

Erkennung ereigniskorrelierte
Potenziale eines
Elektroenzephalogramms durch eine KI

Alexander Reimer und Matteo Friedrich
Gymnasium Eversten Oldenburg

Betreuer: Herr Dr. Glade & Herr Husemeyer

Inhaltsverzeichnis

1 Zusammenfassung	3
2 Einleitung	3
3 Methode und Vorgehensweise	3
3.1 Materialien	3
3.2 Vorgehensweise	5
3.2.1 Elektroenzephalographie	5
3.2.2 Fourier-Analyse	6
3.2.3 Neuronales Netz	8
3.2.4 Roboter	11
4 Ergebnisse	12
5 Diskussion	15
6 Danksagung	16
6.1 Finanzierung	16
6.2 Unterstützung	16
7 Quellen & Referenzen	16

1 Zusammenfassung

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

Um Daten über das Gehirn zu bekommen, nutzen wir einen Elektroenzephalographen, kurz EEG, welcher durch Elektroden an der Kopfhaut die Spannungsdifferenz innerhalb des Gehirns misst. Dies 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, durch das Erkennen verschiedener dieser sogenannten ereigniskorrelierten Potentiale (EKPs) einen Roboter nur mit Gedanken steuern zu können.

Wir haben es geschafft eine KI zu programmieren, die in der Lage ist, anhand der EEG-Daten richtig zu erkennen, ob eine bestimmte Testperson gerade geblinzelt hat oder nicht. Die korrekte Klassifizierung der Daten ist jedoch nur möglich, wenn die Test- und Trainingsdaten unter den exakt gleichen Voraussetzungen aufgenommen wurden (das heißt gleiche Person, gleiche Umgebung, etc.). Diese Problematik könnten wir wahrscheinlich durch das Sammeln von mehr und verschiedenen Daten, sowie das Umpositionieren der Elektroden lösen. Leider hatten wir dafür aber noch nicht genug Zeit gehabt.

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.

Die Idee eines BCI ist nicht neu, sondern wird intensiv erforscht. Aufgrund bereits durchgeführter Experimente ist bekannt, dass BCIs umsetzbar sind [1].

Wir hoffen, neben dem Erlangen von Erfahrung in diesem interessanten Bereich auch selbst dazu beizutragen. Dies wollen wir erreichen durch das Entwickeln einer allgemeinen Anwendung, bei der kein vorheriges Trainieren für eine fremde Person benötigt wird, das Umsetzen mit günstiger, für viele bezahlbare Hardware, sowie ein performantes Programm, welches leicht für die eigenen Zwecke anpassbar ist.

3 Methode und Vorgehensweise

3.1 Materialien

- EEG
 - 4 Channel Ganglion Board von OpenBCI *
 - 2x 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
 - Flux.jl für das neuronale Netz¹ [12] [13]
 - BrainFlow.jl als Schnittstelle zum EEG² [14]
 - FFTW.jl für die Fast Fourier Transformation³ [15]
 - CUDA.jl zum effektiven Nutzen einer NVIDIA GPU⁴ [16]
 - PyPlot.jl zum Plotten⁵ [17]
 - BSON.jl zum Speichern und Laden von Netzwerken⁶
 - ev3dev.jl zum Steuern eines EV3-Roboters durch einen Raspberry Pi⁷ [18]

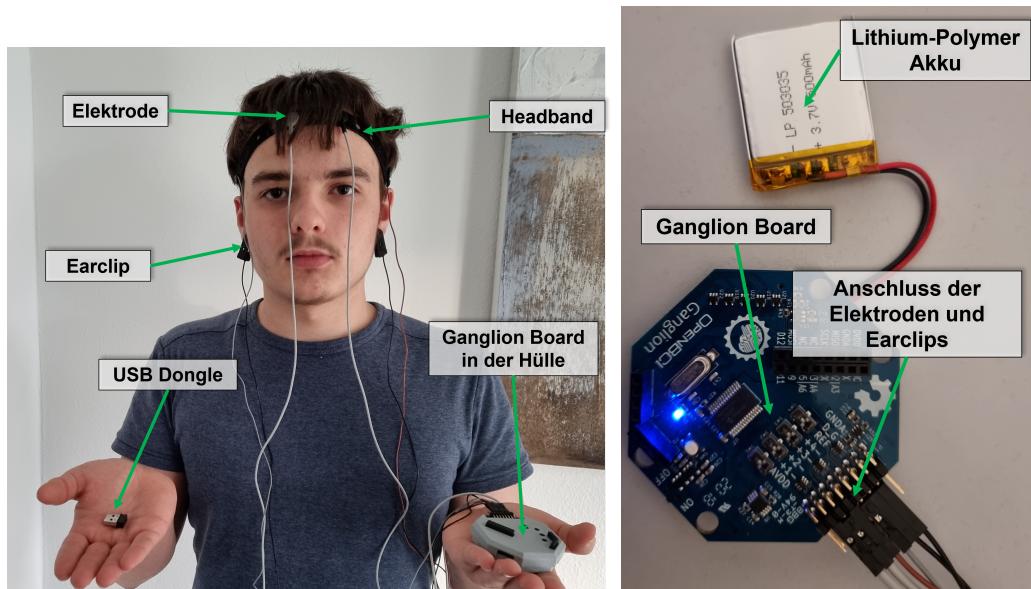


Abbildung 1: Das EEG und Zubehör

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).

¹<https://github.com/FluxML/Flux.jl>

²<https://github.com/brainflow-dev/brainflow>

³<https://github.com/JuliaMath/FFTW.jl>

⁴<https://github.com/JuliaGPU/CUDA.jl>

⁵<https://github.com/JuliaPy/PyPlot.jl>

⁶<https://github.com/JuliaIO/BSON.jl>

⁷<https://github.com/AR102/ev3dev.jl>

3.2 Vorgehensweise

Unser Projekt lässt sich grob in 3 Teile unterscheiden.

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 Spannungsdifferenzen 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 wurde, je mehr Elektroden, desto höher die Genauigkeit. [2]

Wir untersuchen dabei ereigniskorrelierte Potentiale (EKPs). Dies sind bestimmte Spannungsschwankungen („Potentiale“), welche in Zusammenhang mit einem beobachtbaren Ereignis stehen, wie z.B. Blinzeln oder Armbewegungen. [3] [4]

Dabei haben wir uns für das Blinzeln entschieden, da wir es im Graphen zumindest mit bloßen Augen deutlich besser erkennen konnten als Armbewegungen (s. Abbildung 2).

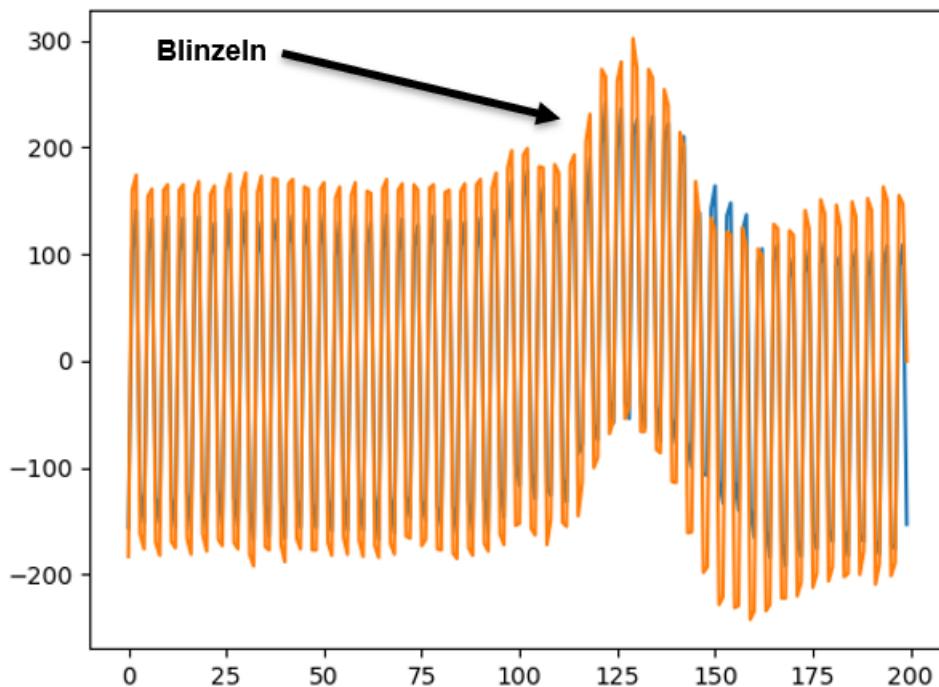


Abbildung 2: Ausschnitt eines EEG mit Blinzeln

Für unser Projekt nutzen wir zwar die Ausrüstung für EEG, aber genau genommen wird das Blinzeln mithilfe von Elektromyographie (EMG) erkannt, denn aufgrund der Platzierung der

zwei verwendeten Elektroden direkt über den Augen werden nicht die Spannungsdifferenzen des Gehirns gemessen, sondern die für das Blinzeln verantwortlichen Augenmuskeln [5].

Wir haben uns für diesen Anfang entschieden, da im EMG ein Blinzeln sehr sicher erkennbar ist und wir so eine mangelnde Genauigkeit des Messgerätes als Fehlerquelle ausschließen konnten, solange wir noch an dem neuronalen Netz und der Datenverarbeitung arbeiten.

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. [1]

Darauf wollen wir erstmal verzichten, da die Konditionierung Zeit benötigen und nicht unser Ziel einer allgemeinen Anwendbarkeit erfüllen würde.

Insgesamt haben wir zwei Earclips, die an den Ohren befestigt werden und zum Filtern von Störungen dienen, und vier Messelektroden, mit einer zeitlichen Auflösung von jeweils 200 Hertz (200 Messungen pro Sekunde).

Zwei dieser Elektroden haben wir auf der Stirn platziert (links: Elektrode 1, rechts: Elektrode 2), zwei jeweils links (Elektrode 3) und rechts (Elektrode 4) auf dem Schädel, etwas über und vor dem Ohr (s. Abbildung 3). Wir haben uns für diese Platzierung entschieden, da die Elektroden so direkt über Muskeln sitzen, die beim Blinzeln bewegt werden.

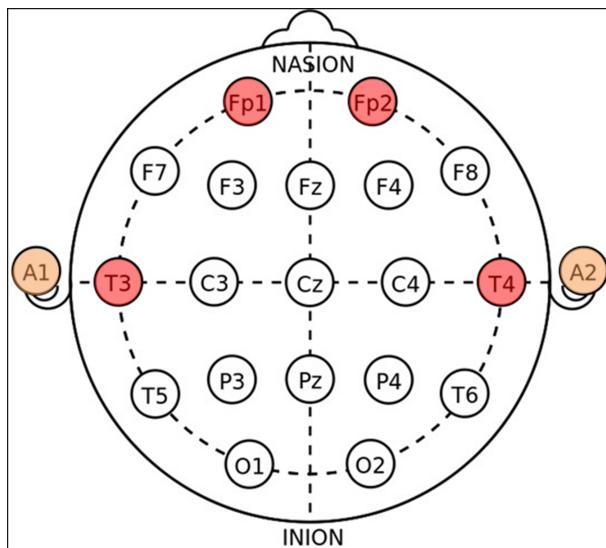


Abbildung 3: 10-20 System mit den von uns genutzten Elektroden rot und Earclips orange gefärbt

Uns ist nach der Aufnahme der Trainingsdaten aufgefallen, dass die Elektroden 3 und 4 deutlich schlechtere Signalqualität hatten als Elektroden 1 und 2, vermutlich weil wir für Elektroden 1 und 2 Flat Electrodes, für Elektroden 3 und 4 aber Spike Electrodes benutzt haben, und die Elektroden 3 und 4 weiter von den Augen entfernt sind.

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

Also kriegen wir $2 * 200 = 400$ Signale pro Sekunde.

3.2.2 Fourier-Analyse

Uns wurde von Professor Everling 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, Funktion, und mehr bestimmen, indem diese in Sinus-Kurven zerlegt wird, sie dient also zur Spektralanalyse.

Wir nutzen dafür die Fast Fourier Transformation (FFT), welche lediglich eine komplexere aber effizientere Form der Diskreten Fourier Transformation (DFT) ist [6].

Grund für die Verwendung einer DFT ist, dass wir ein zeitdiskretes endliches Signal haben, also eine endliche Anzahl (200 pro Elektrode) an Datenpunkten, die auf konkrete Zeitpunkte bezogen sind, und keine unendlichen (iterativen) Funktionen.

Aus der FFT folgt ein Array (eine Liste) an Werten. Der Index dieser Liste bestimmt, für welche Frequenz der Wert gilt (erster Wert: 1 Hertz, zweiter Wert: 2 Hertz, etc.) und der Wert gibt die Amplitude der entsprechenden Sinus-Kurve an [8].

So lässt sich bestimmen, welche Frequenzen am stärksten vorkommen. Außerdem können dann Frequenzen herausgefiltert werden, indem die entsprechenden Indices ignoriert 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.

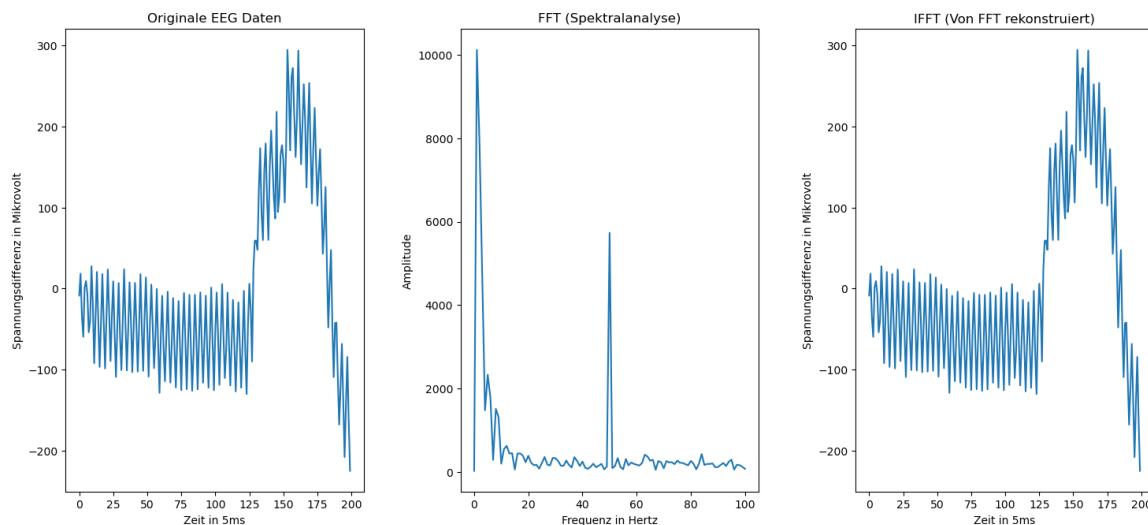


Abbildung 4: Das EEG eines Blinzels, eine FFT dessen, und eine IFFT

Aber vor der Implementation haben wir getestet, ob eine Spektralanalyse für unseren Zweck überhaupt sinnvoll ist. 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. Somit ist es geeignet, da die KI in der Lage sein sollte, diesen Unterschied zu erkennen.

Ob FFT einen 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 Frequenzen nicht so groß und vermutlich für eine sichere Erkennung nicht unbedingt notwendig.

Die große Amplitude bei 50 Hertz ist aufgrund der Verstrahlung der Umgebung, die meist besonders die Frequenz von 50 Hertz betrifft [4].

Um eine Verwirrung der KI zu verhindern, können wir bei Verwendung von FFT also auch einfach die Amplitude für 50 Hertz auf 0 setzen.

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 µs.

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.

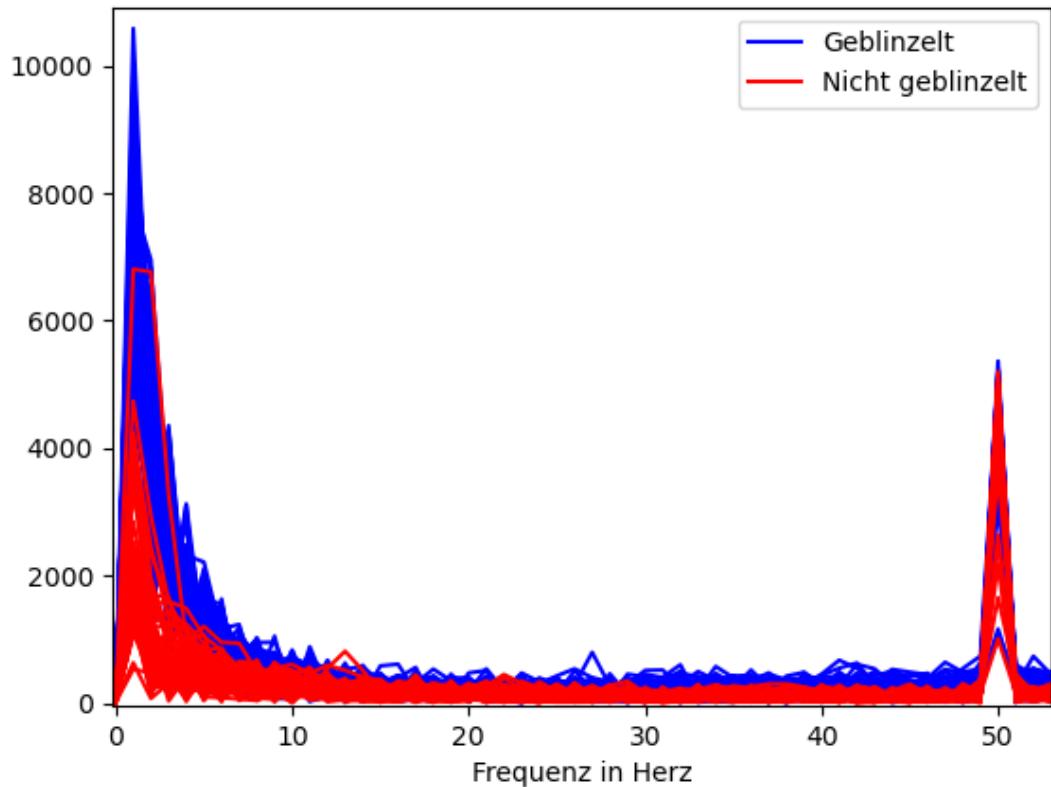


Abbildung 5: Die FFT unserer EEG-Daten

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 eines Neurons gibt es den sogenannten Forward Pass. Dabei beginnt man im ersten Hidden Layer damit, für alle Neuronen den sogenannten Netzininput (auch net input) zu berechnen. Um den Netzininput 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 Netz Input 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. 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 machen. Daher auch der Name Forward Pass. Die Interpretation der Outputs hängt von den Trainingsdaten ab. (s. Backwardpass). Die allgemeine Formel für die Aktivierung eines Neurons lautet also:

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

$$a_i = \text{sig}(\text{netzinput}_i)$$

wobei a_i = die Aktivierung des Neurons i und netzinput_i = der Netzininput des Neurons i .

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 machen. Daher auch der Name Forward Pass. Die Outputs sind dann lediglich die Aktivierungen der Neuronen im Output Layer. Die Interpretation dieser hängt von den Trainingsdaten ab (s. Backpropagation - Theorie). Eine allgemeine Formel für die Aktivierung eines Neurons wäre also:

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

wobei a_j = die Aktivierung des Neurons j , L = der nächste Layer, a_L = alle Aktivierungen des Layers L , W_{Lj} = alle Gewichte zwischen dem Neuron j und den Neuronen des Layers L , und b_j = der Bias des Neurons j . [9]

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.

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

Abbildung 6: Graph der Sigmoidfunktion

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.

Backwardpass

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 ΔW_{ij} = um wie viel das Gewicht W zwischen den Neuronen j und i verändert werden muss, ϵ = die Lernrate (meist ein kleiner Wert wie 0.001), δ_i ≈ die Ableitung der Cost des Neuronen 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 δ_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 $\text{sig}'(x) = \text{sig}(x) * (1 - \text{sig}(x))$, 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 Netz Input 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 δ '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 (δ), 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, δ_L = alle δ '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 δ -Werte für jeden Layer zu berechnen und für die nächsten Berechnungen zu speichern – daher auch der Name Backpropagation. [7] [8] [10]

Flux

Wir haben in unserem letzten Projekt bereits ein neuronales Netzwerk mit dieser Funktionsweise selbst programmiert, um es besser verstehen zu können [19]. Für dieses Projekt haben wir allerdings das Package Flux benutzt, welches die gleichen oder besseren Ergebnisse liefern sollte, mit deutlich besserer Performance, da es stark optimiert und sehr weit entwickelt wurde.

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. 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.

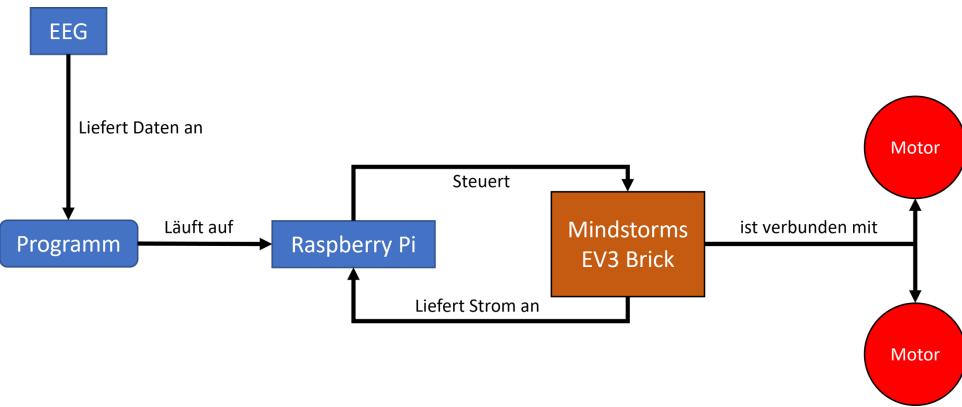


Abbildung 7: Ablauf von Gehirnaktivität zur Motorbewegung

Wir haben dafür das Package ev3dev.jl benutzt, welches wir bereits für die Robotik-AG programmiert hatten. Deshalb ist der Roboter bereits fertig, obwohl das neuronale Netz noch nicht so weit ist.

Mehr technische Details zu der Umsetzung lassen sich beim GitHub Repository des Packages finden [18].

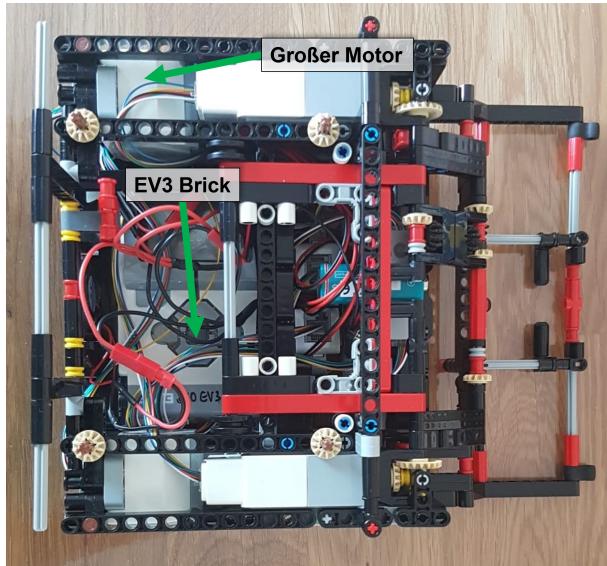


Abbildung 8: Unser Roboter (ohne Raspberry Pi, der über den EV3-Brick gehört)

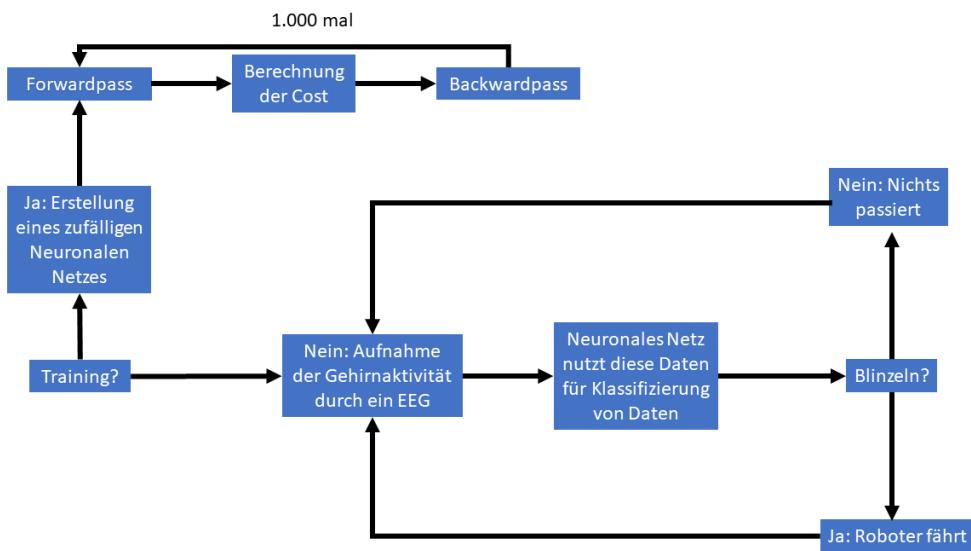


Abbildung 9: Funktionsweise des Programms

4 Ergebnisse

Der gesamte Code sowie der aktuellste Bericht können auf unserem GitHub Repository gefunden werden! [20]

Zuerst haben wir versucht, eine KI zu trainieren, welche erkennen kann, ob eine Person gerade geblinzelt hat oder nicht. Diese könnte man zum Beispiel nutzen, indem man einen Roboter immer dann nach vorne fahren lässt, wenn eine Versuchsperson blinzelt.

Wir haben uns für ein neuronales Netzwerk entschieden, welches als Input eine Sekunde an EEG-Daten nimmt, da sich die Auswirkungen von Blinzeln ungefähr für diesen Zeitraum in den EEG-Daten abbilden. Bei diesem Versuch haben wir keine Fourier Transformation benutzt.

Als Outputs haben wir uns für zwei Neuronen entschieden. Dabei stehen die Werte jeweils für die Sicherheit des Netzwerkes, dass geblinzelt oder nicht geblinzelt wurde.

Unsere Trainingsdaten bestehen dann aus 200 dieser Datensätze, 100 davon mit Blinzeln und 100 ohne. Unser Netzwerk haben wir aber nur mit 90% dieser Datensätze (180) trainiert, damit wir mit den restlichen 20 kontrollieren konnten, ob das Netzwerk auch unbekannte Daten richtig verarbeiten kann.

Für den Aufbau der Hidden Layer des Netzwerkes haben wir uns entschieden, da wir mehrere verschiedene Ansätze ausprobiert haben und diese Struktur konsistent die besten Ergebnisse geliefert hat. Die Struktur ist zu sehen in Abbildung 10.

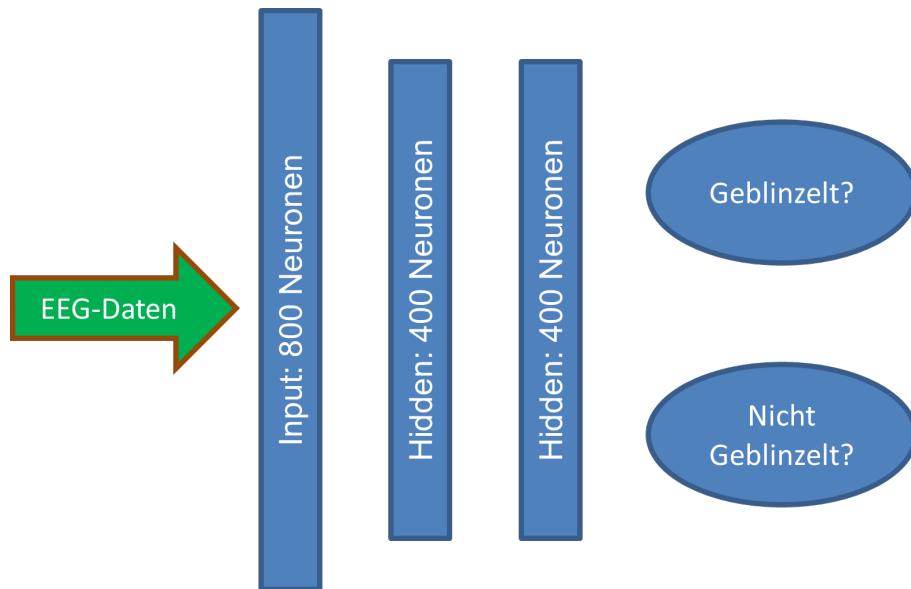


Abbildung 10: Struktur unseres neuronalen Netzwerks

Das Ergebnis war positiv:

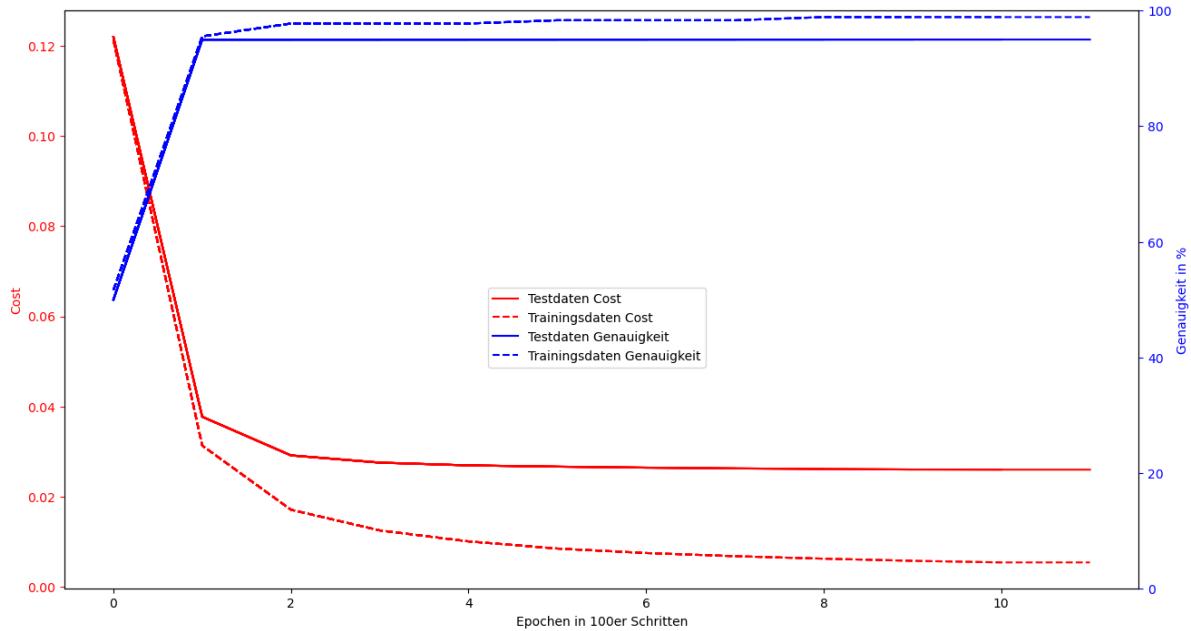


Abbildung 11: Der Cost- und Genauigkeitsverlauf der Test- und Trainingsdaten beim Trainieren des Netwerkes ohne FFT

Wie man an Abbildung 11 sehen kann, erreichte das neuronale Netzwerk in diesem Fall in nur 1000 Epochen (ca. 5 Minuten Rechenzeit) eine Genauigkeit von 95%, bei anderen Durchläufen

erhielten wir auch 100%.

Bei dem Versuch mit Frequenzfiltern durch FFT war die Struktur ähnlich, die Menge an Neuronen in jedem Layer wurde jedoch halbiert, da es nur 100 Inputs gibt, weil die Fast Fourier Transformation bei einer zeitlichen Auflösung von 200 Hertz nur 1 - 100 Hertz berechnen kann. Wir erhielten bei der gleichen Zahl an Wiederholungen („Epochen“) ähnliche Ergebnisse (s. Abbildung 12).

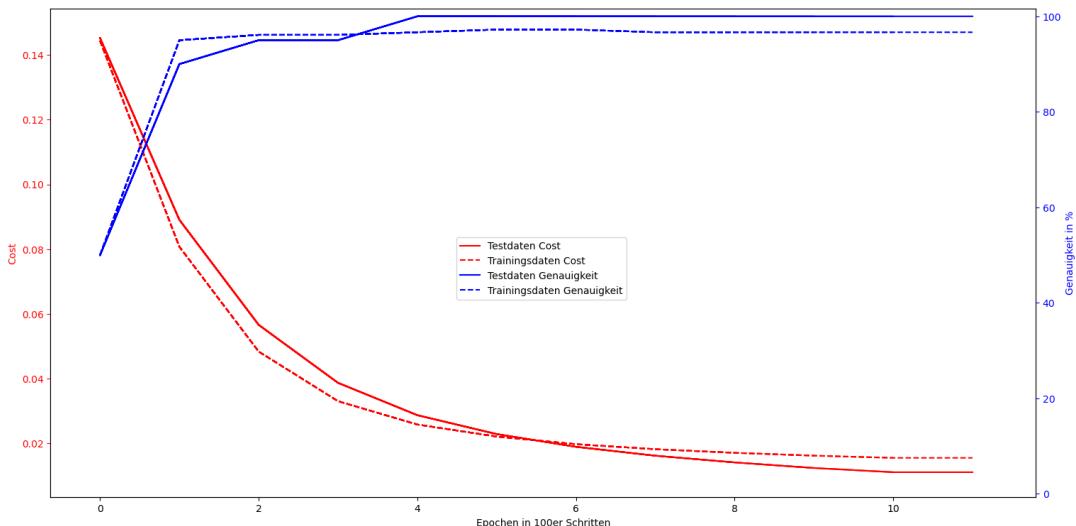


Abbildung 12: Der Cost- und Genauigkeitsverlauf der Test- und Trainingsdaten beim Trainieren des Netzwerkes mit FFT

Dies heißt, dass die FFT wie erwartet die Ergebnisse nicht verschlechtert hat. Jedoch lässt sich auch nicht genau sagen, inwiefern sie besser ist als keine, wenn überhaupt. Denn die kleinen Erfolgs-Unterschiede, die man oft sieht, wie hier auch, stammen meist davon, dass jedes neuronale Netzwerk zufällig initialisiert wird und somit, selbst mit einem ansonsten identischen Verfahren, einen leichten Unterschied in den Ergebnissen zeigen kann.

Zu Beginn hatten wir ein noch größeres Problem: Die Testdaten konnten sicher erkannt werden, aber als wir es mit neuen Daten von anderen Personen und in anderen Räumen versuchten, hat das Netzwerk meist komplett versagt.

Das Ziel der allgemeinen Anwendung war also nicht gegeben gewesen.

Dieses Problem konnten wir allerdings lösen, indem wir einen größeren Raum benutzt und mehr Daten gesammelt haben. So gab es eine größere Diversität an Daten und das neuronale Netzwerk wurde darauf trainiert, auf die Umgebung zurückzuführende Veränderungen zu ignorieren.

Das neuronale Netzwerk konnte Blinzeln danach sicher erkennen. Wie in Abbildung 13 zu erkennen ist, war die Differenz zwischen den beiden Output-Neuronen als nicht geblinzelt wurde fast 1, bei kräftigem Blinzeln auch. Die Klassifizierung des Netzwerks war also mit sehr großer Sicherheit getätigter worden. Bei leichtem Blinzeln hat das erste Output-Neuron (geblinzelt, grüner Graph) das zweite (nicht geblinzelt, roter Graph) zwar nicht oder nur leicht übertragen, war aber dennoch klar erkennbar.

Es fällt auch auf, dass die Summe der beiden Output-Neuronen stets ungefähr 1 ergibt und die Graphen sich an $y = 0.5$ grob spiegeln. Wir vermuten, dass beim neuronalen Netz entweder nur einer der Werte ausgerechnet wird und der andere der Umkehrwert dessen ist ($1 - P$), oder beide fast identisch berechnet werden.

Ein Video mit Aufnahme der Augen, des Graphen, und des Roboters ist auf YouTube verfügbar, auch wenn die Synchronisierung bei zwei der drei Clips nicht automatisch funktioniert hat und deswegen einige Teile vielleicht nicht genau synchron sind [11].

Jedoch merkt man vor allem im letzten Clip (wo wir mithilfe von Audiowellenanalyse die Augen- und Roboteraufnahme synchronisieren konnten) und auch als Testperson, dass eine Verzögerung

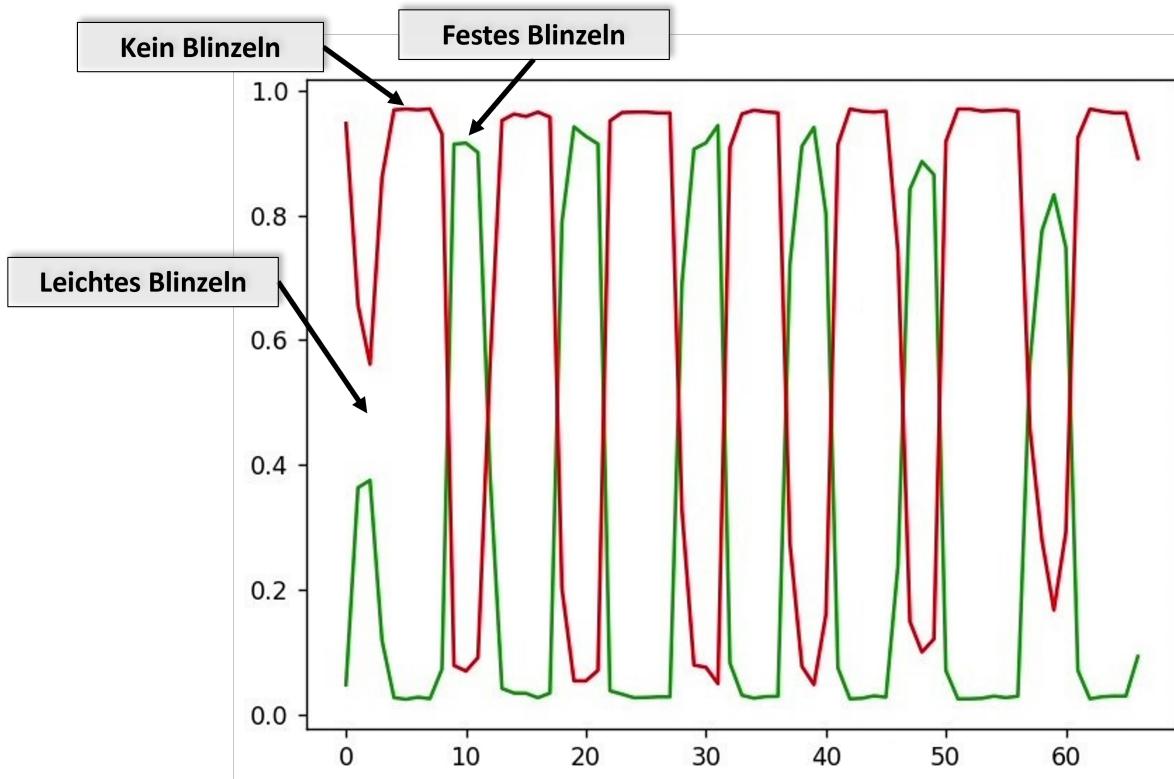


Abbildung 13: Ein Live-Test unseres Projektes, die grüne Linie repräsentiert das erste Output-Neuron (es wurde geblinzelnt), die rote das zweite Output-Neuron (es wurde nicht geblinzelnt). Die x-Achse ist die Zeit in 0,25 Sekunden Schritten

kaum spürbar ist. Grund dafür ist vermutlich, dass wir für diese Tests alle 0,25 Sekunden das EEG der letzten Sekunde abgefragt haben. Somit gibt es nicht eine maximale extra Verzögerung von z.B. 1000 ms, sondern nur 250 ms.

Die Kosten der benutzten Hardware halten sich ebenfalls 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.

5 Diskussion

Wir sind zufrieden damit, dass wir unser erstes Teilziel erreicht haben.

Als ersten folgenden Schritt wollen wir zuerst die aktuell offenstehenden Fragen beantworten, indem wir das Trainieren mit nur einem Output-Neuron ausprobieren und 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.

Der nächste, große Schritt wird dann sein, die Elektroden an anderen Positionen anzubringen, wo sie nicht mehr auf Muskeln sitzen und wir somit kein EMG mehr benutzen, sondern ein tatsächliches EEG.

Wir planen dafür, den Okzipitalen Kortex (mittig am Hinterkopf, leicht über dem Inion) zu verwenden. Denn Professor Everling hat uns informiert, dass dort α -Wellen (7 - 13 Hertz) besonders stark ausgeprägt sind, und er hat uns den Tipp gegeben, dass unser EEG vor allem diesen Frequenzbereich gut aufnehmen könnte. Dies hat sich bereits bestätigt, denn wie man in

Abbildung 5 sehen kann, ist der niedrige Frequenzbereich zumindest beim Blinzeln am stärksten ausgeprägt.

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.⁸

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

Literatur

- [1] Ujwal Chaudhary, Niels Birbaumer und Ander Ramos-Murgialday. „Brain-computer interfaces for communication and rehabilitation“. In: *Macmillan Publishers Limited* 12 (Sep. 2016), S. 513–525.
- [2] Wikipedia. *Elektroenzephalografie – Wikipedia, The Free Encyclopedia*. <http://de.wikipedia.org/w/index.php?title=Elektroenzephalografie&oldid=216289194>. 2022. (Besucht am 13.01.2022).
- [3] N. Birbaumer und R. F. Schmidt. „Biologische Psychologie“. In: Springer Verlag, Berlin Heidelberg, 2010. Kap. Kapitel 20.5, S. 468–493.
- [4] Universität Lübeck. *Elektroenzephalographie (EEG) und Ereigniskorrelierte Potentiale (EKP)*. Praktikumsskript.
- [5] Wikipedia. *Electromyography — Wikipedia, The Free Encyclopedia*. <http://en.wikipedia.org/w/index.php?title=Electromyography&oldid=1070043736>. [Online; accessed 16-February-2022]. 2022.
- [6] 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).
- [7] Larry Hardesty. „Explained: Neural networks“. In: *MIT News* (2017). URL: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>.

⁸<https://www.uwo.ca/bmi/investigators/stefan-everling.html>

Videos

- [8] Grant Sanderson 3blue1brown. *Was ist eine Fourier-Transformation? Eine visuelle Einführung*. YouTube Video. 2018. URL: <https://www.youtube.com/watch?v=spUNpyF58BY> (besucht am 10.01.2022).
- [9] Brotcrunsher. *Neuronale Netze - Backpropagation - Forwardpass*. YouTube Video. 2017. URL: <https://www.youtube.com/watch?v=YIqYBxpv53A> (besucht am 15.01.2022).
- [10] Brotcrunsher. *Neuronale Netze - Backpropagation - Backwardpass*. YouTube Video. 2017. URL: <https://www.youtube.com/watch?v=EAtnQCut6Qno> (besucht am 15.01.2022).
- [11] 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).

Programme

- [12] Michael Innes u. a. „Fashionable Modelling with Flux“. In: *CoRR* abs/1811.01457 (2018). arXiv: 1811.01457. URL: <https://arxiv.org/abs/1811.01457>.
- [13] Mike Innes. „Flux: Elegant Machine Learning with Julia“. In: *Journal of Open Source Software* (2018). DOI: 10.21105/joss.00602.
- [14] Andrey Parfenov et al. *Brainflow*. GitHub Repository. 2018. URL: <https://github.com/brainflow-dev/brainflow>.
- [15] 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.
- [16] 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. arXiv: 1712.03112 [cs.PL].
- [17] Steven G. Johnson. *PyPlot.jl*. GitHub Repository. 2012. URL: <https://github.com/JuliaPy/PyPlot.jl>.
- [18] Alexander Reimer. *ev3dev.jl*. GitHub Repository. 2022. URL: <https://github.com/AR102/ev3dev.jl>.
- [19] Alexander Reimer und Matteo Friedrich. *AI-Composer*. GitHub Repository. 2021. URL: <https://github.com/AR102/AI-Composer.jl>.
- [20] Alexander Reimer und Matteo Friedrich. *Interpreting EEG with AI*. GitHub Repository. 2022. URL: <https://github.com/AR102/Interpreting-EEG-with-AI>.