



Automatic Control Chair, Kiel University

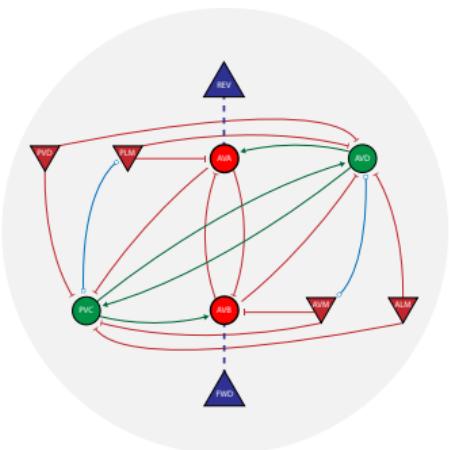
# Eine Anwendung des Reinforcement Learning auf biologische neuronale Netze zur Regelung dynamischer Systeme am Beispiel des inversen Pendels

Abschlussvortrag Bachelorarbeit

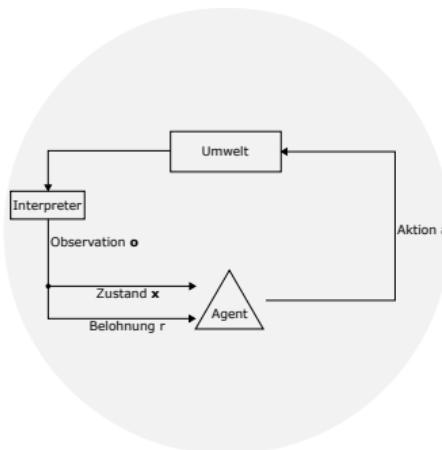
Jonas Helmut Wilinski

Abschlussvortrag Bachelorarbeit, Kiel (Germany), 14. September 2018

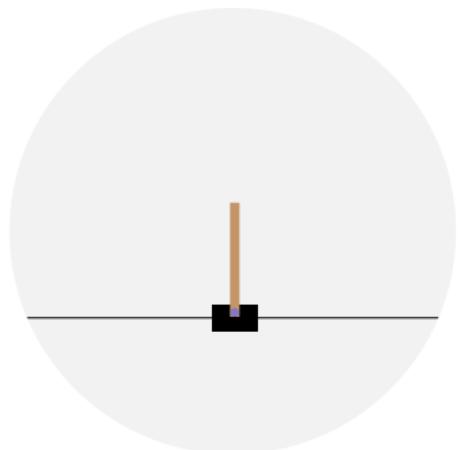
## Übersicht der Themen dieser Bachelorarbeit



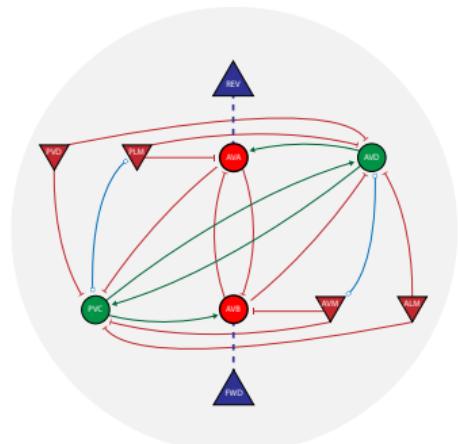
Biologische neuronale  
Netze



Reinforcement Learning



Simulation: inverses Pendel

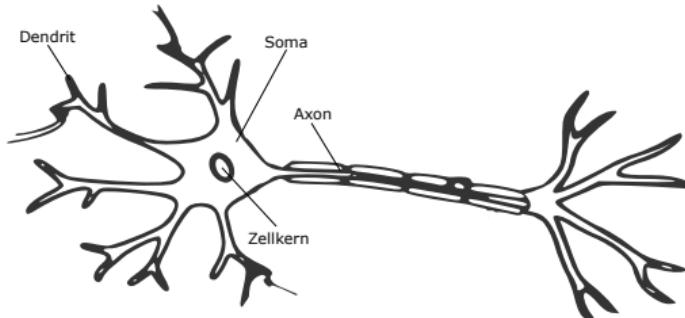


- ▶ Nervenzellen & Synapsen
- ▶ Biologisches neuronales Netzwerk
- ▶ Symmetrisches neuronales Netzwerk

## Biologische neuronale Netze

## Biologische Nervenzellen

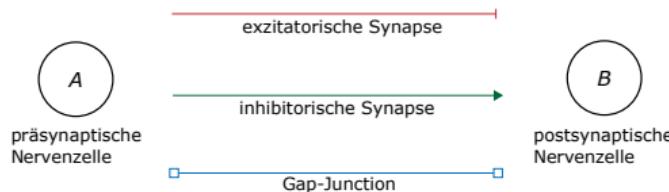
- Neuronale Netze biologischer Lebensformen bestehen aus Nervenzellen und Synapsen bzw. Gap-Junctions
- Neuronale Nervenzellen bestehen aus Dendrit, Soma und Axon. Das Soma enthält den Zellkern und verarbeitet ankommende Informationen.
- Der Dendrit dient der Reizaufnahme von Signalen anderer Nervenzellen übertragen durch Synapsen und Gap-Junctions.
- Das Axon ist ein Nervenzellenfortsatz zur Weiterleitung der Signale von Soma an weitere Synapsen und Gap-Junctions.



## Biologische Synapsen

Eine Synapse dient der Informationsübertragung zwischen zwei Nervenzellen und wird in drei Kategorien unterteilt:

- Exzitatorische Synapsen (chemisch) - wirken anregend auf die postsynaptische Nervenzelle und übertragen die Informationen.
- Inhibitorische Synapse (chemisch) - wirken hemmend auf die postsynaptische Nervenzelle und übertragen die Informationen negativ.
- Gap-Junctions (elektrisch) - wirken bidirektional als synchronisierende Verbindung und gleichen Potentialunterschiede aus.



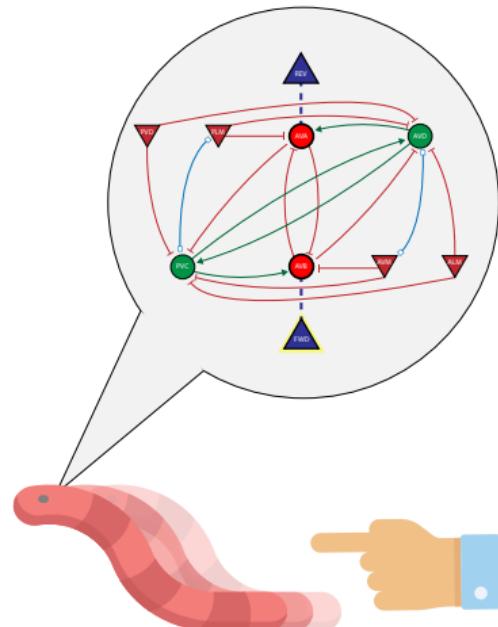
## Biologisches neuronales Netz des C. Elegans

Der *Touch-Withdrawal Circuit* des *C. Elegans* dient des reflexartigen Zurückschnellens bei äußeren Stimuli.

▲ **Motor-Neuronen** sorgen für die Anregung von Muskelgruppen, um bspw. eine Vorwärtsbewegung umzusetzen.

▼ **Sensor-Neuronen** nehmen Signale durch äußere Stimuli auf und übersetzen diese auf Aktionspotentiale innerhalb des neuronalen Netzes.

● **Inter-Neuronen** nehmen durch Synapsen und Gap-Junctions Aktionspotentiale entgegen und summieren diese auf, bis ein Schwellwert erreicht wird - das Inter-Neuron feuert.



## Neuronale Dynamik durch das Leaky Integrate and Fire - Modell

Die neuronale Dynamik und die damit verbundenen Vorgänge im neuronalen Netz werde durch das Leaky Integrate and Fire (LIF) - Modell dargestellt.

Berechnung des Membranpotentials bei anliegenden Synapsen- und Gap-Junction-Strömen:

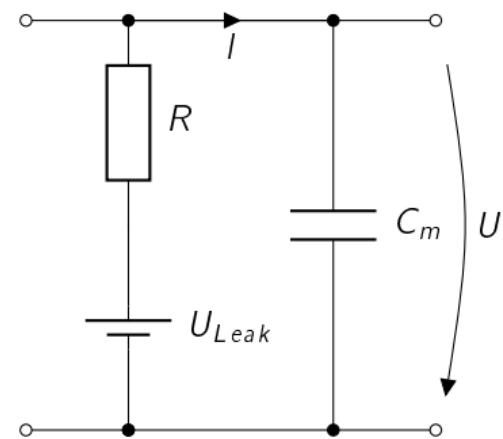
$$\frac{dU}{dt} = \frac{G_{Leak}(U_{Leak} - U) + I_{in}}{C_m} \text{ mit} \quad (1)$$

$$I_{in} = \sum_{i=1}^n I_{Stimuli} + \sum_{i=1}^n I_{Syn} + \sum_{i=1}^n I_{Gap}. \quad (2)$$

Berechnung der Synapsen- und Gap-Junction-Ströme:

$$I_{Syn} = \frac{\omega}{1 + \exp^{\sigma(u_{pre} + \mu)}} (E - u_{post}), \quad (3)$$

$$I_{Gap} = \hat{\omega}(u_{post} - u_{pre}). \quad (4)$$

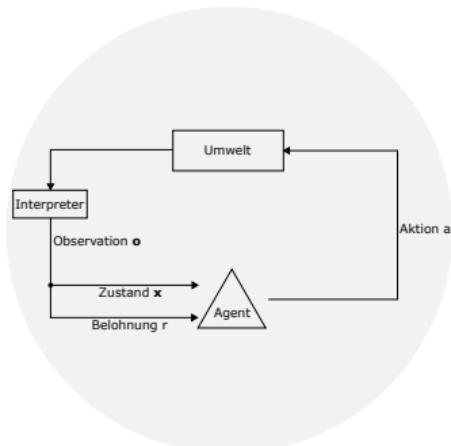


**Figure 1:** Ersatzschaltbild der Zellmembran

## Motivation: Nutzung eines biologischen neuronalen Netzes

Klassische Anwendungen neuronaler Netze stützen sich auf künstlich erstellte Konstrukte, welche durch

- + Profitieren von natürlicher Evolution.
- + Starke Anpassungsfähigkeit durch zwei Eingängen und einem Ausgang.
- + Einfache Erweiterbarkeit und Ausbau des Netzwerks durch Lernalgorithmen.
- + Kurze Trainingsphasen.
- Implementationsaufwand und Simulation aufwendig.
- Wenig Fachliteratur über die Verbindung von biologischen neuronalen Netzen mit Methoden des Reinforcement Learning.
- Hohe Anzahl an zu simulierenden Parametern.



## Reinforcement Learning

- Deep Learning und Grundlagen von Lernalgorithmen
- Vorteile des Belohnungssystems
- Reinforcement Learning durch Random-Search und genetischen Algorithmen

## Reinforcement Learning und das Belohnungssystem

*Reinforcement Learning* aus dem Bereich des *Deep Learning* ist eine Abwandlung des überwachten Lernens mit der Erweiterung eines Belohnungssystems.

- Der Agent ist in der Lage, eine Aktion im gegebenen Aktionsraum zu tätigen.
- Die Aktion hat Auswirkungen auf die Umwelt bzw. Simulation, der neue Zustand der Simulation wird interpretiert und als Observationsvektor  $\mathbf{o}$  ausgegeben.
- Aus der Observation  $\mathbf{o}$  gehen zum einen der Zustandsvektor  $\mathbf{x}$  zum anderen die Belohnung  $r$  hervor.
- Diese Größen werden von dem Agenten interpretiert und dienen als neue Entscheidungsgrundlage der nächsten Aktion.

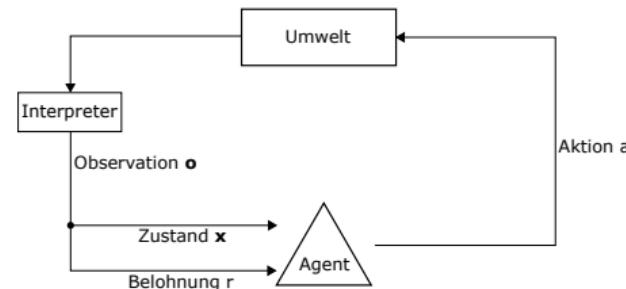


Figure 2: Ablaufplan: Reinforcement Learning

## Anwendung auf das biologische Netzwerk

Durch *Reinforcement Learning* können Parameter im biologischen neuronalen Netz optimiert werden, sodass es zu Feuer-Events kommt.

- Um die neuronale Dynamik durch das *LIF*-Modell richtig abzubilden, damit es bei äußeren Stimuli zu Feuer-Ereignissen kommt, werden insgesamt 46 Parameter simuliert.
- ...
- Durch verschiedene Suchalgorithmen können Parametersätze simuliert und die Performance gemessen werden.

## Suchalgorithmen

### Random-Search

- Parameter werden in einer gegebenen Gleichverteilung mit festen Grenzen pro Episode zufällig generiert.
- Die Performance jedes Parametersatzes wird anhand der erhaltenen Belohnung gemessen.
- Nach Ablauf der Simulationszeit werden diejenigen Parameter gespeichert, welche die höchste Belohnung erhalten haben.

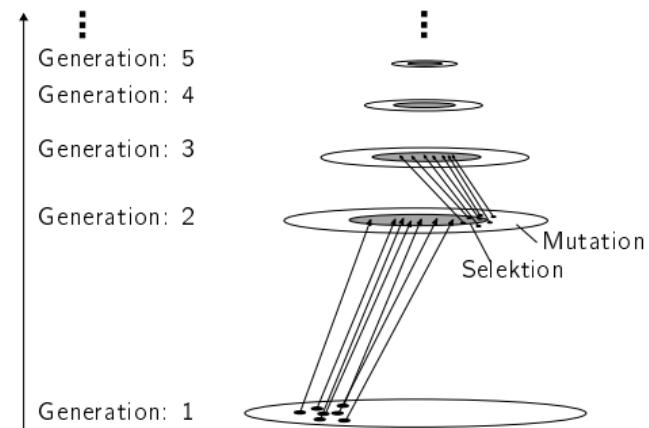
### Weights

- Im Gegensatz zu Random-Search wird Weights zur Optimierung des neuronalen Netzes mit bereits bestehenden Parametern verwendet.
- Es werden bestehende Synapsen mit einem pro Episode zufällig gewählten Gewicht versehen, welches die Reizübertragung schwächt.

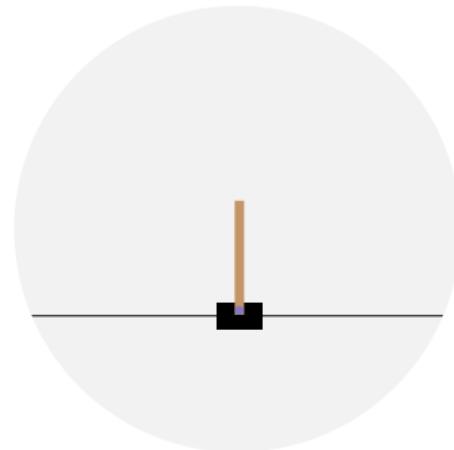
## Suchalgorithmen

### Genetische Algorithmen

- Die erste Generation an Parametern wird in einer gegebenen Gleichverteilung mit festen Grenzen pro Episode zufällig generiert.
- Nach Durchlauf der ersten Generation werden wenige Episoden mit guter Belohnung in eine Selektion aufgenommen.
- Die Grenzen der Gleichverteilung werden entsprechend der Maxima/Minima der Selektion angepasst.
- Die Simulation konvergiert auf ein lokales Maximum für jeden Parameter.



**Figure 3:** Grafische Darstellung des genetischen Algorithmus.



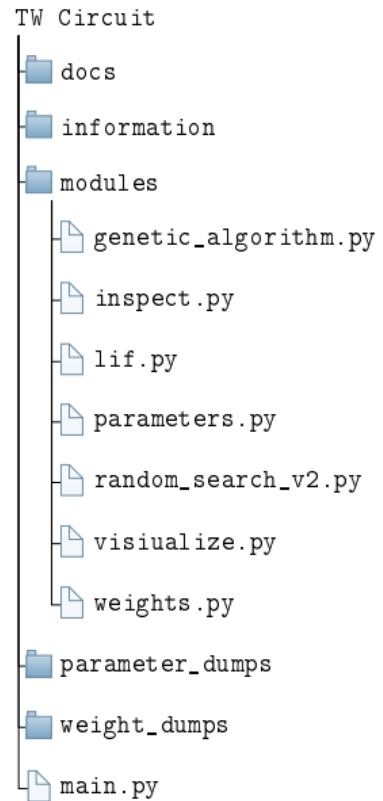
## Simulation: inverses Pendel

- Implementierung des Simulators
- Simulationsumgebung: OpenAI Gym
- Grafische Darstellung und Auswertung der Simulation

## Aufbau des Simulators

Um die neuronale Dynamik des biologischen neuronalen Netzes zu simulieren und die Aktionen auf die Umwelt zu übertragen, wurde ein Simulator in der Programmiersprache Python geschrieben.

- ...

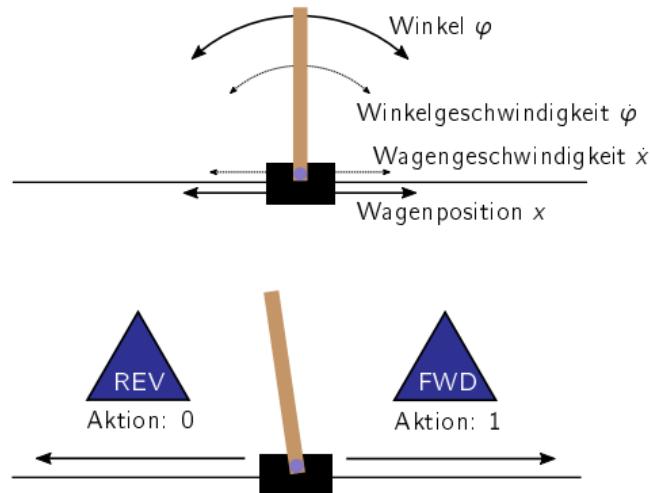


# Implementierung und Simulation des inversen Pendels

## Simulationsumgebung CartPole\_v0

Um die Methode des *Reinforcement Learning* auf biologische neuronale Netze anzuwenden, wird die Simulationsumgebung `CartPole_v0` von OpenAi Gym gewählt.

- Das inverse Pendel auf dem Wagen kann sich in Vorwärts- und Rückwärtsrichtung bewegen.
- Pro Zeitschritt kann ein Schritt in die jeweilige Richtung erfolgen (Aktion: 0 oder 1).
- Der Observationsvektor  $\mathbf{o}$  gibt die Zustandsgrößen  $\varphi$ ,  $\dot{\varphi}$ ,  $x$  und  $\dot{x}$  wieder.
- Es wird ebenfalls eine Belohnung  $r$  zurückgegeben.
- ...



## Simulationsläufe: Auslagerung auf Cloud Server

- Durch rechenintensive Simulationen mit den Algorithmen Random-Search und Weights wird ein externer Server benötigt.
- Eine virtuelle Instanz in der Google Cloud Platform wurde angemietet, um Simulationszeiten über 12 Stunden zu ermöglichen.
- Es wurden Simulationen mit über 10 Mio. Episoden gefahren und analysiert.
- ...



Google Cloud Platform

## Visualisierung der Simulationsergebnisse

- ...

## Simulation des inversen Pendels (Video)

Das Video zeigt das inverse Pendel, welches durch das biologische neuronale Netz und bereits trainierten Parametern stabilisiert wird.

## Zusammenfassung & Ausblick

- ...

Vielen Dank für Ihre Aufmerksamkeit.

## Color and thickness

- Emphasizing text cell
  - \cEmph provides **this text**
  - \ctEmph provides **this text**
  - \tEmph provides **this text**
  - \bEmph provides **this text**
  - \btEmph provides **this text**

## Some examples

```
\begin{displaybox}{0.995\textwidth}
```

Example 1: use 0.995\textbackslash textwidth for full width box

```
\end{displaybox}
```

Example 1: use 0.995\textwidth for full width box

```
\begin{alignbox}{0.75\textwidth}
```

Use for math related environments including text line (blank line below)

```
\begin{align}
```

$$y = x^2$$

```
\end{align}
```

```
\end{alignbox}
```

Use for math related environments including text line (blank line below)

$$y = x^2 \quad (5)$$

## Some examples

```
\begin{alignbox}{0.5\textwidth}
  \begin{align}
    y = x^2
  \end{align}
\end{alignbox}
```

$$y = x^2 \quad (6)$$

This shows the use of an `\texttt{alignbox}` environment

```
\begin{inlinebox}{1cm}
  abc
\end{inlinebox}
```

This shows the use of an `inlinebox` environment

abc

## Inversionsbasierte Trajektorienplanung

- **Differenzielle Flachheit**<sup>[fliess:95]</sup>

Ein System  $\dot{x} = f(x, u)$  wird **differenziell flach** genannt, wenn ein so genannter **flacher Ausgang**  $y = h(x, u)$ ,  $\dim y = \dim u$  existiert, so dass

$$x(t) = \theta_x(y, \dot{y}, \dots, y^{(\beta)}), \quad u(t) = \theta_u(y, \dot{y}, \dots, y^{(\beta+1)}).$$

⇒ Differenzielle Zustands- und Eingangsparametrierung

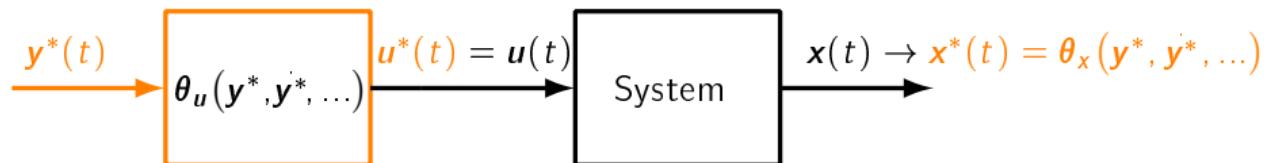
## Inversionsbasierte Trajektorienplanung

- **Differenzielle Flachheit**<sup>[fliess:95]</sup>

Ein System  $\dot{x} = f(x, u)$  wird **differenziell flach** genannt, wenn ein so genannter **flacher Ausgang**  $y = h(x, u)$ ,  $\dim y = \dim u$  existiert, so dass

$$x(t) = \theta_x(y, \dot{y}, \dots, y^{(\beta)}), \quad u(t) = \theta_u(y, \dot{y}, \dots, y^{(\beta+1)}).$$

### ⇒ Differenzielle Zustands- und Eingangsparametrierung



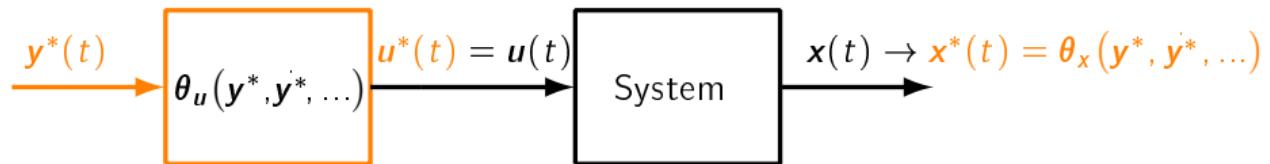
## Inversionsbasierte Trajektorienplanung

- **Differenzielle Flachheit**<sup>[fliess:95]</sup>

Ein System  $\dot{x} = f(x, u)$  wird **differenziell flach** genannt, wenn ein so genannter **flacher Ausgang**  $y = h(x, u)$ ,  $\dim y = \dim u$  existiert, so dass

$$x(t) = \theta_x(y, \dot{y}, \dots, y^{(\beta)}), \quad u(t) = \theta_u(y, \dot{y}, \dots, y^{(\beta+1)}).$$

⇒ Differenzielle Zustands– und Eingangsparametrierung



⇒ Methodische Übertragung auf **verteilt-parametrische Systeme**

# Eine Anwendung des Reinforcement Learning auf biologische neuronale Netze zur Regelung dynamischer Systeme am Beispiel des inversen Pendels

## Abschlussvortrag Bachelorarbeit

### Contact data

Jonas Helmut Wilinski

Chair of Automatic Control  
Faculty of Engineering  
Kiel University

 <http://www.control.tf.uni-kiel.de>

 stu118261@mail.uni-kiel.de