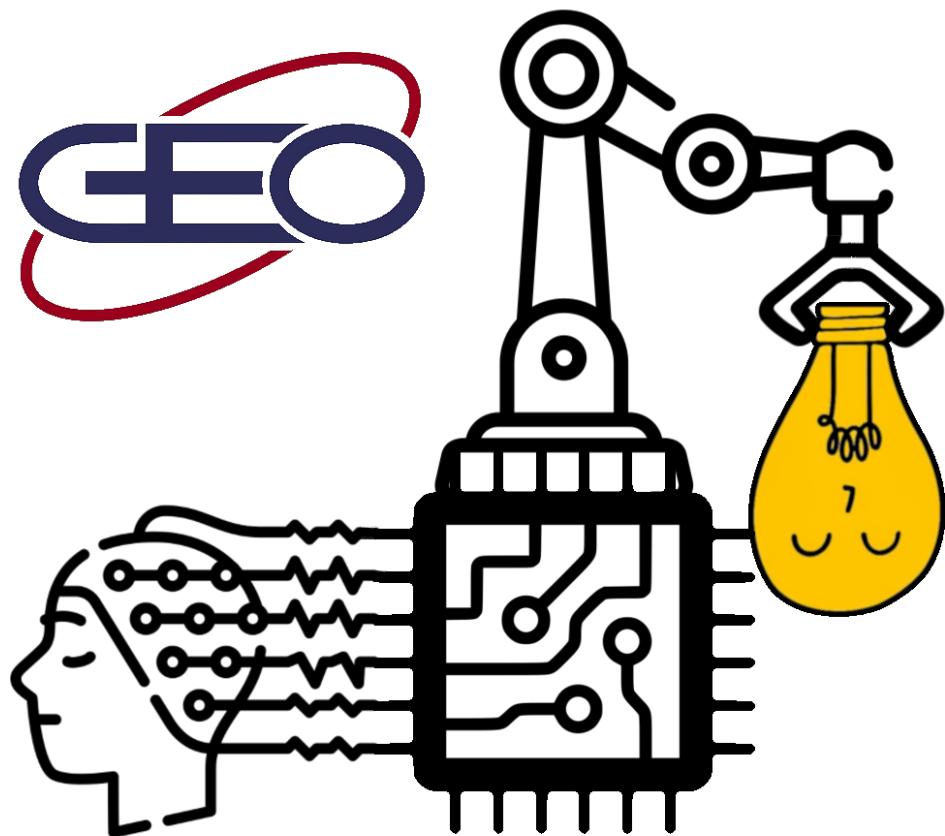


# Erkennung ereigniskorrelierter Potenziale eines Elektroenzephalogramms durch eine KI



Alexander Reimer und Matteo Friedrich  
Gymnasium Eversten Oldenburg

Betreuer: Herr Dr. Glade & Herr Husemeyer

## Inhaltsverzeichnis

<b>1 Kurzfassung</b>	<b>2</b>
<b>2 Einleitung</b>	<b>2</b>
<b>3 Methode und Vorgehensweise</b>	<b>2</b>
3.1 Materialien . . . . .	2
3.2 Vorgehensweise . . . . .	4
3.2.1 Elektroenzephalographie . . . . .	4
3.2.2 Fourier-Analyse . . . . .	6
3.2.3 Neuronales Netz . . . . .	7
3.2.4 Roboter . . . . .	10
<b>4 Ergebnisse</b>	<b>11</b>
<b>5 Diskussion</b>	<b>15</b>
<b>6 Danksagung</b>	<b>16</b>
6.1 Finanzierung . . . . .	16
6.2 Unterstützung . . . . .	16
<b>7 Quellen &amp; Referenzen</b>	<b>16</b>
7.1 Abbildungen . . . . .	16
7.2 Literatur . . . . .	16
7.3 Videos . . . . .	17
7.4 Programme . . . . .	17

## 1 Kurzfassung

In diesem Projekt wollen wir einen Roboter mithilfe bloßer Gedankenkraft steuern.

Um Daten über das Gehirn zu bekommen, nutzen wir einen Elektroenzephalographen, kurz EEG, welches durch Elektroden an der Kopfhaut die Spannungsdifferenzen innerhalb des Gehirns misst. Diese werten wir mithilfe eines neuronalen Netzes aus, welches wir vorher darauf trainiert haben, Muster in diesen Daten zu erkennen. So können wir bestimmte Ereignisse anhand der EEG-Daten ableiten, z.B. ob jemand geblinzelt hat oder sich gerade konzentriert.

Das Ziel ist es dann, einen Roboter nur mit Gedanken steuern zu können. Wir haben bereits ein geeignetes neuronales Netz trainiert, welches Blinzeln sicher erkennen kann. Die Steuerung eines Roboters funktioniert ebenfalls. Unsere Hoffnung ist, in Zukunft den Roboter auch durch Gedanken an bestimmte Richtungen und nicht nur Blinzeln steuern zu können.

## 2 Einleitung

Ziel des Projektes ist es, zuerst ein BCI – Brain-Computer Interface – zu entwickeln, welches verschiedene EKPs erkennen kann. Dieses wollen wir dann testen, indem wir es zur Steuerung eines Roboters nutzen.

BCIs sind Schnittstellen zwischen dem Gehirn und einem Computer, die Kommunikation vom Gehirn zum Computer ermöglichen. Man kann BCIs zum Beispiel für die Steuerung von Prothesen, Drohnen und Robotern nutzen. [1]

Zum Beispiel haben Forscher aus Stanford ein BCI entwickelt, das in der Lage ist, behinderten Personen wieder die Möglichkeit zu geben, nur über ihre Gedanken Nachrichten schreiben zu können [2].

Wir hoffen, bei der Erarbeitung unseres Projektes neben dem Erlangen von Erfahrung in diesem interessanten Bereich auch selbst dazu beizutragen. BCIs, die mit sehr teurer Hardware entwickelt werden, sind nämlich bereits ziemlich gut erforscht. Jedoch sind diese BCIs, aufgrund ihres hohen Preises, für viele Konsumenten nicht nutzbar. Genau bei diesem Problem wollen wir mit unserem Projekt ansetzen. Wir wollen ein BCI entwickeln, welches auch mit günstiger Hardware funktioniert.

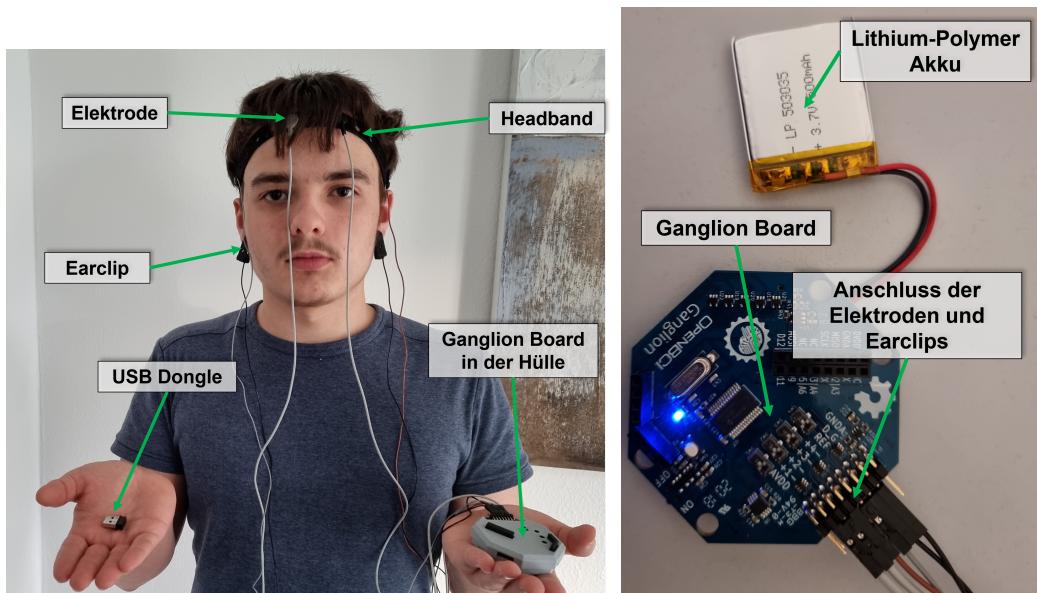
Dabei ist es uns wichtig, dass unser BCI ohne spezielles Training auf genau diese Person für jeden beliebigen Menschen funktioniert, und dass unser BCI auch leicht für die eigenen Zwecke anpassbar ist, z.B. die Erkennung von Atemzügen statt Augenschließen. Deswegen benutzen wir Open Source Software und haben unseren kompletten Code ebenfalls auf GitHub veröffentlicht.

## 3 Methode und Vorgehensweise

### 3.1 Materialien

- EEG
  - 4 Channel Ganglion Board von OpenBCI ★ (s. Danksagung)
  - 4x Spike und 2x Flat Electrodes ★
  - Klettband für die Elektroden ★
  - 2x Earclips ★
  - Lithium-Polymer-Akku und Ladegerät ★
  - Plastik-Hülle für das Ganglion Board
- Roboter
  - Lego Mindstorms EV3 Brick

- Raspberry Pi 3B 8 GB
- 2x EV3 großer Motor
- SD-Karte (8 GB)
- diverse LEGO Teile
- Computer: Aorus 15P
  - CPU: i7-11800H
  - GPU: RTX 3060
  - RAM: 16 GB
- Software
  - Julia als Programmiersprache für unser ganzes Projekt [19]
  - Flux.jl für das neuronale Netz [20] [21]
  - BrainFlow.jl als Schnittstelle zum EEG [22]
  - FFTW.jl für die Fast Fourier Transformation [23]
  - CUDA.jl zum effektiven Nutzen einer NVIDIA GPU [24]
  - PyPlot.jl zum Plotten [25]
  - BSON.jl zum Speichern und Laden von Netzwerken
  - ev3dev.jl zum Steuern eines EV3-Roboters durch einen Raspberry Pi [26]



**Abbildung 1:** Das EEG und Zubehör. Links sieht man die typische Anwendung des EEG im Versuchsaufbau. Rechts ist das Ganglion Board genauer dargestellt.

Unser EEG-Gerät besteht aus einem Klettband mit Löchern für die Elektroden, einer Platine (Ganglion Board), in welche die Elektroden eingesteckt werden, einer Kunststoff-Hülle zum Schutz des Ganglion Boards, und einem USB-Dongle, mit welchem die Signale der Platine kabellos empfangen werden können (s. Abbildung 1).

### 3.2 Vorgehensweise

Unser Projekt lässt sich grob in 3 Bereiche unterteilen. Zum einen gibt es den neurobiologischen Teil. Dieser besteht aus der Messung von Gehirnaktivität und der Umwandlung dieser Aktivität in für uns nutzbare Daten. Der zweite Teil besteht aus der Verarbeitung dieser Signale. Hierfür nutzen wir ein neuronales Netz, welches Muster in den Gehirnaktivitäten erkennen kann. Der letzte Teil von unserem Projekt beinhaltet die Konstruktion und Steuerung eines EV3 Roboters. Je nachdem, was das neuronale Netz ausgibt, soll sich dieser Roboter anders verhalten und so über Gedanken steuerbar sein.

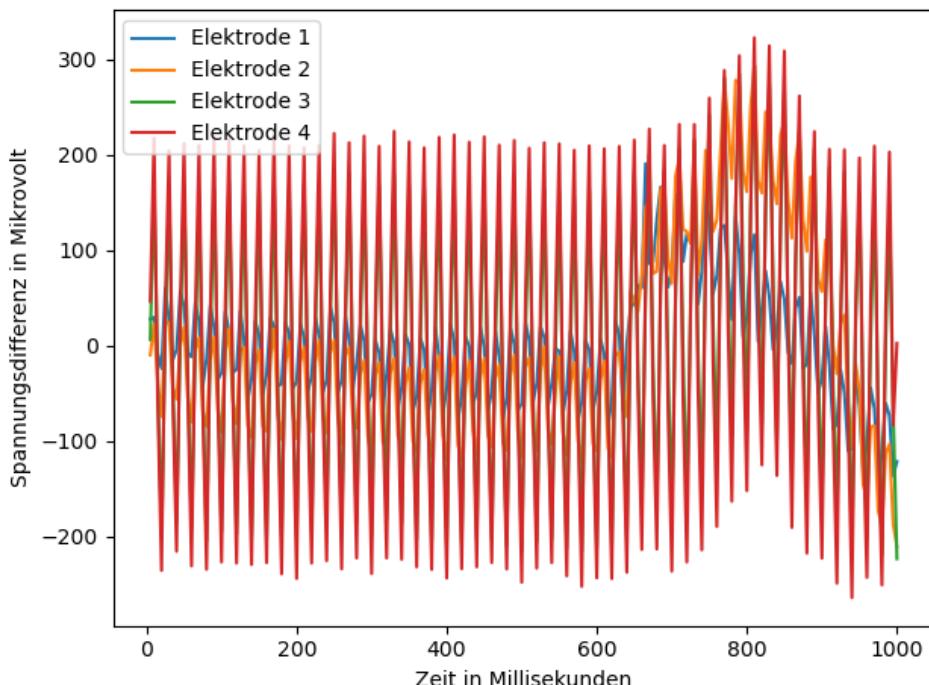
#### 3.2.1 Elektroenzephalographie

Bei der Elektroenzephalographie (EEG) werden Elektroden an der Kopfoberfläche platziert. Diese können sehr kleine Spannungen messen, die durch Reize im Gehirn entstehen und durch den Schädel dringen. Diese Spannungen werden nicht von einzelnen Nervenzellen erzeugt, sondern geben die Summe aller lokalen Spannungen wieder. Man kann also auch nur ungefähr sagen, wo genau im Gehirn ein bestimmter Reiz ausgelöst wurden. Mit mehr Elektroden kann die Genauigkeit erhöht werden. [3] [4]

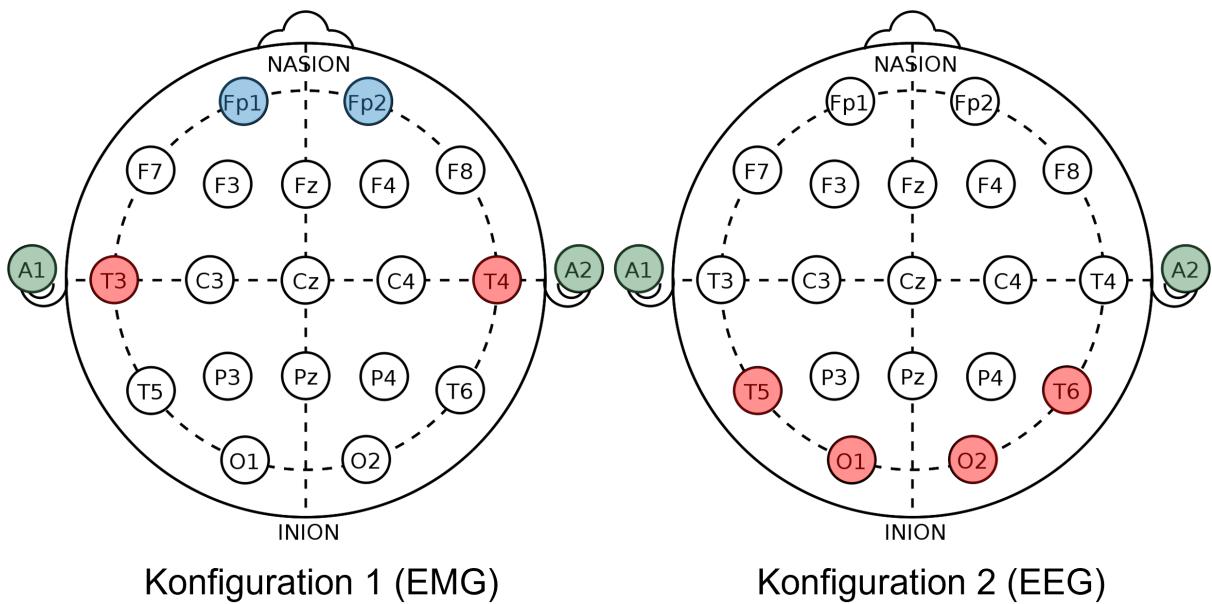
Um eine Differenz bilden zu können, wird eine zweite Elektrode benötigt. Wir benutzen dafür eine Referenzelektrode, welche am Ohrläppchen befestigt wurde, wo sie ein von Gehirnströmen weitgehend unbeeinflusstes, konstantes Potential ableiten kann. Alle vier am Kopf platzierten Elektroden nutzen diese Referenzelektrode. Somit handelt es sich bei allen vier um eine unipolare Ableitung. [5]

Wir untersuchen dabei ereigniskorrelierte Potentiale (EKPs). Dies sind bestimmte Spannungsschwankungen („Potentiale“), welche in Zusammenhang mit einem internen oder externen Ereignis stehen, wie z.B. nach einem lauten Ton, während und nach einer Körperbewegung, oder während hoher Konzentration. [4] [5]

Dabei haben wir uns zuerst für das Blinzeln entschieden, da wir es im Graphen zumindest mit bloßen Augen sehr gut erkennen konnten (s. Abbildung 2).



**Abbildung 2:** Ausschnitt eines EMG mit Blinzeln (Konfiguration 1)



**Abbildung 3:** 10-20 System mit den von uns genutzten Flat Elektroden blau, Spike Elektroden rot, und Grounding & Reference Earclips grün gefärbt. Links für EMG („Konfiguration 1“), rechts für EEG („Konfiguration 2“)

Dafür haben wir zu Beginn zwei Flat Electrodes (Elektroden mit glatter Kontaktfläche) über den Augen und zwei Spike Elektroden (Elektroden mit langen „Spitzen“) an den Schläfen platziert. Flat Elektroden haben zwar aufgrund ihrer größeren Oberfläche auf freier Haut eine etwas bessere Signalqualität, aber die von uns gewählten Stellen an den Schläfen waren leicht behaart. Die Spike Elektroden können durch die Haare leichter Kontakt zur Kopfoberfläche herstellen.

Jedoch handelt es sich dabei eigentlich nicht um EEG. Der Grund für den so starken Ausschlag war, dass wir Elektromyographie (EMG) nutzten. Dabei werden die Spannungsdifferenzen von Muskeln gemessen, in unserem Fall die Muskeln, welche das Augenlid bewegen. [6]

Aber wir haben es trotzdem zu Beginn genutzt, da wir erstmal eine funktionierende Basis schaffen wollten.

Nach EMG haben wir dann tatsächliche Elektroenzephalographie genutzt. Dabei haben wir weiterhin Blinzeln genommen. Die Elektroden haben wir dafür alle nebeneinander am Okzipitalen Kortex (Visueller Kortex) platziert (s. Abb. 3), welcher sich am Hinterkopf befindet und sehr stark mit den Augen und dem Sehvermögen verknüpft ist. [4]

Wir entschieden uns, beim Blinzeln zu bleiben, da wir bereits wussten, dass geschlossene Augen klar erkennbar sein sollten: Bei offenen Augen findet eine sogenannte Alpha-Blockade statt, bei der - vor allem im Okzipitalem Kortex - die Alphawellen stark sinken. Entsprechend steigen sie stark an, wenn die Augen geschlossen sind. [5] [7] [8]

Weiter ist es möglich, nur durch Gedanken eine Steuerung auszuführen. Dies funktioniert jedoch meist durch instrumentelle oder klassische Konditionierung, also durch das Bestrafen und Belohnen auf Basis der Messungen des EEG. So kann das Gehirn darauf trainiert werden, auf Verlangen eine bestimmte, vorher festgelegte Aktivität auszulösen, die dann gemessen und ausgewertet werden kann. [9] Darauf wollen wir verzichten, da die Konditionierung Zeit benötigen und nicht unser Ziel einer allgemeinen Anwendbarkeit erfüllen würde.

Uns ist nach der Aufnahme der Trainingsdaten in der ersten Konfiguration (EMG) aufgefallen, dass bei den Elektroden 3 und 4 das Blinzeln deutlich schlechter erkennbar war als bei den Elektroden 1 und 2, vermutlich weil die Elektroden 3 und 4 Spike Elektroden und weiter von den Augen entfernt waren.

Deswegen haben wir uns entschieden, nur die Elektroden 1 und 2 zu verwenden.

Wir haben immer eine Sekunde an EEG-Daten am Stück aufgenommen, sowohl für die

Trainingsdaten als auch die Live-Tests. Grund für diese Entscheidung war, dass die Auswirkung von Blinzeln immer in diesem Rahmen erkennbar war. Es wäre zwar auch kürzer gegangen, jedoch ist die Fourier-Analyse effektiver, je länger der Zeitraum.

Da jede Elektrode eine zeitliche Auflösung von 200 Hertz (200 Messungen pro Sekunde) hat, messen wir also in der ersten Elektrodenkonfiguration (EMG)  $2 * 200 = 400$  und in der zweiten Elektrodenkonfiguration (EEG)  $4 * 200 = 800$  Signale pro Sekunde.

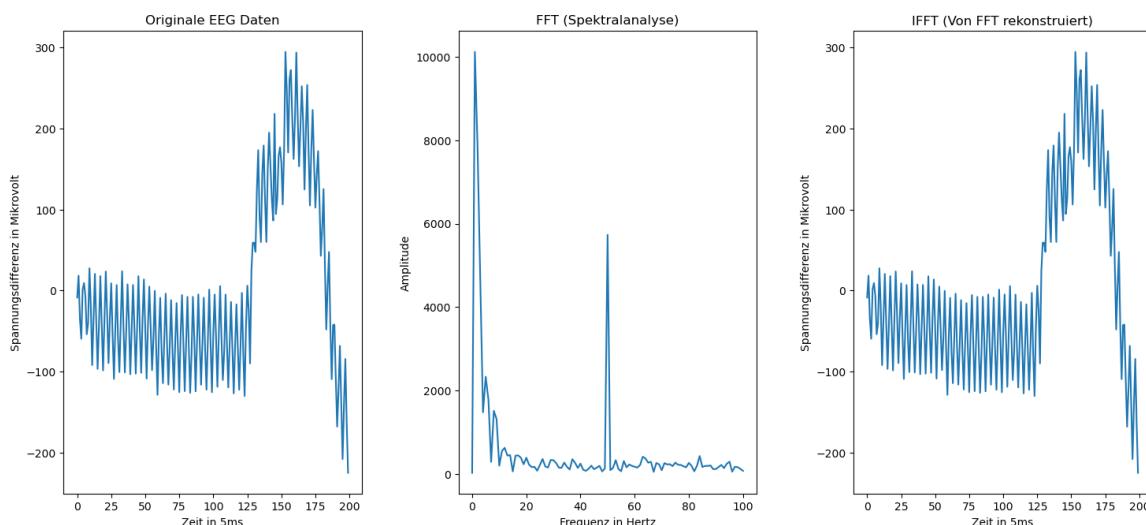
### 3.2.2 Fourier-Analyse

Uns wurde von Professor Everling (s. Danksagung – Unterstützung) empfohlen, die Anwendung der Fourier-Analyse zur Vorbereitung der Daten für das neuronale Netz zu bedenken. Die Fourier-Analyse kann die verschiedenen zugrundeliegenden Frequenzen von Datenfolgen, Funktionen, und mehr bestimmen, indem diese in Sinus-Kurven zerlegt werden, sie dient also zur Spektralanalyse [13]. Wir nutzen dafür die Fast Fourier Transformation (FFT), welche lediglich eine komplexere aber effizientere Form der Diskreten Fourier Transformation (DFT) ist [10].

Aus der FFT folgt ein Array (eine Liste) an komplexen Zahlen. Der Index eines Wertes in der Liste bestimmt, für welche Frequenz der Wert gilt (erster Wert: 1 Hertz, zweiter Wert: 2 Hertz, etc.). Nun muss für jede komplexe Zahl der Abstand zum Ursprung bestimmt werden, also der absolute Betrag. Dieser entspricht dann der Amplitude der Frequenz. So lässt sich bestimmen, welche Frequenzen am stärksten vorkommen. Außerdem können dann Frequenzen herausgefiltert werden, indem die entsprechenden Indices auf 0 gesetzt werden.

Eine FFT kann Elektroenzephalogramme fast verlustfrei repräsentieren. Dies lässt sich erkennen, wenn man mithilfe der durch die FFT entstandenen Spektralanalyse die Daten rekonstruiert (Inverse Fast Fourier Transformation, IFFT) (s. Abbildung 4). Dazu werden die Sinus-Kurven der Frequenzen mit den entsprechenden Amplituden multipliziert und dann addiert.

Eine große Amplitude bei 50 Hertz, wie z.B. in Abbildung 4, kann aufgrund des Wechselstroms in Deutschland entstehen, welcher eine Frequenz von 50 Hertz hat. [5] Um eine Verwirrung der KI zu verhindern, können wir bei der Verwendung von FFT die Amplitude für 50 Hertz auf 0 setzen.

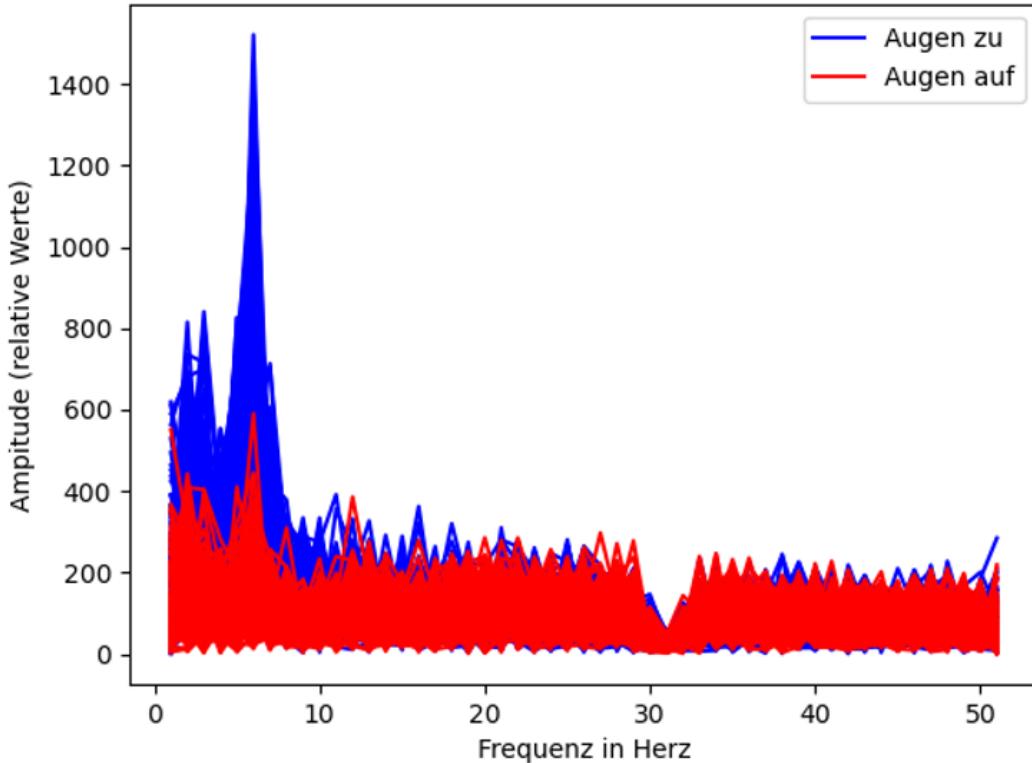


**Abbildung 4:** Das EMG-Signal eines Blinzelns, die Amplituden des FFT dieses Signales, und eine Rekonstruktion des Signals durch IFFT

Vor der Implementation von FFT haben wir aber zuerst getestet, ob eine Spektralanalyse für unseren Zweck überhaupt sinnvoll wäre. Dazu haben wir die Ergebnisse der FFT von Elektroenzephalogrammen mit Blinzeln und ohne Blinzeln verglichen. Wie man in Abbildung 5 sehen kann,

gibt es vor allem im niedrigeren Frequenzbereich einen klaren Unterschied.

Die KI sollte in der Lage sein, diesen Unterschied zu erkennen. Eine FFT wäre also zur Vorverarbeitung der Daten geeignet.



**Abbildung 5:** Die FFT unserer EEG-Daten

Ob die FFT Analyse einen großen Vorteil zu den rohen Daten darstellen wird, werden wir in den Ergebnissen sehen müssen, jedoch liegt es nahe, da die KI sich auf einige wenige, wichtige Inputs konzentrieren kann. Denn wie man sehen kann, ist der Unterschied in den höheren Frequenzbereichen nicht so groß und vermutlich für eine sichere Erkennung nicht unbedingt notwendig.

Die FFT ist außerdem sehr schnell, auf unserem Test-System braucht sie für die Verarbeitung von einer Sekunde Messdaten von zwei Elektroden im Schnitt  $22\mu\text{s}$ . Ein weiterer Vorteil der FFT ist, dass wir weniger Inputs haben. Wir brauchen aller höchstens die Amplituden für 1-100 Hertz, alles darüber kann unser EEG sowieso nicht akkurat wahrnehmen. Mit 100 Inputs (die Frequenzen) statt 200 Inputs (jede einzelne Spannungsdifferenz) kann unsere KI schneller lernen und schneller eine Erkennung durchführen.

Dennoch wollen wir das Programm auch ohne FFT ausprobieren – wenn die Ergebnisse gleich gut sind, würde es sich trotzdem lohnen, auf FFT zu verzichten, da für FFT auch die Installation eines weiteren Packages (FFTW) notwendig ist.

### 3.2.3 Neuronales Netz

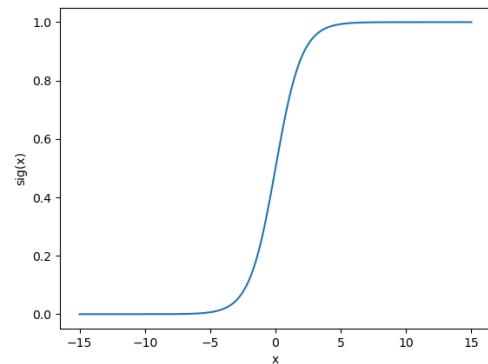
Ein neuronales Netzwerk besteht aus drei Teilen: dem Input Layer, den Hidden Layers und dem Output Layer. Der Input Layer ist eine Liste aus Zahlen zwischen 0 und 1. Er gibt an, welche Eingaben (Inputs) das Netzwerk bekommen soll, z. B. die Grauwerte der Pixel eines Bildes. Die Hidden Layers sind eine Ansammlung von in mehrere Layer (Schichten) unterteilten Neuronen. Jedes Neuron besitzt eine Aktivierung (Activation), die als Zahl zwischen 0 und 1 angegeben

werden kann, und einen Bias (Verzerrung), der eine beliebige Zahl sein kann. Die Neuronen verschiedener Layer sind alle durch sogenannte Gewichte (Weights) verbunden, die ebenfalls einen beliebigen Wert haben können. Die Outputs sind dann lediglich die Aktivierungen der Neuronen im Output Layer.

### Forward Pass

Zur Berechnung der Aktivierung der Neuronen gibt es den sogenannten Forward Pass. Dabei beginnt man im ersten Hidden Layer damit, für alle Neuronen den sogenannten Netzinput (auch net input) zu berechnen. Um den Netzinput eines Neurons zu berechnen, werden alle Aktivierungen des vorherigen Layers mit den von dem Neuron dorthin führenden Gewichten multipliziert und summiert. Der Bias ist eigentlich auch ein Gewicht, jedoch ist er mit einem Neuron verbunden, das immer die Aktivierung 1 hat.

Um aus diesem Netzinput nun die Aktivierung zu berechnen, benötigt man eine Aktivierungsfunktion, die dafür sorgt, dass die Aktivierung zwischen 0 und 1 liegt. Wir haben dafür eine Sigmoidfunktion benutzt, die eine Zahl nimmt und einen Wert zwischen 0 und 1 ausgibt (s. Abbildung 6). Dies wird dann für jedes Neuron in jedem Layer wiederholt. Da man aber immer die Aktivierungen des vorherigen Layers benötigt, muss man das ganze vom Input Layer zum Output Layer durchführen. Daher auch der Name Forward Pass. Die Interpretation der Outputs hängt von den Trainingsdaten ab.



**Abbildung 6:** Graph der Sigmoidfunktion. Wir nutzen die Sigmoidfunktion um die Aktivierung aus dem Netzinput zu berechnen.

Die allgemeine Formel für die Aktivierung eines Neurons lautet also:

$$\text{sig}(x) = \frac{1}{1 + e^{-x}}$$

$$a_j = \text{sig} \left( \sum_L (a_L * W_{Lj}) + b_j \right)$$

wobei  $a_j$  die Aktivierung des Neurons  $j$ ,  $L$  der vorherige Layer,  $a_L$  der Vektor aller Aktivierungen des Layers  $L$ ,  $W_{Lj}$  der Vektor aller Gewichte zwischen dem Neuron  $j$  und den Neuronen des Layers  $L$ , und  $b_j$  der Bias des Neurons  $j$  ist. [14]

### Loss/Cost

Um zu bestimmen, wie gut ein bestimmtes neuronales Netz ist, gibt es die sogenannte Cost-Funktion (auch Loss-Funktion genannt), die mithilfe von Trainingsdaten funktioniert.

Trainingsdaten bestehen aus einer Liste aus Trainingsdatensätzen. Jeder dieser Datensätze beinhaltet Inputs für das Netzwerk und die richtigen Outputs dafür. Machine Learning, das mit solchen Trainingsdaten arbeitet, wird Supervised Learning genannt.

Die Funktion für die Cost des Output-Layers und somit gesamten Netzwerkes (für einen Trainingsdatensatz) lautet wie folgt:

$$C_0 = (a_L - y)^2$$

wobei  $C_0$  = die Cost des Output-Layers,  $L$  = der letzte Layer (Output Layer),  $a_L$  = die Aktivierungen des Layers  $L$ , und  $y$  = die richtigen Outputs für die Inputs, mit denen die Aktivierungen berechnet wurden. Bei  $a_L$  und  $y$  handelt es sich um Vektoren

Um die Cost zu berechnen, muss man also für alle Output Neuronen die Differenz der gegebenen und der richtigen Aktivierungen bilden. Danach muss man diese Differenzen quadrieren und am Ende alle Ergebnisse aufsummieren. Dies kann man für alle Trainingsdatensätze wiederholen und von allen Costs den Durchschnitt nehmen, um die allgemeine Performance eines Netzwerkes zu überprüfen. Diese Art der Cost-Funktion wird mittlere quadratische Abweichung (mean squared error, kurz MSE) genannt.

Zusammenfassend kann man also sagen, dass die Cost die Abweichung von den berechneten und den richtigen Outputs angibt. Aufgrund des Trainingsdatensatzes weiß man nun, wie der Output Layer verändert werden muss.

### Backpropagation

Doch wie verändert man nun die Aktivierungen des Output-Layers? Es müssen alle Gewichte und Biases davor angepasst werden. Um nun zu wissen, wie ein Gewicht verändert werden muss, gibt es folgende Funktion:

$$\Delta W_{ij} = \epsilon * \delta_i * a_j$$

wobei  $\Delta W_{ij}$  = um wie viel das Gewicht  $W$  zwischen den Neuronen  $j$  und  $i$  verändert werden muss,  $\epsilon$  = die Lernrate (meist ein kleiner Wert wie 0.001),  $\delta_i$  ≈ die Ableitung der Cost des Neurons  $i$  im Verhältnis zum Gewicht  $W_{ij}$ , und  $a_j$  = die Aktivierung des Neurons  $j$ . Was dabei oft verwirrend ist:  $j$  bezeichnet das Neuron, welches zuerst kommt, und  $i$  das Neuron, welches danach kommt (Reihenfolge im Forward-Pass), obwohl es bei  $W_{ij}$  andersherum steht.

Für den Bias wird die gleiche Formel benutzt, mit der Ausnahme, dass  $a_j$  immer 1 ist und so wegfällt. Der Grund dafür liegt darin, dass der Bias, wie in der Struktur beschrieben, eigentlich nur ein Gewicht ist, das mit einem Neuron verbunden ist, welches immer eine Aktivierung von eins hat.

Um nun  $\delta_i$  für den Output Layer zu berechnen, gibt es folgende Gleichung:

$$\delta_i = \text{sig}'(\text{netzinput}_i) * (a_i(\text{soll}) - a_i(\text{ist}))$$

wobei  $\text{sig}'(x)$  = die Ableitung von  $\text{sig}(x)$ , also  $\text{sig}'(x) = \text{sig}(x) * (1 - \text{sig}(x))$ ,  $\text{netzinput}_i$  = der Netzinput des Neurons  $i$ ,  $a_i(\text{soll})$  = die Aktivierung, die das Neuron haben sollte (also das gleiche wie  $y$ ), und  $a_i(\text{ist})$  = die Aktivierung, die das Neuron hat.

Mit dieser Formel wird berechnet, welche Aktivierung das Neuron haben sollte, was an  $a_i(\text{soll}) - a_i(\text{ist})$  erkennbar ist. Die Aktivierungsfunktion mit dem Netzinput wird als Faktor mit einberechnet, da möglichst nur die Gewichte stark verändert werden sollen, die bei dem Trainingsdatensatz eine hohe Aktivierung haben, also durch diese Inputs besonders angesprochen werden. So werden zum Beispiel beim Sortieren nur die Neuronen miteinander verknüpft, die für ein bestimmtes Muster verantwortlich sind.

Für die Neuronen der Hidden Layers muss man alle  $\delta$ 's des nächsten Layers mit den von dem Neuron dorthin führenden Gewicht multiplizieren und dann summieren. Dadurch werden die Änderungen, die die Aktivierungen dieser Neuronen brauchen ( $\delta$ ), zusammengerechnet, da natürlich die Aktivierungen im nächsten Layer unterschiedliche Änderungen in dem gleichen Neuron benötigen.

Durch die Multiplikation mit den dahin führenden Gewichten werden diese Änderungen gewichtet, da sie auf einige Neuronen größere Auswirkungen haben als auf andere. Wie beim Output Layer auch wird diese Summe noch mit  $\text{sig}'(\text{netzinput})$  multipliziert, um die Aktivierung durch bestimmte Muster angesprochener Neuronen noch weiter zu erhöhen und weniger/kaum angesprochener Neuronen zu senken, sodass die Ergebnisse besser und eindeutiger werden. Die Formel hierfür lautet:

$$\text{sig}'(\text{netzinput}_i) * \sum_L (\delta_L * W_{Li})$$

wobei  $L$  = der nächste Layer,  $\delta_L$  = alle  $\delta$ 's des Layers  $L$ , und  $W_{Li}$  = alle Gewichte, die ein Neuron des nächsten Layers und Neuron  $i$  verbinden.

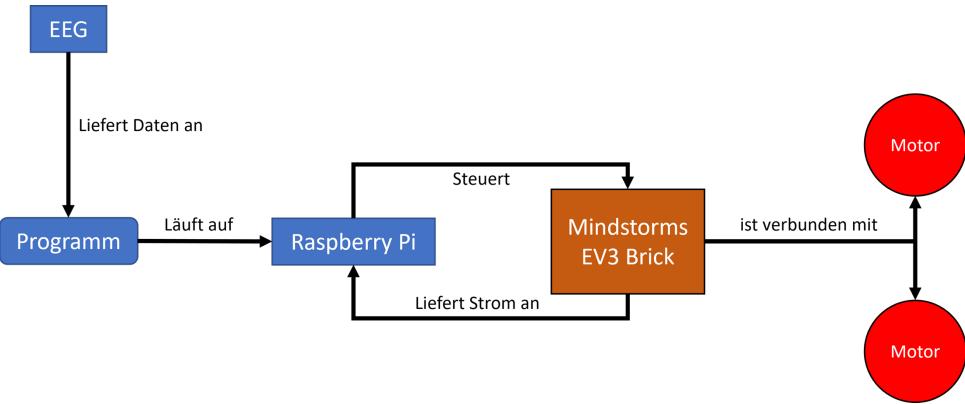
Da immer die nächsten Layer und der Output Layer benötigt werden, ergibt es Sinn, diese Optimierung beim Output Layer zu starten und dann rückwärts die  $\delta$ -Werte für jeden Layer zu berechnen und für die nächsten Berechnungen zu speichern – daher auch der Name Backpropagation. [11] [15] [16]

## Flux

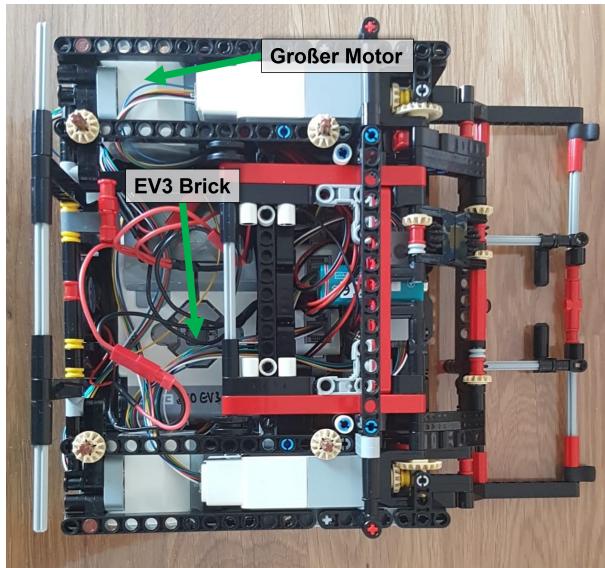
Wir haben in unserem letzten Projekt bereits ein neuronales Netzwerk mit dieser Funktionsweise selbst programmiert, um es besser verstehen zu können. [27] Für dieses Projekt haben wir allerdings das Package Flux benutzt, welches die gleichen oder bessere Ergebnisse liefern sollte, mit deutlich besserer Performance, da es stark optimiert und sehr viel weiter entwickelt wurde. [20]

### 3.2.4 Roboter

Für unseren Roboter haben wir uns entschieden, Lego Mindstorms EV3 Motoren zu benutzen, die von einem EV3-Brick gesteuert werden, der wiederum von einem Raspberry Pi 3B gesteuert wird (s. Abbildung 7 und 8). Der Grund dafür, dass wir keine Motoren direkt mit dem Raspberry Pi steuern, ist, dass wir schon alle Teile für einen EV3 Roboter haben. Somit müssten wir entweder neue Motoren kaufen oder passende Adapter finden, welche meist nur mit C und Python funktionieren und teuer sind.



**Abbildung 7:** Darstellung des Informationflusses in unserem Experiment. Die über das EEG aufgenommen Gehirnaktivitäten werden durch das neuronale Netz analysiert. Das Ergebnis wird an den Raspberry Pi weitergeleitet, welcher mittels des EV3-Bricks die Motoren ansteuert.



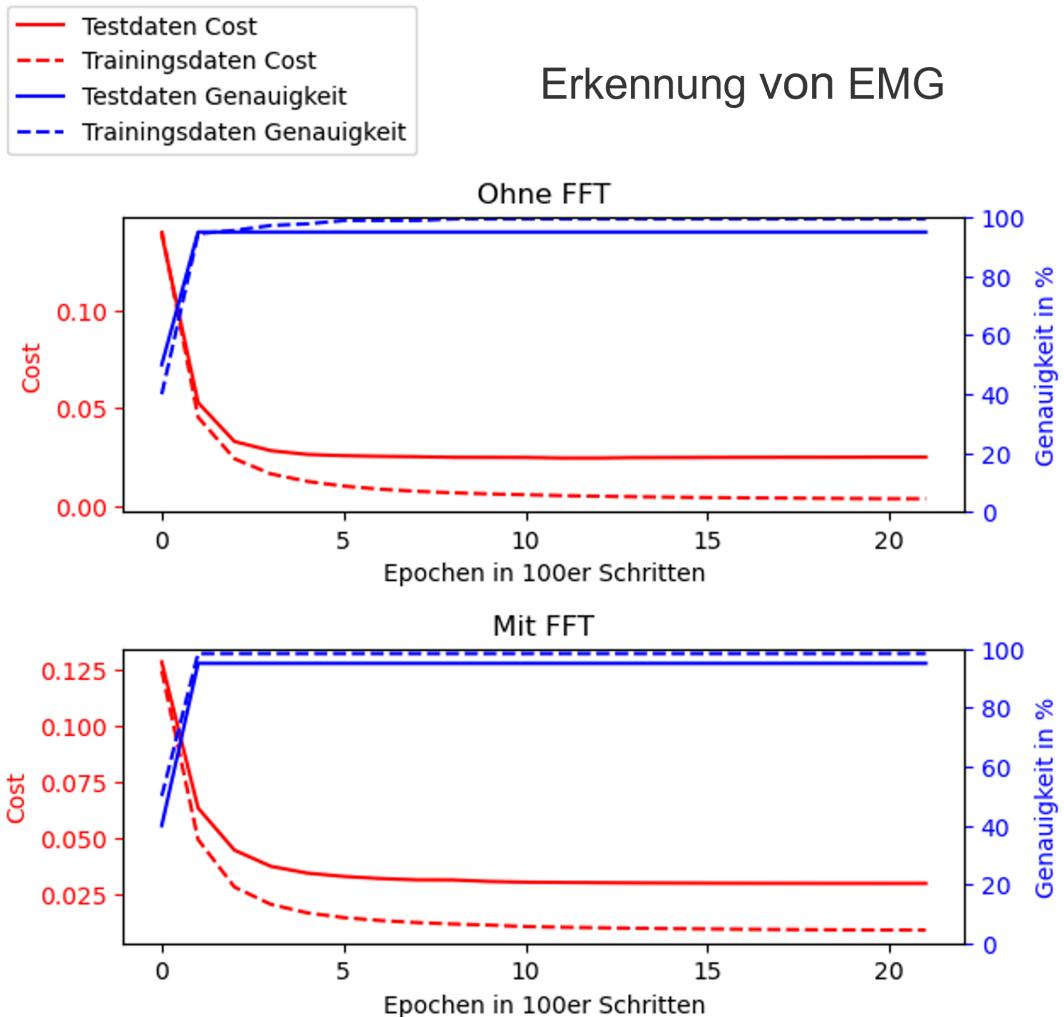
**Abbildung 8:** Unser Roboter (ohne Raspberry Pi, der über den EV3-Brick gehört)

Wir haben dafür das Package `ev3dev.jl` benutzt, welches wir bereits für die Robotik-AG programmiert hatten.

Mehr technische Details zu der Umsetzung lassen sich beim GitHub Repository des Packages finden [26].

## 4 Ergebnisse

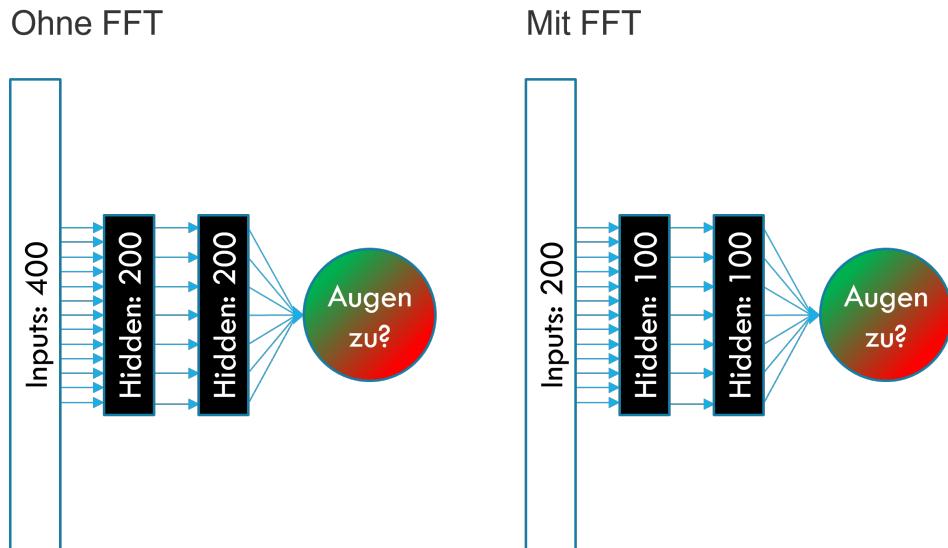
Beim Erkennen von Blinzeln mit EMG (Konfiguration 1) erhielten wir, sowohl mit als auch ohne FFT, eine Testdaten-Genauigkeit von 95% (s. Abb. 9), in einigen undokumentierten Durchläufen auch 100% (). Mit der Genauigkeit ist gemeint, welchen Anteil der gegebenen Datensätze das neuronale Netzwerk korrekt klassifizieren konnte. Dabei ist vor allem die Testdaten-Genauigkeit relevant, da diese die Genauigkeit mit den Testdaten angibt. Die Testdaten sind nicht in den Trainingsdaten enthalten, sodass das neuronale Netzwerk nicht mit ihnen trainiert wurde und so mit ihnen komplett neue, unbekannte Fälle simuliert werden können.



**Abbildung 9:** Es wurde EMG benutzt. Zu sehen ist der Cost- und Genauigkeitsverlauf der Test- und Trainingsdaten beim Trainieren des Netzwerkes, oben ohne Vorverarbeitung durch FFT und unten mit. Eine Epoche entspricht jeweils einer Backpropagation mit allen Trainings-Datensätzen.

Wir hatten jeweils 100 Trainingsdatensätze für die Fälle, dass geblinzelt und nicht geblinzelt wurde.

Für die Struktur der Hidden Layers des Netzwerkes haben wir uns entschieden, da wir einige ausprobiert haben und diese konsistent gute Ergebnisse (90-100% Testdaten-Genauigkeit) mit kurzen Trainingszeiten (ca. 3 Minuten für 1000 Epochen, mit FFT) geliefert haben. Sie ist zu sehen in Abbildung 10.

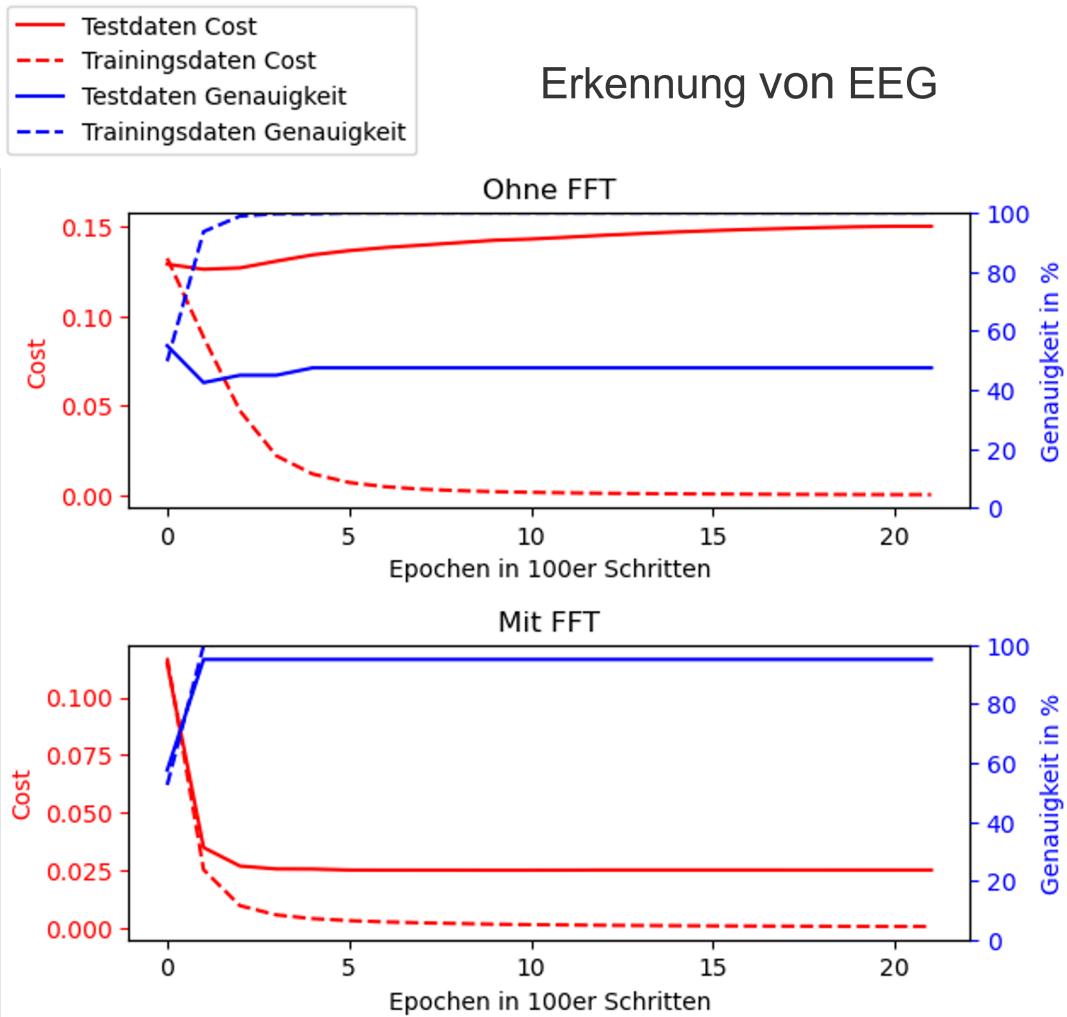


**Abbildung 10:** Struktur unseres neuronalen Netzwerks bei Verwendung von EMG (Konfiguration 1)

Auch ein realer Test mit einer Testperson lief gut, wie man in der Aufnahme davon sehen kann. [17]

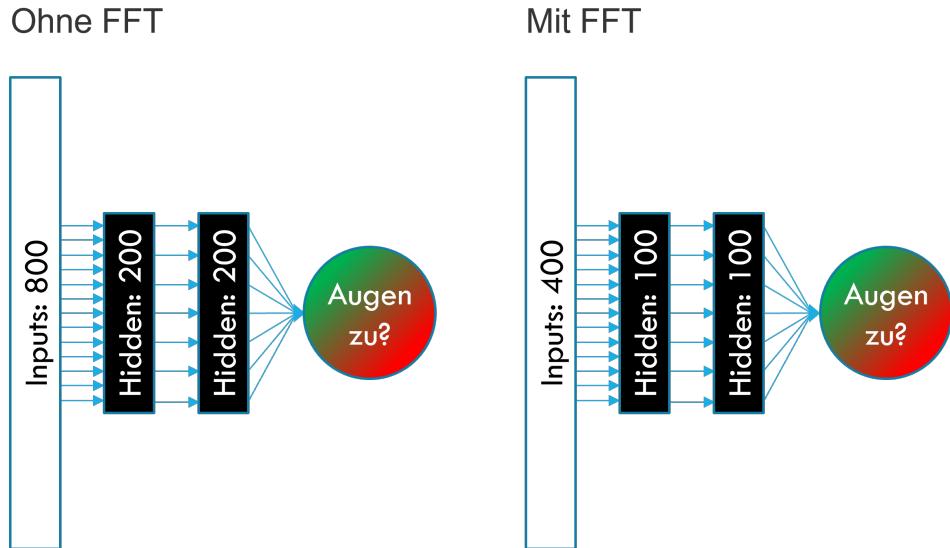
Nach diesem Erfolg gingen wir nun weiter zur Verwendung von EEG (Konfiguration 2). Dabei konnte das neuronale Netzwerk ebenfalls eine Testdaten-Genauigkeit von 95% erreichen, jedoch nur mit FFT. Ohne FFT sah es deutlich schlechter aus (s. Abb. 11): Die Genauigkeit mit den Trainingsdaten konnte zwar 100% erreichen, jedoch blieb die Testdaten-Genauigkeit, welche viel repräsentativer für die Leistung bei der realistischen Verwendung ist, bei 47,5% stehen.

Der Grund dafür ist vermutlich ein Phänomen namens Überanpassung. Dabei passt sich das neuronale Netzwerk zu stark an die Trainingsdatensätze an, wodurch das Netzwerk die Fähigkeit, auch neue, leicht unterschiedliche Daten korrekt zu erkennen, verliert / nicht erlernt. Dafür verantwortlich könnte sein, dass das Netzwerk zu lange trainiert wurde, oder was in unserem Fall wahrscheinlicher ist, ohne FFT keine tieferen Zusammenhänge finden konnte.



**Abbildung 11:** Es wurde EEG benutzt. Zu sehen ist der Cost- und Genauigkeitsverlauf der Test- und Trainingsdaten beim Trainieren des Netwerkes, oben ohne Vorverarbeitung durch FFT und unten mit. Eine Epoche entspricht jeweils einer Backpropagation mit allen Trainings-Datensätzen.

Ein Live-Test mit FFT verlief gut, wie man in der Aufnahme dessen sehen kann. [18]  
Die Netzwerkstruktur für EEG ist in Abbildung 12 zu sehen.



**Abbildung 12:** Struktur unseres neuronalen Netzwerks bei Verwendung von EEG (Konfiguration 2)

## 5 Diskussion

Wir sind zufrieden damit, dass wir unser erstes Teilziel erreicht haben: Der Roboter fährt sicher, und es gibt kaum eine Verzögerung. Die Kosten der benutzten Hardware halten sich noch in Grenzen: Beim Kauf hat die gesamte EEG-Ausstattung ca. 550€ gekostet, was zwar eigentlich eine hohe Summe ist (danke an den Förderverein für die Finanzierung!), aber „professionelle“ Ausrüstung kostet deutlich mehr. Die Performance des Programms kann außerdem zwar noch verbessert werden, jedoch dauern beim Trainieren 1000 Epochen (genug für ca. 100% Genauigkeit) auf dem Test-System bereits lediglich 5 Minuten.

Als ersten folgenden Schritt wollen wir recherchieren, warum unsere Ergebnisse in großen Räumen und draußen so viel besser waren als in kleinen Räumen. Außerdem könnte man die Signalqualität des EEG noch verbessern, indem man Elektrodengel aufträgt.

Zudem wollen wir in fernerer Zukunft versuchen, mithilfe unseres neuronalen Netzes bereits bei allen Menschen vorhandene, selbst kontrollierbare Gehirnaktivität zu finden und sicher zu erkennen, jedoch ohne vorherige Konditionierung. Solche Gehirnaktivität könnte z.B. der allgemeine Gedanke an „Rechts“ sein, was womöglich mit erhöhter Aktivität auf der linken Hirnhälfte in Verbindung stehen könnte.

## 6 Danksagung

### 6.1 Finanzierung

Danke an den Förderverein „Gesellschaft der Freunde des Gymnasium Eversten e.V.“ für die Finanzierung des EEG-Geräts. Alle finanzierten Teile sind in der Materialliste mit ★ gekennzeichnet.

### 6.2 Unterstützung

Vielen Dank an Professor Everling vom „The Brain and Mind Institute“ der Western University in Kanada, der uns geholfen hat, einen Ansatz in diesem komplizierten Thema zu finden und praktische Tipps zum Umgang mit dem EEG gegeben hat.<sup>1</sup>

Dank geht auch an Ino Saathoff, der für uns die Hülle der EEG-Platine mit seinem 3D-Drucker erstellt hat.

Natürlich geht auch Dank Oliver Samkovskij und Ino, die zusammen mit uns den Roboter gebaut haben.

## 7 Quellen & Referenzen

### 7.1 Abbildungen

Die Vorlage des 10-20-Systems in der Abbildung 3 stammt von der Dokumentation von OpenBCI. [12]

Alle anderen Abbildungen und Fotos wurden von uns selbst erstellt.

### 7.2 Literatur

- [1] Andrea Bonci u. a. „An Introductory Tutorial on Brain–Computer Interfaces and Their Applications“. In: *Electronics* 10.5 (Feb. 2021), S. 560. DOI: 10.3390/electronics10050560. URL: <https://doi.org/10.3390/electronics10050560>.
- [2] Francis R. Willett u. a. „High-performance brain-to-text communication via imagined handwriting“. In: *bioRxiv* (2020). DOI: 10.1101/2020.07.01.183384. eprint: [https://www.biorxiv.org/content/early/2020/07/02/2020.07.01.183384](https://www.biorxiv.org/content/early/2020/07/02/2020.07.01.183384.full.pdf). URL: <https://www.biorxiv.org/content/early/2020/07/02/2020.07.01.183384>.
- [3] Wikipedia. *Elektroenzephalografie – Wikipedia, The Free Encyclopedia*. <http://de.wikipedia.org/w/index.php?title=Elektroenzephalografie&oldid=216289194>. 2022. (Besucht am 13.01.2022).
- [4] N. Birbaumer und R. F. Schmidt. „Biologische Psychologie“. In: Springer Verlag, Berlin Heidelberg, 2010. Kap. 20.5, S. 468–493.
- [5] Universität Lübeck. *Elektroenzephalographie (EEG) und Ereigniskorrelierte Potentiale (EKP)*. Praktikumsskript.
- [6] Wikipedia. *Electromyography — Wikipedia, The Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=Electromyography&oldid=1070043736>. 2022. (Besucht am 16.02.2022).
- [7] Helga Peter und Thomas Penzel. „Berger-Effekt“. In: *Springer Reference Medizin*. Springer Berlin Heidelberg, 2020, S. 1–1. DOI: 10.1007/978-3-642-54672-3\_350-1. URL: [https://doi.org/10.1007/978-3-642-54672-3\\_350-1](https://doi.org/10.1007/978-3-642-54672-3_350-1) (besucht am 30.03.2022).
- [8] Wikipedia. *Berger-Effekt — Wikipedia, The Free Encyclopedia*. <http://de.wikipedia.org/w/index.php?title=Berger-Effekt&oldid=208486396>. 2022. (Besucht am 29.03.2022).

---

<sup>1</sup><https://www.uwo.ca/bmi/investigators/stefan-everling.html>

- [9] Ujwal Chaudhary, Niels Birbaumer und Ander Ramos-Murgui alday. „Brain-computer interfaces for communication and rehabilitation“. In: *Macmillan Publishers Limited* 12 (Sep. 2016), S. 513–525.
- [10] Sagar Khillar. „Difference Between FFT and DFT“. In: *Difference Between Similar Terms and Objects* (Sep. 2021). URL: <http://www.differencebetween.net/technology/difference-between-fft-and-dft/> (besucht am 18.02.2022).
- [11] Larry Hardesty. „Explained: Neural networks“. In: *MIT News* (2017). URL: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>.
- [12] *OpenBCI Documentation*. Produkt-Dokumentation.

### 7.3 Videos

- [13] Grant Sanderson 3blue1brown. *But what is the Fourier Transform? A visual introduction*. YouTube Video. 2018. URL: <https://www.youtube.com/watch?v=spUNpyF58BY> (besucht am 10.01.2022).
- [14] Brotcrunsher. *Neuronale Netze - Backpropagation - Forwardpass*. YouTube Video. 2017. URL: <https://www.youtube.com/watch?v=YIqYBxpv53A> (besucht am 15.01.2022).
- [15] Grant Sanderson 3blue1brown. *But what is a Neural Network?* YouTube Playlist. 2017. URL: [https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1\\_67000Dx\\_ZCJB-3pi](https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi) (besucht am 10.01.2022).
- [16] Brotcrunsher. *Neuronale Netze - Backpropagation - Backwardpass*. YouTube Video. 2017. URL: <https://www.youtube.com/watch?v=EAtQCut6Qno> (besucht am 15.01.2022).
- [17] Alexander Reimer und Matteo Friedrich. *Live-Test des Programms für Jugend Forscht 2022*. YouTube Video. 2022. URL: <https://www.youtube.com/watch?v=I2osIVPmt20> (besucht am 18.02.2022).
- [18] Alexander Reimer und Matteo Friedrich. *Live-Test des Projekts für Landeswettbewerb Jugend Forscht 2022*. YouTube Video. 2022. URL: <https://www.youtube.com/watch?v=k7P1B90SHLw> (besucht am 31.03.2022).

### 7.4 Programme

- [19] Jeff Bezanson u. a. „Julia: A Fresh Approach to Numerical Computing“. In: *SIAM Review* 59.1 (Jan. 2017), S. 65–98. DOI: [10.1137/141000671](https://doi.org/10.1137/141000671). URL: <https://doi.org/10.1137/141000671>.
- [20] Michael Innes u. a. „Fashionable Modelling with Flux“. In: *CoRR* abs/1811.01457 (2018). arXiv: [1811.01457](https://arxiv.org/abs/1811.01457). URL: <https://arxiv.org/abs/1811.01457>.
- [21] Mike Innes. „Flux: Elegant Machine Learning with Julia“. In: *Journal of Open Source Software* (2018). DOI: [10.21105/joss.00602](https://doi.org/10.21105/joss.00602).
- [22] Andrey Parfenov et al. *Brainflow*. GitHub Repository. 2018. URL: <https://github.com/brainflow-dev/brainflow>.
- [23] Matteo Frigo und Steven G. Johnson. „The Design and Implementation of FFTW3“. In: *Proceedings of the IEEE* 93.2 (2005). Special issue on “Program Generation, Optimization, and Platform Adaptation”, S. 216–231. DOI: [10.1109/JPROC.2004.840301](https://doi.org/10.1109/JPROC.2004.840301).
- [24] Tim Besard, Christophe Foket und Bjorn De Sutter. „Effective Extensible Programming: Unleashing Julia on GPUs“. In: *IEEE Transactions on Parallel and Distributed Systems* (2018). ISSN: 1045-9219. DOI: [10.1109/TPDS.2018.2872064](https://doi.org/10.1109/TPDS.2018.2872064). arXiv: [1712.03112 \[cs.PL\]](https://arxiv.org/abs/1712.03112).
- [25] Steven G. Johnson. *PyPlot.jl*. GitHub Repository. 2012. URL: <https://github.com/JuliaPy/PyPlot.jl>.

- [26] Alexander Reimer. *ev3dev.jl*. GitHub Repository. 2022. URL: <https://github.com/AR102/ev3dev.jl>.
- [27] Alexander Reimer und Matteo Friedrich. *AI-Composer*. GitHub Repository. 2021. URL: <https://github.com/AR102/AI-Composer.jl>.