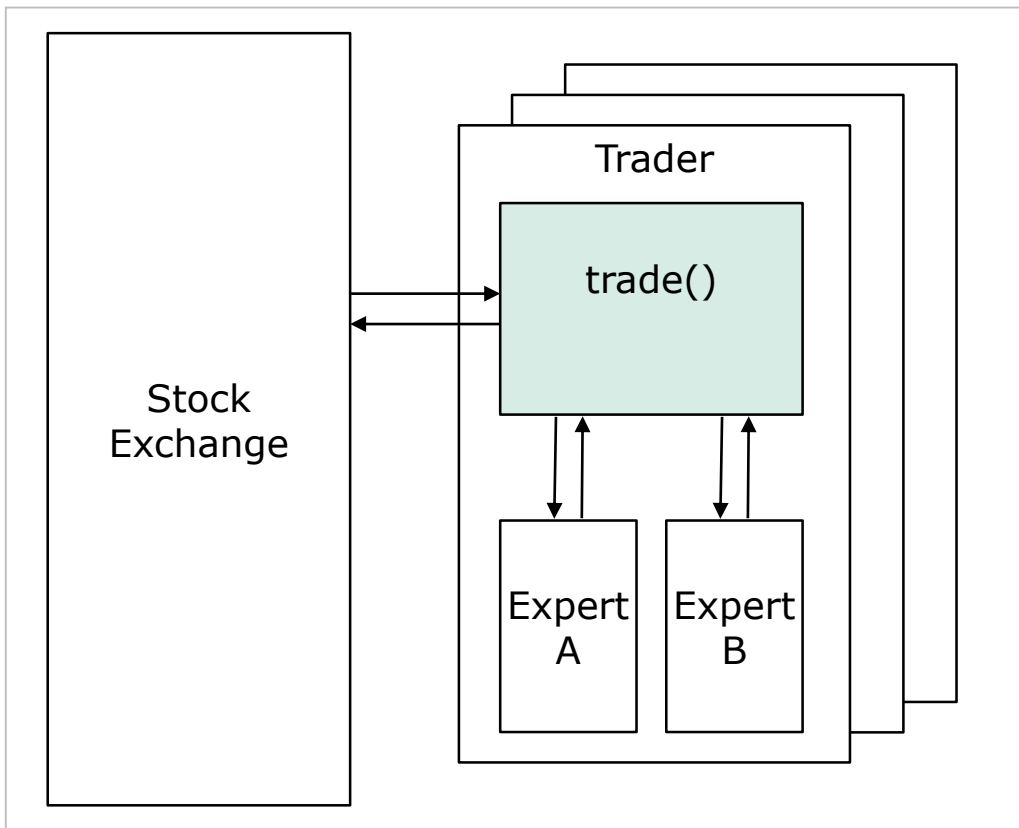
Beschreibung

- Problem: Trainieren von Q für jeweils einzelne neue Werte ist ineffizient
- Lösung: Speichern von Erfahrungen und Training anhand zufälliger Teilmengen der gespeicherten Erfahrungen

-
- Neuronale Netze
 - Deep Q-Learning
 - **Framework**
-

Aufgabe: Baut einen Trader, welcher auf Basis der Eingaben der Börse (Wertpapierkurse) und Experten (Empfehlungen) selbstständig handelt

Setting



Beschreibung

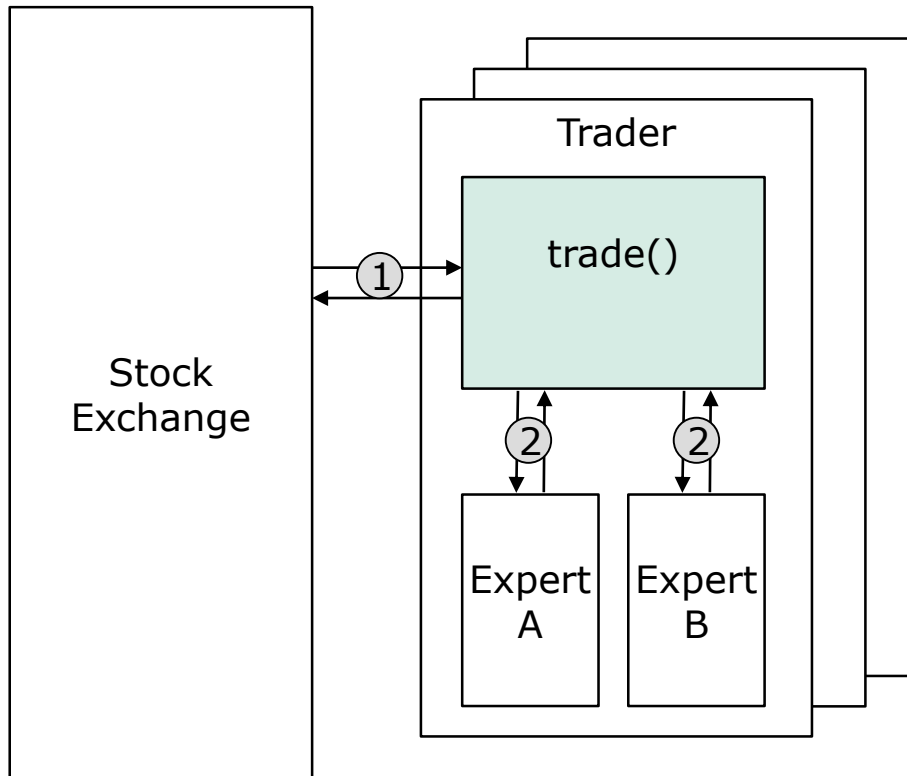
- **Börse:** Handelt Wertpapiere A, B
- **Trader:** Gibt einmal täglich Handelsentscheidungen an Börse
- **Experten:** Geben einmal täglich Empfehlung für Wertpapiere A, B
 - Empfehlung kann **falsch** sein

Anforderungen:

1. Trader handelt vollautomatisch
2. Trader lernt aus Handlungen
3. Lernen des Trades soll mittels Deep Q-Learning realisiert werden

Börse und Trader kommunizieren über trade() Methode; Trader und Experte kommunizieren über vote() Methode

Setting

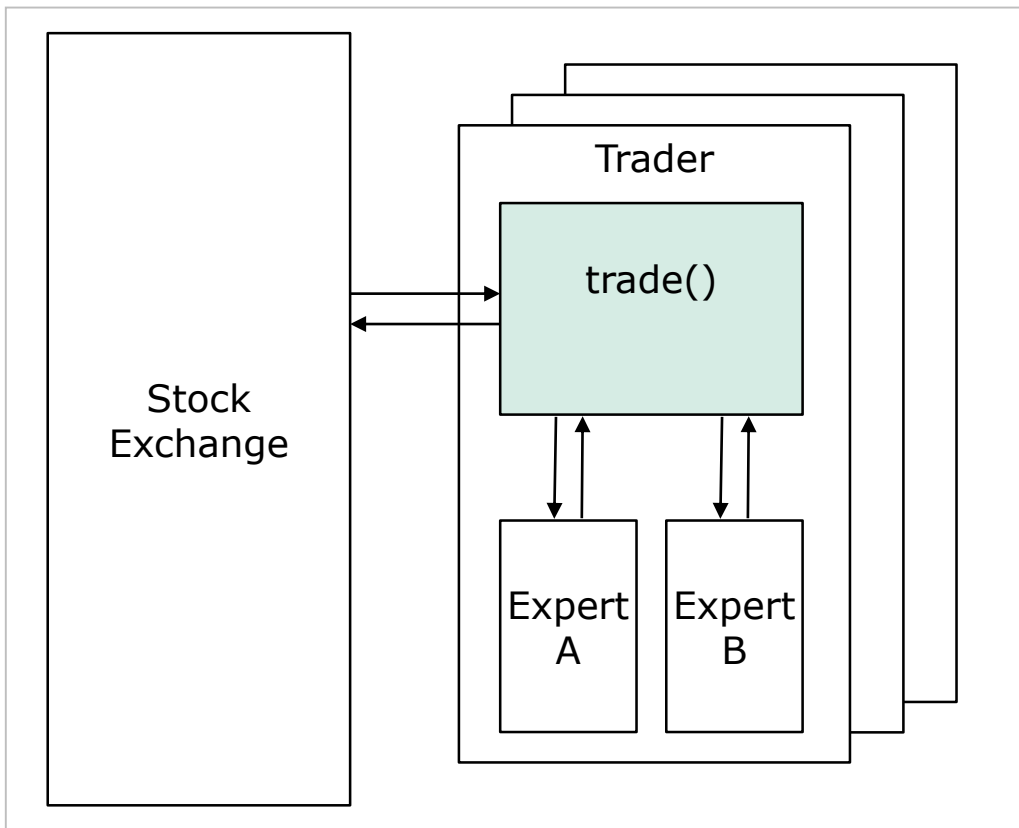


Beschreibung

- Interfaces ITrader und IExpert werden von Tradern und Experten implementiert (Vererbung, ABC)
- ① **ITrader.trade()**
 - Eingabe: aktuelles Portfolio + aktuelle Kurse **aller** Unternehmen
 - Ausgabe: Liste auszuführender Orders
 - ② **IExpert.vote()**
 - Eingabe: alle Kurse **eines** Unternehmens bis zum aktuellen Datum
 - Ausgabe: Empfehlung (BUY, HOLD, SELL) für Wertpapier

Aufbau und Training des neuronalen Netzes des Traders erfolgt im Vorfeld

Setting



Vorgehen (Empfehlung)

1. Implementiert Deep Q-Learning in **trade()** Methode des Traders (und beliebigen Hilfsmethoden)
 - Implementiert Struktur zum Speichern von **State** (Klasse?)
2. Neuronales Netz wird im **Constructor** des Traders erzeugt; Netzstruktur und evtl. notwendige Hilfsvariablen haben wir schon vorgegeben
3. Trainiert neuronales Netze über **main-Methode** in Trader-Datei
4. Testet euren Trader (mit trainiertem neuronalem Netz) über main-Methode der **StockExchange**

Das nötige Framework und ein Lösungsskelett bekommt ihr gestellt (1/6)

Dateien

fau2019-preparation 10 items

- datasets
- experts
- framework
- traders
- .DS_Store
- directories.py
- README.md
- requirements.txt
- stock_exchange.py
- test_runner.py






Beschreibung

- **directories.py**
 - Variablen mit Verzeichnispfaden → ignore
- **README.md**
 - Aufgabenstellung, spannende Links, Erklärungen zum Framework → read/ignore
- **requirements.txt**
 - Benötigte Python Packages → use/ignore
- **stock_exchange.py**
 - Beinhaltet Börse (Klasse StockExchange) und main-Methode zur Evaluierung aller Trader → use
- **test_runner.py**
 - Beinhaltet alle Unit-Tests → use

Das nötige Framework und ein Lösungsskelett bekommt ihr gestellt (2/6)

Dateien

datasets 5 items

-  .DS_Store
-  a_1962-2011.csv
-  a_2012-2015.csv
-  b_1962-2011.csv
-  b_2012-2015.csv

Beschreibung

- **a_1962-2011.csv**
 - Kurse von Wertpapier A der Trainings-Periode → ignore
- **a_2012-2015.csv**
 - Kurse von Wertpapier A der Testing-Periode → ignore
- **b_1962-2011.csv und b_2012-2015.csv**
 - Wie oben, nur für Wertpapier B → ignore

Warum ignore?

Weil die Testdaten automatisch über den Constructor der Klasse StockMarketData geladen werden können

Das nötige Framework und ein Lösungsskelett bekommt ihr gestellt (3/6)

Dateien

experts 5 items

obscurer_expert_data

test

.DS_Store

__init__.py

obscurer_expert.py

Beschreibung

- **obscurer_expert_data**
 - Ordner mit Binärdaten für die Experten → ignore
- **obscurer_expert.py**
 - Beinhaltet Experten (Klasse ObscureExpert) für Wertpapiere A und B → use

Das nötige Framework und ein Lösungsskelett bekommt ihr gestellt (4/6)

Dateien

framework 14 items

- test
- .DS_Store
- __init__.py
- company.py
- interface_expert.py
- interface_trader.py
- logger.py
- order.py
- period.py
- portfolio.py
- stock_data.py
- stock_market_data.py
- utils.py
- vote.py

Beschreibung

- **company.py**
 - Enum für Wertpapiere A, B → use
- **interface_expert.py**
 - Beinhaltet Interface für Experte → use
- **interface_trader.py**
 - Beinhaltet Interface für Trader → use
- **logger.py**
 - Python Logging Mechanismus → use/ignore
- **order.py**
 - Beinhaltet eine Handlung (Order) → use
- **period.py**
 - Enum für Zeiträume → use

Das nötige Framework und ein Lösungsskelett bekommt ihr gestellt (5/6)

Dateien

framework 14 items

- test
- .DS_Store
- __init__.py
- company.py
- interface_expert.py
- interface_trader.py
- logger.py
- order.py
- period.py
- portfolio.py
- stock_data.py
- stock_market_data.py
- utils.py
- vote.py

Beschreibung

- **portfolio.py**
 - Beinhaltet Portfolio eines Traders, d.h. Cash und Anzahl Wertpapiere A und B → use
- **stock_data.py**
 - Beinhaltet historische Kurse eines Wertpapiers → use
- **stock_market_data**
 - Beinhaltet historische Kurse aller Wertpapiere → use
- **utils.py**
 - Beinhaltet Speichern/Laden eines neuronalen Netzes mit Keras → use
- **vote.py**
 - Enum für Empfehlung eines Experten (BUY, HOLD, SELL) → use

Das nötige Framework und ein Lösungsskelett bekommt ihr gestellt (6/6)

Dateien

traders 7 items

📁 dql_trader_data

📁 test

📄 .DS_Store

📄 __init__.py

📄 buy_and_hold_trader.py

📄 deep_q_learning_trader.py

📄 trusting_trader.py

Beschreibung

- **dql_trader_data**
 - Ordner zum Speichern des trainierten neuronalen Netzes → ignore
 - Nutzt Methode `save_trained_model()` sowie Constructor im DQL Trader
- **buy_and_hold_trader.py**
 - Trader, welcher nur Wertpapiere A und B hält → use/ignore
- **deep_q_learning_trader.py**
 - Trader, welcher mit Deep Q-Learning (DQL Trader) gelernt hat → implement
- **trusting_trader.py**
 - Trader, welcher immer den Experten vertraut → use/ignore

Folgende Hinweise machen euch das Leben leichter



Hinweise

- Benutzt Python 3.7 und IDE (z.B. PyCharm)
- Schaut euch Framework und Beispiel-Trader an
- Benutzt den gegebenen Pseudocode für Deep Q-Learning als Vorlage
- Versucht alle drei Tricks zu implementieren
- Aufbau und Training eines optimalen Netzes ist experimentell und sehr Hardware-intensiv
- Kleine Unterschiede in Parametern: Große Auswirkungen auf Trainingszeit oder Lernerfolg
- Anfangs einfachen Actionspace nutzen: Z.B. Aktie entweder komplett kaufen oder komplett verkaufen (also 4 statt 10 Actions insgesamt)
- Einfache Reward-Funktion nutzen: Ignoriert zukünftige Rewards
- Rechtzeitig viele Fragen stellen!



Vielen Dank!

Dr. Richard Müller
Managing Consultant

Richard.Mueller@senacor.com



Dr. Christian Neuhaus
Senior Consultant

Christian.Neuhaus@senacor.com

