# Praktikum: Selbstlernende Systeme

## Reinforcement Learning
## Artificial Neural Networks
## Deep Q-Networks

30.10.2019

Universität Augsburg
Institut für Informatik
Lehrstuhl für Organic Computing

# Gliederung

ORGANIC COMPUTING

1. Reinforcement Learning

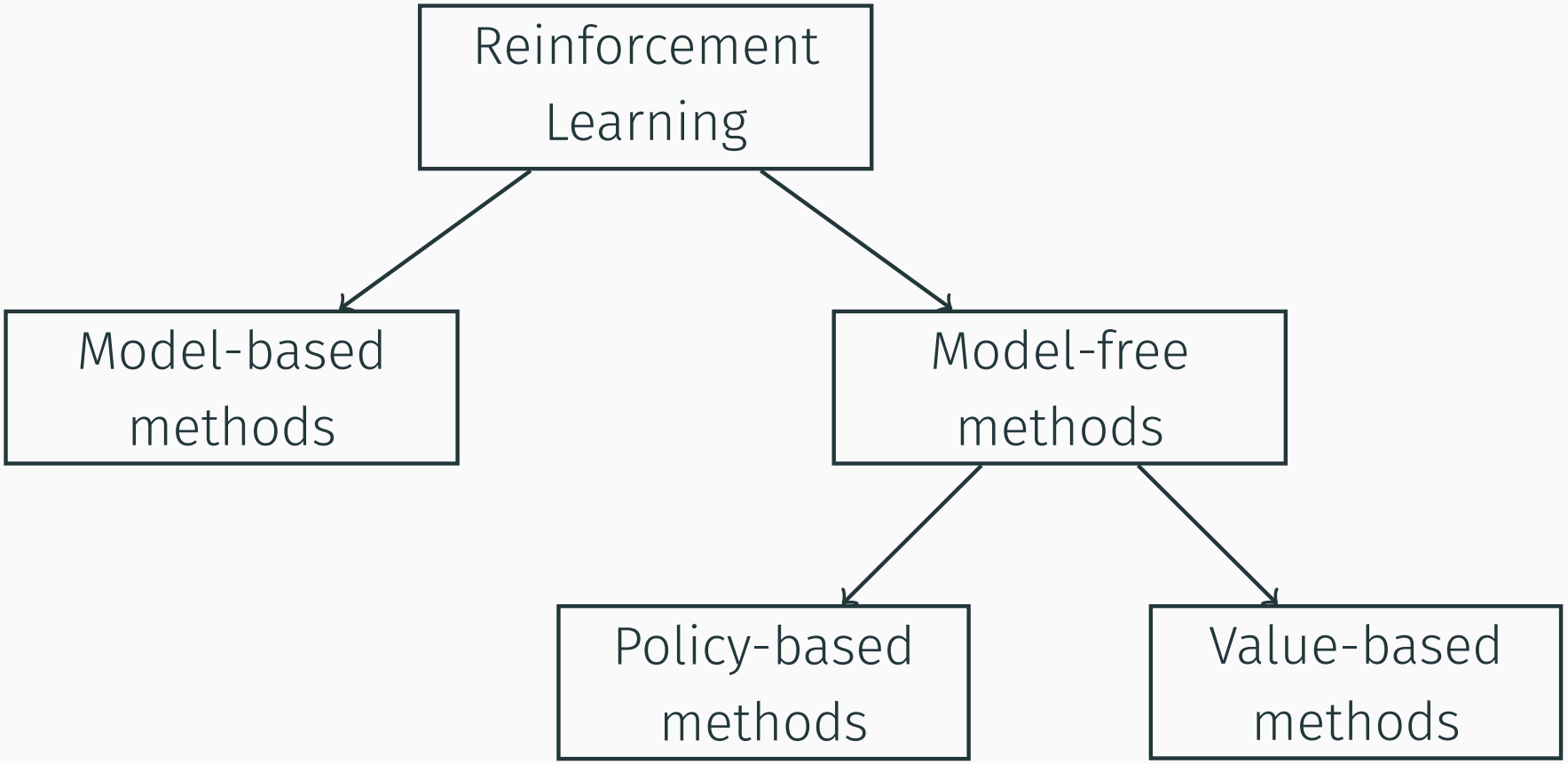
2. Artificial Neural Networks

3. Deep Q-Network

4. Quellen

# Reinforcement Learning

# Overview


Reinforcement Learning
Model-based methods
Model-free methods
Policy-based methods
Value-based methods

# Model-based methods

ORGANIC COMPUTING

- Agent lernt eine Modell des Environments
- Aktion $a_1$ in Zustand $s_1 \rightarrow$ Zustand $s_2$ und Reward $r_2$
- $\Rightarrow$ Verbesserung der Schätzungen von $T(s_2|s_1, a_1)$ und $R(s_1, a_1)$
- Sobald das Modell ausreichend gut ist $\rightarrow$ Policy

Beispiel: Value Iteration und Policy Extraction

# Model-free methods

ORGANIC COMPUTING

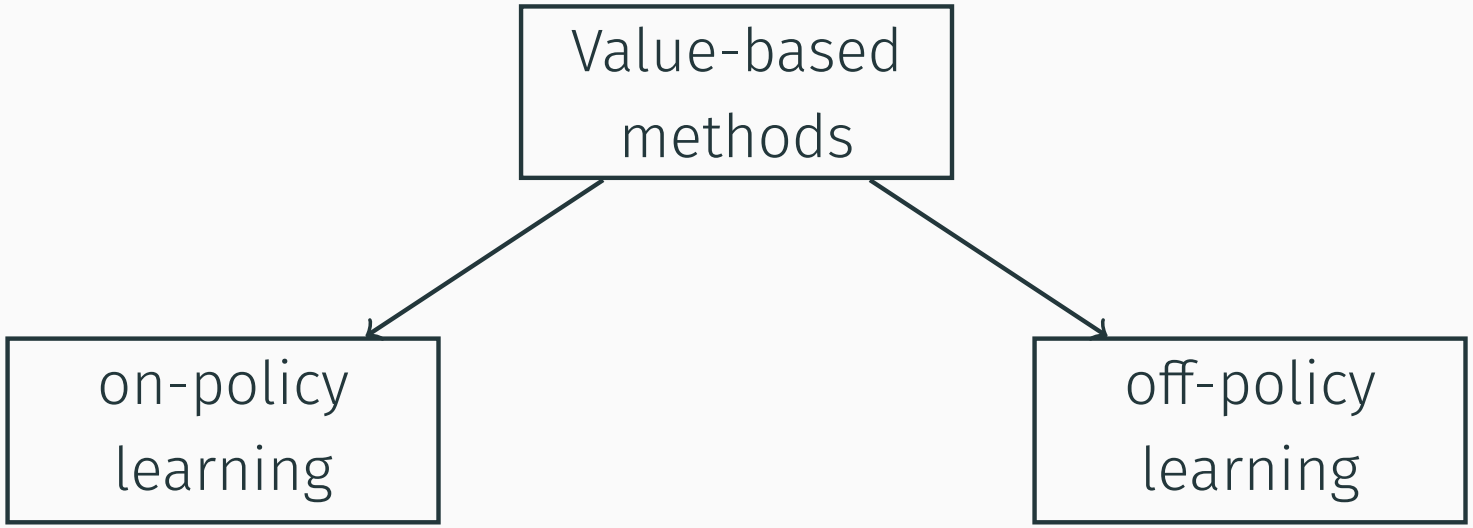
## Value-based methods

basiert auf *temporal difference learning*, lernt die Value Function $V^{\pi}$ oder $V^{*}$ oder die Q-Function $Q^{\pi}$ oder $Q^{*}$

## Policy-based methods

lernt direkt die optimale Policy $\pi^{*}$ (oder versucht die optimale Policy zu approximieren wenn die echte optimale Policy nicht errichbar ist)

# Value-based methods




lernt die Werte der Policy $\pi$

lernt die Werte einer anderen Policy

# on-policy learning

ORGANIC COMPUTING

- Agent lernt den Wert der Policy die benutzt wird um Entscheidungen zu treffen
- geschätzte Value Function wird durch die Ergebnisse der Aktionen geupdatet, die durch die Policy $\pi$ gewählt wurden

Beispiel: SARSA

# off-policy learning

ORGANIC COMPUTING

- geschätzte Value Function kann durch hypothetische Aktionen geupdatet werden (Aktionen die nicht explizit ausprobiert wurden)

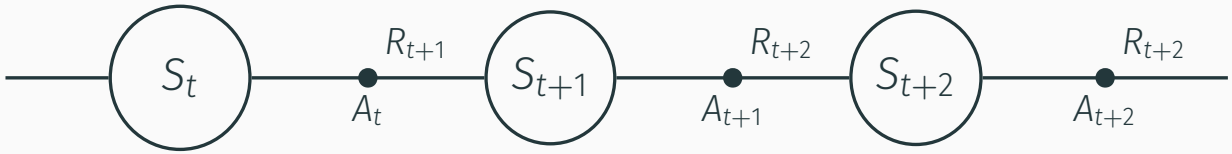
$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)]$$

- Der Agent lernt Strategien, die er nicht unbedingt in der Trainingsphase ausprobiert hat

Beispiel: Q-Learning

ORGANIC COMPUTING

- on-policy
- eine Episode besteht aus einer wechselnden Sequenz von Zuständen und Zustands-Aktions Paaren:




- Lernen der State-Action Paare:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

Quintupel: $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}) \rightarrow$ SARSA

# Q-learning vs SARSA

**SARSA:** lernt $Q^{\pi}(S_t, A_t)$

$$Q^{\pi}(S_t, A_t) \leftarrow Q^{\pi}(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1}) - Q^{\pi}(S_t, A_t)\Big]$$

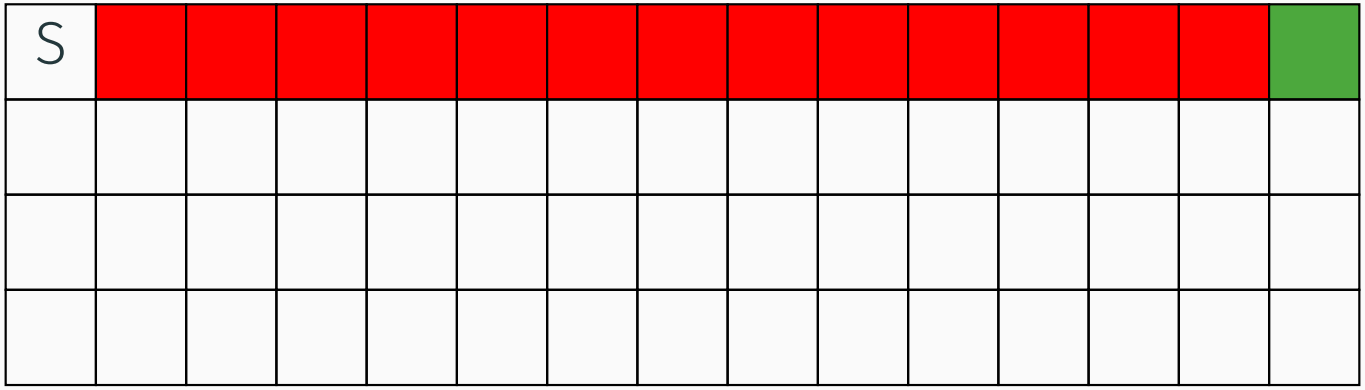
**Q-Learning:** lernt $Q^{*}(S_t, A_t)$

$$Q^{*}(S_t, A_t) \leftarrow Q^{*}(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_{a} Q^{*}(S_{t+1}, a) - Q^{*}(S_t, A_t)\big]$$

Beide konvergieren zu $Q^{\pi}$ bzw. $Q^{*}$, wenn genug Samples von jedem State-Action Paar gegeben sind

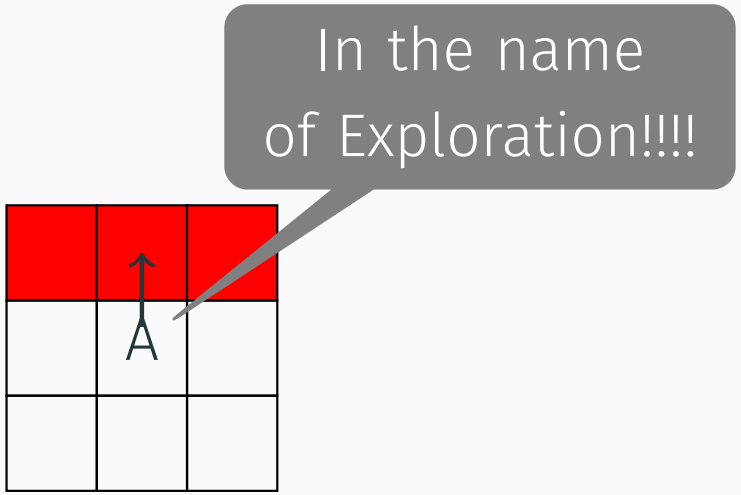
# Cliff Szenario

- Agent startet in S
- Ziel: grüner Endzustand (positiver Reward)
- Aber: Klippe auf dem Weg (negativer Reward und von vorne beginnen)

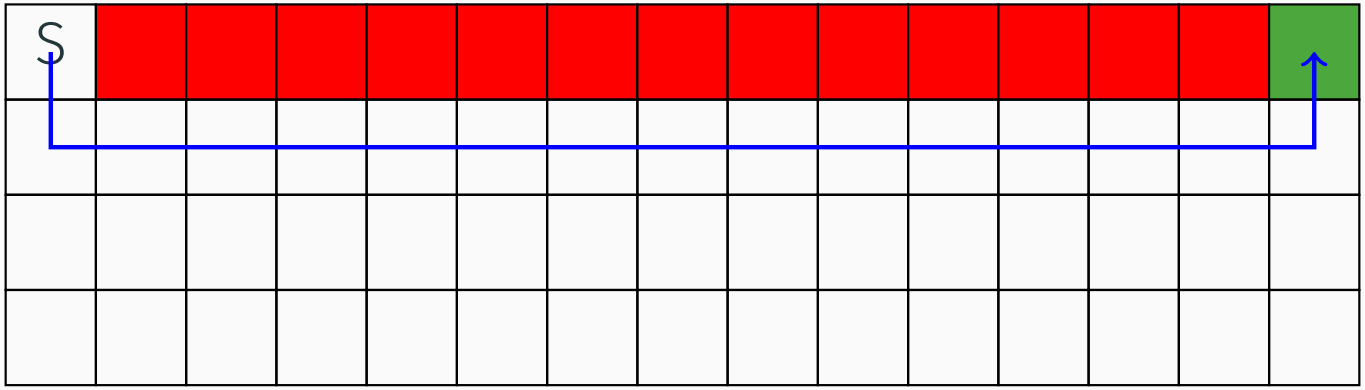


Q-Learning → höchster Action-Value

ABER: Exploration → zufällige Aktion

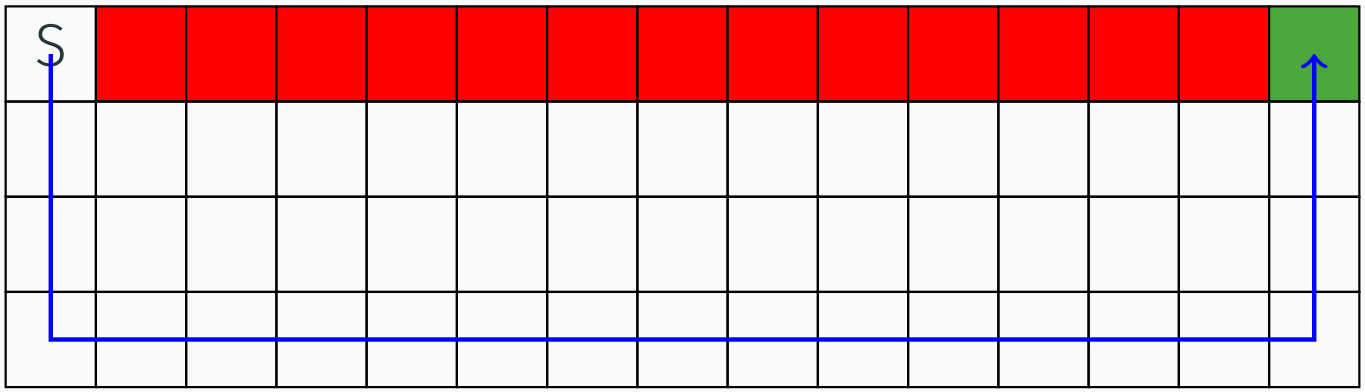


# Cliff Szenario - Q-learning




- optimaler/schnellster Weg
- unsicher

# Cliff Szenario - SARSA




- längerer Weg
- sicher
- gute online performance

# Pseudocode

**Algorithm 1:** Q-Learning

Initialize $Q(s, a)$ arbitrarily;
**repeat**
    Initialize $s$;
    Chose $a$ from $s$ using policy derived from $Q$ ($\epsilon$-greedy);
    **repeat**
        Take action $a$, observe $r$, $s'$;
        Chose $a'$ from $s'$ using policy derived from $Q$ ($\epsilon$-greedy);
        $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$;
        $s \leftarrow s'$;
        $a \leftarrow a'$;
    **until** *$s$ is terminal*;
**until** *$Q$ is converged*;

# Policy-based methods

ORGANIC COMPUTING

- direktes lernen der (optimalen) Policy
- Ein Beispiel in Markov Decision Processes:

$$\pi_\theta(s) = \max_{a \in A} \theta(s, a)$$

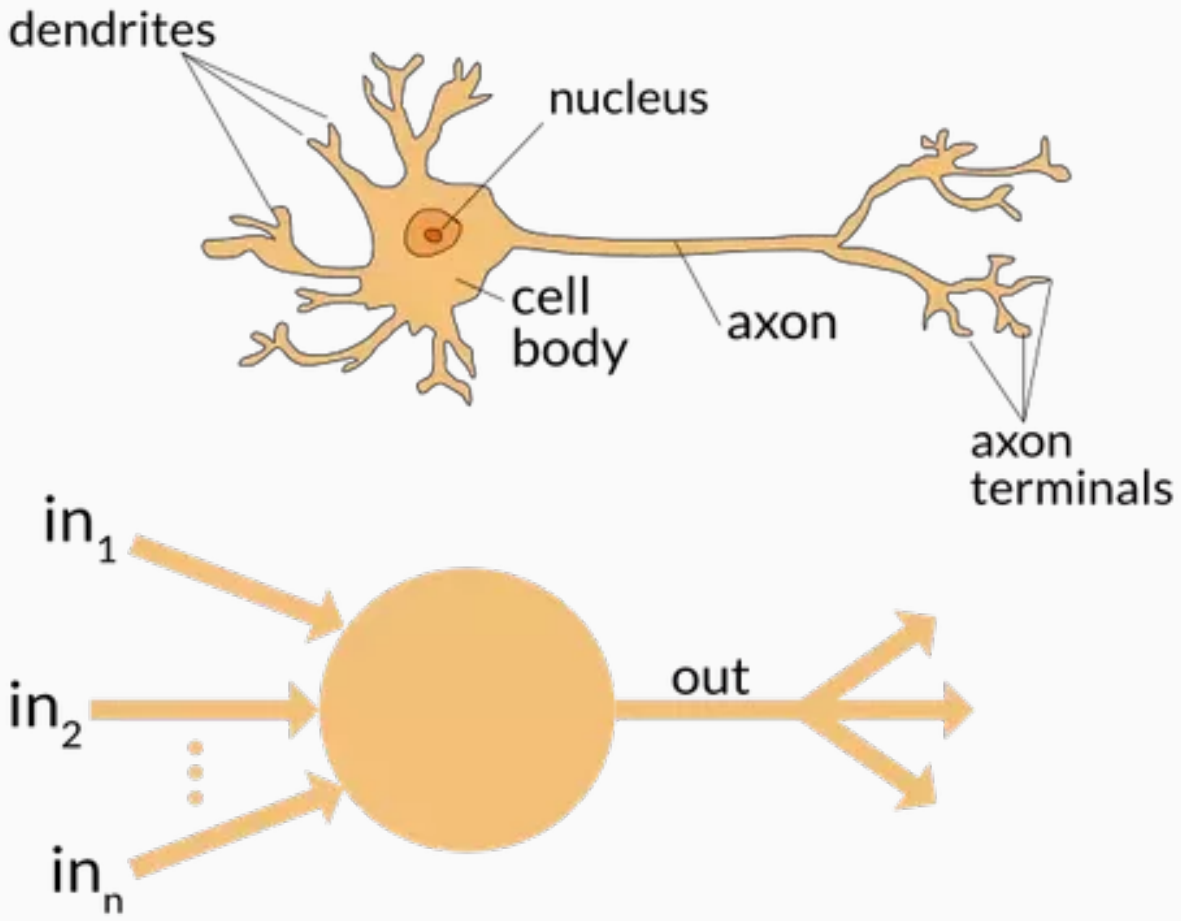
- $\theta$ sind hier beliebige Parameter und nicht zwangsläufig eine Value Function

Später mehr...

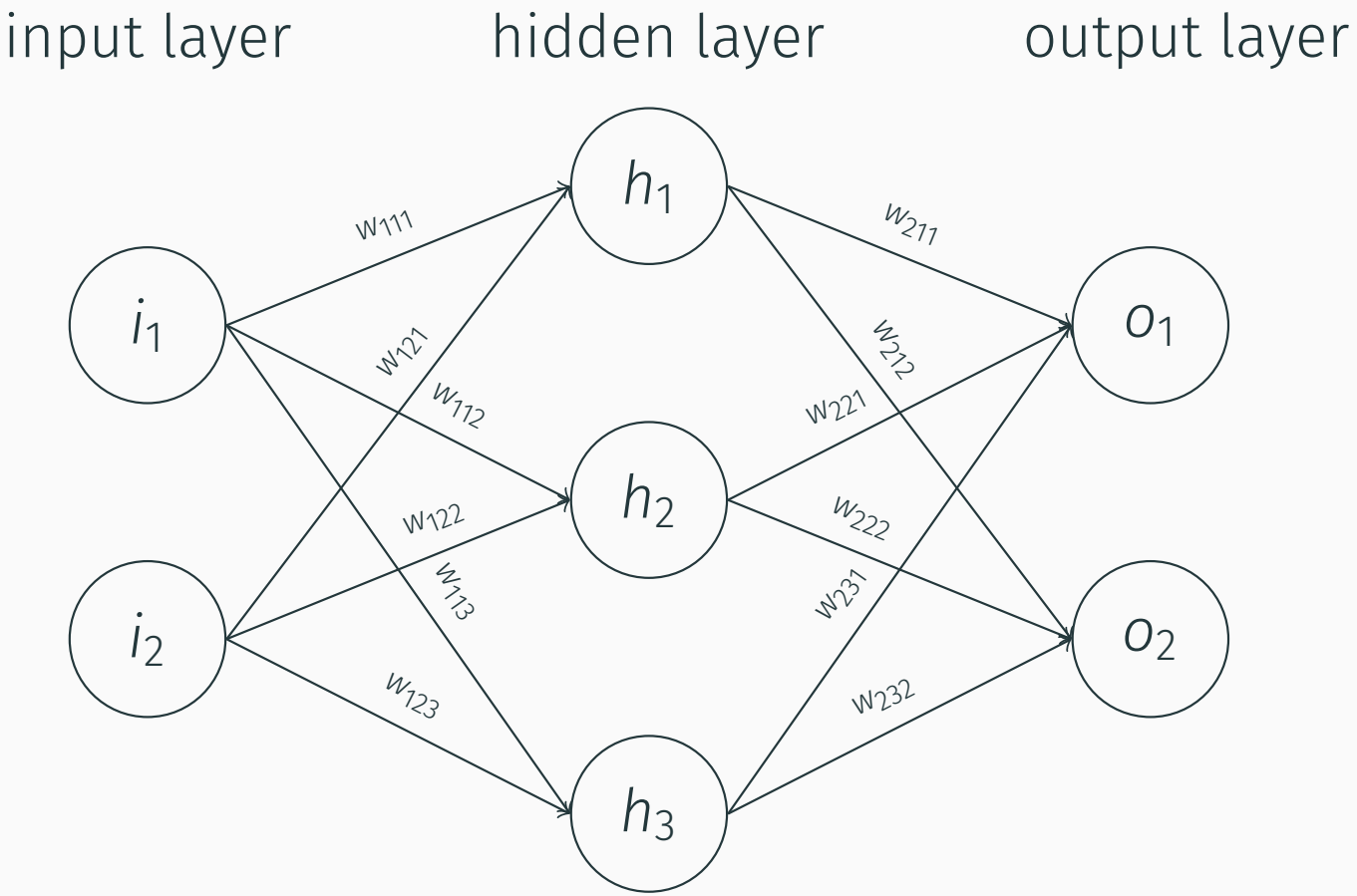
# Artificial Neural Networks

# Neuronen

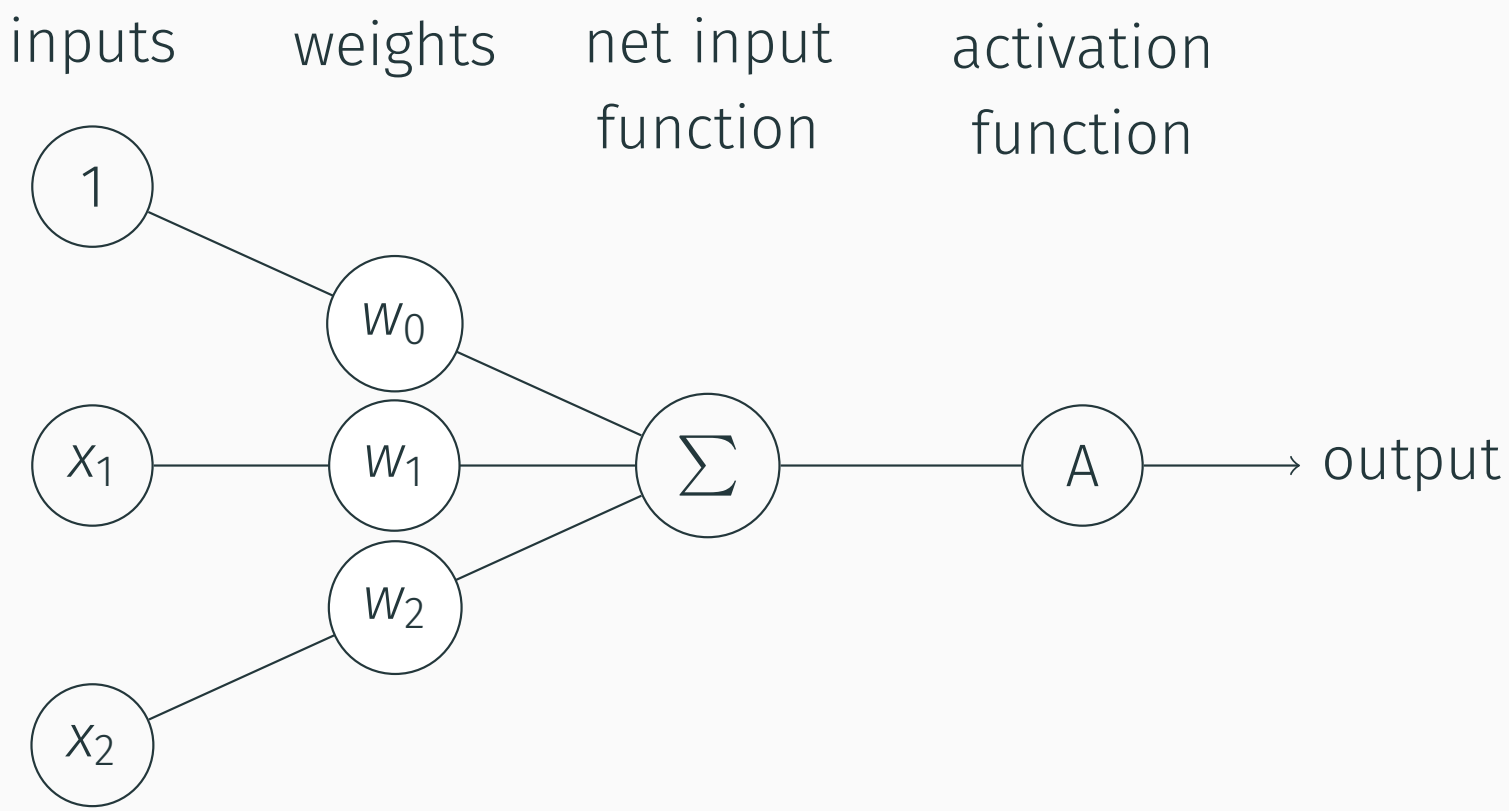



dendrites
nucleus
cell body
axon
axon terminals
in1
in2
inn
out

# Neuronales Netz



# Nodes


inputs
weights
net input function
activation function
1
$x_1$
$x_2$
$w_0$
$w_1$
$w_2$
$\sum$
A
output

# Nodes

ORGANIC COMPUTING

- Ein Knoten feuert wenn die Inputs bestimmte Anforderung erfüllen
- input function: Summiert alle Inputs ($input \times weight$)
- activation function: Entscheidet ob und wie stark ein Signal abgefeuert wird

# Gradient Descent

ORGANIC COMPUTING

- optimization function die Gewichte anhand dem Fehler den diese verursachen anpasst
- Gradient ist ein anderes Wort für Steigung
  - hier: Beziehung zwischen einzelnem Gewicht und Fehler des Netzwerks
- langsame Anpassung viele Gewichte $\rightarrow$ welches Signal hat welche Bedeutung
- Kettenregel:

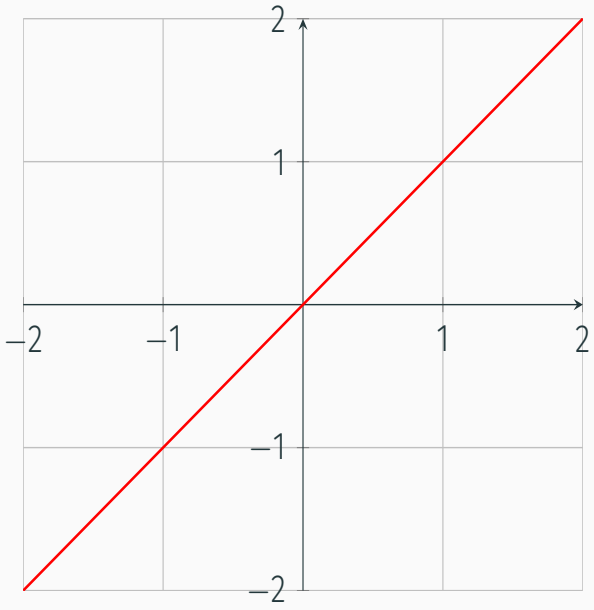
$$\frac{dError}{dweight} = \frac{dError}{dactivation} * \frac{dactivation}{dweight}$$

- Lernen = Gewichte so lange Anpassen, bis der auftretenden Fehler nicht weiter verringert werden kann

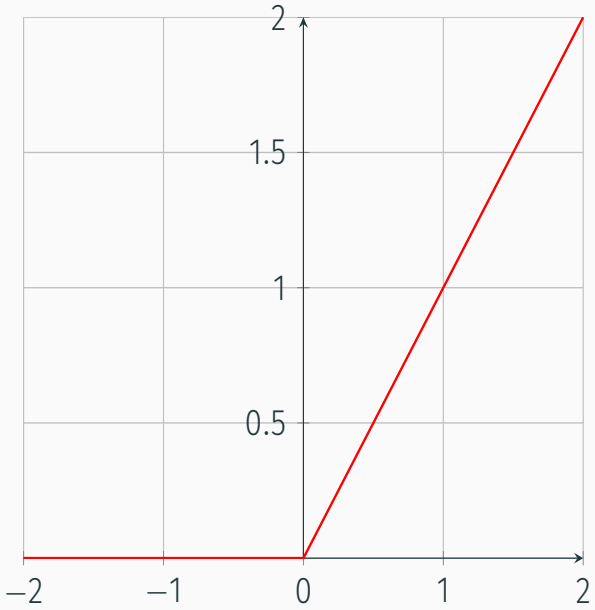
# Activation Functions

ORGANIC COMPUTING

## linear:




## ReLu (Rectifier Linear Unit):



# Error and Loss Function

ORGANIC COMPUTING

- meistens wird der Fehler als Differenz zwischen dem tatsächlichen und dem vorhergesagten Output definiert

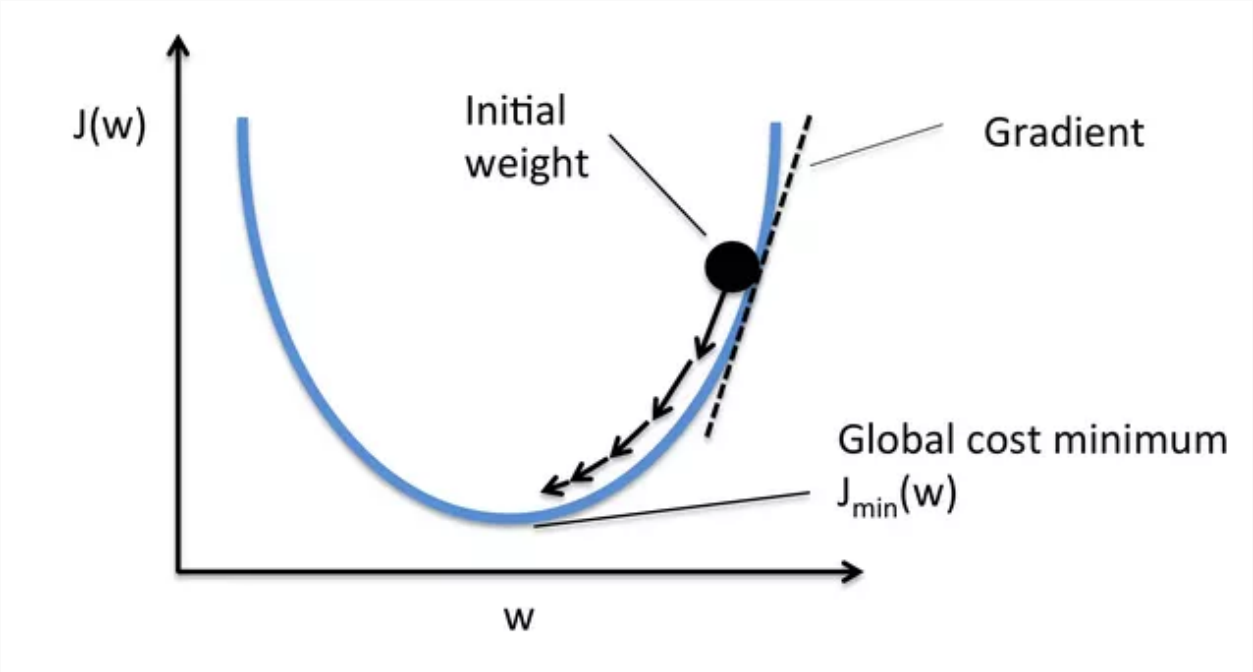
$$J(\theta) = p - \hat{p}$$

- Die Funktion die den Fehler berechnet wird Loss Function $J$ genannt
- unterschiedliche Funktionen $\rightarrow$ unterschiedliche Fehler
- häufig genutzte Loss Function: mean square error (MSE)

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (p_i - \hat{p}_i)^2$$

# Back Proragation

- $J(\theta) = J(w) =$ Funktion interner Parameter (Gewichte und Bias)
- Fehler wird von Layer zu Layer von hinten nach vorne durchgereicht



# keras



- high-level neural networks API
- im Hintergrund können verschiedene Bibliotheken laufen (Tensorflow, CNTK oder Theano)
- schnell und einfach zu bedienen
- www.keras.io

# keras - einfaches Netz

ORGANIC COMPUTING

```
1  from keras.models import Sequential
2  from keras.layers import Dense
3  from keras.optimizers import RMSprop
4
5  model = Sequential()
6
7  # input and first hidden layer
8  model.add(Dense(units=128, activation'relu', input_dim=2))
9  # second hidden layer
10 model.add(Dense(units=256, activation='relu'))
11 # output layer
12 model.add(Dense(units=8, activation='linear'))
13
14 model.compile(loss='mse',
15                optimizer=RMSprop(lr=0.00025))
16
17 # train network
18 model.fit(x_train, y_train, batch_size=32)
19
20 # predict on trained network
21 prediction = model.predict(x_test)
```

# Deep Q-Network

# Grenzen von standard Q-Learning

ORGANIC COMPUTING

- großer Zustandsraum und/oder Aktionsraum $\rightarrow$ sehr große Q-Tabelle
- $\Rightarrow$ approximieren der Q-Tablle durch Neuronales Netz
- NN kann sein Wissen von besuchten Zuständen auf nicht besuchte Zustände generalisieren
- abstrahieren von Mustern und verstehen von Aktionen auf Basis von bereits gesehen Mustern

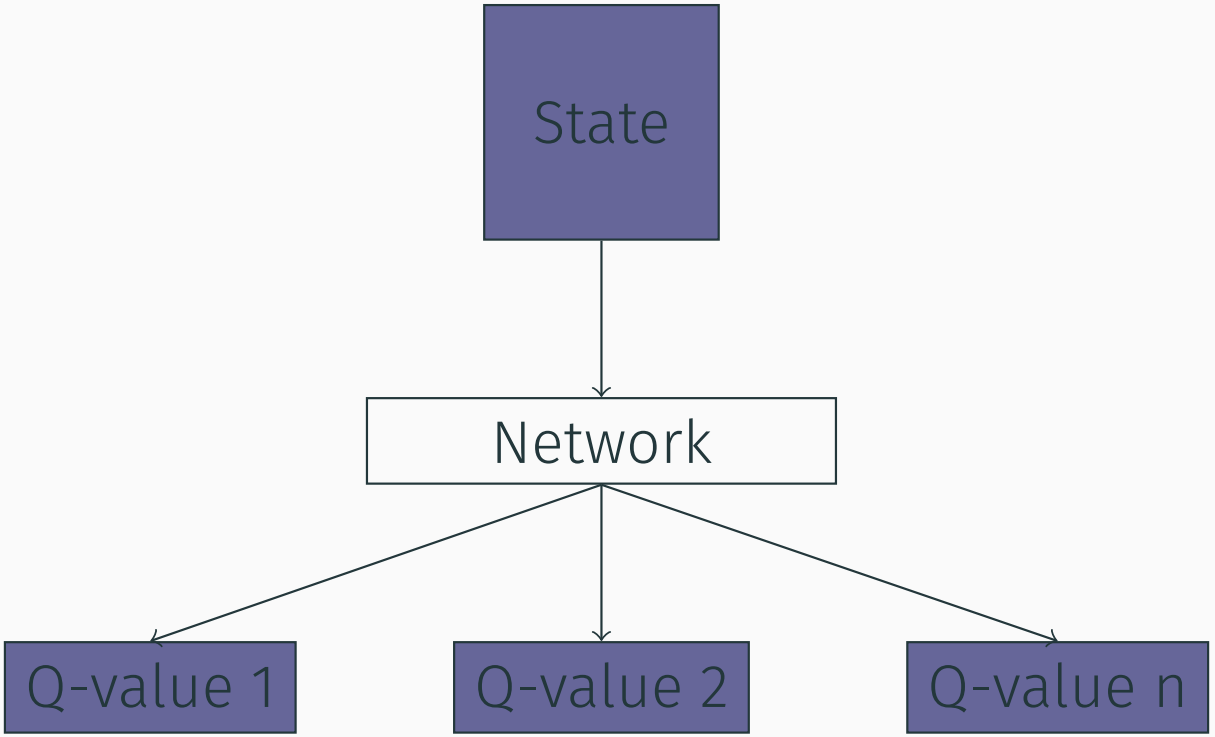
# Q-Network

- Zustandsraum repräsentiert als Vektor
- Loss Function:

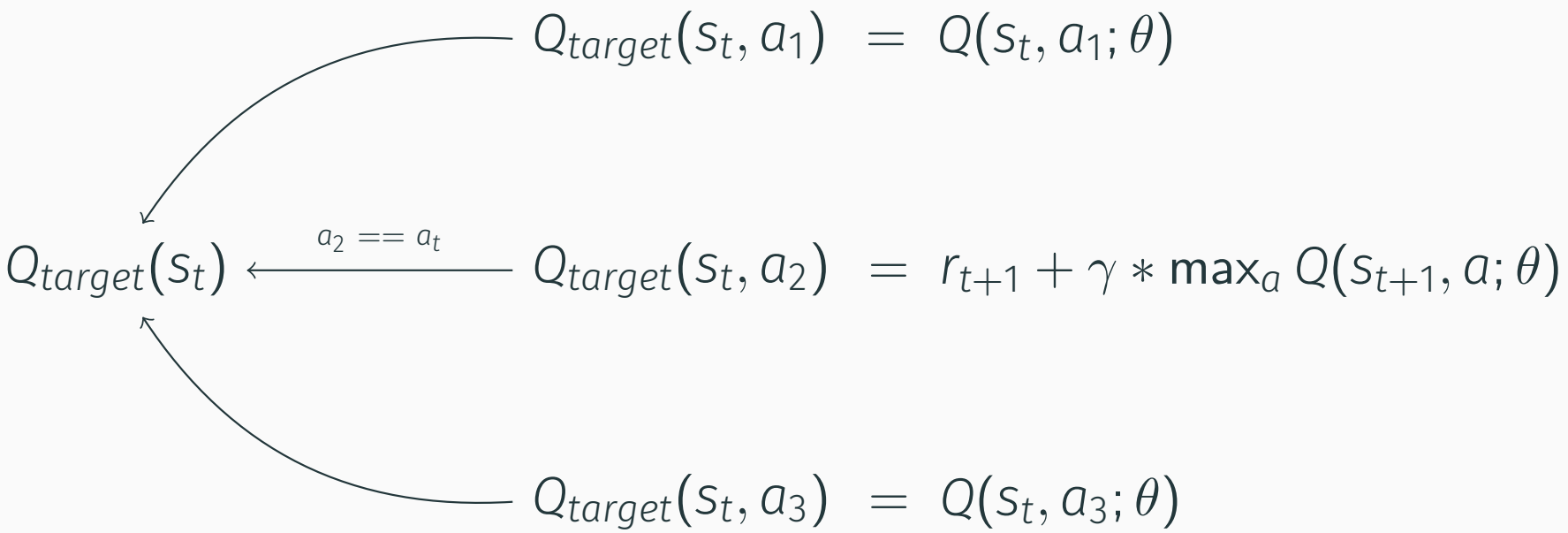
$$J(\theta) = \sum (Q - Q_{target})^2$$

$$Q(s_t, a_t; \theta) \leftarrow Q(s_t, a_t; \theta) + \alpha \Big[ \underbrace{r_t + \gamma \max_a Q(s_{t+1}, a; \theta)}_{target} - Q(s_t, a_t; \theta) \Big]_{TD-error}$$

# Modellierung von $Q(s, a)$



# Q-target


$Q_{target}(s_t, a_1) = Q(s_t, a_1; \theta)$
$Q_{target}(s_t)$
$a_2 == a_t$
$Q_{target}(s_t, a_2) = r_{t+1} + \gamma * \max_a Q(s_{t+1}, a; \theta)$
$Q_{target}(s_t, a_3) = Q(s_t, a_3; \theta)$

# instabiles Lernen

ORGANIC COMPUTING

- nicht linearer Funktionsapproximator (ANN) $\rightarrow$ instabiles Lernen/Divergenz
- Gründe:
  - Korrelation der einzelnen Beobachtungen
  - kleine Anpassungen des Q-Werts können zu signifikanten Veränderungen der Policy führen und so die Verteilung der Daten verändern
  - Korrelation der Action-Values($Q$) und der Target-Values

# Experience Replay

ORGANIC COMPUTING

- merken der letzten $N$ Übergänge $(s_t, a_t, r_{t+1}, s_{t+1}, done)$
- anstatt nur vom letzten Übergang zu lernen, zieht man in jedem Schritt ein zufälliges Minibatch (size = 32) aus dem experience replay
- Q-Learning Updates auf dem Minibatch
- FiFo

$$\Downarrow$$

entfernt Korrelation zwischen den einzelnen Beobachtungen und glättet Veränderungen in der Datenverteilung.
Übergänge werden öfters zum lernen genutzt $\rightarrow$ Dateneffizienz

# Target Network

ORGANIC COMPUTING

- In jedem Schritt verschieben sich die Werte des Q-Networks
- $\Rightarrow$ Feedback Loops zwischen den Target- und den vorausgesagten Q-Values
- zweites Neuronales Netz das während dem Training genutzt wird (Target-Network)
- Berechnet den Target-Q-Value für die Loss Function
- wird langsam und periodisch geupdatet (alle $C$ Schritte)

$\Downarrow$

Reduziert die Korrelation der Action-Values mit den Target-Values

# Error Clipping

ORGANIC COMPUTING

- festlegen des maximalen und minimalen Fehlers

$$\left[r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right] \in [-1, 1]$$

$$\Downarrow$$

stabileres Lernen

# Target Network - Loss Function

$\theta$ = Gewichte des Q-Netwerks

$\theta^-$ = Gewichte des Target-Netzwerks

Loss Function:

$$J(\theta) = \sum \left( r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2$$

ORGANIC COMPUTING

**Algorithm 2:** deep Q-learning with experience replay.

Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
**for** *episode=1,M* **do**
  **for** *t=1,T* **do**
    With probability $\epsilon$ select random action $a_t$
    otherwise select $a_t = \mathbf{argmax}_a\, Q(s_t, a; \theta)$
    Execute action $a_t$ and observe reward $r_{t+1}$ and state $s_{t+1}$
    Store transition $(s_t, a_t, r_{t+1}, s_{t+1}, done)$ in $D$
    Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1}, done)$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if } done \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    perform a gradient descent step on $\left(y_j - Q(s_j, a_j; \theta)\right)^2$ with respect to the network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$

# Quellen

# Bilder



- https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network
- https://www.quora.com/In-neural-networks-how-important-is-back-propagation-What-is-its-significance