Neuronale Netze

zur Auswertung von Detektorsignalen

Bachelorarbeit

im Fachgebiet Physik

vorgelegt von:	Dennis Schmidt
Studienbereich:	Physik
Matrikelnummer:	3979743
Erstgutachter:	Prof. Dr. Reinhard Dörner
Zweitgutachter:	Dr. Achim Czasch

Eidesstattliche Erklärung

Ich versichere, die Bachelorarbeit selbstständig und lediglich unter Benutzung der angegebenen Quellen und Hilfsmittel verfasst zu haben.
Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Frankfurt am Main, den 27. September 2012

Inhaltsverzeichnis

Inhaltsverzeichnis

1. Schreibweisen

Um einen direkten Bezug zu dem hier erarbeiteten Programm deutlich zu machen, werde ich alle Parameter und Funktionen nach denen im Programmcode benennen. Dies scheint im Text manchmal nicht sonderlich passend, ist aber übersichtlicher als mehrere Bezeichnungen zu verwenden. Zudem habe ich meine Notation dem Skript „Neuronale Netze“ von D. Kriesel angepasst. Das Skript ist zu finden unter http://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-2col-dkrieselcom.pdf

2. Physikalischer Hintergrund und Motivation

Delay Line Anoden gehören zu den wichtigsten Detektortypen zur Ortsbestimmung von Elektronen. Dabei werden die einzelnen Elektronen zunächst mit Hilfe einer Micro-Channel-Plate (MCP) zu einer Elektronenwolke vervielfacht. Ein MCP besteht aus vielen schmalen Kanälen (ca. 25 μm). Schlägt ein Elektron an den Rand eines Kanals, so löst es dabei weitere Elektronen aus. Dabei werden die Elektronen durch ein elektrisches Potential weiter beschleunigt. Verlässt diese Wolke nun die MCP, kann man dank der fehlenden Ladungen einen kurzzeitigen Strom messen, der zur Zeitbestimmung des Einschlags benutzt wird.

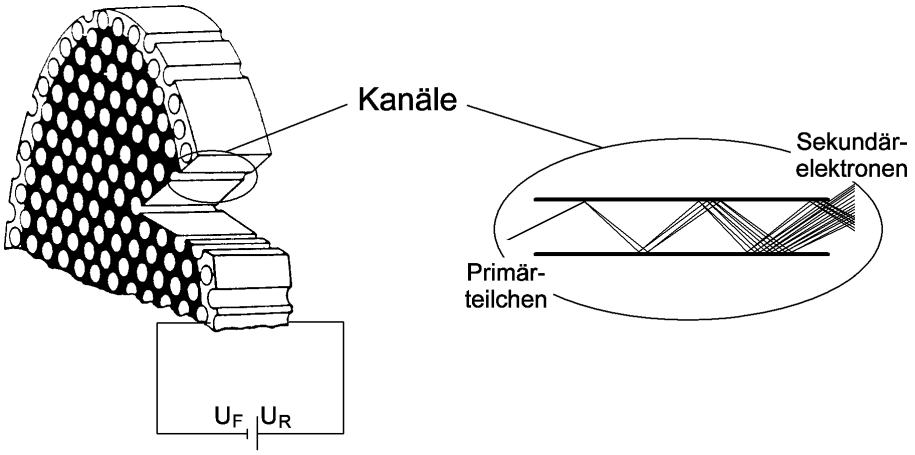

Abbildung 2.1.: Schematische Darstellung eines MCP. Ein Primärteilchen löst eine Elektronenwolke aus.[Ach99]

Der Ort der Elektronenwolke wird anschließend mit Hilfe einer Hex-Anode festgestellt. Eine Hex-Anode besteht aus drei hexagonal angeordneten Anoden. Eine Anode besteht aus einem sehr eng gewickelten Drahtpaar. Einer der Drähte des Paares wird dabei gegenüber dem anderen auf ein etwas positiveres Potenzial vorgespannt. Dieser Draht sammelt die Elektronen. Das Drahtpaar leitet

das Signal dann als Lecherleitung. Für die Aufgabenstellung dieser Arbeit sind die Details der Signalauskopplung nicht relevant, sodass im Folgenden der Einfachheithalber von einem Draht gesprochen wird. Trifft die Elektronenwolke auf den Draht, läuft ein Stromimpuls vom Einschlagsort zu beiden Enden des Drahtes. In der Praxis werden mehrere Windungen von der Elektronenwolke getroffen. Die einzelnen Stromimpulse überlagern sich, wodurch man am Ende des Drahtes eine Verteilung messen kann und so eine bessere Auflösung erreichen kann. Den Ort des Einschlags kann man aus diesen sechs Signalen konstruieren.

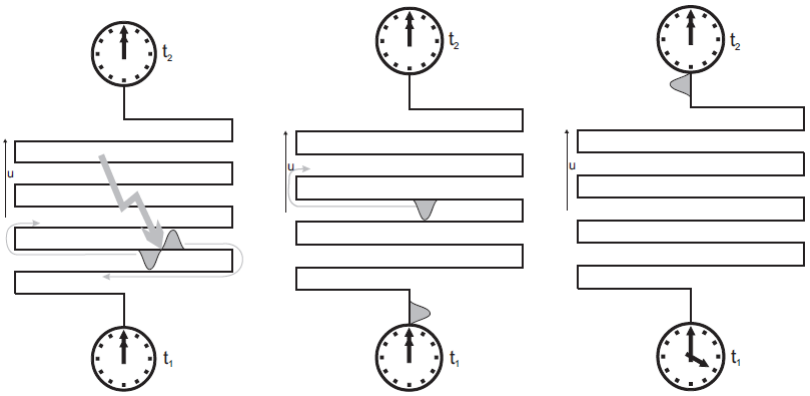

Abbildung 2.2.: Das Elektron trifft auf die Anode, wodurch ein Stromimpuls erzeugt wird(links). Das Signal breitet sich zu beiden Enden des Drahts aus und wird zur Zeit t_1 am einen Ende (mitte) und zur Zeit t_2 am anderen Ende (rechts) aufgezeichnet.[Mec06]

Theoretisch reichen zwei Anoden, um den Ort einfacher Einschläge zu bestimmen. Treffen jedoch mehrere Teilchen gleichzeitig auf den Detektor, so kann es schnell zu Situationen kommen, in denen die einzelnen Signale nicht mehr eindeutig den verschiedenen Einschlägen zugeordnet werden können. Dieses Problem wird durch eine Hex-Anode zwar nicht vollständig behoben, tritt jedoch durch die zusätzliche Anode erst bei mindestens drei Teilchen auf. Die Zuordnung der einzelnen Signale ist auf diese Weise in den meisten Fällen zwar lösbar, jedoch keinesfalls trivial. Aktuell wird hierfür der Algorithmus von Dr. Achim Czasch verwendet. Einzelne Einschläge ordnet er sehr zuverlässig zu. Bei zunehmender Zahl der Signale hat er jedoch Probleme, eindeutige Lösungen zu finden. Ziel dieser Bachelorarbeit ist es, für dieses Problem einen ganz anderen Lösungsansatz zu testen: Künstliche Neuronale Netze.
Ziel des hier trainierten neuronalen Netzes ist es jedoch nicht, den bisherigen Algorithmus komplett zu ersetzen. Vielmehr soll eine ungefähre Ortsinforma-

tion von beiden Teilchen gewonnen werden, die dazu genutzt werden kann, die verschiedenen Signale eindeutig zuzuordnen.
Dabei wird hier auf die bereits implementierte JAVA-Bibliothek »SNIPE« [1] zurückgegriffen. Die erste Aufgabe ist es, diese Bibliothek an die gegebenen Umstände anzupassen und ein einfaches neuronales Netz mithilfe von generierten Daten zu trainieren. Ziel ist es zunächst, einfache Ereignisse zu erkennen. Danach sollen mit Hilfe eines komplexeren Netzes auch die kritischen Mehrfachhits ausgewertet werden können. Am Ende werden noch einige Feinabstimmungen vorgenommen, um ein möglichst brauchbares Netz zu generieren.

[1] Die Bibliothek »SNIPE« ist zu finden unter http://www.dkriesel.com/tech/snipe

3. Neuronale Netze

Als Vorbild für die hier verwendeten künstlichen neuronalen Netze (KNN) dienen die Nervenzellen im Gehirn. Diese bestehen aus einer Vielzahl von Neuronen, die untereinander mit Synapsen verknüpft sind. Synapsen verbinden zwei Neuronen gerichtet. Sie können dabei unterschiedlich stark ausgeprägt sein. Eine starke Bindung leitet eine Anregung des ausgehenden Neurons zu großem Teil an das eingehende Neuron weiter. Die Ausprägung kann sich dabei mit der Zeit ändern, was einen Lerneffekt zur Folge hat.

Formale Definition

»Ein Neuronales Netz ist ein sortiertes Tripel (N,V,w) mit zwei Mengen N, V sowie einer Funktion w, wobei N die Menge der Neuronen bezeichnet und V eine Menge $\{(i,j)|i,j \in N\}$ ist, deren Elemente Verbindungen von Neuron i zu Neuron j heißen. Die Funktion $w : V \rightarrow \mathbb{N}$ definiert die Gewichte, wobei w((i, j)), das Gewicht der Verbindung von Neuron i zu Neuron j, kurz mit $w_{i,j}$ bezeichnet wird.« [Kri05]

Die Gewichtsfunktion w ist zu Beginn völlig unbestimmt. Erst durch einen Lernprozess stellen sich hier sinnvolle Werte ein. Hierbei gibt es verschiedene Lernverfahren:

Formen des Lernens

Überwachtes Lernen: Es gibt eine Trainingsmenge, in der die richtigen Ausgabedaten bekannt sind. Die Gewichte werden für alle Eingangsdaten so angepasst, dass der Fehler zu den richtigen Daten minimal wird.

Unüberwachtes Lernen: Es sind keinerlei Ausgabedaten bekannt. Das Netz muss selbstständig zu verschiedenen Eingabedaten sinnvolle Klassifizierungen erkennen.

Bestärkendes Lernen: Es sind keine Ausgabedaten bekannt. Es gibt jedoch Informationen darüber, ob eine vorgenommene Änderung der Gewichte sinnvoll war oder nicht.

Ich werde mich in dieser Arbeit ausschließlich mit überwachtem Lernen beschäftigen. Hierbei wird ein hinreichend großer Trainingsdatensatz benötigt, in dem jeweils zu den Detektordaten auch der richtige Einschlagort des Teilchens bekannt ist.

3.1. Topologie des Netzes

Für den Aufbau von neuronalen Netzen hat man eine Vielzahl an Möglichkeiten. Mit steigender Komplexität bekommt man jedoch schnell das Problem eines geeigneten Algorithmus zum Anpassen der Gewichte. Man kann schon bei reativ kleinen Netzen kaum mehr sagen, in welche Richtung welches Gewicht angepasst werden soll. Daher werde ich mich auf eine grobe Vereinfachung beschränken, auf das Perzeptron.
Ein Perzeptron ist ein künstliches neuronales Netz, in dem die Neuronen in mehreren Schichten (mindestens 2) angeordnet sind. Dabei bildet die erste Schicht die Eingabeschicht und die letzte Schicht die Ausgabeschicht. Jedes Neuron ist mit allen Neuronen der Folgeschicht verknüpft. Da die mittleren Schichten von außen nicht einsehbar sind, werden sie auch »versteckte Schichten« genannt.

3.2. Informationsverarbeitung der Neuronen

Neuronen verarbeiten alle eingegangenen Informationen. Dabei ist zunächst nötig, alle eingehenden Signale zu sammeln. Dies geschieht über eine gewichtete Summe über alle Eingangssignale. Das Ergebnis wird dann mit einer Aktivierungsfunktion weiterverarbeitet.
Als Aktivierungsfunktion werden meist Funktionen mit eindeutigem Schwellwert genommen, z.B. die Heaviside-Funktion, die unterhalb des Schwellwerts den Wert 0 und oberhalb den Wert 1 annimmt. Auf diese Weise sind alle Neuronen in einem Intervall beschränkt. Der Schwellwert wird mithilfe des sogenannten Biaswertes gespeichert. Der Biaswert wird in SNIPE für jedes

Neuron einzeln bestimmt und mittrainiert. In der Praxis werden anstatt der Heaviside-Funktion oft ähnliche Funktionen, wie die Fermifunktion oder der Tangens Hyperbolicus verwendet, da diese auf ihrem gesamten Intervall stetig differenzierbar sind.

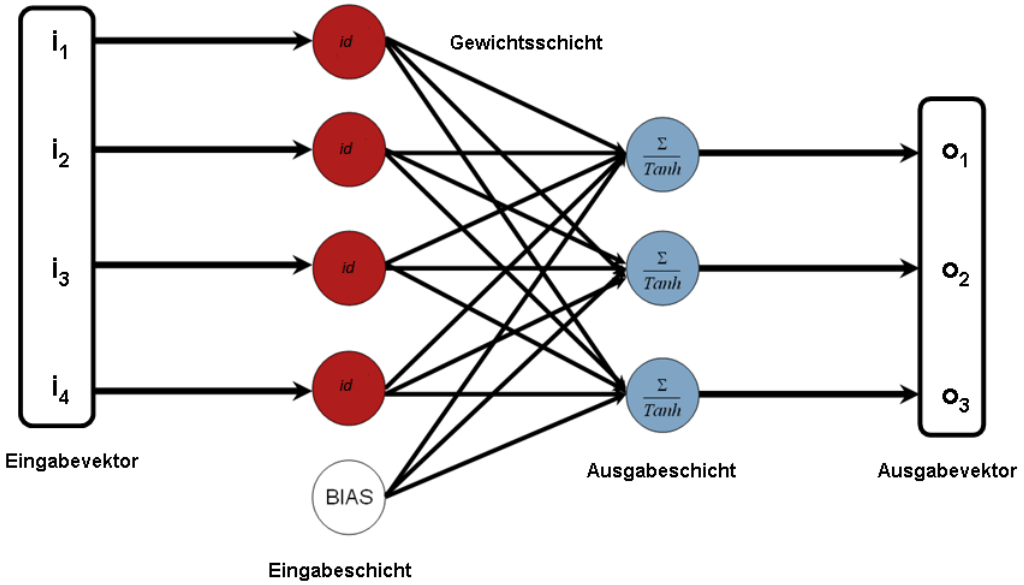

Abbildung 3.1.: Aufbau eines Singlelayerperzeptrons. Der Eingabevektor wird von der Eingabeschicht direkt übernommen. Die Eingabeschicht ist mit allen Neuronen der Ausgabeschicht vernetzt. Dort werden bei jedem Neuron alle Signale gesammelt und mit dem Tangens Hyperbolicus weiterverarbeitet. Die Ausgabeschicht bildet danach direkt den Ausgabevektor. Der Biaswert wird für alle nichttrivialen Neuronen ebenfalls trainiert.

3.3. Bestimmung der Gewichte

Bei einem Single-Layer Perzeptron (SLP) können die Gewichte zunächst gleich (z.B. alle mit 0) initialisiert werden. Das Training erfolgt dann über die Delta-Regel:

$$\Delta\omega_{i,\Omega} = \eta \cdot o_i \cdot (t_\Omega - o_\Omega) = \eta o_i \delta_\Omega$$

Wobei $\Delta\omega_{i,\Omega}$ die Änderung des Gewichts vom Eingangsneuron i zum Outputneuron Ω,
η die Lernrate,
o_i der Output vom Eingangsneuron i,

t_Ω der korrekte Output des Outputneurons Ω (Teaching Output),
und o_Ω der aktuelle Output des Netzes am Neuron Ω ist.

Beim Multi-Layer Perzeptron (MLP) ist dies jedoch komplizierter, da die Neuronen in der versteckten Schicht zunächst einmal keine Interpretation zulassen. Erst durch eine zufällige Initialisierung aller Gewichte erhalten sie eine (abstrakte) Bedeutung. Doch auch dann ist es nicht trivial zu bestimmen, in welche Richtung welches Gewicht bei gegebenen Trainingsdaten geändert werden muss. Um dieses Problem zu lösen, wird die Delta-Regel verallgemeinert. Nach einigen Schritten [Kri05] erhält man dabei folgende Form:

$\Delta\omega_{k,h} = \eta o_k \delta_h$
mit
$\delta_h = f'_{act}(net_h) \cdot (t_h - y_h) \rightarrow$h außen
$\delta_h = f'_{act}(net_h) \cdot \sum_{l \in L}(\delta_h \cdot \omega_{h,l}) \rightarrow$ h innen

Wobei $f_{act}(net_h)$ der Wert der Aktivierungsfunktion von Neuron h bei gegebener Netzeingabe net_h ist.

Diese Regel ist unter dem Namen „Backpropagation of Error“ (Backprop) bekannt. Backprop bearbeitet also zuerst die hinterste Schicht und arbeitet sich dann immer weiter in die vorderen Schichten vor. Dabei werden die Gewichte stets in Richtung der lokal stärksten Minimierung des Fehlers verändert. Die Schrittweise dieses Gradientenverfahrens ist dabei durch die Lernrate η gegeben.

4. Implementierung

4.1. Klassen

SnipeNetwork

SnipeNetwork ist die Hauptklasse und bildet die Schnittstelle zwischen den Daten und SNIPE. Auch die graphische Ausgabe wird durch die Funktion GUI() aufgerufen.

Generator

Die Klasse Generator besitzt Funktionen zum simulieren von Einschlägen im Detektor. Es ist auch möglich, mehrere Einschläge gleichzeitig zu simulieren und so sich überlappende Signale zu erzeugen. Die Funktion *testsession speichert eine Folge von Einschlägen in einem Array, das leicht von SNIPE genutzt werden kann.

Convert

Die Klasse Convert beinhaltet Funktionen zum Umwandeln von Daten:

-Umwandlung zwischen verschiedenen Koordinaten: Mit MCP-Koordinaten wird ein zweidimensionaler Bereich [MCP_Radius, MCP_Radius] beschrieben. Mit NeuronCoords wird ein zweidimensionaler Bereich [0,outputNeurons-Root] beschrieben.

-Umwandlung eines diskreten Neuronenausschlags in eine Gaußverteilung, bei der alle Neuronen verschieden stark angesprochen werden.

Analyze

Die Klasse Analyze beinhaltet Funktionen zum Auswerten von trainierten Netzen. Sie beinhaltet die Berechnung verschiedener Fehlermaße. Auch befinden sich hier Funktionen, mit denen zwischen verschiedenen Einschlägen unterschieden werden kann.

Plot

Die Klasse Plot liefert die Swing-Umgebung, die von der graphischen Ausgabe genutzt wird.

4.2. Parameter

Schwellwert

Zur genaueren Bestimmung der Position liegt es nah, nicht nur das stärkste Neuron zu betrachten, sondern lokal über die stärksten Ausschläge zu mitteln. Die hierbei gewählte Grenze hat starke Auswirkungen auf die Genauigkeit. Eine zu hohe Grenze hat kaum Effekt, während eine zu niedrige Grenze die Vorhersage stets in Richtung der Mitte verschiebt.

Ausgabeneuronen

Die Anzahl der Ausgabeneuronen hat einen direkten Einfluss auf die Genauigkeit, da man bei gleichem Schwellwert bereits einen kleineren Bereich abdeckt. Insbesondere bei Mehrfachhits kann eine zu kleine Ausgabeschicht oft die verschiedenen Hits nicht mehr unterscheiden, da sie sich überlappen. Sie sind jedoch auch der entscheidende Faktor, der die Komplexität des Netzes erhöht. So wächst die In dieser Arbeit werden ausschließlich 144 Ausgabeneuronen benutzt, da die nötige Trainingszeit einer größeren Schicht den zeitlichen Rahmen gesprengt hätte. Da die Trainingszeit jedoch linear mit der Anzahl der Ausgabeneuronen wächst, ist hier für weitere Berechnungen in der Zukunft noch viel Spielraum.

Lernrate

Die Lernrate bestimmt die Veränderung der Gewichte pro Lerndurchlauf. Eine hohe Lernrate führt zu schnellem Lernen. Allerdings können dadurch auch schnell Änderungen rückgängig gemacht und so wieder vergessen werden. Außerdem werden bei einer hohen Lernrate oft Minima übersprungen. Eine niedrige Lernrate verändert das Netz nur minimal. Man erreicht jedoch am Ende eine genauere Einstellung des Netzes. Es hat sich gezeigt, dass eine variable Lernrate, die zu Beginn relativ groß ist und dann abnimmt, die besten Ergebnisse liefert. Da in SNIPE ein Trainingsdurchlauf mit einer festen Lernrate durchgeführt wird, werden mehrere Durchläufe hintereinander ausgeführt und die Lernrate nach jedem Durchlauf mit dem Faktor dLernrate multipliziert.

Aktivierungsfunktion

Als Aktivierungsfunktion hat sich in der Vergangenheit der Tangens Hyperbolicus bewährt. Um Rechenzeit zu sparen, wird in dieser Arbeit eine mitgelieferte Approximation genutzt (TangensHyperbolicusAnguita).[Kri05] In der Eingabeschicht kann es darüber hinaus sinnvoll sein, die Identitätsfunktion zu verwenden, da diese Schicht die einzige Aufgabe hat, den Eingangsvektor an das Netz zu übergeben.

Versteckte Neuronen

Die Anzahl der verstecken Neuronen beim MLP ist nicht durch die Problemstellung gegeben. Da die Arbeitsweise von neuronalen Netzen jedoch kaum vorherzusagen ist, kann man hier a priori nur schwer eine Voraussage treffen. Hier muss eine gute Einstellung experimentell ermittelt werden.

4.3. Generierung der Daten

Neuronale Netze benötigen zum trainieren einen umfangreichen Satz an Daten. In dieser Arbeit werden generierte, optimale Datensätze benutzt. Die Funktion createEvent (MCP_Radius) in der Klasse Generator erzeugt einen zufälligen

Einschlag im Radius des MCP, der dann in den benötigten Eingangsvektor umgewandelt wird.

4.3.1. Eingangsvektor

Die Funktion toInput wandelt den Einschlagsort in das entsprechende MCP-Signal um.

```
1  // Festsetzen des MCP-Signals
2  timeA[6] =   60.0; // MCP signal
3
4  // Berechnen aller Mittelpunkte
5  timeA[0] =   0.5 *x*f + timeA[6];
6  timeA[1] = -0.5 *x*f + timeA[6];
7  timeA[2] =   0.25*(x-y*Math.sqrt(3.))*f + timeA[6];
8  timeA[3] = -0.25*(x-y*Math.sqrt(3.))*f + timeA[6];
9  timeA[4] =   0.25*(x+y*Math.sqrt(3.))*f + timeA[6];
10 timeA[5] = -0.25*(x+y*Math.sqrt(3.))*f + timeA[6];
11
12 for (int ch=0;ch<7;ch++) {
13   for (int i=0;i<120;i++) {
14     double dx = i-timeA[ch];
15     input_layer[i+i*ch] = Math.exp(-dx*dx/(width*2.35));
16   }
17 }
```

timeA[0-5] beschreibt die Maxima der 6 Kanäle der Hexanode. TimeA[6] beschreibt die Mitte des gesamten MCP-Signals. Es wird durch eine Zeitmessung des Einschlags bestimmt. Da hier jedes Ereignis bereits einzeln betrachtet wird, ist dies ein konstanter Wert. Aus den Mittelpunkten werden 7 Kanäle mit einer Samplerate von 120 erstellt. Die Maxima werden dort in eine statistische Verteilung umgewandelt. Da das neuronale Netz genau einen Eingangsvektor benötigt, werden die Kanäle anschließend hintereinander in einem Array gespeichert. Der Eingangsvektor besteht daher stets aus 840 Einträgen.

Man kann der Funktion toInput zusätzlich einen bereits erstellten Eingangsvektor geben. Anschließend werden die Daten auf den alten Vektor dazu addiert, wodurch ein überlagertes Signal entsteht.

4.3.2. Ausgangsvektor

Die Wahl des Ausgangsvektors ist durch die Problemstellung nicht so festgeschrieben wie der Eingangsvektor. Die Bedingung ist nur, dass man aus dem Ausgangsvektor einfach auf möglichst genaue Vorhersagen für den tatsächlichen Einschlagsort kommt.

Koordinatenform

Die einfachste Möglichkeit ist es, die x- und y-Koordinate direkt als Ausgangsvektor mit zwei Einträgen zu verwenden. Dies würde allerdings voraussetzen, dass es wirklich nur einen Einschlag gab. Da das aber nicht gegeben ist, wird diese Darstellung in dieser Arbeit nicht weiter untersucht.

NeuronCoords

Hier werden n^2 Neuronen benutzt, die ein Quadrat mit Kantenlänge 2*MCP_Radius bilden. Jedes Neuron deckt also einen Bereich von $(2 * MCP\_Radius/n)^2$ ab. Um dies für die weitere Bearbeitung etwas übersichtlicher zu gestalten, wird vorher eine einfache Koordinatentransformation durchgeführt:

```
1 neuronCoord[0]=(MCP_position [0] + MCP_radius)*(outputNeuronsRoot)/(2*
      MCP_radius);
2 neuronCoord[1]=(MCP_position [1] + MCP_radius)*(outputNeuronsRoot)/(2*
      MCP_radius);
```

Die Neuronen haben nun unabhängig von MCP_Radius und outputNeuronRoot die Kantenlänge 1. Die Einschlagkoordinaten sind nun im Bereich [0,outputNeuronRoot].

Alle Vorgänge, die auf Ebene der Neuronen stattfinden, werden in NeuronCoords betrachten. Zur Interpretation und Analyse wird dann wieder auf MCPCoords transformiert.

Binäres Neuron

Das Neuron, in welchem der Einschlag war, wird nun mit 1 gespeichert. Alle anderen Neuronen haben den Wert 0.

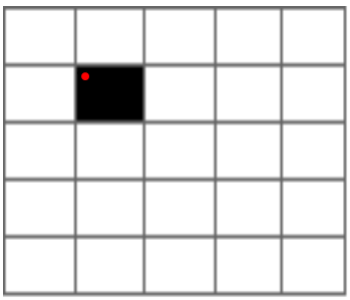

Abbildung 4.1.: Binäres Neuron

Auf diese Weise geht jedoch ein großer Teil der Ortsinformation verloren. Dies hat für die Auswertung jedoch keinen so entscheidenden Einfluss wie es zunächst scheint, da das Netz zum trainieren eine Vielzahl von Trainings durchläuft. Es sollte sich also eine Verteilung einstellen, über die man mitteln kann. Dadurch kann man in der Auswertung einiges an Genauigkeit zurückgewinnen, obwohl man mit niedriger Genauigkeit trainiert hat.

Gaußverteilung

Da man am Ende über eine Verteilung mittelt, stellt sich die Frage, ob man an Genauigkeit gewinnen kann, wenn man nicht mit einem diskreten Neuron, sondern bereits mit einer Verteilung trainiert. Aus diesem Grund wird Ort mit einer Gaußglocke umhüllt. Jedes Neuron hat den Wert dieser Verteilung an seinem Mittelpunkt.

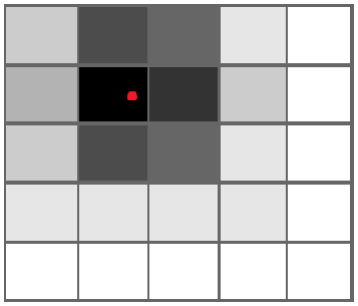

Abbildung 4.2.: Gaußglocke

Ziel ist es nun, den Trainigsfehler zu minimieren. Die zweidimensionale Funktion GaußTest, gibt den bei der Rücktransformation entstandenen Fehler in

Abhängigkeit des Schwellwerts *threshold* und der Varianz *var* aus. Abbildung 3.3 zeigt einen Plot dieser Funktion mit 144 Ausgabeneuronen.

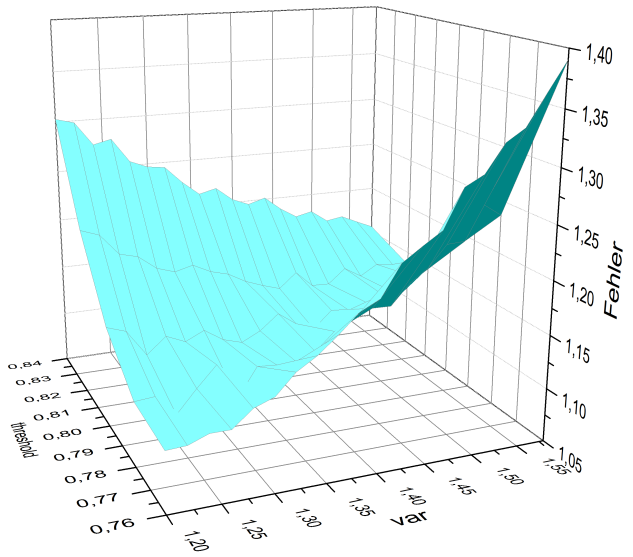

Abbildung 4.3.: 3D-Plot des Transformationsfehlers

Der Schwellwert bezeichnet den Anteil vom Maximum, bis zu dem am Ende gemittelt wird. Da der Ausgabevektor begrenzt ist, muss der Schwellwert relativ hoch gewählt sein. Ansonsten wird das Signal stets in Richtung der Mitte verschoben. Der Plot ist sehr lokal, da aus den oben genannten Gründen nur in der Nähe von threshold = 0.8 nach einem Minimum gesucht habe. Die gewählten Werte für Training Samples sind:
$threshold = 0.8$
$var = 1.4$

Der mittlere Fehler beträgt hierbei 1,12.

Benutzte Datensätze

Zum Training des Netzes wurden insgesamt 4 verschiedene Datensätze mit jeweils 60000 Daten generiert und diese für alle Trainings benutzt.

I. Binäres Neuron / Single Hit
II. Gaußverteilung / Single Hit
III. Binäres Neuron / Double Hit

IV. Gaußverteilung / Double Hit

Die Datensätze mit Double Hits bestehen zu 40% aus zwei überlagerten Ereignissen und 60% einfachen Ereignissen.

4.4. Training

Zum Trainieren wird die Funktion Perceptron() in der Klasse SnipeNetwork benutzt. Die Parameter werden entweder in der Funktion selbst bestimmt oder übergeben. Parameter, die kaum geändert wurden, müssen in der Funktion geändert werden.

```
1 NeuralNetworkDescriptor desc = new NeuralNetworkDescriptor(inputNeurons,
      hiddenNeurons,outputNeurons);
2 desc.setSettingsTopologyFeedForward();
```

Im ersten Schritt wird die Topologie des Netzes festgelegt. Es werden alle nötigen Neuronen erstellt, sowie Synapsen von allen Neuronen einer Ebene zu allen Neuronen der nächsten Ebene.

```
1 desc.setNeuronBehaviorOutputNeurons(new TangensHyperbolicusAnguita());
2 desc.setNeuronBehaviorHiddenNeurons(new TangensHyperbolicusAnguita());
3 desc.setNeuronBehaviorInputNeurons(new TangensHyperbolicusAnguita());
```

Danach wird das Verhalten aller Neuronen geregelt. Die Funktion TangensHyperbolicusAnguita beschreibt .....

```
1 NeuralNetwork net = new NeuralNetwork(desc);
```

Da die Topologie nun feststeht, kann das NeuralNetwork net erstellt werden, auf dem alle weiteren Operationen laufen.

```
1 File in = new File(File\_input);
2 File out = new File(File\_output);
3
4 double [][] inputs = SnipeNetwork.readFile(in,runs,inputNeurons);
5 double [][] Desiredoutputs = SnipeNetwork.readFile(out,runs,outputNeurons);
6
7 TrainingSampleLesson lesson=new TrainingSampleLesson(inputs,Desiredoutputs);
```

Hier wird die TrainingSampleLesson lesson erstellt. Dabei sind die im letzten Kapitel beschriebenen Datensätze die Grundlage (Dateien in und out).

```
for (int i=0; i<trainruns/runs ; i++){
    net.trainBackpropagationOfError(lesson, runs, learnrate);
    lesson.shuffleSamples();
    learnrate= learnrate * dlearnrate;
}
// Speichere Netz in saveNet*.txt
Convert.saveString(net.exportToString(), saveNet);
```

Anschließend wird lesson benutzt, um das Netz mithilfe von Backpropagation of Error zu trainieren. Dabei wird jeweils nach 60000 Durchläufen die Lernrate angepasst und die Reihenfolge der einzelnen Durchläufe zufällig permutiert. Zum Schluss wird das trainierte Netz in Textform gespeichert.

4.5. Auswertung

Zur Bewertung der einzelnen Netze habe ich verschiedene Fehlermaße definiert, um mir ein Bild von der Güte und Varianz des Netzes zu machen. Diese Fehlermaße werden sowohl auf ein einfaches Ereignis, als auch auf ein doppeltes angewandt. Zur Erkennung der Peaks habe ich den Ort über die stärksten Ausschläge gemittelt. Alle Fehlermaße sind somit nicht direkt aus dem Netz auslesbar, sondern werden durch von mir implementierte Algorithmen berechnet. Diese Berechnung kann mit anderen Algorithmen mit Sicherheit noch optimiert werden.

4.5.1. Fehlermaße

Mittlerer Fehler

$\bar{\mathrm{d}} = \frac{1}{n}\sum_{k=1}^{n}\sqrt{(x_k - E(x_k)^2) + (y_k - E(y_k)^2)} = \frac{1}{n}\sum_{k=1}^{n} d_k$
Der mittlere Fehler $\bar{\mathrm{d}}$ gibt den durchschnittlichen Abstand zwischen korrektem und vorhergesagtem Einschlag an(in MCP-Koordinaten). Es wird dabei über die radialen Abstände d_k summiert und durch die Anzahl der Testläufe n geteilt. Der mittlere Fehler ist das wichtigste Merkmal für die Güte des Netzes.

Standardabweichung des Fehlers

$\sigma \approx \sqrt{\frac{1}{n}\sum_{k=1}^{n} d_k^2}$ Die Standardabweichung des Fehlers ist ein Indikator für die Verlässlichkeit des Netzes. Eine große Standardabweichung spricht dafür, dass die vom Netz vorhergesagten Einschlagsorte häufig sehr weit von den tatsächlichen Einschlagsorten entfernt waren.

richtig klassifizierte Events

Die Anzahl der Hits kann man aus dem MCP leicht auslesen. Oft erkennt das neuronale Netz jedoch eine andere Anzahl von Hits. Diese Auswertungen sind dann nicht ohne weitere Überarbeitung zu gebrauchen. Die relative Anzahl der richtig klassifizierten Events r ist zunächst durch die Auflösung der Ausgabeschicht begrenzt. Jedoch können auch Interferenzeffekte zu falschen Ergebnissen führen.

erkannte Hits

Als erkannte Hits zählen alle Hits, die innerhalb eines festen Fehlers f liegen. Da das hier betrachtete neuronale Netz lediglich die Aufgabe hat, den ungefähren Einschlagsort zu bestimmen, wurden hier 5mm und 8mm als Fehlergrenzen festgelegt. Dieses Fehlermaß wird nur auf Doublehits angewandt, da nur hier diese Information nötig ist.

4.5.2. Graphische Ausgabe

Die graphische Ausgabe des Netzes für einzelne Eingaben ist ein sehr hilfreiches Werkzeug, um Schwachstellen des Netzes zu erkennen und zu analysieren. Starke Ausschläge sind in der Ausgabe dunkler als schwache Ausschläge, wobei der stärkste Ausschlag stets komplett schwarz ist und der Rest entsprechend linear angepasst ist. Die korrekten Einschläge sind durch rote Punkte gekennzeichnet, alle vom Netz erkannten Einschläge mit gelben Punkten.

4.5.3. Double Hit

Die Auswertung von mehreren gleichzeitigen Einschlägen hat sich als kompliziert herausgestellt, da verschiedene Treffer richtig erkannt werden müssen. Diese Arbeit beschränkt sich auf eine sehr lokale Mittelung über die Maxima. Es wird im ersten Schritt das Maximum gesucht. Danach wird über alle umliegenden Neuronen, deren Ausschlag größer als *threshold* ist, gemittelt. Danach wird nach weiteren signifikanten ($> threshold$) Maxima gesucht etc. In der graphischen Ausgabe wird unter anderem die Anzahl der erkannten Einschläge angezeigt ($countedHits$).

Als Fehlermaße bieten sich auch hier der mittlere Fehler $\bar{d}$ und die Standardabweichung σ an. Dabei werden nur die Fälle betrachtet, in denen die Anzahl der Hits korrekt bestimmt wurde. Dies ist akzeptabel, weil man die Anzahl der Einschläge direkt aus dem MCP auslesen kann und somit einen Vergleichswert hat. Zum Bestimmen der Fehlermaße liegt der Mindestabstand zweier Einschläge bei 5mm.

5. Auswertung des Netzes

In dieser Arbet wurden ca. 40 Einstellungen der Netze getestet. Da jedoch viele Einstellungen keine brauchbaren, oder ähnliche Ergebnisse brachten, werden nur die besonders guten und die besonders interessanten Netze beschrieben. Eine Auflistung mit allen Ergebnissen befindet sich im Anhang. Als allgemein beste Standardeinstellung haben sich 10 Durchläufe mit je 60.000 Daten erwiesen. Die Lernrate beginnt bei 0,07 und wird nach jedem Durchlauf mit dem Faktor 0,7 multipliziert. Als Aktivierungsfunktion in der Eingabeschicht wird die Identitätsfunktion benutzt. Versteckte- und Ausgabeschicht werden durch den Tangens Hyperbolicus aktiviert.

5.1. SLP

Im ersten Schritt soll ein lauffähiges SLP trainiert werden..

Binäres Neuron

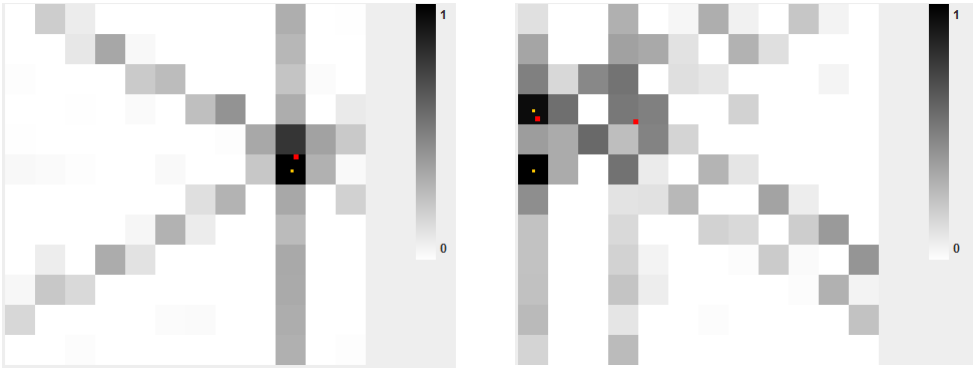

Abbildung 5.1.: Singlelayerperzeptron mit binärem Output. Bei einem einfachen Einschlag ist die Detektorstruktur deutlich erkennbar (links). Bei mehreren Einschlägen führt das jedoch zu Interferenzen (rechts).

$\bar{d}_{1Hit}$	$\bar{d}_{2Hit}$	σ_{1Hit}	σ_{2Hit}	r_{1Hit}	r_{2Hit}	$in5mm$	$in8mm$
2,82	6,84	3,86	8,77	0,9	0,36	0,81	0,86

Trainiert wurde mit Datensatz I und Standardeinstellungen. Man kann beim SLP ganz deutlich die Struktur des Detektors erkennen. Dies erschwert jedoch auch die Erkennung von mehreren Einschlägen, da hier mehrere Überlagerungen zu erwarten sind. Dies kann man an der sehr geringen Anzahl erkannter Doublehits sehr deutlich erkennen.

Gaußverteilung

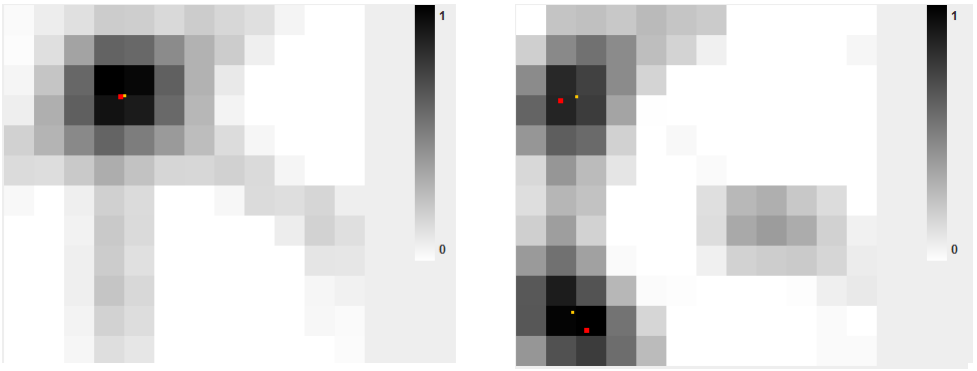

Abbildung 5.2.: Singlelayerperzeptron mit Gaußoutput. Die Detektorstruktur bei einfachen Hits hat im Vergleich zur Gaußglocke etwas weniger Gewicht (links). Die Auflösung von mehreren Einschlägen wird dadurch leicht verbessert (rechts).

$\bar{d}_{1Hit}$	$\bar{d}_{2Hit}$	σ_{1Hit}	σ_{2Hit}	r_{1Hit}	r_{2Hit}	$in5mm$	$in8mm$
2,1	3,29	2,54	3,48	0,99	0,63	0,84	0,98

Trainiert wurde mit Datensatz II und Standardeinstellungen. Auch hier ist die Struktur des Detektors noch zu erkennen. Der Einschlagsort hat jedoch hier etwas mehr Gewicht. Das ist mit der in der Verteilung enthaltenen Metrik erklärbar, die Ausschläge mit großer Entfernung zum Einschlagsort stärker bestraft. Der mittlere Fehler und die Standardabweichung sind somit hier etwas geringer.

5.2. MLP

Im zweiten Schritt soll ein etwas aufwendigeres MLP trainiert werden, um die Fehler weiiter zu minimieren.

5.2.1. Training mit Single Hits

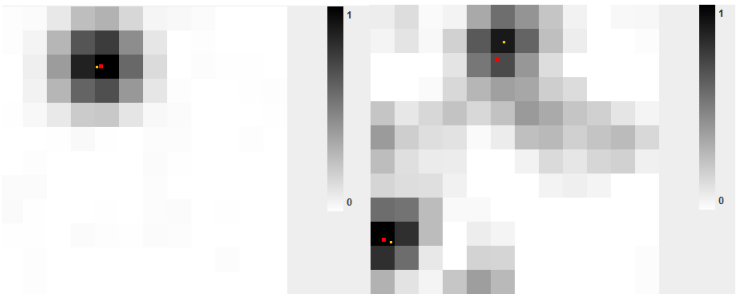

Abbildung 5.3.: Multilayerperzeptron mit Gaußoutput und Singlehit-Training. Einfache Hits (links) sind klar und ohne Störungen erkennbar. Bei doppelten Hits (rechts) treten Störeffekte auf.

$\bar{d}_{1Hit}$	$\bar{d}_{2Hit}$	σ_{1Hit}	σ_{2Hit}	r_{1Hit}	r_{2Hit}	$in5mm$	$in8mm$
1,22	2,64	1,41	2,57	1	0,67	0,92	0,99

Abb. 5.3 zeigt ein MCP mit Standardeinstellungen. Trainiert wurde nur mit Datensatz II. Single Hits sind hier sehr deutlich erkennbar. Auch zwei gleichzeitige Einschläge können gut bestimmt werden. Allerdings sind hier noch immer Neuronen angesprochen, die weit von beiden Einschlägen entfernt sind. Damit ist auch das relativ schlechte Verhältnis erkannter Doublehits zu erklären.

5.2.2. Training mit Double Hits

Das Training mit Double Hits brachte die entscheidende Verbesserung zum Erkennen von Mehrfachhits.

$\bar{d}_{1Hit}$	$\bar{d}_{2Hit}$	σ_{1Hit}	σ_{2Hit}	r_{1Hit}	r_{2Hit}	$in5mm$	$in8mm$
1,39	2,21	1,57	2,41	1	0,85	0,96	0,996

Auch hier wurde mit Standardeinstellungen trainiert. Diesmal wurde mit Datensatz IV trainiert. Mittlerer Fehler und Standartabweichung sind vergleichbar mit dem Ergebniss von Datensatz II; tendenziell sogar etwas besser. Die Anzahl der erkannten Hits ist jedoch eindeutig besser. Zwei Einschläge, die

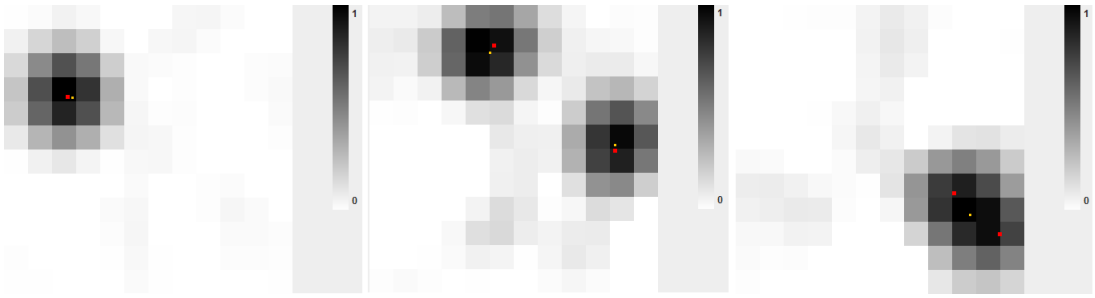

Abbildung 5.4.: Multilayerperzeptron mit Gaußoutput. Einfache (links) und doppelte (mitte) Hits werden relativ gut erkannt. Sind zwei Einschläge jedoch zu nah, so reicht die Auflösung der Ausgabeschicht nicht aus(rechts).

sehr nah beieinander liegen, werden jedoch fast immer falsch erkannt. Dies könnte man möglicherweise nur durch eine größere Ausgabeschicht verbessern. Bemerkenswert ist auch die sehr geringe Standardabweichung bei zwei Hits, wodurch das Netz vergleichsweise verlässlich wird. Strebt man eine Genauigkeit von +-8mm an, was zur Weiterverarbeitung mit dem aktuellen Algoritmus bereits hilfreich wäre, so liegen nur 4 Promille aller Vorhersagen falsch.

Zum Vergleich wurde ein Netz mit gleichen Einstellungen und Datensatz III trainiert.

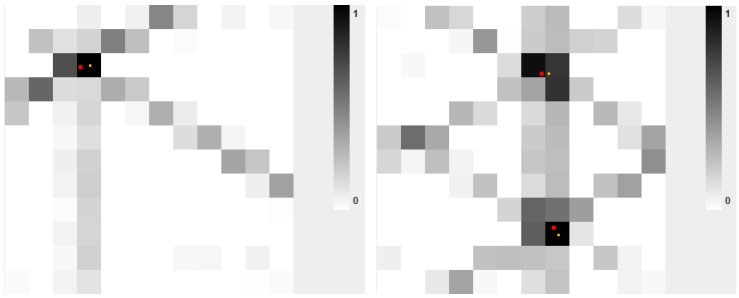

Abbildung 5.5.: Multilayerperzeptron mit binärem Output. Die Detektorstruktur ist hier wieder deutlich zu erkennen, was zu einer schlechteren Auflösung führt. links: ein Hit; rechts: zwei Hits

$\bar{d}_{1Hit}$	$\bar{d}_{2Hit}$	σ_{1Hit}	σ_{2Hit}	r_{1Hit}	r_{2Hit}	$in5mm$	$in8mm$
2,23	4,47	2,67	6,34	0,97	0,39	0,91	0,94

Einfache Hits werden im Vergleich zum vorherigen Netz deutlich schlechter erkannt. Insbesondere der mittlere Fehler bei Mehrfachhits ist hier deutlich größer. Außerdem werden auch nur 38 % überhaupt korrekt als Zweifachhits klassifiziert. Das Training mit Gaußoutput liefert in allen Bereichen deutlich bessere Ergebnisse.

6. Fazit und Ausblick

Ziel dieser Bachelorarbeit war es, zu testen, ob sich künstliche neuronale Netze zur Auswertung von Hex-Anoden eignen. Dabei gab es beim bisherigen Algorithmus insbesondere mit Mehrfachhits Probleme mit der Zuordnung der einzelnen Signale. Das in Abschnitt 5.2.2 vorgestellte Multilayerperzeptron kann diese Aufgabe relativ zuverlässig lösen. Die Anzahl der Einschläge wird hier häufig korrekt erkannt und die vorhergesagten Orte sind tendenziell richtig. Insbesondere kann man beim Vergleich von binärem Output und Gaußoutput feststellen, dass die Interferenzen durch die Detektorstruktur beim Training mit Datensatz IV (Gauß) deutlich abgeschwächt sind, was eine signifikante Verbesserung der Vorhersage zur Folge hat. Sinnvoll einsetzbar sind die trainierten Netze jedoch noch nicht. Aktuell werden von den hier trainierten Netzen nur exakt gleichzeitige Ereignisse betrachtet. Das MCP-Signal besteht stets aus einem Peak bei 60 ms (also in der Mitte des gesamten Signals). Eine zeitlich Verschiebung hätte ebenfalls eine Verschiebung aller anderen Signale zur Folge. Der nächste Schritt für eine sinnvolle Anwendung ist daher, ein Netz mit nur fast gleichzeitigen Ereignissen zu trainieren. Außerdem können ohne großen Aufwand Optimierungen vorgenommen werden, um den Fehler weiter zu minimieren. Vor allem durch eine Vergrößerung der Ausgabeschicht könnte mehr Genauigkeit erreicht werden. Dies hätte jedoch eine merkbare Verlängerung der Trainingsphase zur Folge, was mit entsprechenden Rechnern und genügend Zeit jedoch leicht möglich wäre.
Außerdem hat sich der hier implementierte Algorithmus zur Erkennung von verschiedenen Hits als noch nicht ausgereift erwiesen. Man könnte hier zum Beispiel noch die Information über die Anzahl der tatsächlichen Einschläge einfließen lassen.
Die hier verwendete SNIPE-Bibliothek hat zudem Möglichkeiten, auch komplexere Netze zu konstruieren. Hier ist noch viel Spielraum für weitere Versuche. Bisher wurden Signale über einen Zeitraum von 120 ms aufgenommen und erst danach ausgewertet. Man könnte hier zum Beispiel ein zeitabhängiges

Netz konstruieren, das mit nur 7 Eingabeneuronen die Detektorsignale direkt empfängt und sofort weiterverarbeitet.

Literaturverzeichnis

[Ach99] Achler, M.: *Untersuchung von Symmetrieeffekten in der Photodoppelionisation von Helium mit zirkular polarisiertem Licht*, Goethe-Universitaet Frankfurt am Main, Diss., 1999 2.1, 6

[Kri05] Kriesel, David: *Ein kleiner Überblick über Neuronale Netze.* 2005 3, 3.3, 4.2

[Mec06] Meckel, M.: *Strong-Field Ionization of Aligned Oxygen*, Goethe-Universitaet Frankfurt am Main, Diplomarbeit, 2006 2.2, 6

Abbildungsverzeichnis

A. Trainierte Netze

Es folgt eine Übersicht über alle im Rahmen dieser Arbeit trainierten Netze. Der Name der jeweiligen Netze entspricht dem Namen der Textdatei, in der das entesprechende Netz gespeichert ist.

A. Trainierte Netze

Tabelle A.1.:

Name	Aktivierung	Hidden	Daten	Runs	Rate	d Rate	Gauss	2Hit Training	Fehler
saveNet150Hidden1M	TangensH	150	60000	1000000	0,001	1	Nein	Nein	viel
saveNet150Hidden300k	TangensH	150	60000	300000	0,001	1	Nein	Nein	36
saveNet300k	TangensH	0	60000	300000	0,001	1	Nein	Nein	5,6
saveNet300k_01	TangensH	0	60000	300000	0,001	1	Nein	Nein	9,2
saveNet300k_02	TangensH	0	60000	300000	0,002	1	Nein	Nein	6,1
saveNetg60k_02	TangensH	0	60000	60000	0,002	1	Ja	Nein	10,56
saveNetg60k_08	TangensH	0	60000	60000	0,008	1	Ja	Nein	7,32
saveNetg60k_08-	TangensH	0	60000	60000	0,008	0,6	Ja	Nein	5,29
saveNetg100k_08	TangensH	0	60000	100000	0,008	1	Ja	Nein	5,2
saveNetg150Hid300k_20-	TangensH	150	60000	300000	0,020	0,6	Ja	Nein	4,04
saveNetg150Hid300k_30-	TangensH	150	60000	300000	0,030	0,6	Ja	Nein	2,44
saveNetg150Hid480k_60-	TangensH	150	60000	480000	0,060	0,6	Ja	Nein	1,5
saveNetg180k_10	TangensH	0	60000	180000	0,010	1	Ja	Nein	1,88
saveNetg300k_02	TangensH	0	60000	300000	0,002	1	Ja	Nein	5,09
saveNetg300k_04	TangensH	0	60000	300000	0,004	1	Ja	Nein	4,3
saveNetg1080k_10–	TangensH	0	60000	1080000	0,010	0,6	Ja	Nein	30
saveDoubNetg150Hid480k_60-	TangensH	150	60000	480000	0,060	0,6	Ja	Ja	1,43
saveDoubNetg250Hid480k_60-	TangensH	250	60000	480000	0,060	0,7	Ja	Ja	1,5
saveDoubNetg150Hid600k_60-	TangensH	150	60000	600000	0,060	0,7	Ja	Ja	1,44
saveDoubNetg50Hid480k_60-	TangensH	50	60000	480000	0,060	0,7	Ja	Ja	1,73
saveDoubNetg250Hid600k_60-	TangensH	250	60000	600000	0,060	0,7	Ja	Ja	1,51
saveDoubNetg150Hid600k_60-8	TangensH	150	60000	600000	0,060	0,8	Ja	Ja	1,37
saveDoubNetg150Hid180k_50-7	TangensH	150	60000	180000	0,050	0,7	Ja	Ja	1,75
saveDoubNetg150Hid900k_100-8	TangensH	150	60000	900000	0,100	0,8	Ja	Ja	1,35
saveDoubNet150Hid480k_60-	TangensH	150	60000	480000	0,060	0,6	Nein	Ja	2,1
saveDoubNetg130Hid480k_60-	TangensH	130	60000	480000	0,060	0,7	Ja	Ja	1,5
saveDoubNetg170Hid480k_60-	TangensH	170	60000	480000	0,060	0,7	Ja	Ja	1,57
save2DoubNetg150Hid480k_60-	TangensH	150	60000	480000	0,060	0,7	Ja	Ja	1,55
save3DoubNetg150Hid480k_60-6	TangensH	150	60000	480000	0,060	0,6	Ja	Ja	1,58
saveDoubNetg150Hid600k_60-6	TangensH	150	60000	600000	0,060	0,6	Ja	Ja	1,54
saveDoubNetg150Hid600k_70-7	TangensH	150	60000	600000	0,070	0,7	Ja	Ja	1,41
SaveDoubNetg150Hid600k_80-7	TangensH	150	60000	600000	0,080	0,7	Ja	Ja	2,1
SaveDoubNetg150Hid600k_600-7	TangensH	150	60000	600000	0,600	0,7	Ja	Ja	1,88
saveDoubNetg150Hid600k_60-7	TangensH	150	60000	600000	0,060	0,7	Ja	Ja	1,42

Tabelle A.2.:

Name	Aktivierung	Hidden	Daten	Runs	Rate	d Rate	Gauss	2Hit Training	Fehler
Save2DoubNetg150Hid600k_70-7	Id-TanH-TanH	150	60000	600000	0,070	0,7	Ja	Ja	1,39
SaveSLPNet300k_70-7	Id-TanH	0	60000	300000	0,070	0,7	Nein	Nein	7,2
SaveSLPNet600k_70-7	Id-TanH	0	60000	600000	0,070	0,7	Nein	Nein	2,82
Save3DoubNetg150Hid600k_70-7	Id-TanH-lin(-1)	150	60000	600000	0,070	0,7	Ja	Ja	NaN
SaveSLPDoubNet600k_70-7	Id-TanH	0	60000	600000	0,070	0,7	Nein	Ja	3,32
SaveSLPNetg600k_70-7	Id-TanH	0	60000	600000	0,070	0,7	Ja	Nein	2,1
SaveDoubNet150Hid600k_70-7	Id-TanH-TanH	150	60000	600000	0,070	0,7	Nein	Ja	2,23
SaveNetg150Hid600k_70-7	Id-TanH-TanH	150	60000	600000	0,070	0,7	Ja	Nein	1,22

Tabelle A.3.:

Name	Fehler(2Hit)	Sigma	Sigma(2Hit)	proper	proper(2Hih)	in5mm(2Hit)	in8mm(2Hit)
Save2DoubNetg150Hid600k_70-7	2,21	1,57	2,41	1	0,85	0,96	1
SaveSLPNet600k_70-7	6,84	3,86	8,77	0,9	0,36	0,81	0,86
SaveSLPNetg600k_70-7	3,29	2,54	3,48	0,99	0,63	0,84	0,98
SaveDoubNet150Hid600k_70-7	4,47	2,67	6,34	0,97	0,39	0,91	0,94
SaveNetg150Hid600k_70-7	2,64	1,41	2,57	1	0,67	0,92	0,99