



Lehrstuhl für Regelungstechnik, Christian-Albrechts-Universität Kiel

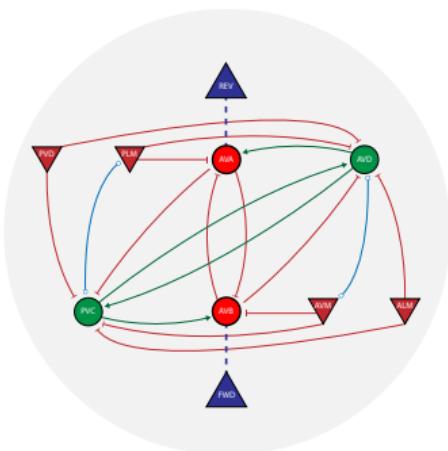
# Eine Anwendung des Reinforcement Learning auf biologische neuronale Netze zur Regelung dynamischer Systeme am Beispiel des inversen Pendels

Abschlussvortrag Bachelorarbeit

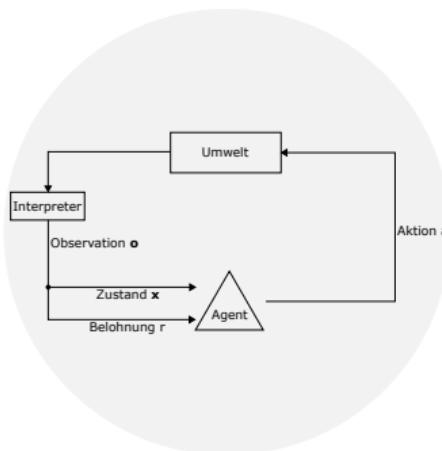
Jonas Helmut Wilinski

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

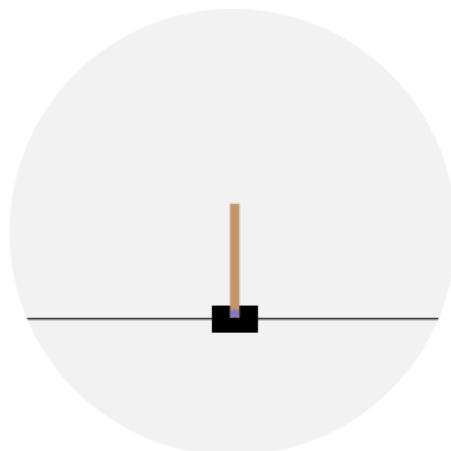
## Übersicht der Themen dieser Bachelorarbeit



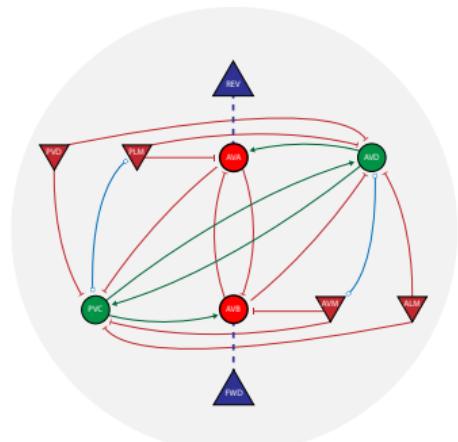
Biologische neuronale  
Netze



Reinforcement Learning



Simulation: inverses Pendel



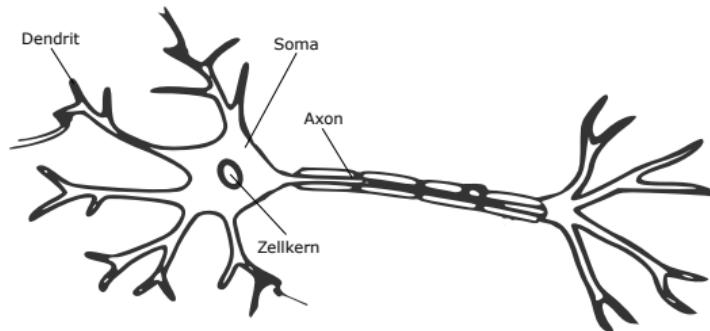
- ▶ Nervenzellen & Synapsen
- ▶ Biologisches neuronales Netzwerk
- ▶ Simulation neuronaler Dynamik

## Biologische neuronale Netze

## Biologische Nervenzellen

Neuronale Netze biologischer Lebensformen bestehen aus Nervenzellen und Synapsen bzw. Gap-Junctions.

- Der **Dendrit** dient der Reizaufnahme von Signalen anderer Nervenzellen übertragen durch Synapsen und Gap-Junctions.
- Das **Soma** enthält den Zellkern und verarbeitet ankommende Informationen.
- 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.



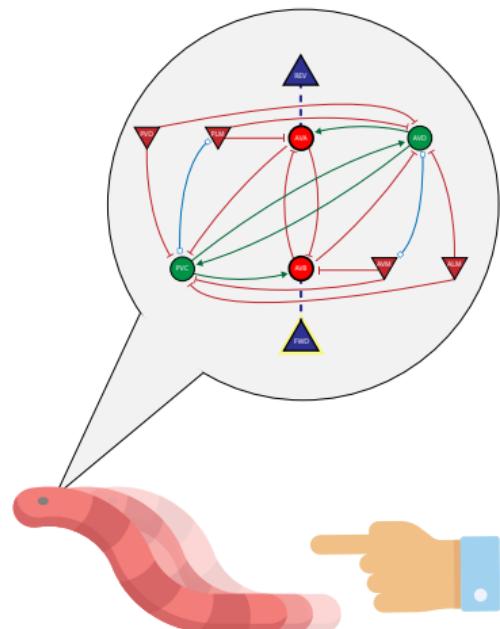
**Sensor-Neuronen** nehmen Signale durch äußere Stimuli auf und übersetzen diese auf Aktionspotentiale innerhalb des neuronalen Netzes.



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



**Motor-Neuronen** sorgen bei entsprechenden Feuer-Events für die Anregung von Muskelgruppen, um bspw. eine Vorwärtsbewegung umzusetzen.



## 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:

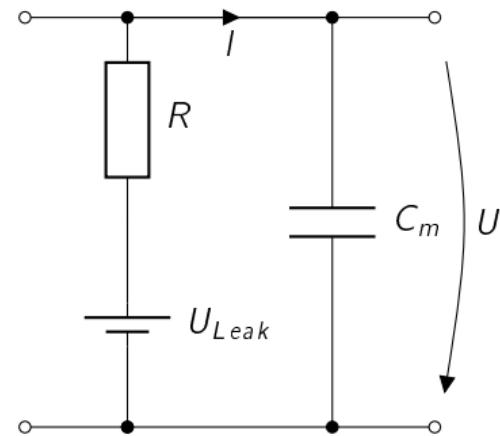
$$\frac{dU}{dt} = \frac{G_{\text{Leak}}(U_{\text{Leak}} - U) + I_{in}}{C_m} \quad \text{mit} \quad (1)$$

$$I_{in} = \sum_{i=1}^n I_{\text{stimuli}} + \sum_{i=1}^n I_{\text{syn}} + \sum_{i=1}^n I_{\text{gap}}. \quad (2)$$

Berechnung der Synapsen- und Gap-Junction-Ströme:

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

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

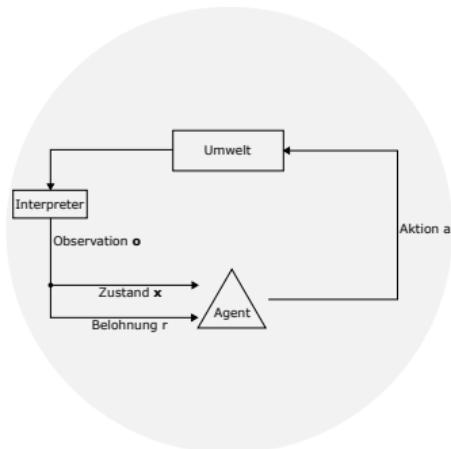


**Abbildung 1:** Ersatzschaltbild der Zellmembran

## Motivation: Nutzung eines biologischen neuronalen Netzes

Klassische Anwendungen des *Deep Learning* durch neuronale Netze stützen sich auf künstlich erstellte Modelle. Die Nutzung bereits bestehender Netzwerke birgt einige Vorteile:

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



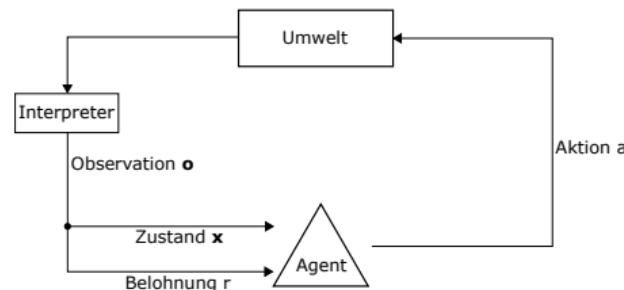
## Reinforcement Learning

- 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.

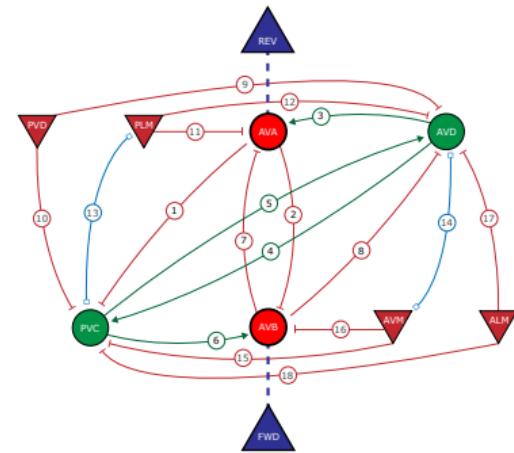


**Abbildung 2:** Funktionweise des Reinforcement Learning

## Anwendung auf das biologische Netzwerk

Durch *Reinforcement Learning* können Parameter im biologischen neuronalen Netz optimiert werden, sodass es zu Feuer-Events kommt und ein gegebenes System stabilisiert werden kann.

- Es werden insgesamt 46 Parameter generiert und optimiert, um die neuronale Dynamik richtig abzubilden.
- Die Belohnung nach jeder Episode (Simulationslauf eines bestimmten Parametersatzes) gibt Aufschluss über die Güte der generierten Parameter.
- Es erfolgt eine nachgelagerte Optimierung durch weitere Gewichtung der Synapsen und Gap-Junctions.



**Abbildung 3:** Biologisches neuronales Netzwerk mit Synapsengewichtung.

## Suchalgorithmen

### Random-Search

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

- ▶ Erfolgswahrscheinlichkeit hoch
- ▶ Ressourcenaufwand hoch

### Weights

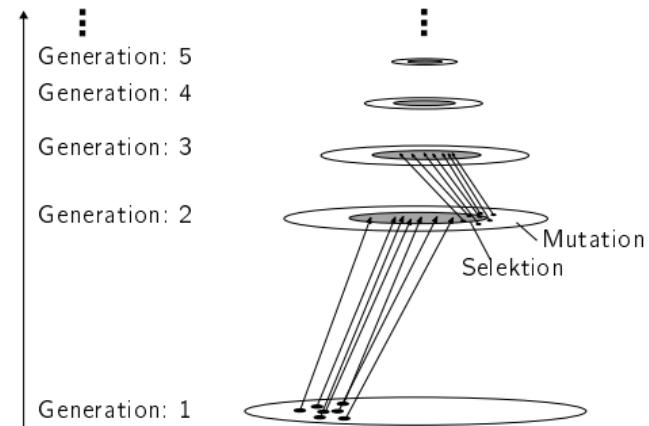
- Im Gegensatz zu Random-Search wird Weights zur Optimierung des neuronalen Netzes mit bereits bestehenden Parametern verwendet.
- Es werden bestehende Synapsen mit einem pro Episode zufällig gewählten Gewicht versehen, welches die Reizübertragung schwächt.
- Die Güte des Gewichtsatzes wird ebenfalls anhand der erhaltenen Belohnung gemessen.
- Nach Ablauf der Simulationszeit werden die Gewichte gespeichert, welche die höchste Belohnung erhalten haben.

- ▶ Erfolgswahrscheinlichkeit hoch
- ▶ Ressourcenaufwand hoch

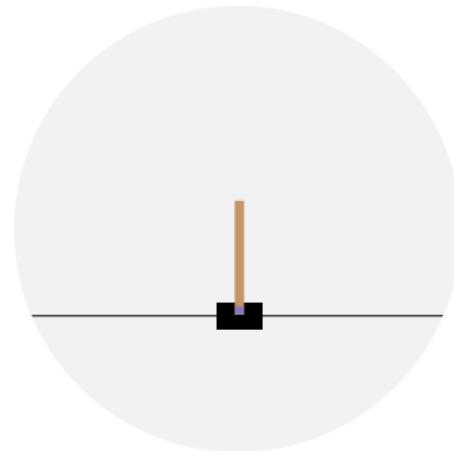
## Suchalgorithmen

### 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 Belohnungs-Maximum für jeden Parameter.



**Abbildung 4:** Grafische Darstellung des genetischen Algorithmus.



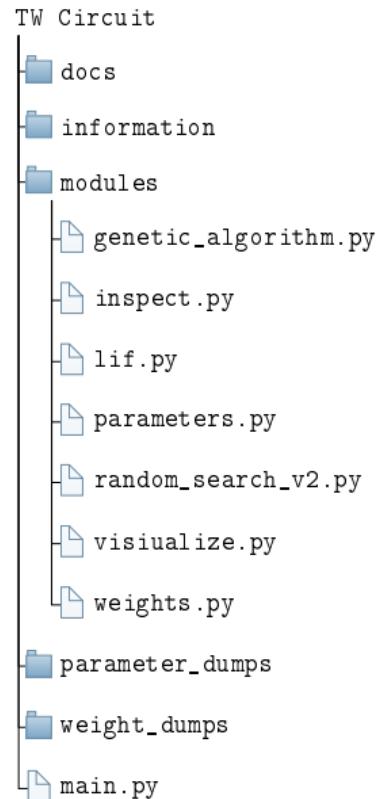
- Implementierung des Simulators
- Simulationsumgebung: OpenAI Gym
- Grafische Darstellung und Auswertung der Simulation

## Simulation: inverses Pendel

## 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.

- Wichtige Programmfonktionen werden in Modulen zusammengefasst und sind uneingeschränkt aufrufbar.
- Es existiert ein zentraler Punkt zum Ändern von Parametern des neuronalen Netzes sowie anderer Simulationsvariablen.
- Parameter und Gewichte werden nach Simulationen mit entsprechenden Informationen gespeichert und sind plattformunabhängig aufrufbar.
- Der Programmcode ist ausführlich kommentiert und es wurde eine entsprechende Dokumentation verfasst, um den Simulator nach eigenen Anforderungen zu erweitern.

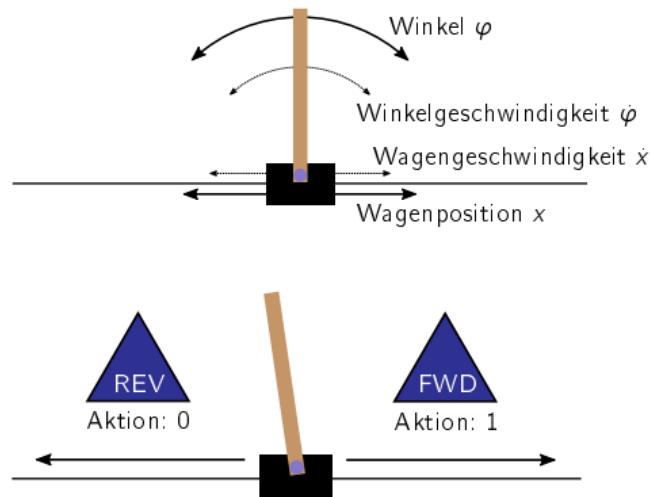


# Implementierung und Simulation des inversen Pendels

## 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.
- Die Ein- bzw. Ausgangsgrößen werden mit den eingangs erwähnten Sensor- und Motor-Neuronen verbunden.



## Simulationsläufe: Auslagerung auf Cloud Server

Durch rechenintensive Simulationen mit den Algorithmen Random-Search und Weights wird ein externer Server mit größeren Ressourcen 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.
- Die Ergebnisse dieser Simulationen werden in Form von Parameter- und Weight-„Dumps“ bereitgestellt und können auf Heimrechnern visualisiert werden.

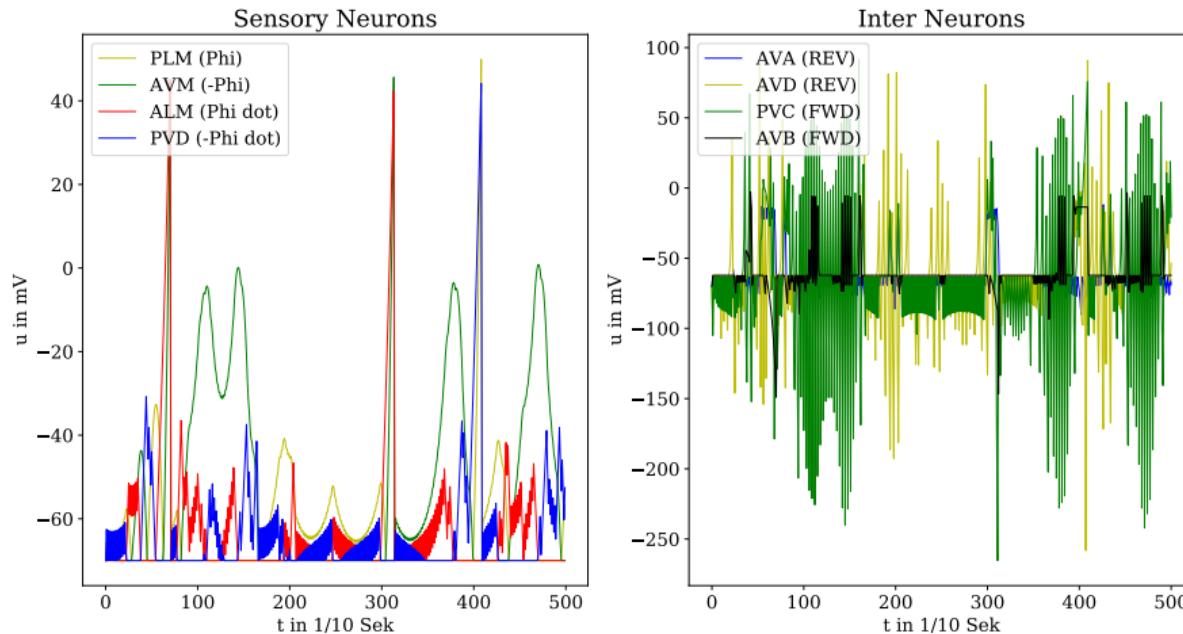
GCP virtuelle Instanz	
	4 vCPUs - Intel XEON
	12 GB DDR5 RAM
	Frankfurt, Deutschland
	Ubuntu 18.04 LTS
	Externe IPv4 Adresse



# Implementierung und Simulation des inversen Pendels

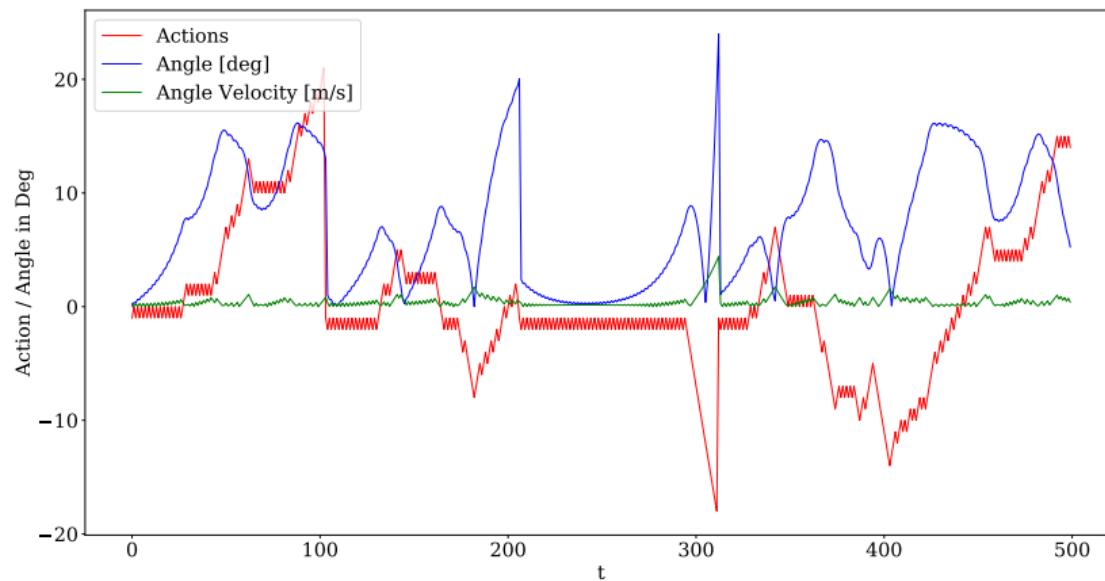
## Visualisierung der Simulationsergebnisse

Darstellung der Membranpotentiale über eine Simulationsdauer von 5 Sekunden:



## Visualisierung der Simulationsergebnisse

Darstellung der Aktion des neuronalen Netzes mit Winkel  $\varphi$  des Pendels über eine Simulationsdauer von 5 Sekunden:



## Simulation des inversen Pendels (Video)

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

## Zusammenfassung

- Ein Simulator für neuronale Dynamik wurde zur Simulation genereller Abläufe innerhalb eines biologischen neuronalen Netzes entwickelt.
- Such- und Optimierungsalgorithmen im Bereich des *Reinforcement Learning* sind implementiert worden, um die Parametersuche zu realisieren.
- Die Stabilisierung des inversen Pendels kann durch richtig gefundene Parameter bzw. Gewichte gewährleistet werden.
- Probleme treten in der Optimierung der Parameter sowie in der internen Verschaltung des gegebenen neuronalen Netzes auf.

## Ausblick

- Durch den modularen Aufbau des Simulators kann dieser erweitert und verbessert werden.
- Die neuronale Dynamik kann durch erweiterte Modelle präziser dargestellt werden.
- Das gegebene neuronale Netz des *C. Elegans* könnte durch erweiterte Literaturrecherche realitätsgetreuer implementiert werden.
- Weitere Simulationsumgebungen (bspw. aus OpenAI Gym) können hinzugefügt werden.

---

Vielen Dank für Ihre Aufmerksamkeit.

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

## **Abschlussvortrag Bachelorarbeit**

### **Kontaktdaten**

Jonas Helmut Wilinski

Lehrstuhl für Regelungstechnik  
Technische Fakultät  
Christian-Albrechts-Universität zu Kiel

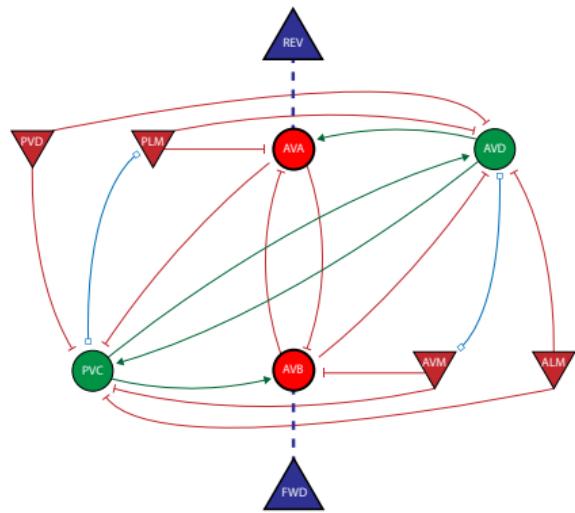
 <http://www.control.tf.uni-kiel.de>  
 stu118261@mail.uni-kiel.de

# **BACKUP**

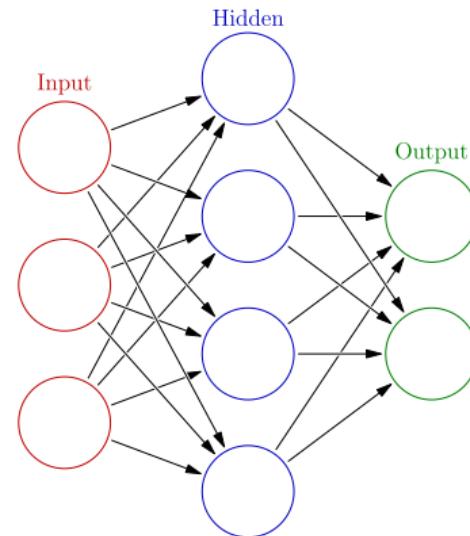
---

# **BACKUP**

## Unterschied biologisches neuronales Netz vs. künstliches neuronales Netz



**Abbildung 6:** Biologisches neuronales Netz des *C. Elegans*.



**Abbildung 7:** Modell eines künstlichen neuronalen Netzes

## Herleitung des Leaky Integrate and Fire - Modells (1/3)

Um eine geeignete Differenzialgleichung herzuleiten, wird zuerst das erste Kirchhoffsche Gesetz angewendet

$$I = I_R + I_C. \quad (5)$$

Der Strom  $I_R$  ist einfach durch das Ohmsche Gesetz wie folgt zu berechnen

$$I_R = \frac{U_R}{R} = \frac{U - U_{Leak}}{R}. \quad (6)$$

Der Strom  $I_C$  wird durch die Definition eines Kondensators  $C_m = \frac{q}{U}$  zum kapazitiven Strom

$$I_C = \frac{dq}{dt} = C_m \frac{dU}{dt}. \quad (7)$$

Hierbei steht  $q$  für die elektrische Ladung und  $U$  für die anliegende Spannung.  
Einsetzen von (6) und (7) in (5) ergibt

$$I = \frac{U - U_{Leak}}{R} + C_m \frac{dU}{dt}. \quad (8)$$

## Herleitung des Leaky Integrate and Fire - Modells (2/3)

Wird diese Gleichung mit  $R$  multipliziert und umgestellt, bildet sich die folgende lineare Differenzialgleichung erster Ordnung:

$$RC_m \frac{dU}{dt} = (U_{Leak} - U) + RI. \quad (9)$$

Der Strom  $I$  wird als Eingangsgröße verstanden und im weiteren Verlauf als  $I_{in}$  bezeichnet. Nach Division durch  $RC$  und Einführung des Leitwerts  $G_{Leak} = \frac{1}{R}$  entsteht die gewollte Form:

$$\frac{dU}{dt} = \frac{G_{Leak}(U_{Leak} - U) + I_{in}}{C_m}. \quad (10)$$

In dieser Gleichung stehen die Variablen  $G_{Leak}$ ,  $U_{Leak}$  und  $C_m$  für Parameter der betrachteten Nervenzelle, während  $I_{in}$  stellvertretend für alle eingehenden Ströme aus Stimuli, chemischen Synapsen und Gap-Junctions steht

$$I_{in} = \sum_{i=1}^n I_{Stimuli} + \sum_{i=1}^n I_{Syn} + \sum_{i=1}^n I_{Gap}. \quad (11)$$

## Herleitung des Leaky Integrate and Fire - Modells (3/3)

Die anliegenden Synapsenströme sind durch folgenden formalaren Zusammenhang zu berechnen:

$$I_{Syn} = \frac{w}{1 + \exp^{\sigma(u_{pre} + \mu)}}(E - u_{post}). \quad (12)$$

Synapsenströme sind grundsätzlich von den pre- und postsynaptischen Potentialen der jeweiligen Nervenzellen  $u_{pre}$  und  $u_{post}$  abhängig. Weiterhin können diese chemischen Synapsen exzitatorisch oder inhibitorisch wirken. Diese Eigenschaft wird durch das sog. Nernstpotential  $E \in [-90 \text{ mV}, 0 \text{ mV}]$  beschrieben. Weitere Größen dieser Gleichung bilden die Kreisfrequenz  $w$ , die Standardabweichung  $\sigma$  und der Erwartungswert  $\mu$ .

Gap-Junctions bilden die Ausnahme, denn sie dienen als Ausgleichsglied und wirken bidirektional. Ihr Strom wird wie folgt berechnet:

$$I_{Gap} = \hat{w}(u_{post} - u_{pre}). \quad (13)$$

Für die Berechnung des Gap-Junction Stroms benötigt es ebenfalls das pre- und postsynaptische Potenzial der jeweiligen Nervenzellen  $u_{pre}$  und  $u_{post}$ , sowie die Kreisfrequenz  $\hat{w}$ .

## Aktionsraum und Übersetzungsvorschrift der Sensor-Neuronen

Um die jeweiligen Größen durch die Sensorneuronen zu übersetzen, werden folgende Funktionen für die jeweils positive und negative Sensorneurone  $S_{positiv}$  und  $S_{negativ}$  angenommen:

$$S_{positiv} := \begin{cases} -70 \text{ mV} & x \leq 0 \\ -70 \text{ mV} + \frac{50 \text{ mV}}{x_{min}} x & 0 < x \leq x_{min} \\ -20 \text{ mV} & x > x_{max} \end{cases} \quad (14)$$

$$S_{negativ} := \begin{cases} -70 \text{ mV} & x \geq 0 \\ -70 \text{ mV} + \frac{50 \text{ mV}}{x_{min}} x & 0 > x \geq x_{min} \\ -20 \text{ mV} & x < x_{max} \end{cases} \quad (15)$$

$x \in [x_{min}, x_{max}]$  ist eine messbare, dynamische Systemvariable, welche in den gegebenen Grenzen  $x_{min}$  und  $x_{max}$  auftritt. Lediglich eine Fallunterscheidung wird getroffen: nimmt  $x$  einen positiven Wert an, wird Sensorneurone  $S_{positiv}$  aktiviert, bei negativem  $x$ -Wert, agiert die Sensorneurone  $S_{negativ}$ .