

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 Zusammenfassung</b>	<b>3</b>
<b>2 Einleitung</b>	<b>3</b>
<b>3 Methode und Vorgehensweise</b>	<b>4</b>
3.1 Materialien . . . . .	4
3.2 Vorgehensweise . . . . .	5
3.2.1 Elektroenzephalographie . . . . .	5
3.2.2 Neuronales Netz . . . . .	8
3.2.3 Roboter . . . . .	12
<b>4 Ergebnisse</b>	<b>13</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>17</b>

## 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 durchgeföhrter 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 bezahlbarer 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
- Software
  - Flux.jl für das neuronale Netz <sup>1</sup> [10] [11]
  - BrainFlow.jl als Schnittstelle zum EEG <sup>2</sup> [12]
  - FFTW.jl für die Fast Fourier Transformation <sup>3</sup> [13]
  - CUDA.jl zum effektiven Nutzen einer NVIDIA GPU <sup>4</sup> [14]
  - PyPlot.jl zum Plotten <sup>5</sup> [15]
  - BSON.jl zum Speichern und Laden von Netzwerken <sup>6</sup>
  - ev3dev.jl zum Steuern eines EV3-Roboters durch einen Raspberry Pi <sup>7</sup> [16]

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).

---

<sup>1</sup><https://github.com/FluxML/Flux.jl>

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

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

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

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

<sup>6</sup><https://github.com/JuliaIO/BSON.jl>

<sup>7</sup><https://github.com/AR102/ev3dev.jl>

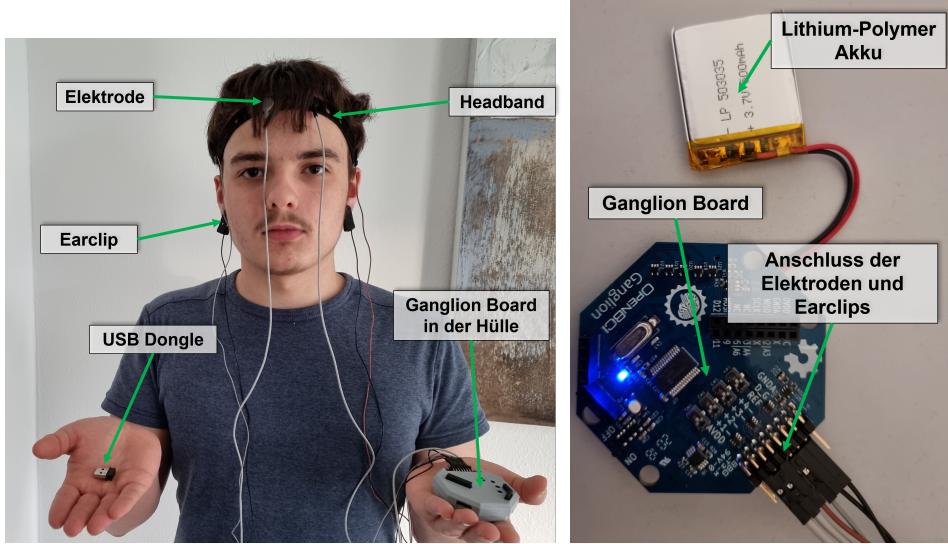


Abbildung 1: Das EEG und Zubehör

### 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 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]

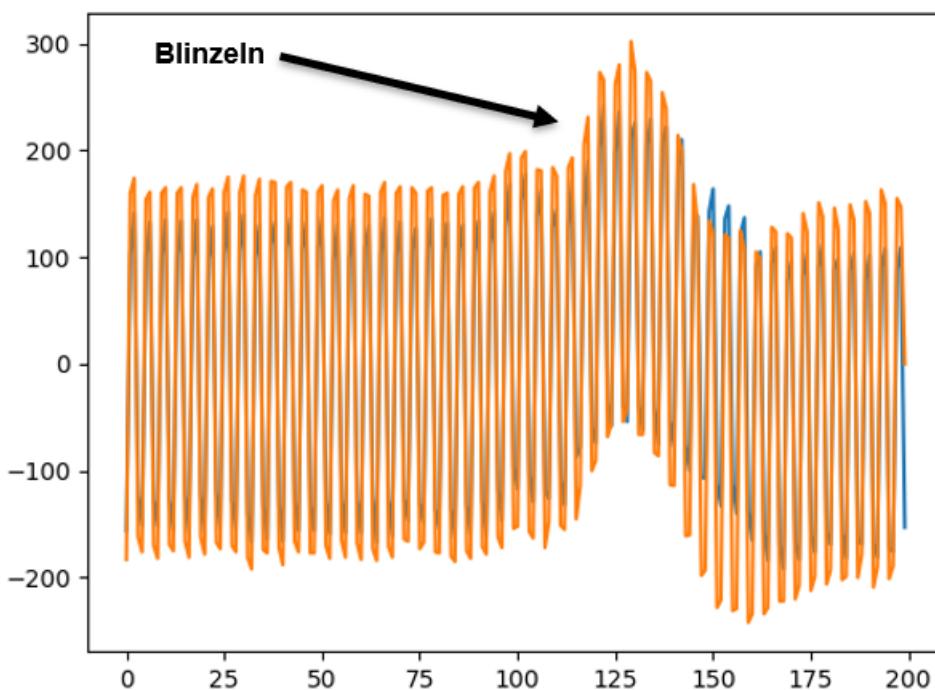
Weiter ist es möglich, nur durch Gedanken eine Steuerung auszuführen. Dies funktioniert jedoch meist durch instrumentelle oder klassische Konditionierung, also durch das Bestrafen und Belohnen auf Basis der Messungen des EEG. So kann das Gehirn darauf trainiert werden, auf Verlangen eine bestimmte, vorher festgelegte Aktivität auszulösen,

die dann gemessen und ausgewertet werden kann. [1]

Dies wollen wir nicht machen, da die Konditionierung Zeit benötigt, nicht unser Ziel einer allgemeinen Anwendbarkeit erfüllt und voraussichtlich bessere Ausrüstung erfordert.

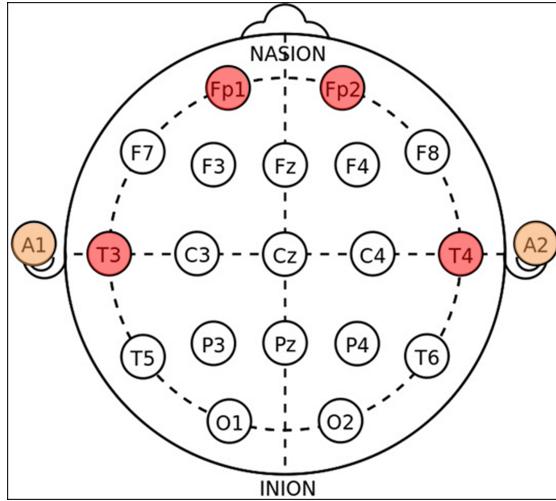
Zuerst probieren wir deshalb die Erkennung von EKPs, da diese deutlich leichter zu klassifizieren sind und wir so erstmal das Konzept eines neuronalen Netzes in einem solchen Kontext ausprobieren können. Denn wenn unser neuronales Netz EKPs noch nicht sicher erkennen kann, wären komplexere Zusammenhänge ohne vorherige Konditionierung erst recht nicht möglich.

Spezifisch probieren wir das Blinzeln aus, da dies in den EEG-Daten sogar von Menschen sehr einfach erkannt werden kann, wie man in Abbildung 2 sehen kann.



**Abbildung 2:** Ausschnitt eines EEG mit Blinzen

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, zwei jeweils links und rechts auf dem Schädel, etwas über dem Ohr, da dies die Standard-Aufstellung in der Dokumentation ist (s. Abbildung 3) [5]. Wir planen, sie basierend auf unseren Ergebnissen wo nötig anzupassen.



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

Also kriegen wir  $4 * 200 = 800$  Signale pro Sekunde.

Uns wurde von Professor Everling empfohlen, die Fourier Transformation zur Vorbereitung der Daten für das neuronale Netz zu nutzen.

Die Fourier Transformation kann die verschiedenen zugrundeliegenden Frequenzen einer Zahlenfolge extrahieren, indem diese in Sinus-Kurven zerlegt wird.

Daraus folgt ein Array (eine Liste) an Werten. Der Index bestimmt, für welche Frequenz der Wert gilt (erster Wert: 1 Hertz, zweiter Wert: 2 Hertz, etc.) und der Wert gibt an, womit die entsprechende Sinus-Funktion multipliziert werden muss, um die originale Funktion wiederzuerlangen. [7]

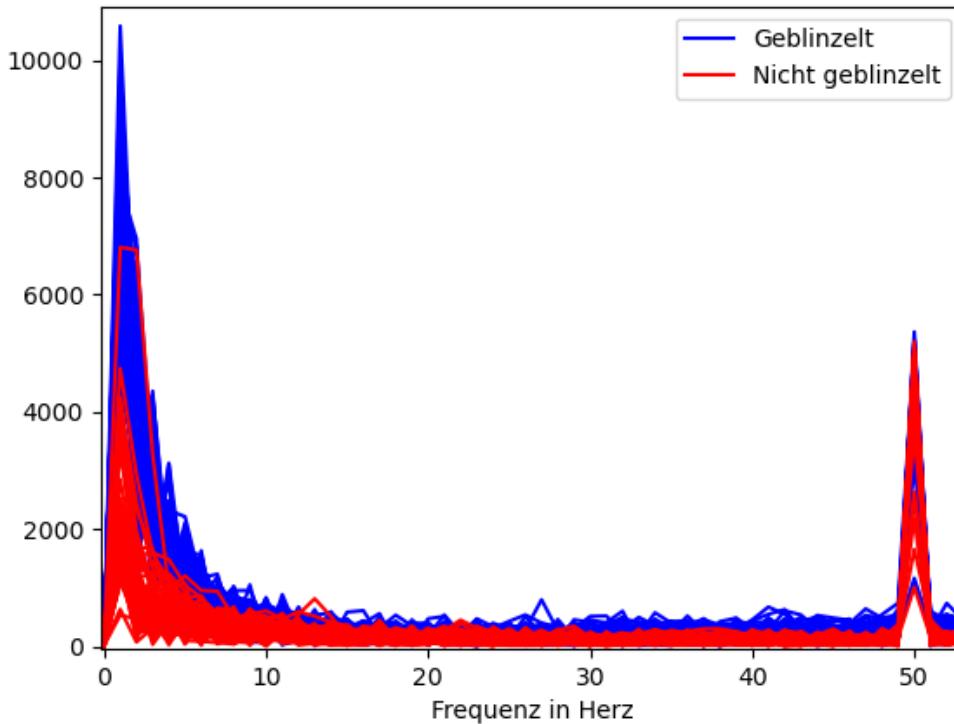
So lässt sich bestimmen, welche Frequenzen am stärksten vorkommen. Außerdem können dann Frequenzen herausgefiltert werden, indem die entsprechenden Indices ignoriert werden.

Wir haben dazu spezifisch die Fast Fourier Transformation (FFT) des Julia-Pakets FFTW genutzt. Zuerst haben wir die Sinnhaftigkeit einer Fourier Transformation für unser Projekt getestet, indem wir sie auf unsere Daten angewandt und dies in einem Plot ausgegeben haben.

Wie man in Abbildung 4 sehen kann, ist dabei selbst mit den bloßen Augen ein Unterschied zwischen geblinzelt und nicht geblinzelt in allen Frequenzen zu sehen, vor allem bei den niedrigeren Frequenzen.

Der plötzliche Anstieg bei 50 Hertz ist aufgrund der Verstrahlung der Umgebung, die meist besonders die Frequenz von 50 Hertz betrifft [4].

Aufgrund dieses Tests haben wir uns entschieden, auch auszuprobieren, das neuronale Netzwerk mit EEG-Daten zu trainieren, welche vorher mit der Fast Fourier Transformation verarbeitet wurden.



**Abbildung 4:** Die FFT unserer EEG-Daten

### 3.2.2 Neuronales Netz

Ein neuronales Netzwerk besteht aus drei Teilen: dem Input Layer, den Hidden Layers und dem Output Layer.

Der Input 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  $\text{netzinput}_i$  = der Netzinput 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:

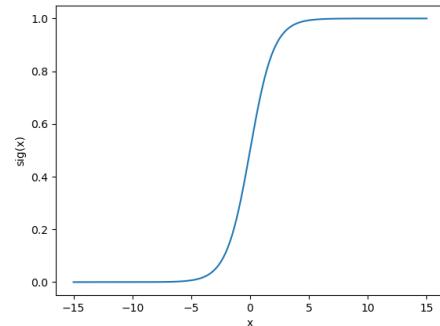
$$a_j = \text{sig} \left( \sum_L (a_L * W_{Lj}) + b_j \right)$$

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  $j$ . [8]

### **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.



**Abbildung 5:** Graph der Sigmoidfunktion

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 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  $\Delta W_{ij}$  = 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 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  $\delta_i$  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 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.

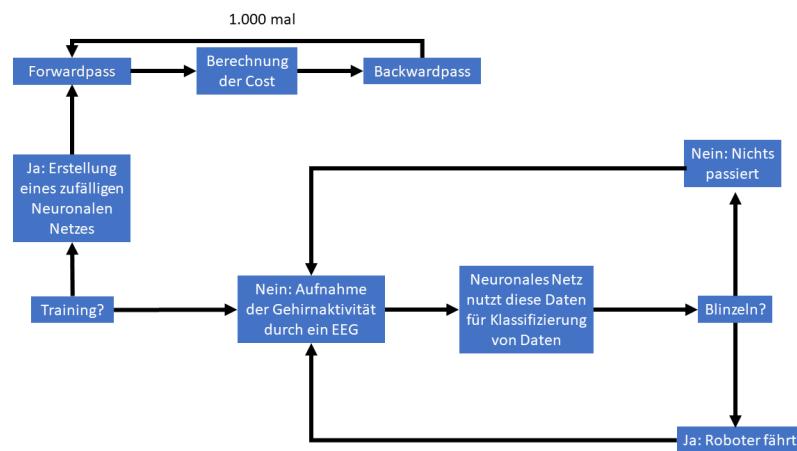
Für die Neuronen der Hidden Layers muss man alle  $\delta$ 's des nächsten Layers mit den von dem Neuron dorthin führenden Gewicht multiplizieren und dann summieren. Dadurch werden die Änderungen, die die Aktivierungen dieser Neuronen brauchen ( $\delta$ ), zusammengerechnet, da natürlich die Aktivierungen 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. [6] [7] [9]



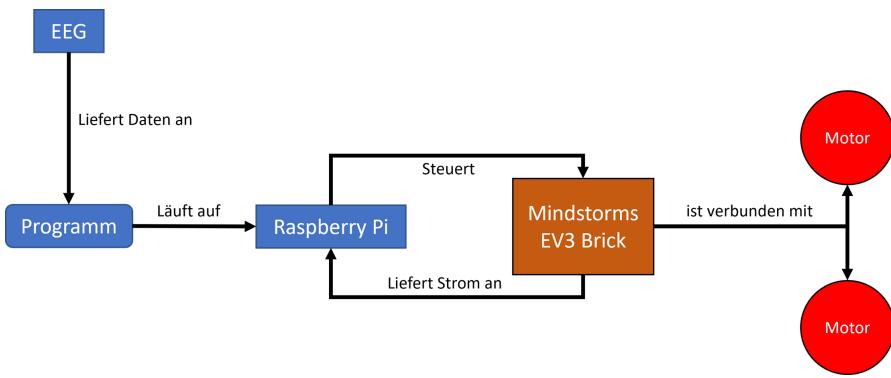
**Abbildung 6:** Funktionsweise des Programms

## Flux

Wir haben in unserem letzten Projekt bereits ein neuronales Netzwerk mit dieser Funktionsweise selbst programmiert, um es besser verstehen zu können [17]. Für dieses Projekt haben wir allerdings das Package Flux benutzt, welches die gleichen Ergebnisse liefern sollte, jedoch mit deutlich besserer Performance, da es sehr stark optimiert wurde.

### 3.2.3 Roboter

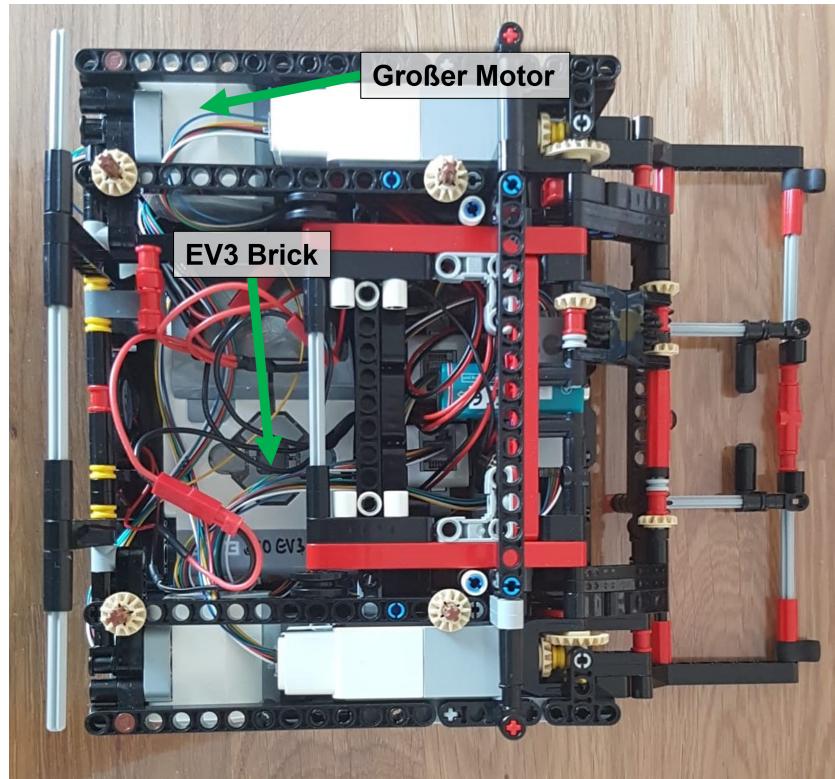
Für unseren Roboter haben wir uns entschieden, Lego Mindstorms EV3 Motoren zu benutzen, die von einem EV3-Brick gesteuert werden, der wiederum von einem Raspberry Pi 3B gesteuert wird. Der Grund dafür, dass wir keine Motoren direkt mit dem Raspberry Pi steuern, ist, dass wir schon alle Teile für einen 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 [16].



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

## 4 Ergebnisse

Der gesamte Code sowie der aktuellste Bericht können auf unserem GitHub Repository gefunden werden! [18]

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.

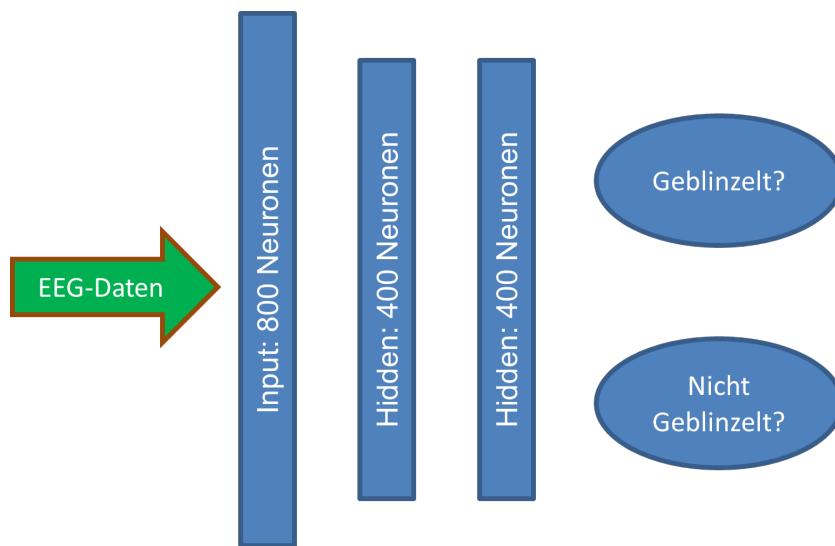
Wir haben uns für ein neuronales Netzwerk entschieden, welches als Input eine Sekunde an EEG-Daten nimmt, da sich die Auswirkungen von Blinzeln ungefähr für diesen Zeitraum in den EEG-Daten abbilden. Bei diesem Versuch haben wir keine Fourier Transformation benutzt.

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.

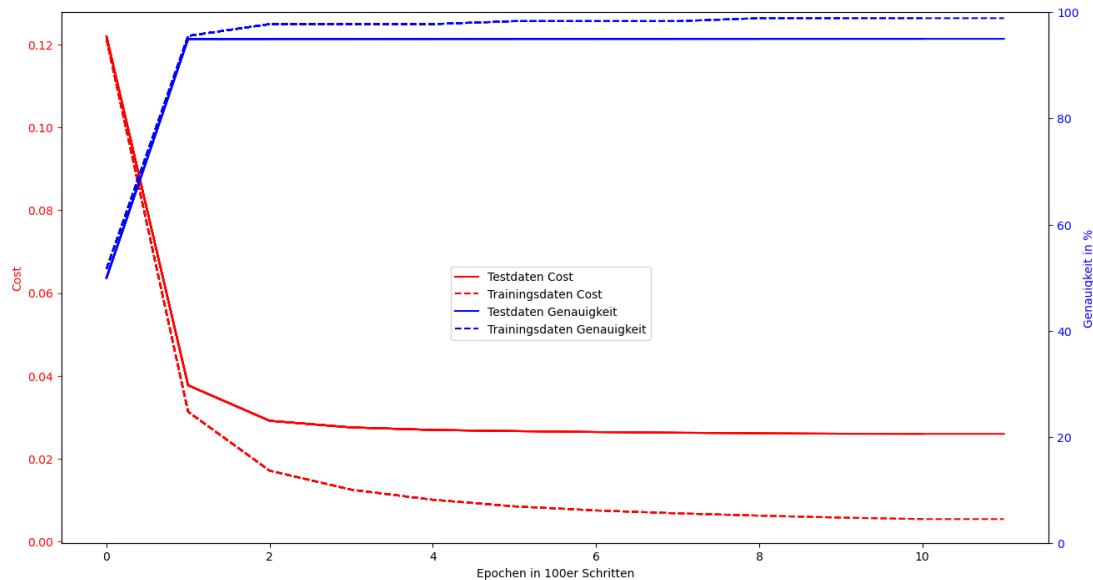
Für den Aufbau der Hidden Layer des Netzwerkes haben wir uns entschieden, da wir mehrere verschiedene Ansätze ausprobiert haben und diese Struktur konsistent die besten

Ergebnisse geliefert hat. Die Struktur ist zu sehen in Abbildung 9.



**Abbildung 9:** Struktur unseres neuronalen Netzwerks

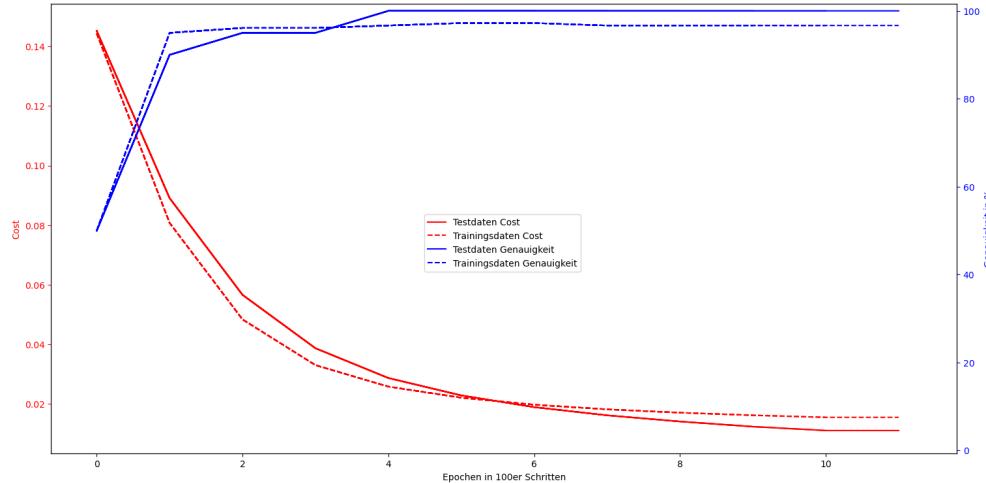
Das Ergebnis war positiv:



**Abbildung 10:** Der Cost- und Genauigkeitsverlauf der Test- und Trainingsdaten beim Trainieren des Netwerkes ohne FFT

Wie man an Abbildung 10 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 11).



**Abbildung 11:** Der Cost- und Genauigkeitsverlauf der Test- und Trainingsdaten beim Trainieren des Netzwerkes mit FFT

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

Es gab aber noch ein 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 ist also noch nicht gegeben.

Die Kosten der benutzten Hardware halten sich jedoch noch in Grenzen: Beim Kauf hat die gesamte EEG-Ausstattung ca. 550€ gekostet.

Die Performance des Programms kann außerdem zwar noch verbessert werden, jedoch dauern 1000 Epochen auf unseren persönlichen Systemen bereits lediglich 3 Minuten.

## 5 Diskussion

Ein großes Problem, welches wir lösen wollen, ist die oben genannte Tatsache, dass das neuronale Netzwerk für neue Daten fast nutzlos ist. Wir haben bereits einige Ideen, wie wir dies verbessern können:

Wir glauben, dass hauptsächlich die fehlende Diversität der Daten verantwortlich ist. Denn alle Trainings- und Testdaten wurden im selben Raum, am selben Tag, von

derselben Person aufgenommen. Somit lernt das neuronale Netzwerk nicht, zwischen tatsächlichen Unterschieden in den Daten und unwichtigen Unterschieden, die z.B. durch andere Hintergrundstrahlung oder Person verursacht werden, zu differenzieren.

Um dieses Problem zu lösen, wollen wir also versuchen, mehr Trainingsdaten zu sammeln, die unter verschiedenen Bedingungen aufgenommen wurden, um unser Ziel der allgemeinen Nutzbarkeit erfüllen zu können. Dafür haben sich bereits Freunde freiwillig erklärt.

Außerdem vermuten wir auch Probleme mit der Platzierung der Elektroden. Denn wir nehmen die EEG-Daten von Blinzeln direkt an der Stirn, über den Augen, auf. Da das Auge ein Dipol ist (zwei unterschiedlich geladene Seiten hat) und beim Blinzeln der Augapfel sich automatisch mit der negativ geladenen Vorderseite nach oben (in Richtung der Elektroden) dreht, könnte so ein direkter elektrischer Ausschlag entstehen – ohne eine bestimmte Frequenz. Wir denken, dies könnte andere Signale, die vom Gehirn kommen, etwas „übertönen“.

Darum wollen wir die Elektroden eher im Bereich des Okzipitalen Kortexes platzieren. Dort sollte es optimal sein, denn dort sind außerdem die Alpha-Wellen des Gehirns besonders stark ausgeprägt, und wie Abbildung 4 gezeigt hat, ist vor allem dieser niedrige Frequenzbereich von Bedeutung.

Sobald wir komplett neue Daten sicher erkennen können, wollen wir das Programm mit Blinzeln direkt den Roboter steuern lassen.

Zudem wollen wir in 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

Danke 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.<sup>8</sup>

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

Danke an Oliver Samkovskij und Ino, die zusammen mit uns den Roboter gebaut haben.

---

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

## 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)*. Praktikum.
- [5] *OpenBCI Documentation*. Produkt-Dokumentation.
- [6] Larry Hardesty. „Explained: Neural networks“. In: *MIT News* (2017). URL: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>.

### Videos

- [7] Grant Sanderson 3blue1brown. *Was ist eine Fourier-Transformation? Eine visuelle Einführung*. YouTube Video. 2018. URL: <https://www.youtube.com/watch?v=sPUmpyF58BY> (besucht am 10.01.2022).
- [8] Brotcrunsher. *Neuronale Netze - Backpropagation - Forwardpass*. YouTube Video. 2017. URL: <https://www.youtube.com/watch?v=YIqYBxpv53A> (besucht am 15.01.2022).
- [9] Brotcrunsher. *Neuronale Netze - Backpropagation - Backwardpass*. YouTube Video. 2017. URL: <https://www.youtube.com/watch?v=EAtQCut6Qno> (besucht am 15.01.2022).

### Programme

- [10] Michael Innes u. a. „Fashionable Modelling with Flux“. In: *CoRR* abs/1811.01457 (2018). arXiv: 1811.01457. URL: <https://arxiv.org/abs/1811.01457>.
- [11] Mike Innes. „Flux: Elegant Machine Learning with Julia“. In: *Journal of Open Source Software* (2018). DOI: 10.21105/joss.00602.
- [12] Andrey Parfenov et al. *Brainflow*. GitHub Repository. 2018. URL: <https://github.com/brainflow-dev/brainflow>.

- [13] 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.
- [14] 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].
- [15] Steven G. Johnson. *PyPlot.jl*. GitHub Repository. 2012. URL: <https://github.com/JuliaPy/PyPlot.jl>.
- [16] Alexander Reimer. *ev3dev.jl*. GitHub Repository. 2022. URL: <https://github.com/AR102/ev3dev.jl>.
- [17] Alexander Reimer und Matteo Friedrich. *AI-Composer*. GitHub Repository. 2021. URL: <https://github.com/AR102/AI-Composer.jl>.
- [18] Alexander Reimer und Matteo Friedrich. *Interpreting EEG with AI*. GitHub Repository. 2022. URL: <https://github.com/AR102/Interpreting-EEG-with-AI>.