ECAP-2010-031

Diplomarbeit

Ereignisklassifikation mit Hilfe von Neuronalen Netzen für das ANTARES Neutrino-Teleskop

Klaus Geyer

März 2010

Physikalisches Institut der Friedrich-Alexander-Universität Erlangen-Nürnberg

Einleitung

Auf der Suche nach einem tieferen Verständnis für das Universum in dem wir leben, haben die Menschen ihren Blick schon lange auf den Himmel gerichtet. Mit Hilfe von optischen Teleskopen konnte J. Kepler im 16. Jahrhundert die Planetenbewegungen unseres Sonnensystems bestimmen. Die Ergebnisse seiner Arbeit sind als die Keplerschen Gesetze bekannt.

Zu Beginn der Astronomie wurde der Himmel mit bloßen Augen nach astrophysikalischen Objekten abgesucht. Mit der Zeit verfeinerten sich die Apparaturen, an die Stelle des menschlichen Auges traten elektronische Halbleiter und die Teleskope wurden größer und leistungsfähiger. Neben dem für Menschen sichtbaren Licht, wird ein weites Frequenzspektrum vom Röntgenbereich über Infrarot bis hin zur Radioastronomie abgedeckt. Diese Teleskope beobachten einen Teil der elektromagnetischen Strahlung, die von astrophysikalischen Objekten ausgesendet wird und die Erde erreicht. Die unbegrenzte Lebensdauer der Photonen machen diese zu einem geeigneten Informationsträger über intergalaktische Entfernungen. Bereits das menschliche Auge kann die Existenz von Sternen bestätigen. Da die Photonen an der elektromagnetischen Wechselwirkung teilnehmen, können sie leicht von Materie, wie interstellaren Staubwolken, abgeschirmt werden. Je nach Frequenzspektrum sind die Informationen dahinter liegender Objekte verloren.

Neben Photonen erreicht auch kosmische Strahlung die Erde. Sie besteht zum größten Teil aus Protonen, die z.B. durch Supernovaexplosionen beschleunigt werden. Neben Elektronen und Atomkernen, gehören auch Neutrinos zur kosmischen Strahlung. Die Astroteilchenphysik befasst sich mit dem Nachweis und der Analyse der kosmischen Strahlung, die die Erde trifft. Um Informationen von intergalaktischen Objekten übertragen zu können, sind gewisse Eigenschaften, wie eine lange Lebensdauer, erforderlich. Das Neutrino bietet sich an, da es darüberhinaus nur durch die schwache Wechselwirkung mit anderen Teilchen wechselwirkt. Anders als das Proton oder Elektron, die aufgrund ihrer Ladung elektromagnetisch wechselwirken, wird das Neutrino nicht von Magnetfeldern abgelenkt. Das Neutrino bringt alle Eigenschaften mit, die es als Informationsträger qualifizieren, bisher verborgene Objekte zu entdecken und sogar in astrophysikalische Objekte hinein zu blicken. Ihr hohes Druchdringungsvermögen mag eine wichtige Eigenschaft als Informationsträger sein, doch ist es dadurch auch sehr

schwer Neutrinos nachzuweisen.

Neutrino-Teleskope wie der ANTARES-Detektor im Mittelmeer, weisen Neutrinos indirekt über Cherenkov-Licht nach. Aus Myonneutrinos entstehen in einer charged-current-Reaktion geladene Myonen. Bei hinreichend hoher Geschwindigkeit emittieren diese Cherenkov-Licht, welches von den Neutrino-Teleskopen registriert wird. Da die Teilchen der kosmischen Strahlung mit der Atmosphäre der Erde wechselwirken, entsteht eine Vielzahl weiterer Teilchen, die ebenfalls genug Energie besitzen um ein Signal im Detektor zu erzeugen. Aus diesem Grund blicken Neutrino-Teleskope, wie ANTARES nicht in die Richtung des Himmels, sondern durch die Erde und nutzen diese als Abschirmung gegen die atmosphärische Strahlung.

Die Unterscheidung zwischen Teilchen, die sich von oben, bzw. von unten durch den Detektor bewegen, ist ein notwendiger Schritt um Neutrinos nachzuweisen. Bisherige Strategien versuchen, jedes Ereignis zu rekonstruieren und anhand der rekonstruierten Richtung zu entscheiden, um welchen Ereignistyp es sich handelt. Dabei kommt es häufig zu Fehlrekonstruktionen.

Diese Arbeit untersucht, ob sich der atmosphärische Untergrund bereits vor einer Rekonstruktion durch ein künstliches Neuronales Netz heraus filtern lässt. Es werden Parameter gesucht, die für diese Aufgabenstellung geeignet erscheinen. Anschließend werden verschiedene Auswahlverfahren auf die Parameter angewendet, um die geeignetsten herauszufiltern. Unterschiedliche Neuronale Netze wurden trainiert und getestet, um eine geeignete Netztopologie zu finden. Verschiedene Aktivierungsfunktionen der Netze und Skalierungen der Daten wurden getestet um, die Untergrundunterdrückung weiter zu steigern.

Inhaltsverzeichnis

Kapitel 1

Das ANTARES-Experiment

Der ANTARES-Detektor ist ein Neutrino-Detektor im Mittelmeer. Er befindet sich in einer Tiefe von 2400m unter Wasser. Diese Wasserschicht über dem Detektor soll einen Großteil der Teilchen aus Luftschauern abschirmen, die durch kosmische Strahlung erzeugt werden. Das allein ist aber leider nicht ausreichend, da eine große Anzahl von atmosphärischen Myonen immer noch den Detektor erreicht. Der gesamte Detektor ist daher nicht zur Wasseroberfläche, sondern auf den Meeresboden ausgerichtet. Er nutzt also die gesamte Erde als Schild gegen die atmosphärische Strahlung.

1.1 Physikalische Zielsetzung

1.1.1 Kosmische Neutrinos

Die Detektion kosmischer Neutrinos ist ein wesentliches Ziel des ANTARES-Experiments. Von großem Interesse sind dabei Neutrino-Punktquellen, da die Messung ihres Flusses und Energiespektrums Aufschluss über astrophysikalische Prozesse in der Quelle geben kann.

Mögliche Neutrino-Punktquellen sind aktive galaktische Kerne. Im Zentrum aktiver galaktischer Kerne befindet sich ein schwarzes Loch, das Materie aus der umgebenden Galaxie anzieht. Dabei entsteht eine Akkretionsscheibe. Zusätzlich können sich, ausgehend von dem schwarzen Loch, Jets bilden, die in der Lage sind Teilchen zu beschleunigen.

Auch Supernovae gelten als Quellen hochenergetischer Teilchen und Neutrinos. Sie leiten die letzte Phase eines Sterns ein, nachdem der Brennstoff in seinem Inneren verbraucht ist. Dabei werden Teilchen mit einer Gesamtmasse, die das Mehrfache der Masse unserer Sonne entspricht, in den Weltraum abgestoßen. Je nach der Größe der Masse des ursprünglichen Sterns, entstehen anschließend z.B. Neutronensterne oder

schwarze Löcher. Die Supernova 1987A, in der Großen Magellanschen Wolke und einer Entfernung von etwa 51 kpc[1] zur Erde, erhöhte die gemessene Anzahl der Neutrinos auf der Erde um 24 zusätzliche Neutrinos (relativ zum Untergrund) innerhalb von 10 Sekunden [1]. Diese Neutrinos können von ANTARES jedoch nicht gesehen werden, da sie sich in einem zu niedrigen Energiebereich (MeV) befinden. Neutrinos höherer Energie (TeV) werden von Supernovae erzeugt, wenn deren Schockfront in das Interstellare Medium läuft. Bisher konnten diese Neutrinos aber noch nicht beobachtet werden.

Als weitere Quelle von Neutrinos gelten Gamma-Ray-Bursts (GRB), die wohl die hellsten Objekte mit der größten Energie im Universum darstellen. Bei den GRB handelt es sich um kurze, einige Sekunden bis einige Minuten andauernde Energieausbrüche, hochenergetischer elektromagnetischer Strahlung. Ihr Ursprung ist Gegenstand aktueller Forschung.

Die genannten Beispiele sind vermutlich Quellen der hochenergetischen kosmischen Strahlung. In Abbildung 1.1 ist der gemessene Fluss in Abhängigkeit der Energie aufgetragen. Sie besteht zum größten Teil aus Protonen (90%), Alphateilchen (9%) und einigen schwereren Kernen [2].

Trifft ein Proton mit ausreichend Energie auf andere Protonen oder schwere Kerne, entstehen Pionen und Hadronen wie in Formel 1.1 dargestellt ist.

$$p + X(Nukleon, \gamma, e^-) \rightarrow \pi^{\pm} + Hadronen \tag{1.1}$$

Das $\pi^{\pm}$ zerfällt nach einer Lebensdauer von etwa $2,603 \cdot 10^{-8} s$ (aus [4]) in ein Myon und ein Neutrino wie hier dargestellt:

$$\begin{aligned} \pi^+ &\rightarrow \mu^+ + \nu_\mu \\ \pi^- &\rightarrow \mu^- + \bar{\nu_\mu} \end{aligned} \tag{1.2}$$

Auch das Myon zerfällt nach einer Lebensdauer von $2,197 \cdot 10^{-6} s$ und erzeugt dabei weitere Neutrinos:

$$\begin{aligned} \mu^+ &\rightarrow e^+ + \nu_e + \bar{\nu_\mu} \\ \mu^- &\rightarrow e^- + \bar{\nu_e} + \nu_\mu \end{aligned} \tag{1.3}$$

Demnach sind alle Orte, an denen die oben genannten Teilchen beschleunigt bzw. erzeugt werden, potentielle Quellen von Neutrinos.

Der indirekte Nachweis Dunkler Materie ist ebenfalls über Neutrinos und dadurch mit dem ANTARES-Detektor möglich. Neutralinos werden in super-symmetrischen Theorien gefordert und sind Majorana-Teilchen, bilden demnach ihre eigenen Antiteilchen. Als WIMP (Weakly Interacting Massive Particle) gelten sie als ein Kandidat für

[1] 1kpc $\approx 3,26$ Lichtjahre

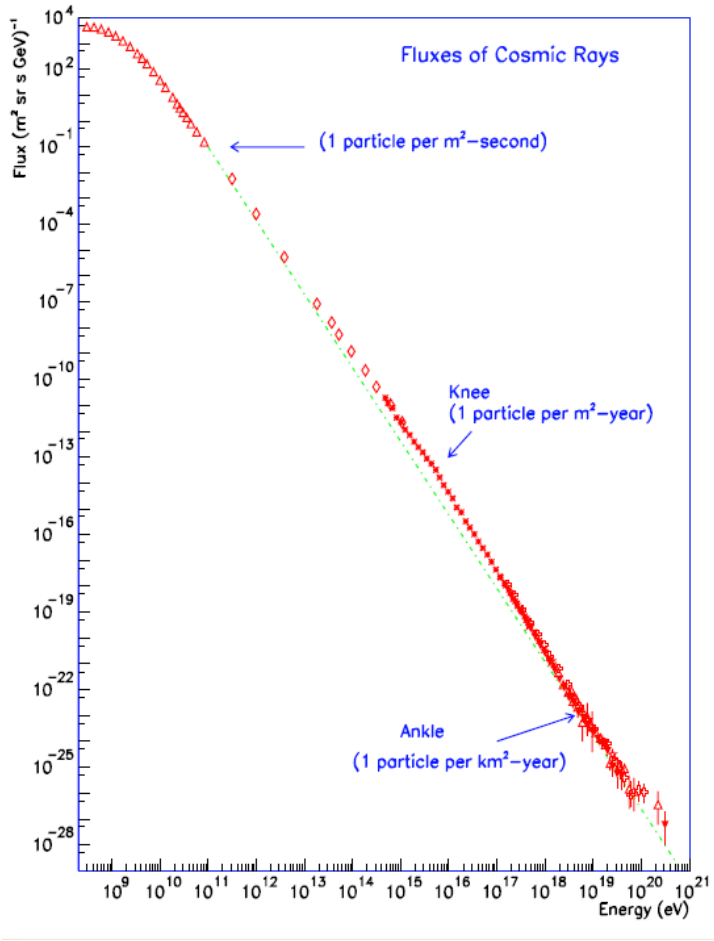


Abbildung 1.1: Energie und Fluss kosmischer Strahlung, aus [3]

die Kalte Dunkle Materie. Durch Neutralino-Annihilation entstehen Neutrinos, die im ANTARES-Detektor nachgewiesen werden können. Als Quellen von Neutralinos kommen z.B. Annihilationen im freien Raum und dem galaktischen Halo in Frage. Durch die Gravitation können Neutralinos in massereichen Objekten, wie der Sonne, angereichert werden. Je höher die Neutralinodichte, desto höher ist die Annihilationsrate und damit die Neutrinoproduktion in diesen Objekten.

1.1.2 Nachweis von Neutrinos

1.1.2.1 Entdeckung

Bereits 1930 postulierte W. Pauli [5] das Neutrino, um das kontinuierliche Spektrum des Elektrons beim Betazerfall des Neutrons zu erklären und die Drehimpulserhaltung zu gewährleisten [5].

Der geringe Wirkungsquerschnitt der Neutrinos, welcher es ermöglicht weit entfernte intergalaktische Objekte zu beobachten, macht es auf der anderen Seite sehr schwer Neutrinos zu detektieren. Da es keine elektrische Ladung besitzt und nur an der schwachen Wechselwirkung teilnimmt, wurde es erst 20 Jahre später experimentell nachgewiesen.

Den ersten experimentellen Nachweis von Neutrinos lieferte 1950 das Experiment von Cowan und Reines [5]. Die Experimentatoren machten sich die Reaktion eines Neutrinos (genauer: Antielektronneutrino) mit einem Proton zunutze, bei der die beiden nachweisbaren Teilchen Neutron und Positron erzeugt werden:

$$\bar{\nu}_e + p \rightarrow n + e^+ \tag{1.4}$$

Der Einfang von Neutrinos durch den inversen Betazerfall, wie er von Cowan und Reines verwendet wurde, konnte zum Nachweis von Neutrinos verwendet werden, weil sich das Experiment in unmittelbarer Nähe zu einem Kernreaktor befand. Die Anzahl der verfügbaren Neutrinos war dementsprechend groß. Die Abbildung 1.2 zeigt den Wirkungsquerschnitt von Myon-Neutrinos und -Antineutrinos in Abhängigkeit der Energie.

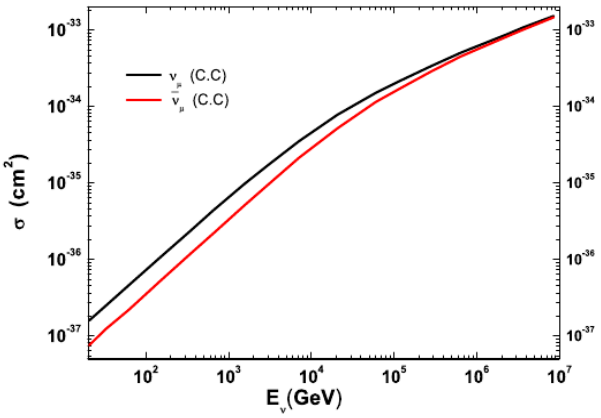


Abbildung 1.2: Wirkungsquerschnitt des Myonneutrinos in Abhängigkeit seiner Energie, aus [6].

1.1.2.2 Wechselwirkungsprozesse

Erst die Wechselwirkung mit anderen Teilchen macht es möglich Neutrinos nachzuweisen. Neutrinos besitzen keine elektrische Ladung und nehmen nur an der schwachen Wechselwirkung teil. Die entsprechenden Wechselwirkungsteilchen sind die geladenen W^+ und W^- -Bosonen sowie das neutrale Z^0 -Boson, die an Quarks und Leptonen koppeln. Je nachdem welche der oben genannten Bosonen an der Reaktion teilnehmen, spricht man von 'Charged Current' - (für die W^+ und W^--Bosonen) und von 'Neutral Current' – Reaktionen, die im Folgenden mit 'CC' und 'NC' abgekürzt werden. Alle Reaktionen mit Neutrinos lassen sich grob in diese beiden Bereiche einteilen, wobei im

Fall einer CC-Reaktion nicht nur ein Impuls-, sondern auch ein Ladungsübertrag statt findet.

Bei der quasi-elastischen Streuung von Neutrinos an Nukleonen handelt es sich um Reaktionen der Form:

$$\nu_l + n \rightarrow p + l^- \tag{1.5}$$

$$\bar{\nu_l} + p \rightarrow n + l^+ \tag{1.6}$$

mit $l = e, \mu, \tau$ Dies sind CC-Reaktionen, bei denen ein geladenes W-Boson als Austauschteilchen dient. Die zweite Formel wurde von Cowan und Reines, für den Fall für Elektronen, benutzt, um Neutrinos nachzuweisen. Die untere Grenze der Energie des Neutrinos (bei ruhendem Nukleon) beträgt ca. 1.8MeV [3] und ergibt sich aus den Schwerpunktenergien der beteiligten Teilchen:

$$E^t h_\nu = \frac{(m_n + m_e)^2 - m_p^2}{2m_p} \backsimeq 1.806 MeV \tag{1.7}$$

Wie der Abb. 1.3 zu entnehmen ist, sinkt der Wirkungsquerschnitt für quasi-elastische Prozesse bei zunehmender Energie. An ihre Stelle treten tief-inelastische Streuungen. Die Reaktionen lauten wie folgt:

$$\begin{aligned} \nu_l + N &\rightarrow l^- + X \\ \bar{\nu_l} + N &\rightarrow l^+ + X \end{aligned} \tag{1.8}$$

$$\begin{aligned} \nu_l + N &\rightarrow \nu_l + X \\ \bar{\nu_l} + N &\rightarrow \bar{\nu_l} + X \end{aligned} \tag{1.9}$$

($l = e, \mu, \tau, N = Nukleon, X = Hadron$, siehe [7]) Im Fall von 1.8 sind die Austauschteilchen die geladene W^+ und W^--Bosonen, für 1.9 dagegen das Z^0-Boson.

1.1.2.3 Der Cherenkov-Effekt

Während das Experiment von Cowan und Reines die Existenz des Neutrinos selbst nachweisen sollte, sehen sich Projekte wie ANTARES dem Problem gegenübergestellt, dass sie möglichst jedes Neutrino erfassen wollen, welches den Detektor durchquert. Das gilt insbesondere für die verschiedenen Flavour, also Elektron-, Myon- und Tauneutrinos. Darüber hinaus geht es nicht nur um die bloße Existenz von Neutrinos, sondern auch um die Richtung aus der es kam und die Bestimmung seiner Energie.

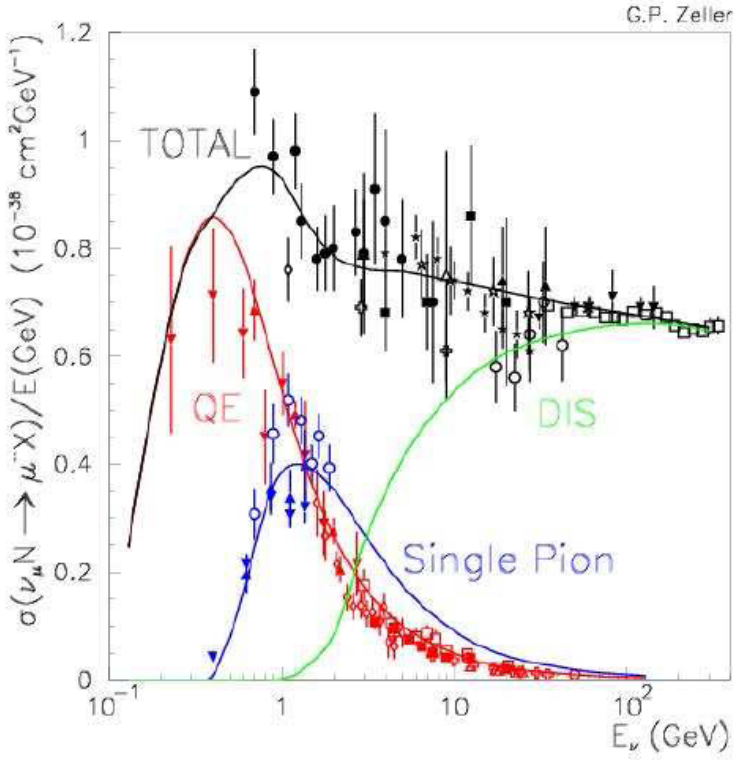


Abbildung 1.3: Wirkungsquerschnitte von Neutrinoreaktionen. QE = quasi-elastische Streuung, DIS = tief-inelastische Streuung. Die Single-Pion Prozesse sind der tief-inelastischen Streuung zuzurechnen [7].

Neutrino-Teleskope sind daher gezwungen einen anderen Weg zum Nachweis von Neutrinos zu wählen. In Kapitel 1.1.2.2 wurde beschrieben, dass bei der Kollision eines Neutrinos mit einem Nukleon, unter anderem Leptonen, also geladenen Teilchen, entstehen. Überschreitet die Geschwindigkeit dieser geladenen Teilchen die Geschwindigkeit von Licht im selben Medium, kommt es zum Cherenkov-Effekt, bei dem Licht unter einem charakteristischen Winkel zur Bewegungsrichtung des Teilchens abgestrahlt wird. Aufgrund seiner Geschwindigkeit entsteht ein Lichtkegel hinter dem Teilchen, wie beispielhaft in Abbildung 1.4 skizziert ist.

Der Winkel, unter dem Cherenkov-Licht abgestrahlt wird, ist abhängig von der Geschwindigkeit des Teilchens und dem Brechungsindex des Mediums, durch das es sich bewegt. Der Cherenkov-Winkel kann wie folgt berechnet werden:

$$cos\Theta_C = \frac{\triangle x_{Photon}}{\triangle x_{Teilchen}} = \frac{\triangle tv_{Photon}}{\triangle tv_{Teilchen}} = \frac{c}{n\beta c} = \frac{1}{n\beta} \tag{1.10}$$

Da sich die geladenen Teilchen, die bei Reaktionen hochenergetischer Neutrinos entstehen, mit einer Geschwindigkeit von ca. 30cm/ns bewegen und die Lichtgeschwindigkeit in Wasser dagegen nur etwa 22cm/ns beträgt, entsteht Cherenkov-Licht. Setzt man

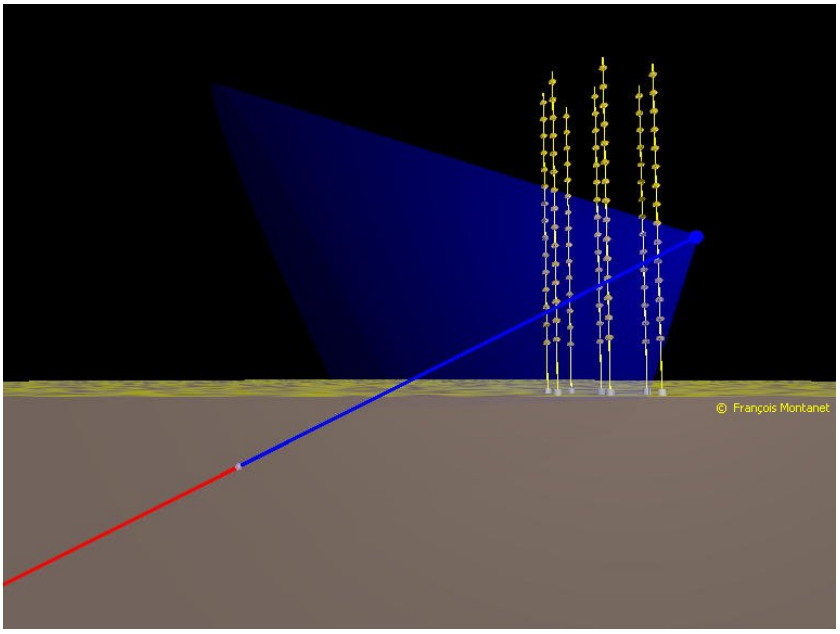


Abbildung 1.4: Schematische Darstellung eines Neutrinos (rot) das im Innern der Erde reagiert und ein geladenes Teilchen wie z.B. ein Myon (blau) erzeugt. Dieses durchquert den Detektor (hier als gelbe Linien dargestellt) unter Aussendung von Cherenkov-Licht, das hier als blauer Kegel dargestellt ist.

die vorherrschenden Gegebenheiten am Standort des ANTARES-Detektors ($n \approx 1,35$ und $\beta \approx 1$) in die Formel 1.10 ein, erhält man 42° für den Cherenkov-Winkel. Die Signatur eines geladenen Teilchens im Detektor wird maßgeblich durch diesen Winkel bestimmt, was eine gezielte Suche ermöglicht.

ANTARES macht sich genau diesen Effekt zunutze, um die Leptonen mit elektrischer Ladung aus den Reaktionsgleichungen in Kapitel 1.1.2.2 nachzuweisen. Der Detektor erfasst dafür den genauen Ort, den Zeitpunkt und die Intensität des Cherenkov-Lichts und ermöglicht so eine Rekonstruktion der Bewegung des Teilchens.

1.1.2.4 Energieverlust von Leptonen in Materie

Bewegen sich geladene Leptonen durch Materie, können diese durch verschiedene Prozesse Energie verlieren. Besitzt das Lepton eine Ladung und ist seine Geschwindigkeit ausreichend, wird Cherenkov-Licht erzeugt. Der dabei entstehende Energieverlust ist im Vergleich zu den anderen Prozessen jedoch gering. Den Großteil ihrer Energie verlieren sie durch Ionisation, Bremsstrahlung, Paarbildung und tief-inelastische Streuung an Kernen. Die Abbildung 1.5 zeigt den Energieverlust eines Myons in Wasser in Abhängigkeit der Energie.

In Abhängigkeit des Flavours des Neutrinos, werden wie in Kapitel 1.1.2.2 beschrie-

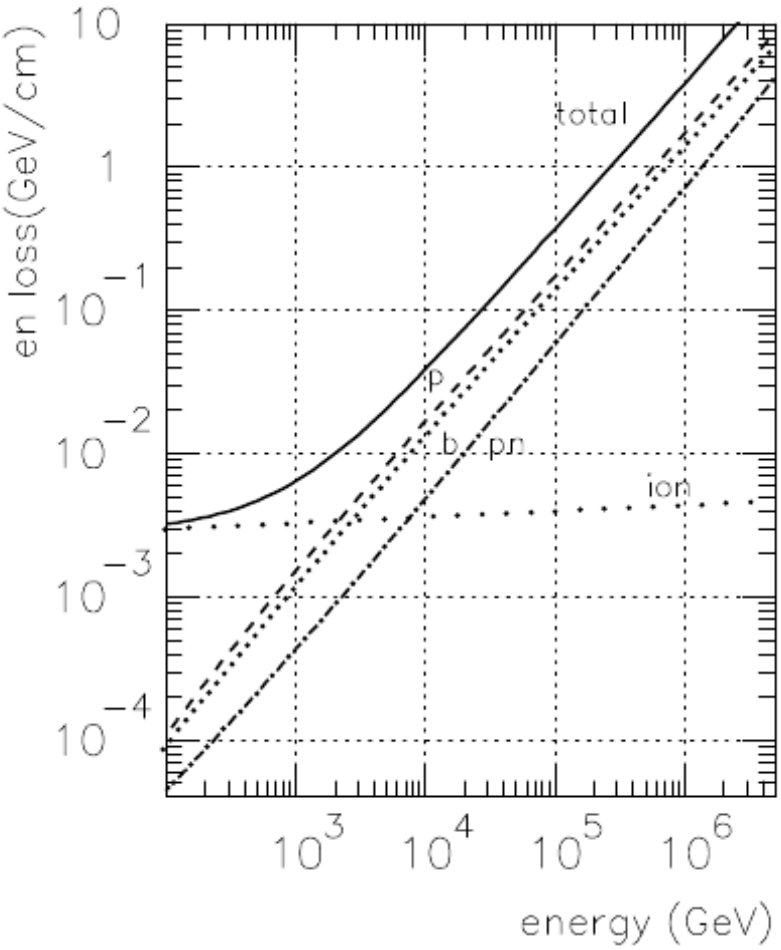


Abbildung 1.5: Energieverlust eines Myons, das sich durch Wasser bewegt. Neben dem Gesamtverlust (total) sind die verschiedenen Prozesse, die zum Energieverlust führen, zu sehen: Ionisation (ion), Bremsstrahlung (b), Paarbildung (p) und tief-inelastische Prozesse (pn)

ben, unterschiedliche Teilchen erzeugt. Die Spuren, die diese hinterlassen sind charakteristisch für das jeweilige Neutrino bzw. das dazugehörige Lepton. Die Abbildung 1.6 zeigt Beispiele von Signaturen, wie sie verschiedene Neutrinos hinterlassen.

Hochenergetische Elektronneutrinos erzeugen durch eine CC-Reaktion hochenergetische Elektronen (Abb. 1.6(c)). Diese werden durch Stoßprozesse abgebremst, die dabei entstehende Bremsstrahlung kann, genügend Energie vorausgesetzt, durch Paarbildung weitere Elektronen und Positronen erzeugen. Die daraus resultierende Bremsstrahlung erzeugt einen elektromagnetischen Schauer. Die Ausdehnung dieser EM Schauer sind mit einer Länge von ca. 10m und einer sehr geringen lateralen Ausdehnung von ca. 10cm, im Umfeld des Detektors klein [6], weshalb sie als Punktlichtquelle erscheinen.

Bei der CC-Reaktion des Tauneutrinos (Abb. 1.6(b)) entsteht neben einem τ ein

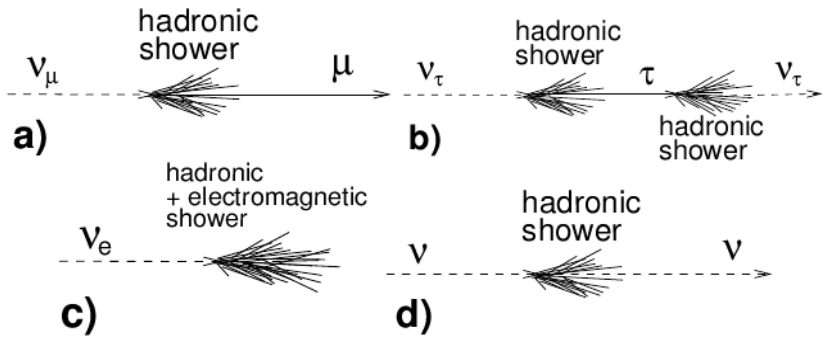


Abbildung 1.6: Signaturen verschiedener Ereignisse.

Teilchenschauer. Mit seiner relativ kurzen Lebenszeit von etwa $2,91 \cdot 10^{-13} s$ [4], zerfällt das τ in einem weiteren Schauer unter Erzeugung eines Tauneutrinos. Da das τ elektrisch geladen ist, emittiert es Cherenkov-Licht und hinterlässt so eine Spur. Natürlich kann sowohl der Beginn, wie auch das Ende der Spur des Taus innerhalb und außerhalb des Detektors liegen. Die Ähnlichkeit zur CC-Reaktion des Myonneutrinos macht es schwierig diese voneinander zu unterscheiden. Insbesondere dann, wenn das Teilchen den Detektor nur streift oder nur der Anfang dieser Reaktion innerhalb des Detektors liegt.

Das Myonneutrino (Abb. 1.6(a)) erzeugt einen hadronischen Schauer und ein Myon. Das Myon bewegt sich mit annähernd der selben Richtung wie das einfallende Neutrino fort, wie durch die folgende Formel beschrieben [6]:

$$\Theta_{\nu\mu} \leqq \frac{0.6^\circ}{\sqrt{E_\nu(TeV)}} \tag{1.11}$$

Aufgrund seiner Lebensdauer von $2,197 \cdot 10^{-}6s$ [4] muss es nicht im Detektor entstehen, sondern kann bereits innerhalb der Erde erzeugt werden und dennoch den Detektor erreichen. Als geladenes Teilchen erzeugt es dort Cherenkov-Licht entlang seiner Bewegungsrichtung. Durch die oben genannten Prozesse, die zum Energieverlust des Myons in Materie führen, ist zu erwarten, dass sich die Richtung des Myons ändert. Wie sich aber zeigt, ist der zu erwartende Winkel für diese Ablenkung, im interessanten Energiebereich, geringer als der Winkel aus Gleichung 1.11 und kann daher vernachlässigt werden [6]. Die Abbildung 1.7 zeigt die Reichweite von Teilchen, entstanden aus Neutrinoreaktionen, in Wasser. Wie man sieht, kann diese in Abhängigkeit der Energie, im Fall für Myonen einige Kilometer betragen, was das effektive Detektorvolumen erhöht, da es nicht zwingend notwendig ist, dass die Reaktion innerhalb des Detektors stattfindet.

Die NC-Reaktionen (Abb. 1.6(d)) haben, unabhängig vom Flavour, immer einen Schauer und ein austretendes Neutrino zur Folge. Durch die Teilchen in hadronischen

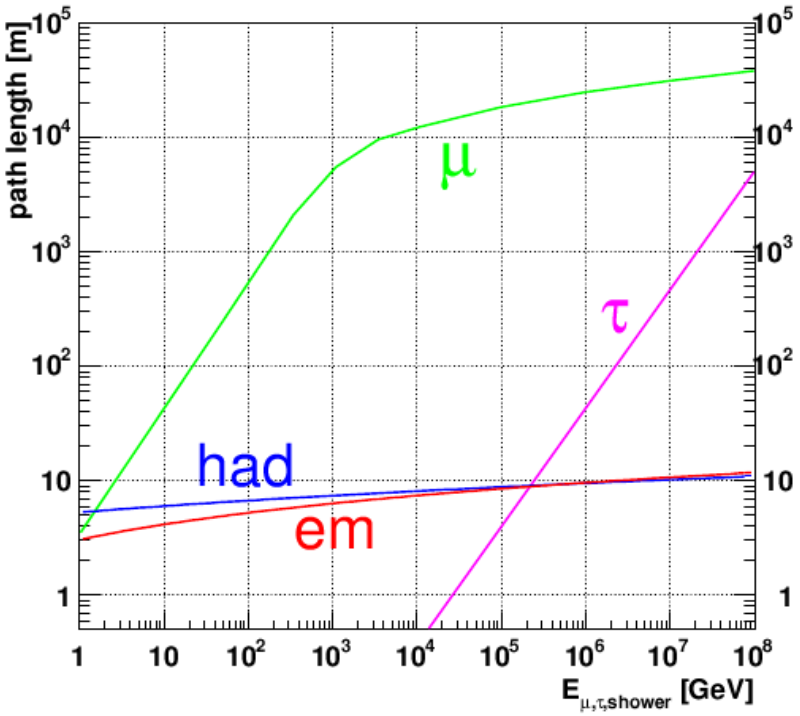


Abbildung 1.7: Spurlängen von Teilchen in Wasser, erzeugt durch Neutrinoreaktionen in Abhängigkeit der Energie. [8]

Schauern kommt es z.B. durch den Pionzerfall zu weiteren Myonen, die mit genügend Energie ebenfalls Cherenkov-Licht erzeugen können. Den größeren Teil des emittierten Lichts, bei Energien über 1TeV, erzeugen jedoch elektromagnetische Schauer [9]. Mit einer Länge von wenigen Metern (siehe [6]), erscheinen auch diese Ereignisse, aufgrund der Maße des Detektors, als Punktlichtquellen.

1.2 Aufbau des Detektors

Seit Mai 2008 ist der ANTARES-Detektor komplett aufgebaut und zeichnet mit seinen 12 Strings Daten auf. Jeder String ist etwa 480m hoch. Am Boden des Meeres verankert, wird jeder String durch eine Boje an ihrem oberen Ende, in einer aufrechten Position gehalten, wie es in Abb. 1.8 dargestellt ist. Die Anordnung der Strings auf dem Meeresboden ist in Abb. 1.9 zu sehen. Der Abstand benachbarter Strings beträgt 60-75m [8].

An jedem String sind 75 optische Module angebracht, die das Cherenkov-Licht detektieren. Immer drei optische Module bilden ein Stockwerk und sind in einer Höhe, im Abstand von 120° um den String angebracht. Fünf Stockwerke bilden wiederum einen

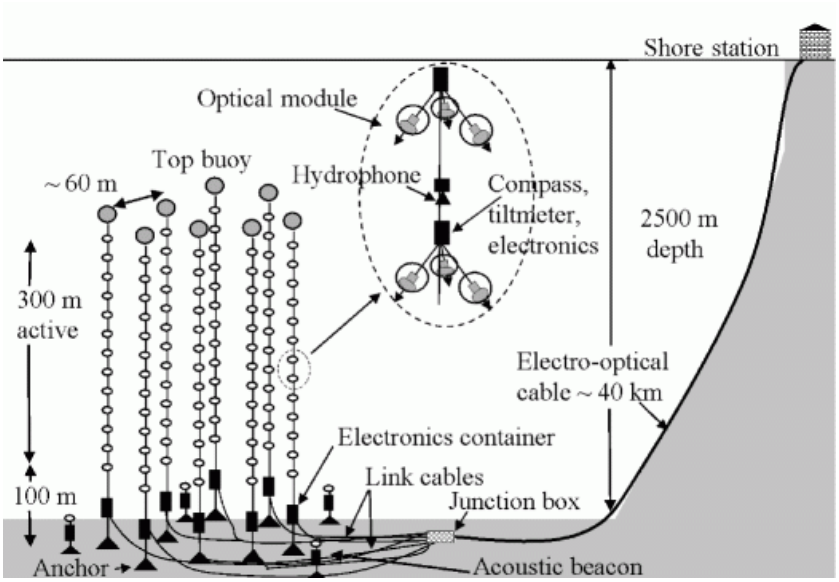


Abbildung 1.8: Schematischer Aufbau des ANTARES-Detektors.

Sektor. In Abbildung 1.10 ist der Aufbau eines Strings dargestellt. Insgesamt stehen bei ANTARES 885 optische Module zur Verfügung. Jedes dieser Module ist in einem Winkel von 45° nach unten abgewinkelt, was die Orientierung des Detektors bestimmt. Die Elektronik zur Datenaufnahme befindet sich in einem Titanzylinder, an dem die

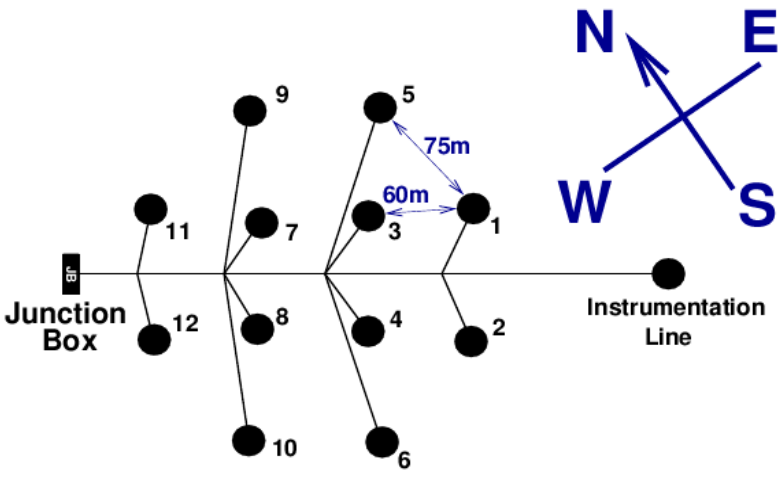


Abbildung 1.9: Die Anordnung und Nummerierung der Strings auf dem Meeresboden [8].

optischen Module jedes Stockwerks angebracht sind. Die Stockwerke sind durch Kabel miteinander verbunden. Jeder String ist mit der sogenannten Junction-Box verbunden. Diese bündelt die Daten und verschickt sie über ein Glasfaserkabel an die Küstenstation

zur Auswertung. Im ANTARES-Kontrollraum, an der Küste, steht eine Computer-Farm bereit, um die Daten weiter zu verarbeiten und zu speichern.

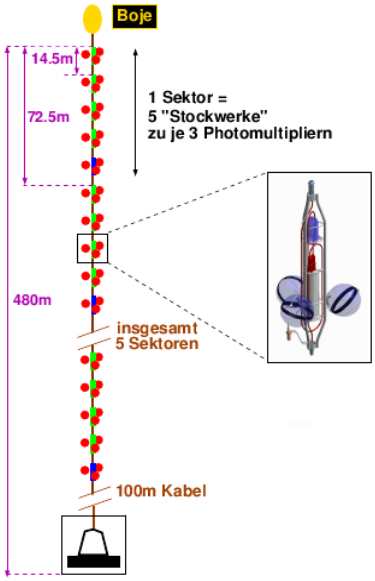


Abbildung 1.10: Schematischer Aufbau eines Strings von ANTARES.

Einige Stockwerke sind zusätzlich mit Systemen zur Positionierung ausgestattet. Durch Kompasse und Neigungsmesser, sowie der Erfassung der Meeresströmung, wird die Verschiebung und Torsion der Strings bestimmt. Hydrophone dienen der akustischen Positionsbestimmung der Stockwerke. Zur Zeitkalibration der optischen Module, werden in regelmäßigen Abständen kontrollierte Lichtimpulse, von zusätzlich angebrachten LED-Modulen ausgesendet. Spezielle Datennahme-Runs dienen der Ladungskalibration.

1.3 Datenaufnahme

Da der Mensch beim ANTARES-Experiment keinen Einfluss auf den Zeitpunkt des Eintreffens eines Teilchens in den Detektor hat, wie es sich beispielsweise bei Experimenten an einem Teilchenbeschleuniger verhält, ist es notwendig, dass der Detektor ununterbrochen Daten erfasst und analysiert. Da die dabei anfallenden Datenmengen für heutige Verhältnisse zu groß sind, um abgespeichert zu werden, ist es zwingend notwendig diese intelligent zu verringern.

Die Daten werden dafür von Triggern nach bestimmten Bedingungen durchsucht, sie entscheiden anschließend ob die Daten aufgezeichnet oder verworfen werden. Die Trigger

sind zu diesem Zweck in unterschiedliche Ebenen aufgeteilt. Der sogenannte L1-Trigger sucht nach sehr großen Amplituden an einzelnen Photomultipliern oder nach Signalen von benachbarten Photomultipliern eines Stockwerks, mit einer typischen Koinzidenz von 25ns. Der L2-Trigger sucht in den Treffern des L1-Triggers, die über mehrere Stockwerke und Lines verteilt sein können, nach weiteren Zusammenhängen, bei denen die Ausbreitungsgeschwindigkeit von Licht in Wasser berücksichtigt wird. Der Unterschied der Zeitpunkte, zweier getriggerter Signale i und j, müssen dabei folgender Bedingung genügen:

$$|\triangle t| = |t_i - t_j| < \frac{|\vec{r_i} - \vec{r_j}|}{v_g} = \frac{|\triangle \vec{r}|}{v_g} \tag{1.12}$$

Wobei v_g die Gruppengeschwindigkeit von Licht in Wasser, $\vec{r_{i,j}}$ den Ort und $t_{i,j}$ den Zeitpunkt der Signale bezeichnet. Können nun mindestens 5 L1-Signale auf diese Weise miteinander verknüpft werden, spricht man von einem Ereignis. Erst wenn ein Ereignis auf diese Weise erkannt wurde, beginnt die Datenaufzeichnung. Um sicher zu stellen, dass keine Informationen verloren gehen, werden nicht nur die getriggerten, sondern alle Signale, inklusive Untergrund, die 2200ns vor und nach dem Ereignis auftraten, mit gespeichert. In den Messdaten finden sich also alle aufgenommenen Signale, aller Photomultiplier in einem bestimmten Zeitfenster.

Für die Stufe L2 in der Triggerhierarchie stehen verschiedene Trigger zur Verfügung, die unterschiedliche Verfahren anwenden und so unterschiedlich oft auslösen. In Abhängigkeit der Triggerrate werden Trigger hinzu- oder abgeschaltet.

1.4 Untergrund

Wie jedes andere Experiment auch, nimmt der ANTARES-Detektor nicht nur Signale gewünschter Prozesse auf. Es gibt einen nicht unerheblichen Untergrund, der sich aus verschiedenen Quellen zusammen setzt.

Einen Teil des optischen Untergrunds bildet das radioaktive Nuklid ^{40}K, wobei das Elektron aus dem Betazerfall Cherenkov-Licht erzeugt:

$$^{40}\mathrm{K} \rightarrow ^{40}\mathrm{Ca} + e^- + \bar{\nu_e} \tag{1.13}$$

Aber auch Biolumineszenz, z.B. von Kleinstlebewesen und Fischen, erzeugt Lichtsignale im Detektor.

Im Folgenden wird diese Art des Untergrunds als Rauschen bezeichnet. In Abhängigkeit von verschiedenen Faktoren, wie Jahreszeit und Strömungsgeschwindigkeit, beträgt das Rauschen in guten Fällen 60kHz pro Photomultiplier, in schlechten Fällen sogar weit darüber. Überschreitet die Rauschrate einige 100kHz, wird die Spannung an

den Photomultipliern, zu deren Schutz reduziert. Im Extremfall kann es auch vorkommen, dass der Detektor abgeschaltet werden muss. Eine zuverlässige Datenauswertung erscheint bei solch hohen Raten praktisch nicht mehr möglich und somit ist es besser die Photomultiplier durch diese Maßnahme zu schonen.

Ein weiterer Untergrund für die Suche nach kosmischen Neutrinos, stammt von geladenen Teilchen, die von oben in den Detektor eindringen. Diese finden ihren Ursprung in den Luftschauern, erzeugt in der Atmosphäre der Erde. Zwar gelingt es Teilchen, wie Protonen und Elektronen nicht in die Tiefe des Detektors vorzudringen, doch werden dabei auch atmosphärischen Myonen erzeugt, die ein wesentlich höheres Durchdringungsvermögen besitzen. Die atmosphärischen Myonen treffen den Detektor in einem Verhältnis von $1 : 10^6$ zu den aus atmosphärischen Neutrinos erzeugten Myonen. Abbildung 1.11 verdeutlicht das Ausmaß dieses Untergrunds. Die Winkelverteilung zeigt aber

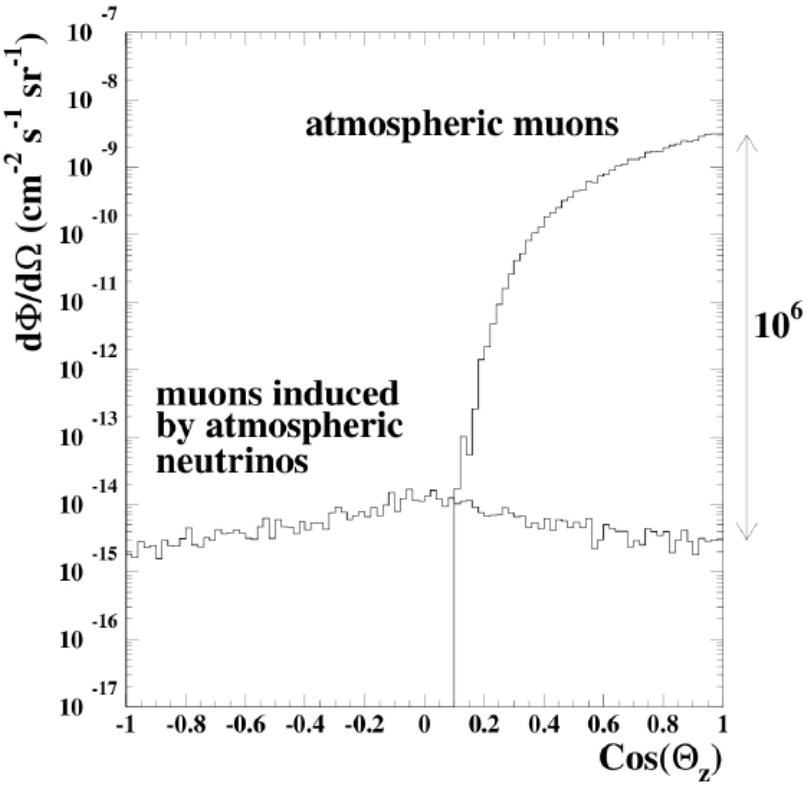


Abbildung 1.11: Winkel aufgelöste Darstellung des Flusses von atmosphärischen Myonen und Myonen mit einer Energie > 1 TeV aus atmosphärischen Neutrinos in einer Wassertiefe von 2300m

auch, dass der Schutzschild Erde die atmosphärischen Myonen von der gegenüberliegenden Seite vollständig ausfiltert. Ein Ereignis, welches von unten kommt gehört demnach mit hoher Wahrscheinlichkeit zu einem Neutrino, welche der Detektor nachweisen soll. Im folgenden werden diese atmosphärischen Myonen, bzw. von oben kommende Ereignisse als Untergrund bezeichnet.

Die atmosphärischen Myonen sind ein großer Störfaktor bei der Suche nach Neutrinos. Bisher werden Rekonstruktionsalgorithmen verwendet, die versuchen jedem Ereignis eine Teilchenspur zuzuweisen. Die Richtung der Spur entscheidet damit über den Ereignistyp. Leider kommt es dabei zu Fehlrekonstruktionen, die in Anbetracht der Häufigkeit des Untergrunds ein großes Problem darstellen.

Kapitel 2

Ansatz zur Entwicklung einer Ereignisklassifikation

Um die Rekonstruktion verschiedener Ereignisse zu erleichtern, wäre eine Klassifikation hilfreich, die die Signaturen aus Kapitel 1.1.2.4 erkennt. Nach einer Klassifizierung könnte eine, auf diese Ereignisart spezialisierte Rekonstruktion angewendet und die Fehlrekonstruktionen verringert werden. Die Klassifikation muss für diesen Zweck möglichst unabhängig von einer Rekonstruktion sein.

Diese Arbeit konzentriert sich auf die Unterscheidung zwischen Myonen aus Myonneutrinos, die von unten in den Detektor eindringen (im Folgenden Signal genannt) und atmosphärischen Myonen (im Folgenden Untergrund genannt), die von oben kommen.

Es gibt verschiedene multivariate Techniken, die zur Klassifikation von Ereignissen verwendet werden. In dieser Arbeit wird eine bestimmte Klasse von künstlichen Neuronalen Netzen konfiguriert und trainiert. Künstliche Neuronale Netze sind in der Lage aus gegebenen Eingangsdaten Muster bzw. Gesetzmäßigkeiten zu erlernen, um diese dann auf ähnliche Daten anzuwenden und eine Entscheidung über ihre Klassifikation zu treffen. Daher erscheint die Methodik für die Anwendung auf die gegebene Problemstellung, die Erkennung von 2 Ereignisklassen im Detektor, als vielversprechend.

2.1 Software für künstliche Neuronale Netze

Künstliche Neuronale Netze werden von Computern berechnet. Es gibt viele Implementierungen in Form von Softwarebibliotheken zur (mehr oder weniger) freien Verfügung. Eine eigene Implementierung zu schreiben erscheint, aufgrund dieses Angebots, überflüssig. Daher wurde auf die frei verfügbare Software FANN (Fast Artificial Neural Network Library)[10] zurückgegriffen, die neben hoher Flexibilität auch ein einfach zu verstehendes Interface und eine gute Dokumentation besitzt. Geschrieben in der Programmiersprache C, gibt es Bindings für die meisten gängigen Programmiersprachen

wie C++, Java, Perl, Python und viele mehr.

2.2 Software des ANTARES-Experiments

Aufgrund der Datenmengen, die bei ANTARES entstehen, werden alle Analysen von Computerprogrammen übernommen. Um die gemeinsame Arbeit zu erleichtern und um zu vermeiden, dass dieselben Schritte in einer Analysekette von jedem selbst neu programmiert werden müssen, steht ein umfangreiches und einfach zu handhabendes Framework mit dem Namen Seatray [11] zur Verfügung. Geschrieben in der Programmiersprache C++, bietet es die Möglichkeit, die Analysekette mit Hilfe von Pythonmodulen individuell, schnell und einfach zusammenzusetzen. Dem Nutzer ist es möglich eigene Module zu schreiben und in die eigene Analysekette einzubinden. Diese Arbeit nutzt dieses Framework, um die Daten zu gewinnen, mit denen die künstlichen Neuronalen Netze trainiert und getestet werden.

2.3 Simulation

Da die Analyse von Messdaten mit schwer erkennbaren systematischen Unsicherheiten behaftet sein kann, sind zum Verständnis komplexer Experimente Simulationen nötig, die den gesamten Detektor und alle Umgebungsvariablen, wie z.B. den Untergrund und das Rauschen, nachbilden. Diese Prozesse werden unter der Bezeichnung Monte Carlo Simulation (kurz MC) zusammengefasst.

Diese Arbeit stützt sich auf MC-Daten, die bereits fertig zur Verfügung standen. Der Vollständigkeit wegen soll hier aber trotzdem kurz auf einzelne Aspekte eingegangen werden. Weiterführende Informationen finden sich z.B. unter [8].

Um atmosphärische Myonen zu simulieren standen in erster Linie Daten, die mit dem Programm CORSIKA [12] erstellt wurden, zur Verfügung. Die Simulationen umfassen, neben verschiedenen Primärteilchen wie Fe, He, Mg, p und N, die mit Teilchen der Atmosphäre wechselwirken und Luftschauer erzeugen, auch verschiedene Energien und Eintrittswinkel. Die Teilchen treten in erster Linie von oben in den Detektor ein. Diese Ereignisse werden im Folgenden als von oben kommende Ereignisse oder als Untergrund bezeichnet.

Die (Anti-)Neutrinos werden mit dem Programm GENHEN [8], ebenfalls mit unterschiedlichen Energien und Eintrittswinkeln generiert. Ereignisse aus diesen Simulationen werden hier als von unten kommend oder als Signal (im Gegensatz zu Untergrund) bezeichnet.

Das Rauschen wird durch eine zufällige Verteilung von Signalen an einzelnen Photomultipliern simuliert. Die Rauschrate, die in dieser Arbeit verwendet wird, beträgt 60kHz, ein Wert der auch in den Messdaten erreicht wird.

Kapitel 3

Auswahl und Funktionsweise künstlicher Neuronaler Netze (KNN)

3.1 Natürliche Neuronale Netze

Der Aufbau künstlicher Neuronaler Netze wurde von der Funktion des biologischen Gehirns inspiriert. In einem Gehirn dienen die Nervenzellen zur Speicherung und Verarbeitung von Informationen. Das Bild 3.1 zeigt eine Aufnahme einiger Nervenzellen eines Gehirns. Die Nervenzellen erhalten Impulse durch sogenannte Dendriten von anderen Zellen, siehe Abbildung 3.2. Eine Nervenzelle kann durch bis zu 200000 Dendriten mit anderen Nervenzellen verbunden sein. Dabei bedeutet ein einlaufendes Signal nicht zwangsweise die Aktivierung der Nervenzelle. Die Anzahl der ankommenden Signale und die Entfernung der Dendriten zum Zellkern sind ausschlaggebend für die Signalstärke. Überschreiten die Signale in ihrer Anzahl bzw. Intensität eine gewisse Reizschwelle, wird die Nervenzelle aktiviert. Das Signal wird über das sich verzweigende Axon und die Synapsen zu weiteren Zellen im Netzwerk transportiert. Eine Nervenzelle kann mit bis zu 10000 Synapsen mit anderen Zellen verbunden sein.

3.2 Aufbau künstlicher Neuronaler Netze

Im Vergleich zu einem echten Gehirn erscheinen heutige künstliche Neuronale Netze (im Folgenden KNN, Künstliches Neuronales Netz, genannt) geradezu unterentwickelt. Abgesehen von der riesigen Anzahl von erforderlichen Neuronen und Verbindungen, die nötig wären, um ein echtes Gehirn zu simulieren, sind es fehlende Funktionen, sowohl im Aufbau als auch während des Verarbeitens der Signale, die ein künstliches Neuronales Netz von einem echten unterscheiden. Ein biologisches Gehirn z.B. lernt durch jede neue Erfahrung dazu oder erweitert sein Gedächtnis. Diese Eigenschaften fehlen den KNN, welche hier beschrieben werden komplett [13]. Nichtsdestotrotz kann man sagen,

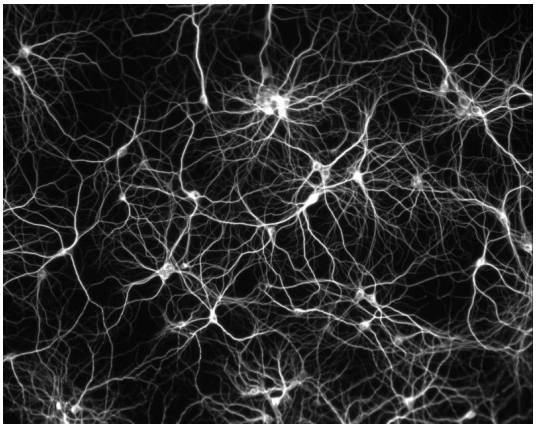


Abbildung 3.1: Abbildung einiger Nervenzellen des Gehirns einer Ratte.

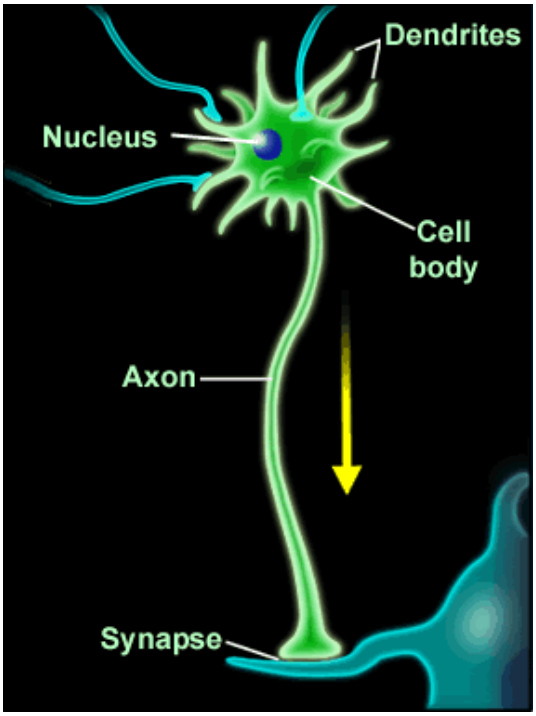


Abbildung 3.2: Aufbau einer Nervenzelle

dass die grundlegende Funktionalität der biologischen Neuronen durch die künstlichen Gegenstücke abgebildet wird. Sie verfügen über die Fähigkeit einfachere Zusammenhänge zu erlernen und anzuwenden. Künstliche Neuronale Netze finden ihre Anwendung heute z.B. in der Qualitätskontrolle, Optimierung und Bildverarbeitung sowie in vielen weiteren Bereichen der Medizin und Industrie [14].

3.2.1 Das Neuron eines KNN

Die kleinsten Einheiten biologischer Neuronaler Netze bilden die Neuronen bzw. die Nervenzellen eines Gehirns. Künstliche Neuronen versuchen die Funktion der Nervenzellen so gut wie möglich nachzubilden, was natürlich nur modellhaft geschehen kann. Das künstliche Neuron kann in drei Bereiche eingeteilt werden, Abbildung 3.3 zeigt den schematischen Aufbau. Die Signale bewegen sich von links nach rechts. In diesem Beispiel hat das künstliche Neuron drei Eingänge, die Eingangssignale sind Zahlenwerte, die mit sogenannten Gewichten (w1, w2 und w3) multipliziert werden. Die so gewichteten Zahlenwerte werden aufsummiert

$$s = \sum_{i=0}^{n} w_i x_i \tag{3.1}$$

und an eine sogenannte Aktivierungsfunktion gegeben. Der Wert der Summe und der Verlauf der Aktivierungsfunktion entscheiden über die Höhe des Ausgangssignals. Die Werte x_i sind die Eingangswerte des Neurons. Die Anzahl der Eingänge kann je nach Position des Neurons und Topologie des Netzwerks variieren.

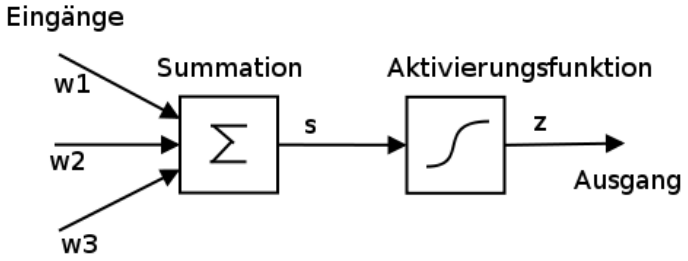


Abbildung 3.3: Aufbau eines Neurons

Als Aktivierungsfunktion werden in erster Linie differenzierbare Funktionen verwendet, da dies eine Bedingung für den Lernalgorithmus ist, der in Kapitel 3.3.1 erklärt wird. In Anlehnung an die Funktion der Nervenzelle, die ab einem bestimmten Schwellwert aktiviert wird, werden als Aktivierungsfunktionen vor allem Sprungfunktionen, wie die Sigmoid- und Tangenshyperbolikus-Funktion verwendet, die in Abbildung 3.4 dargestellt sind [13].

Das künstliche Neuron wird also in Abhängigkeit des Wertes der Summe 3.1 und der Aktivierungsfunktion aktiviert. Die gleichen Eingangswerte haben immer den gleichen Ausgangswert zur Folge. Nervenzellen können aber auch schon bei kleineren oder erst bei höheren Eingangswerten aktiviert werden. Diese Fähigkeit besitzt das künstliche Neuron in dieser Form nicht. Das dabei entstehende Problem kann nicht durch höhere oder niedrigere Gewichte umgangen werden, wenn die Aktivierungsfunktion beispielsweise die tangh-Funktion ist. Besitzen alle Eingänge eines Neurons den Wert 0, kann der

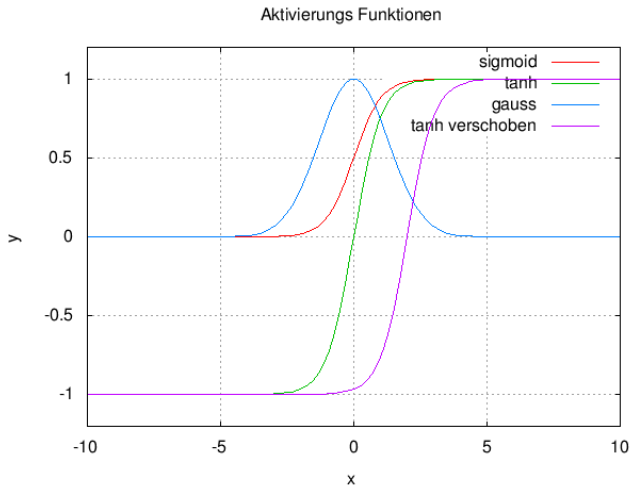


Abbildung 3.4: Verschiedene Aktivierungsfunktionen eines künstlichen Neurons. Die Sigmoidfunktion, ist ein Beispiel einer nicht-symmetrischen Funktion. Die Gaussfunktion wird unter anderem in RBF-Netzen verwendet (siehe Kapitel 3.2.2). Als Beispiel einer Symmetrischen Funktion ist der tangh abgebildet, in einem Fall um einen festen Wert auf der x-Achse verschoben.

Ausgang ebenfalls nur den Wert 0 annehmen. Mit Hilfe eines zusätzlichen Parameters t in der Aktivierungsfunktion wird dieses Einschränkung behoben:

$$f(x) = f(x + t) \tag{3.2}$$

Durch diese Erweiterung wird die Aktivierungsfunktion auf der x-Achse verschoben und das Neuron wird bei einem anderen Wert aktiviert.

3.2.2 Neuronen als Netzwerk

Die Funktion eines einzelnen Neurons erscheint recht einfach. Seine Eigenschaft komplexe Informationen zu erlernen und zu verarbeiten, erhält das System erst, wenn mehrere Neuronen zu einem Netzwerk zusammengeschlossen werden.

Die Komplexität und der Informationsfluss bestimmen die Klassifizierung unterschiedlicher Netztypen. Werden die Signale innerhalb eines Netzes nur in eine Richtung an die folgenden Neuronen weitergegeben, spricht man von Feedforward-Netzen. Besitzt das Netz wenigstens eine Rückkopplung, spricht man von Recurrent-Netzen [13].

Die Feedforward-Netze teilen sich in zwei weitere Bereiche auf, die sich durch die Anzahl der versteckten Schichten unterscheiden. Die Abbildung 3.5 zeigt ein MLP-Netz (MultiLayer Perceptron) mit zwei versteckten Schichten. Die Neuronen sind als Kreise dargestellt, die Verbindungen als Pfeile. Die Kreise, die den Neuronen der versteckten Schichten zugeordnet sind, sind grün gefüllt. Daneben werden noch die Eingangsschicht

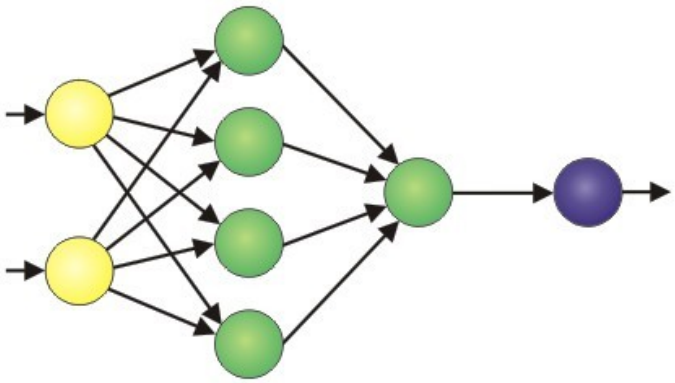


Abbildung 3.5: Mehrere Neuronen sind zu einem Feedforward-Netzwerk zusammengeschlossen. Die Kreise sind die Neuronen des Netzwerks, wobei die gelb markierten als Eingangs-, die grünen als Versteckte- und die lila gefüllten als Ausgangsneuronen bezeichnet werden.

(gelb) und die Ausgangsschicht (lila) unterschieden. Besitzt das Netz keine versteckten Neuronen, spricht man von Single Layer Feedforward-Netzen.

Es gibt weitere Unterarten von Feedforward-Netzen, wie z.B. die RBF-Netze (Radial Basis Function), deren Unterschied in den Aktivierungsfunktionen der einzelnen Schichten liegt. MLP-Netze verwenden in erster Linie Sigmoidfunktionen, während in der versteckten Schicht der RBF-Netze Gaussfunktionen und in der Ein- und Ausgangsschicht lineare Funktionen verwendet werden. Das Prinzip der Signalpropagation bleibt dabei gleich.

Während dieser Arbeit wurden verschiedene Feedforward-Netze getestet, indem ihre Aktivierungsfunktionen geändert wurden. Die MLP-Netze schnitten dabei am besten ab, weswegen sich diese Arbeit auf diesen Typ konzentrierte.

Ein MLP-Netz mit einer versteckten Schicht ist in der Lage, jede stetige Funktion, die ein Element aus einem endlich dimensionalen Raum in einen anderen abbildet, beliebig genau zu approximieren, wenn die Anzahl der Neuronen ausreichend groß gewählt wurde. Für die Klassifikation sind diese Netze dagegen nicht geeignet [15]. Erst wenn dem MLP-Netz eine zusätzliche versteckte Schicht hinzugefügt wird, ist es in der Lage auch komplexere Aufgaben zu lösen. Aus diesem Grund, aber auch durch entsprechende Tests bestätigt, werden in dieser Arbeit vor allen MLP-Netze mit zwei Schichten verwendet. Mehr Schichten sind zwar möglich bringen aber keine Verbesserung, wie Tests zeigten.

Während die Anzahl der Neuronen in den Eingangs- und Ausgangsschichten von den verfügbaren Daten und den gewünschten Ausgaben abhängen, ist die Anzahl der Neuronen in den versteckten Schichten frei wählbar. In der Literatur finden sich viele Faustformeln, über die sich die Anzahl der versteckten Schichten berechnen (besser:

abschätzen) lassen. Die benutzten Informationen variieren von Formel zu Formel. Sehr viele Umstände nehmen Einfluss auf die optimale Anzahl der versteckten Neuronen in unterschiedlichen Schichten. Eine Faustformel beziffert die Anzahl der versteckten Neuronen zwischen der Anzahl der Ein- und Ausgänge. Eine Andere sagt aus, dass nie mehr versteckte Neuronen als die doppelte Anzahl der Eingänge benötigt werden[16]. Neben der Anzahl der Ein- und Ausgänge spielt aber auch die Komplexität, verwendete Aktivierungsfunktion und nicht zuletzt die Netztopologie eine wichtige Rolle. Eine genaue Aussage ist demnach kaum möglich, und genannte Formeln sollten als Hinweise betrachtet werden. Letztendlich läuft es auf Versuche mit unterschiedlich großen Netzen hinaus, die zeigen, wie groß das Netz sein muss.

Eine sehr genaue Aussage über die Anzahl der Neuronen und den Aufbau des Netzes trifft das Theorem von Kolmogorov [15]. Es besagt, dass jede stetige Funktion $y(x_1, ..., x_d)$ mit d Variablen und einem Ausgangswert y, von einem Neuronalen Netz mit 2 versteckten Schichten exakt nachgebildet werden kann, welches in der ersten versteckten Schicht $d(2d+1)$ Neuronen und in der zweiten versteckten Schicht $(2d+1)$ Neuronen besitzt. Die Anforderungen an die Aktivierungsfunktionen sind dabei sehr genau. In der ersten versteckten Schicht kommen demnach nur strikt monotone Funktionen in Frage, während die Funktionen der zweiten versteckten Schicht sogar von $y(x_1, ..., x_d)$ abhängen. Das Theorem bestätigt allerdings nur die Existenz dieser Funktionen und es existiert noch keine Möglichkeit diese zu finden [15]. Aufgrund dieser Tatsache ist dieses Theorem nicht praxisrelevant, da KNN im Allgemeinen eine unbekannte Funktion nachbilden sollen.

3.3 Training und Test eines KNN

Bevor ein KNN eingesetzt werden kann, muss es an das jeweilige Problem heran geführt werden, d.h. es muss lernen. Zwar gibt es auch die Möglichkeit, ein ungelerntes Netz einzusetzen, die Ergebnisse eines gelernten Netzes sind dagegen aber ungleich besser. Für den Lernprozess sind Daten notwendig, die die möglichen Ein- und Ausgangswerte enthalten, d.h. zu gegebenen Eingangsparametern müssen die Ausgangswerte bekannt sein. Das setzt nicht zwangsweise voraus, dass das Problem vollständig beschrieben werden kann bzw. verstanden ist und eine Lösung auf andere Weise, wie z.B. analytisch gefunden werden kann. Die Daten können aus reinen Beobachtungen gewonnen werden in denen der Eingangszustand und der Endzustand, nicht aber das, was dazwischen passiert, bekannt ist. Bevor die Daten an das KNN gegeben werden, müssen die Wertebereiche der Daten auf einen gemeinsamen Wertebereich reskaliert werden.

Die Intelligenz des KNN liegt in den Gewichten, welche die Neuronen miteinander verbinden. Neben der Aktivierungsfunktion, welche aber in den wenigsten Fällen bzw.

nie in einem Lernprozess verändert wird, sind die Gewichte die einzigen Parameter, durch die man das Netz trainieren kann. Auch der zusätzliche Parameter t in Formel 3.2 wird nicht durch den üblichen Lernalgorithmus verändert. Um die Funktion dieses Parameters, wie in 3.2.1 beschrieben, nicht zu verlieren, wird jeder versteckten Schicht ein weiteres Neuron hinzugefügt, wie in Abbildung 3.6 zu sehen ist. Auf diese Weise muss die Aktivierungsfunktion nicht geändert werden, da die Verschiebung nun über die zusätzlichen Gewichte variiert werden kann.

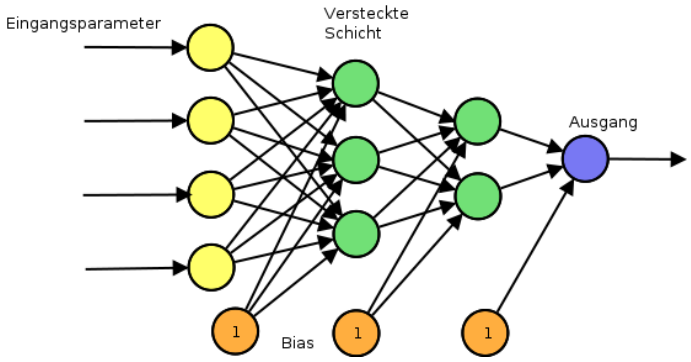


Abbildung 3.6: Ein KNN mit zwei versteckten Schichten. Für jede Schicht (außer der Eingangsschicht) ist ein zusätzliches Biasneuron eingefügt worden, dessen Eingang immer mit +1 besetzt ist.

Die Software-Bibliothek, die für diese Arbeit verwendet wurde, nutzt einen Lernalgorithmus der als Backpropagation Algorithm (BPA) bezeichnet wird. Es gibt weitere Prinzipien zur schrittweisen Anpassung der Gewichte, über die man sich beispielsweise in [13] näher informieren kann. Da diese aber nicht von der gewählten Software-Bibliothek verwendet werden, soll hier nicht weiter darauf eingegangen werden. Eine Beschreibung des BPA ist in Kapitel 3.3.1 zu finden.

Wie gut das Netz gelernt hat, sieht man während und nach dem Training anhand eines Fehlerwertes. Je niedriger dieser ist, desto besser konnte das Netz das Problem erlernen. Im Fall der Bibliothek FANN [10] wird dieser Fehler wie folgt berechnet:

$$\frac{1}{M}\sum_{m}^{M}\frac{1}{N}\sum_{n}^{N}(d_{m,n}-y_{m,n})^2 \tag{3.3}$$

Wobei M die Anzahl der Trainingsdaten, N die Anzahl der Ausgänge des KNN, $d_{m,n}$ der tatsächliche Ausgangswert des n'ten Ausgangs und m'ten Trainingssatzes, und $y_{m,n}$ der erforderliche Ausgangswert ist. Dieser Wert dient in der hier verwendeten Software lediglich als Ausgabewert.

Abbildung 3.7 zeigt ein Beispiel einer Lernkurve. Der Fehlerwert wird über die Anzahl der Epochen aufgetragen. Eine Epoche entspricht dabei dem einmaligen Durchlaufen der verfügbaren Lerndaten. Mit zunehmender Anzahl an Epochen flacht die Kurve

ab, das KNN lernt nicht mehr viel Neues aus den Wiederholungen der vorhandenen Daten.

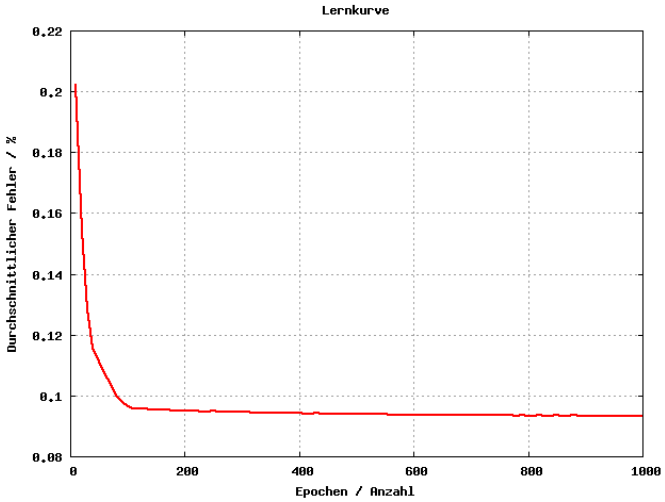


Abbildung 3.7: Beispiel einer Lernkurve eines KNN.

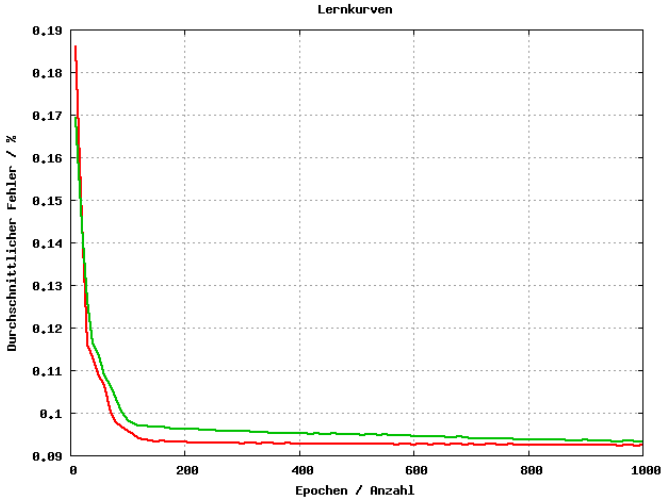


Abbildung 3.8: Zwei Netze mit identischen Bedingungen, aber unterschiedlichem Lernverhalten.

Inwiefern das gefundene Minimum der Fehlerrate einem globalen bzw. lokalen Minimum entspricht, kann nicht festgestellt werden. Da die Gewichte zu Beginn mit Zufallsdaten initialisiert werden, reagiert der Lernalgorithmus bei Wiederholungen nicht immer gleich. Es empfiehlt sich daher, mehr als ein KNN mit derselben Konfiguration und den gleichen Daten zu trainieren und dann das beste Netz auszuwählen. In Abbildung 3.8 sind zwei Lernkurven zu sehen, deren Netze unter gleichen Bedingungen trainiert wurden. Die Lerndaten und die Netzkonfiguration waren identisch.

Ist die Lernphase zu lang, kommt es zum sog. Übertraining, bei dem sich das KNN sehr stark auf die Gegebenheiten, wie sie in den Trainingsdaten vorhanden sind, eingestellt hat [15]. Ist dieser Punkt erreicht, verliert das KNN die Fähigkeit auf ungelernte Ereignisse angemessen zu reagieren. Der einzige Weg, Übertraining auszuschließen, liegt darin, das KNN nach jedem Lernzyklus mit anderen Daten zu testen. Diese Testdaten sollten nicht den Trainingsdaten entsprechen. Es ist zu erwarten, dass der mittlere Fehler der Testdaten höher als bei den Trainingsdaten liegt. Der optimale Zeitpunkt, das Lernen zu beenden, ist demnach das Minimum des Fehlers aus den Testdaten. In Abb. 3.9 sind zwei Kurven zu sehen, die Testkurve (grün) ist deutlich am ansteigenden Ende zu erkennen. Der Punkt, an dem das Training beendet werden sollte, ist das Minimum dieser Kurve, hier also etwa zwischen 4000 und 5000 Epochen. Ein Training darüber hinaus macht keinen Sinn, da sich das Netz zu sehr auf die Trainingsdaten eingestellt hat.

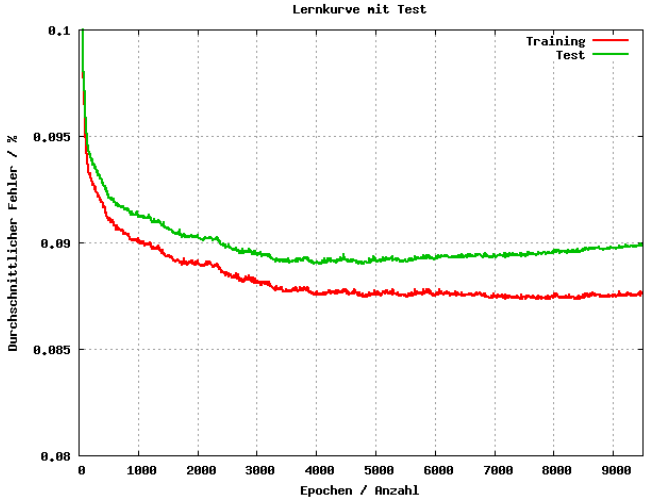


Abbildung 3.9: Training eines Netzes, bei dem nach jedem Lernschritt der Fehler für die Testdaten berechnet wurde.

3.3.1 Der Backpropagation Algorithmus

Der Backpropagation Algorithmus nutzt den Fehler jedes Ausgangsneurons um die Gewichte an den Neuronen der vorhergehenden Schicht anzupassen. Die Lernschritte durchlaufen das KNN also von hinten nach vorne, wie der Name bereits vermuten lässt. Startpunkt ist ein ungelerntes Netz mit zufällig gesetzten Gewichten.

Der Lernalgorithmus legt nun das erste Beispiel aus den Lerndaten an und propagiert dieses vorwärts durch das Netz. Jedes Neuron berechnet eine gewichtete Summe seiner

Eingänge wie folgt (vergleiche Kapitel 3.2.1):

$$a_j = \sum_i w_{ji} z_i \tag{3.4}$$

Wobei z_i die Ausgangswerte der vorherigen Neuronen und w_{ji} die Gewichte der Verbindungen zu dem Neuron j sind. Die Aktivierungsfunktion g und diese Summe bilden anschließend den Ausgangswert z_j dieses Neurons:

$$z_j = g(a_j) \tag{3.5}$$

Nun werden mit Hilfe der Fehlerfunktion 3.6, die über alle vorhandenen Datensätze summiert, Korrekturen der Gewichte berechnet.

$$E = \sum_n^N E^n \tag{3.6}$$

Wobei $E^n = E^n(y_1, ..., y_c)$, direkt von den Ausgangswerten des Neuronalen Netzes abhängt (hier mit insgesamt c Ausgängen). Das Ziel ist nun, diese Funktion zu minimieren. Dafür werden die Ableitungen von E nach den Gewichten w_{ji} berechnet. Wegen der Summe in 3.6 reicht es ein E^n zu betrachten. Die Ausgangswerte der einzelnen Neuronen werden von dem verwendeten Datenset n bestimmt. Der Übersichtlichkeit wegen, wird diese Beziehung in den folgenden Formeln weggelassen. Der Fehler E^n hängt über die Summation 3.4 von den Gewichten w_{ji} ab. Mit Hilfe der Kettenregel ergibt sich:

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{3.7}$$

Mit 3.4 ergibt sich der letzte Bruch zu:

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{3.8}$$

Ersetzt man den ersten Burch durch:

$$\delta_j := \frac{\partial E^n}{\partial a_j} \tag{3.9}$$

lässt sich 3.7 schreiben als:

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j z_i \tag{3.10}$$

Da es sich bei den Werten z_i um die Ausgangswerte der jeweils vorhergehenden Neuronen handelt, sind diese bekannt. Um die partielle Ableitung der Fehlerfunktion E^n nach ∂w_{ji} zu bestimmen, reicht es aus, die Werte für δ_j zu berechnen.

An dieser Stelle muss zwischen Neuronen der Ausgangsschicht und denen der versteckten Schicht unterschieden werden. Für ein Ausgangsneuron k ist $z_k = y_k$, wobei

y_k der Ausgangswert des Ausgangsneurons ist. Mit 3.5 und unter Berücksichtigung der Kettenregel, folgt:

$$\delta_k = \frac{\partial E^n}{\partial a_k} = g^{'}(a_k)\frac{\partial E^n}{\partial y_k} \tag{3.11}$$

Für Neuronen der versteckten Schicht, ist die Berechnung der δ_j etwas anders. Berücksichtigt man die Abhängigkeit der folgenden Neuronen von den jeweils vorherigen und verwendet die Kettenregel, lassen sich die δ_j darstellen als:

$$\delta_j = \frac{\partial E^n}{\partial a_j} = \sum_k \frac{\partial E^n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{3.12}$$

wobei die Summe über alle Neuronen k läuft, die ein Signal von Neuron j erhalten. Die Abbildung 3.10 verdeutlicht die Anordnung. Die folgenden Neuronen k können

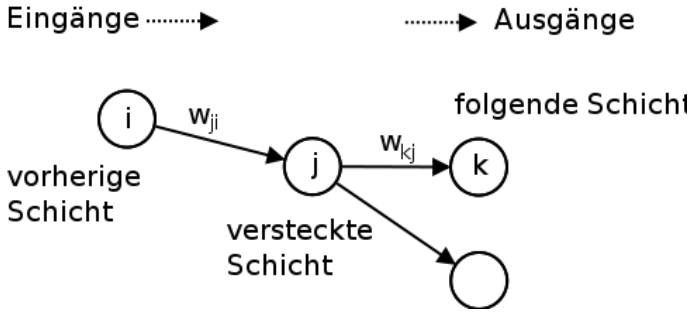


Abbildung 3.10: Schema der beteiligten Neuronen zur Berechnung der Fehler für versteckte Schichten.

Neuronen aus einer weiteren versteckten Schicht oder Ausgangsneuronen sein. Durch Verwendung der Definition in 3.9 und einsetzen von 3.4 und 3.5 ergibt sich folgende Formel:

$$\delta_j = g^{'}(a_j)\sum_k w_{kj}\delta_k \tag{3.13}$$

Wie man sieht, hängen die Fehler δ_j von den δ_k der folgenden Schicht ab. Da die Fehler der Ausgangsschicht durch 3.11 zu berechnen sind, können die Fehler der vorherigen Neuronen durch rekursives Einsetzen in 3.13 berechnet werden.

Wie zu sehen ist, wird dafür die erste Ableitung der Aktivierungsfunktion g verwendet. Eine wesentliche Anforderung des BPA an diese Funktion ist demnach ihre Differenzierbarkeit.

Aus den berechneten Fehlern δ_j lassen sich anschließend mit 3.10 die Änderungen jedes Gewichts in dem Netzwerk bestimmen:

$$w_j k^{neu} = w_j k^{alt} - \eta\frac{\partial E^n}{\partial w_{ji}} \tag{3.14}$$

Hier ist die konstante Lernrate η berücksichtigt, mit deren Hilfe die Größe der Gewichtsänderung beeinflusst werden kann. Eine zu große Lernrate führt zu großen Änderungen der Gewichte, so dass der BPA nicht in der Lage ist ein geeignetes Minimum zu finden, da die Sprünge zu groß sind. Bei einer zu kleinen Lernrate kommt der BPA unter Umständen nicht mehr aus einem gefundenen lokalen Minimum heraus. In [15] werden Möglichkeiten beschrieben, die die Lernrate nicht konstant halten.

Dem Algorithmus steht es frei, die Änderungen nach jedem Datenset n durchzuführen oder erst nachdem die Fehler aus allen Datensets berechnet wurden:

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E^n}{\partial w_{ji}} \tag{3.15}$$

Die für diese Arbeit verwendete Softwarebibliothek ändert die Gewichte des KNN nach jedem einzelnen Datensatz. Daher ist es sinnvoll auf die Reihenfolge der verwendeten Daten zu achten, da sich das Netz bei großen Mengen an Daten eventuell bereits stark an die ersten Beispiele adaptiert hat.

Aus Sicht des BPA handelt es sich bei den sogenannten Biasneuronen (siehe Kapitel 3.3) um zusätzliche Gewichte innerhalb des Netzwerks. Damit ist es nicht nötig, diese gesondert zu behandeln. Der Lernalgorithmus wendet daher dasselbe Verfahren wie für die anderen Gewichte an.

3.3.2 Beispiel des Informationsflusses in KNN

Um die Berechnungen innerhalb eines KNN und die Funktion des BPA zu verdeutlichen, ist im Folgenden der Signalverlauf zwischen den Neuronen aufgezeigt. Die Formeln aus Kapitel 3.3.1 können auf die Zahlenbeispiele angewendet werden, um die Änderungen der Gewichte während der Lernphase zu verfolgen. Das Problem, welches das KNN erlernen soll, ist die XOR-Funktion mit der Wahrheitstabelle 3.1:

Tabelle 3.1: XOR Wahrheitstabelle

Eingang1	Eingang2	Ausgang
0	0	0
0	1	1
1	0	1
1	1	0

Für die hier gezeigten Bilder wurde ein Javaapplet[1] verwendet. Das Netz ist so einfach wie möglich gewählt, um das Verfahren des BPA nachverfolgen zu können. Es sei

[1]Das Applet steht kostenlos im Internet [17] bereit und bietet dem Leser die Möglichkeit selbst zu experimentieren. Für das gezeigte Beispiel wurde die Version 1.2 verwendet

aber darauf hingewiesen, dass dieses eindimensionale Netz das Problem des XOR nur aufgrund der zusätzlichen Biasneuronen erlernen konnte. Im Anschluss ist ein zweidimensionales Netz gezeigt, welches in der Lage ist dieses Problem ohne Bias zu lösen.

In den gezeigten Netzen, in Abbildung 3.12, findet der Informationsfluss von oben nach unten statt, die Rechtecke sind die Neuronen des Netzes, die Zahlen innerhalb und die Füllhöhe der Rechtecke stehen für den Ausgang des jeweiligen Neurons. Die Größe der Gewichte wird durch die farbigen Verbindungen dargestellt, siehe Abb. 3.11. Die Zahlen an den Verbindungen der Neuronen sind die Gewichte w_{ji}. Die Aktivierungsfunktion ist eine nicht symmetrische Sigmoidfunktion (siehe auch Abb. 3.4)

$$g(x) = \frac{1}{1 + e^{-x}}, \tag{3.16}$$

die keine negativen Werte annehmen kann. Die Elemente des Netzes, die für die aktuelle Berechnung benutzt werden, sind blau unterlegt.

Zahlen, die in Klammern innerhalb der Neuronen zu sehen sind, entsprechen den berechneten Fehlern δ_j, der jeweiligen Neuronen, während eines Lernschrittes. Als Fehlerfunktion wird, wie auch in Formel 3.3, der quadratische Fehler des Ausgangswertes benutzt:

$$E^n = \frac{1}{2}(d_n - y_n)^2 \tag{3.17}$$

Hier wurde jedoch berücksichtigt, dass sich dieser Wert nur auf einen Ausgang und nur ein Datenset beschränkt. Analog zu Formel 3.3 ist d_n der Istwert des Netzes und y_n der Sollwert des Datensatzes n. Der Faktor $1/2$ erleichtert Berechnungen mit der Ableitung. Die Lernrate η beträgt in diesen Beispielen 0.35.

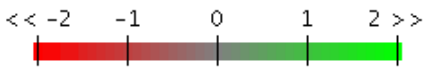
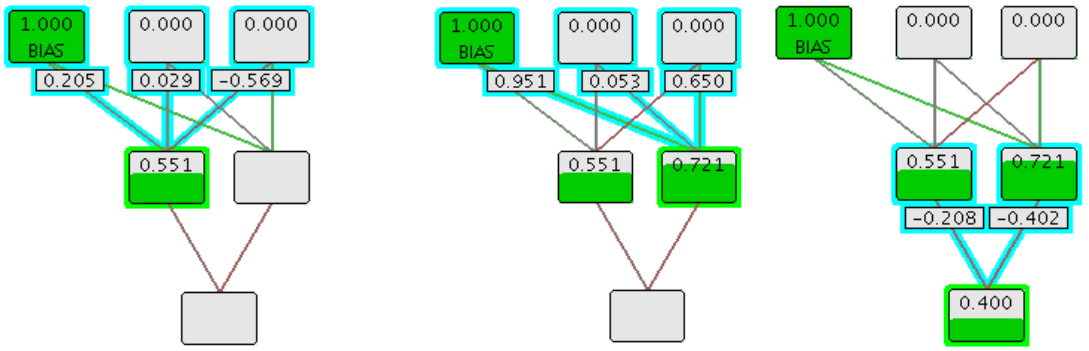


Abbildung 3.11: Farbcode der Gewichte in den gezeigten Beispielen in Kapitel 3.3.2.

Das in Abbildung 3.12 gezeigte Datenset an den Eingängen des Neuronalen Netzes entspricht der ersten Zeile in Tabelle 3.1. Dieses wird vorwärts durch das Netz propagiert, wie in den Abbildungen in 3.12 dargestellt. Die Werte der Neuronen der Eingangsschicht werden mit den Gewichten multipliziert und entsprechend Formel 3.4 summiert. Dieser Wert wird an die Aktivierungsfunktion 3.16 gegeben, was den Ausgangswert der Neuronen in der versteckten Schicht ergibt. Diese Berechnung setzt sich nun für jedes Neuron in jeder Schicht bis zum Ausgangsneuron fort. Da es sich um ein ungelerntes Netz handelt, stimmt der Ausgangswert des Netzes nicht mit dem gewünschten Ausgangswert in Tabelle 3.1 überein. Das Prinzip ist für ein trianiertes Netz absolut identisch.



(a) (b) (c)

Abbildung 3.12: Berechnung des Ausgangssignals eines Netzes. Der erste und notwendige Schritt um das Netz trainieren zu können. Die Werte werden für jedes Neuron berechnet bis das Ausgangsneuron erreicht ist.

Nun wird der Fehler mit Hilfe der Formel 3.11 berechnet und in den Bildern in Klammern dargestellt. Mit den Formeln 3.14 und 3.10 werden nun die Gewichte zu den vorherigen Neuronen verändert. Die Bilder in 3.13 geben einen Überblick über die Zahlenwerte und die Reihenfolge der Berechnungen. Wurde der Fehler bis zu den Eingangsneuronen zurückverfolgt, ist der erste Lernschritt abgeschlossen und das nächste Datenset wird geladen. Die Prozedur wiederholt sich für jeden Datensatz in den Lerndaten. Wurden alle Lerndaten einmal verwendet, spricht man von einer Epoche.

In der Abbildung 3.14 sind Netze abgebildet, die das Verhalten des XOR erlernt haben. Die Umgebungsvariablen zum Trainieren waren bei allen drei Netzen gleich und dennoch sind drei verschiedene Lösungen gefunden worden, wie an den unterschiedlichen Gewichten zu sehen ist. Das bedeutet, dass es offenbar mehrere Minima der Fehlerfunktion gibt, die das Problem lösen. Bei komplexeren Aufgabenstellungen ist es daher unbedingt notwendig, mehrere Netze mit den gleichen Parametern zu trainieren, um sicher zu sein, dass eine gute Lösung gefunden wurde.

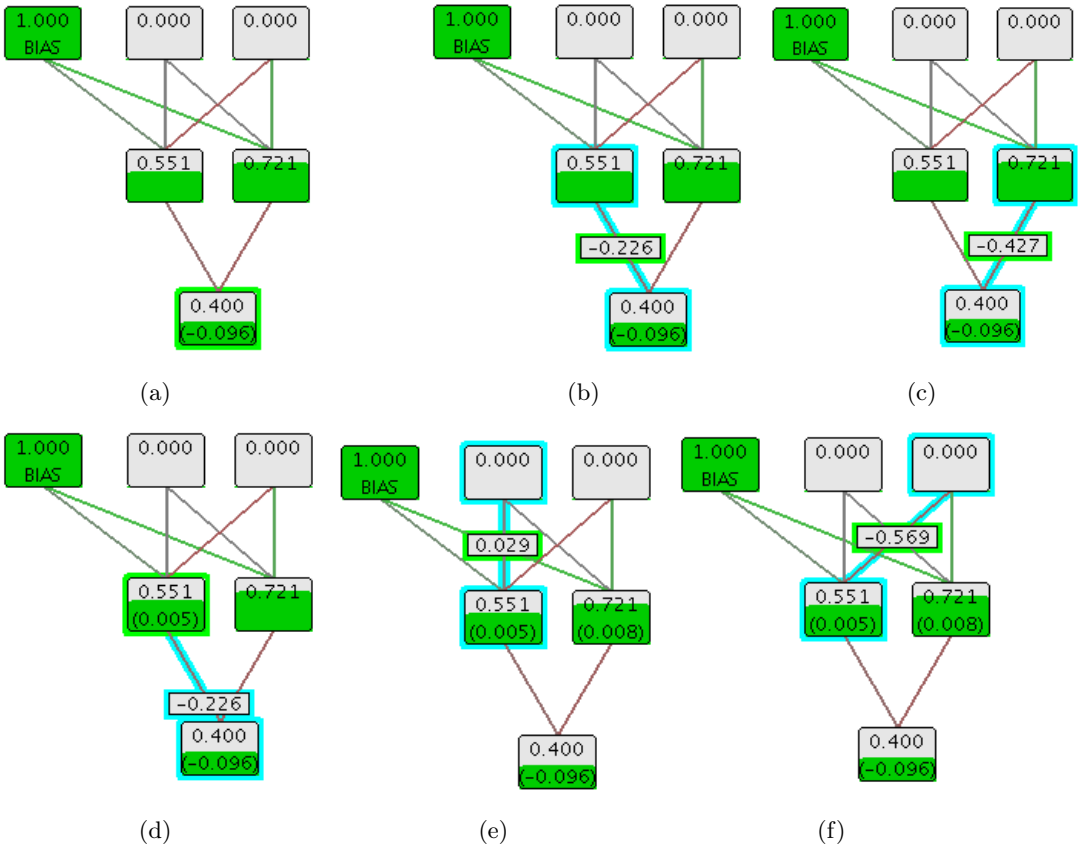


Abbildung 3.13: Der Backpropagationalgorithmus verändert nun die Gewichte in umgekehrter Reihenfolge zurück zu den Eingansneuronen. Wurde das letzte Gewicht and den Eingangsneuronen geändert, ist ein Lernschritt abgeschlossen und ein neues Beispiel kann aus den Daten geladen werden.

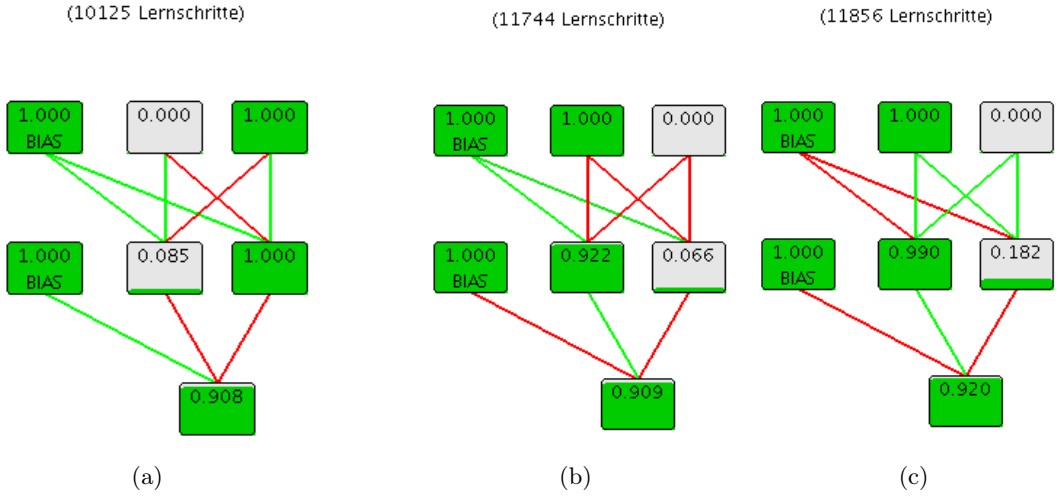


Abbildung 3.14: Fertig trainierte Netze die das Verhalten des XOR nachbilden können. Man erkennt die unterschiedlichen Gewichtungen an den Farben. Jedes Netz durchlief mehrere 1000 Lernschritte bis es in der Lage war, die Funktion des XOR nachzubilden.

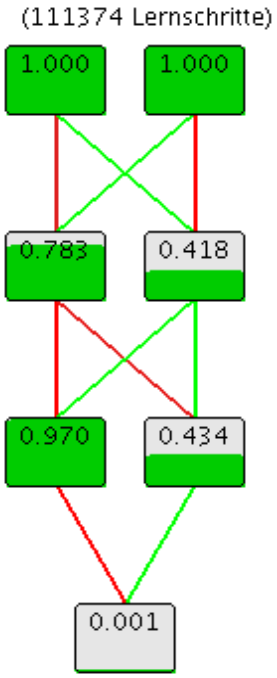


Abbildung 3.15: Ohne Bias sind zwei versteckte Schichten nötig um ein XOR zu simulieren.

Kapitel 4

Untergrundunterdrückung mit künstlichen Neuronalen Netzen

Diese Kapitel beschreibt die Vorgehensweise zur Konfiguration und zum Training eines künstlichen Neuronalen Netzes, das die Unterscheidung zwischen von oben und von unten kommenden Ereignissen ermöglicht. Die Ergebnisse werden in Kapitel 5 gezeigt. Die Leistungsfähigkeit der KNN wurde mit Hilfe von Kennlinien, wie sie in Abbildung 4.7 zu sehen sind, beurteilt. Die Effizienz berechnet sich zu:

$$\text{Eff} = \frac{S_{\text{gefunden}}}{S_{\text{vorhanden}}} \tag{4.1}$$

Die Untergrundunterdrückung wie folgt:

$$\text{Bck} = \frac{U_{\text{gefunden}}}{U_{\text{vorhanden}}} \tag{4.2}$$

Ein Signal wird als gefunden gezählt, wenn das KNN es als solches identifiziert hat und es sich tatsächlich um ein Signal handelt (S_{gefunden}). Untergrund gilt als zurück gewiesen, wenn ihn das KNN nicht als Signal klassifiziert hat und es sich tatsächlich um Untergrund handelt (U_{gefunden}). Vorhandener Untergrund ($U_{\text{vorhanden}}$) und vorhandene Signale ($S_{\text{vorhanden}}$) beziehen sich dabei auf den kompletten Untergrund bzw. auf alle Signale in den Testdaten. Das Ziel war es eine möglichst gute Effizienz bei gleichzeitig möglichst guter Untergrundunterdrückung zu gewährleisten.

Da ein KNN keinen binären sondern einen kontinuierlichen Ausgang besitzt, der alle Werte entsprechend der Aktivierungsfunktion des Ausgangsneurons annehmen kann (hier -1 bis 1, da als Aktivierungsfunktion der tanh verwendet wird), kann die Effizienz und die Untergrundunterdrückung durch dieses Kriterium beeinflusst werden. Die einzelnen Punkte in den Kennlinien entstehen durch Verschieben dieser Grenze.

4.1 Datengewinnung, Metadaten

Bevor nun ein KNN trainiert werden kann, müssen möglichst viele Informationen zusammen getragen werden, die das Problem der Klassifikation möglichst gut beschreiben. Wie in Kapitel 3.3 beschrieben, ist es dafür nicht nötig eine exakte mathematische Beschreibung liefern zu können. Es genügt, wenn in den vorhanden Eingangsparametern Informationen enthalten sind, welche es dem KNN erlauben, sich an die Problemstellung zu adaptieren. Die fundamentalsten Daten, die bei ANTARES anfallen, sind Informationen über den Ort, die Zeit und die Amplitude von Hits an den einzelnen optischen Modulen. Damit lässt sich aber kein KNN trainieren, da im Prinzip für jedes optische Modul und für jede Information, wie Zeit und Amplitude, ein eigener Eingang existieren müsste und sich aus diesen Daten noch keine übergeordnete Logik, für die Unterscheidung von Signal und Untergrund, ableiten lässt. Künstliche Neuronale Netze benötigen Metadaten als Informationsträger, welche verschiedene Eigenschaften für ein Ereignis bereitstellen. Dabei werden die vorhandenen Informationen zusammengetragen und auf verschiedene Arten miteinander verknüpft. Metadaten können z.B. die Anzahl der angeregten optischen Module oder auch die durchschnittlich deponierte Ladung bei einem Ereignis sein. Der Fantasie sind hier buchstäblich keine Grenzen gesetzt und so bildeten über 60 Parameter die Basis für weitere Untersuchungen in dieser Arbeit. Ebenso wichtig wie die Daten, die an das KNN weiter gegeben werden, ist der Ort in der Analysekette, an dem sie gewonnen werden. Es steht zu erwarten, dass die aus den Rohdaten gewonnenen Informationen zu sehr durch Untergrundrauschen, verursacht z.B. durch den K40-Zerfall, belastet sind. Da beim ANTARES-Experiment die Datennahme von Triggern bestimmt wird, liegt es nahe, auch die Metadaten erst nach diesen Triggern zu erzeugen und den Einfluss von rauschendem Untergrund so zu minimieren. Alle Daten in dieser Arbeit werden nach den L1 und L2 Triggern gewonnen, Abb. 4.1 zeigt das Prinzip der Funktionskette. Der Anhang A zeigt das Pythonscript, mit dessen Hilfe die Metadaten berechnet wurden.

Als L2 Trigger wurde in der Softwareumgebung Seatray das Modul I3AntTriggerSimulator verwendet, in dem nur der 3N Trigger aktiviert wurde. Dieser Trigger läuft in der Regel auch bei der Onlinedatennahme. Da die Trigger die Basis der Datengewinnung bilden, ist es besonders bei Vergleichen zwischen MC- und Messdaten wichtig, durch Anwendung gleicher Trigger, identische Voraussetzungen zu schaffen. Die KNN, die im Laufe dieser Arbeit trainiert wurden, sind demnach nur mit diesem Trigger sinnvoll zu verwenden. Will man den Trigger ändern, ist es notwendig ein neues KNN zu trainieren. Sollte das der Fall sein, kann man sich auf die Ergebnisse dieser Arbeit stützen und die Auswahl der Parameter und der Netztopologie stark verkürzen.

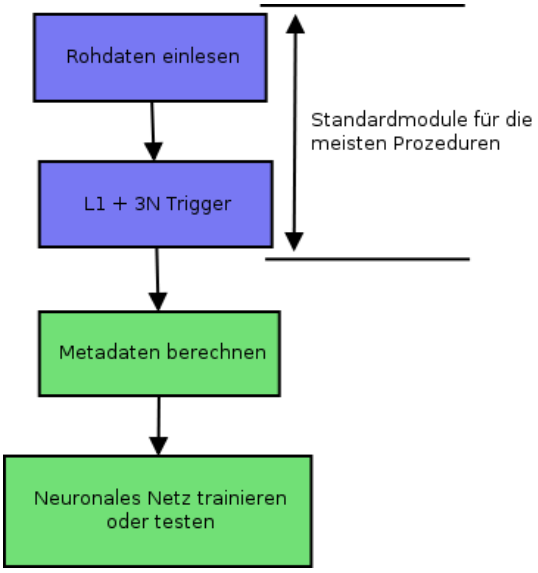


Abbildung 4.1: Prinzip der Datengewinnung zum Trainieren und Testen von Neuronalen Netzen. Als Ausgangspunkt diente das Framework Seatray. Die Metadaten werden direkt nach den L1- und L2-Triggern gewonnen und stützen sich somit nur auf fundamentalste Daten.

4.2 Auswahlkriterien geeigneter Eingangsparameter

Von den erwähnten 60 Parametern, welche zu Beginn zur Verfügung standen, um KNN zu trainieren, mussten viele wieder verworfen werden. Es ist wichtig nur solche Daten zu verwenden und an das KNN zu geben, welche direkten Einfluss auf die Klassifizierung haben. Jeder zusätzliche Eingang beeinflusst das KNN und verschlechtert die Ergebnisse. Werte, die den Detektor beispielsweise in unterschiedlich große Teile zerlegen, aber im Grunde die gleichen Werte berechnen, beinhalten mit hoher Wahrscheinlichkeit ähnliche Informationen. Es sollten also nicht alle benutzt werden, sondern nur die, die den größten Einfluss auf das Ergebnis haben. Im Folgenden werden die Verfahren beschrieben, mit deren Hilfe geeignete Parameter identifiziert wurden. Unabhängig von der Methode, die verwendet wird, um geeignete Parameter zu erkennen, ist es immer ratsam das Ergebnis zu testen. Bei 60 Parametern ist es natürlich kaum möglich jede Kombination zu trainieren und zu testen. Wurde aber erst einmal eine kleine Auswahl getroffen, ist es leichter einige viel versprechende Parameter hinzuzufügen oder wegzulassen und das Netz neu zu trainieren.

Aufgrund der Tatsache, dass die Gewichte eines neuen Netzes mit Zufallszahlen belegt werden und das Netz bei mehreren Trainingseinheiten unterschiedlich gut lernt, wie in 3.3 beschrieben, wurden nach jeder Änderung der Parameter immer verschiedene Netzkonfigurationen verwendet, welche immer mehrmals trainiert wurden, um ein aussagekräftiges Ergebnis über die Änderung zu erhalten.

- Zuerst wurden einzelne Parameter einfach weggelassen, die wie im obigen Beispiel beschrieben, offensichtlich voneinander abhängen.
- In einem zweiten Schritt wurde ein Netz trainiert und die Gewichte, aus der Eingangsschicht hin zu den Neuronen der ersten versteckten Schicht, betrachtet. Sind die Gewichte eines Parameters sehr klein, so kann man annehmen, dass dieser Eingang nicht von großer Bedeutung für das Problem ist. Natürlich schließt das eine Überprüfung nicht aus, bei der der so gefundene Eingang, wie im ersten Schritt weggelassen wird und ein neues Netz trainiert und getestet werden muss. In Abbildung 4.2 ist exemplarisch der erste Eingang durch geringe Gewichte zur nächsten Schicht verbunden. Je höher das Gewicht, desto dicker ist der Pfeil, der es repräsentiert.

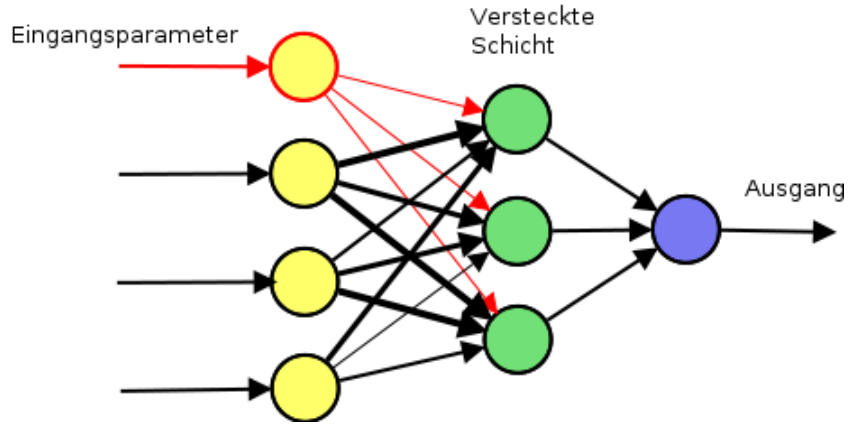
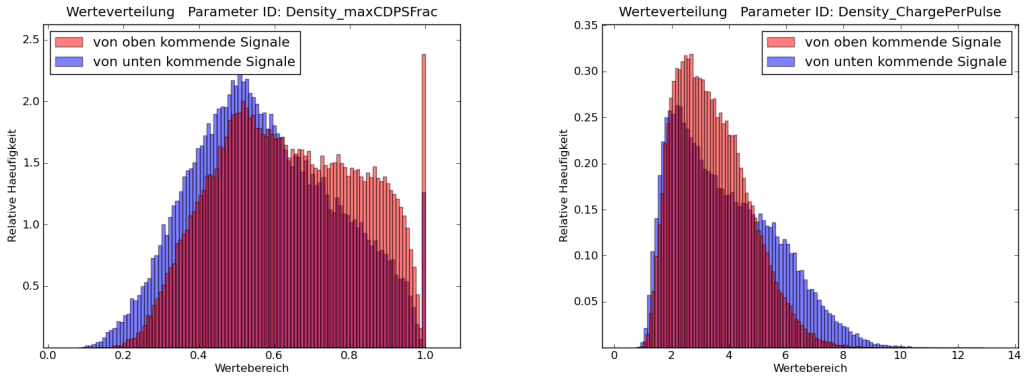


Abbildung 4.2: Die Informationen des ersten Eingangs dieses Neuronalen Netzes werden offenbar kaum verwendet. Je dicker der Pfeil, desto stärker die Verknüpfung. Im nächsten Schritt sollte dieser Eingang nicht mehr verwendet werden. Eine endgültige Entscheidung kann aber nur durch einen Test getroffen werden.

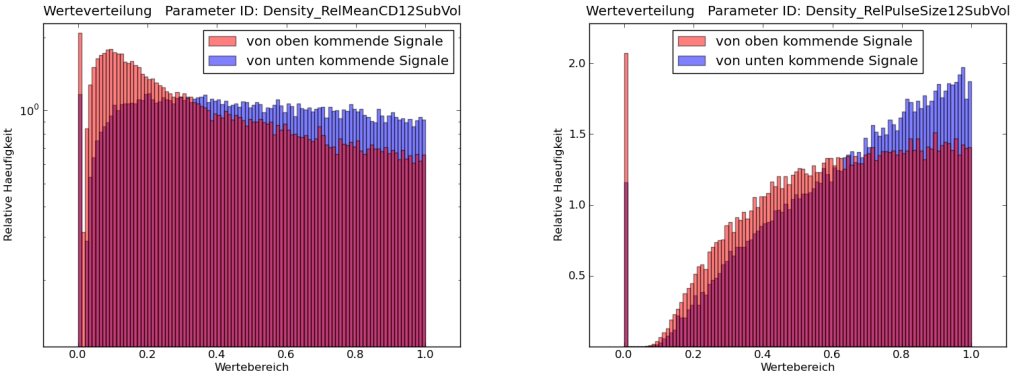
- Auch der Wertebereich einzelner Parameter lässt Rückschlüsse auf den Nutzen des Eingangs zu. Besitzt ein Parameter einen größeren Wertebereich oder einen gänzlich anderen, je nachdem welcher Ausgangswert erwartet wird, kann auch dieser Parameter wertvolle Informationen für das gewünschte Ergebnis beitragen. Die Histogramme in 4.3 bis 4.5 vergleichen die Wertebereiche einiger Parameter für von oben in den Detektor eintretende Untergrundsignale und von unten eintretende Neutrino-induzierte Myonen. Aufgrund der Form und der Breite der Verteilung konnten weitere Eingänge eliminiert werden. Dabei wurde in erster Linie auf Unterschiede geachtet, die darauf hindeuten, dass Informationen in den Werten enthalten sind, die das Netz erkennen kann und die Klassifizierung ermöglichen.

In den Histogrammen 4.3(a) bis 4.5(c) sind die Parameter zu sehen, die für das



(a) (b)

Abbildung 4.3: Vergleich der Werteverteilungen einiger Parameter für Untergrund und Signal. Hier sind einige Beispiele abgebildet die sich als Informationsträger für KNN eignen.

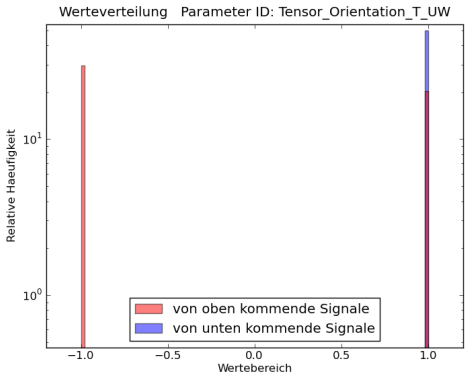


(a) (b)

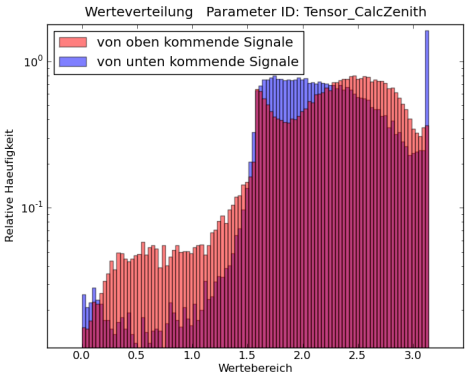
Abbildung 4.4: Fortsetzung: Vergleich der Werteverteilungen einiger Parameter für Untergrund und Signal. Hier sind einige Beispiele abgebildet die sich als Informationsträger für KNN eignen.

endgültige Netz benutzt wurden. Das Hauptaugenmerk wurde dabei auf die Form der Verteilung gelegt und nicht unbedingt auf unterschiedliche Wertebereiche, wie sie 4.6(b) und 4.6(c) aufweisen. Die Verwendung dieser und ähnlich gestalteter Parameter führte nach Tests zu keiner Verbesserung, sondern zu einer Verschlechterung der Ergebnisse. Neben den genannten Beispielen gibt es auch viele Parameter, die keinen Unterschied in ihrer Werteverteilung aufweisen, wie im Beispiel von 4.6(a) zu sehen ist. Auch hier konnte die Verwendung dieser Parameter nicht zu Verbesserung des KNN beitragen.

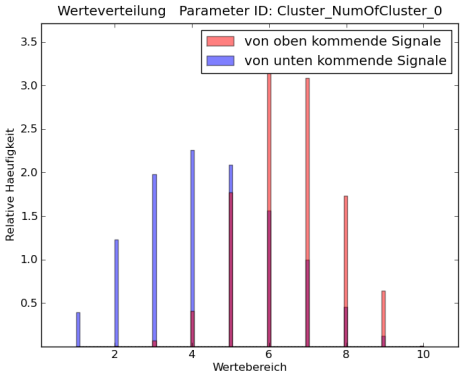
Diese Auswahlverfahren erlaubten es, die Adaption des KNN immer weiter zu verbessern. Die Kennlinien in Abb. 4.7 zeigen die Erkennungsleistung verschiedener Netze und Konfigurationen nach unterschiedlichen Änderungen. Zu Beginn waren die Ergeb-



(a)

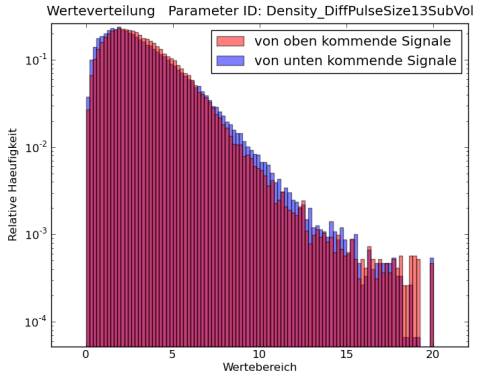


(b)

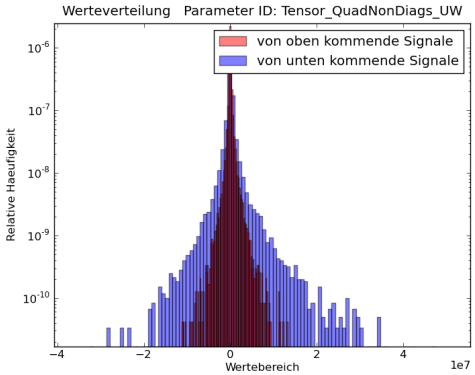


(c)

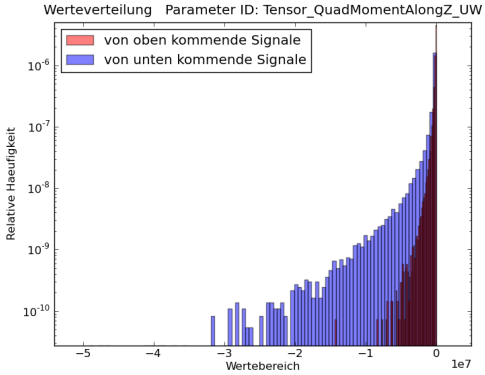
Abbildung 4.5: Fortsetzung: Vergleich der Werteverteilungen einiger Parameter für Untergrund und Signal. Hier sind einige Beispiele abgebildet die sich als Informationsträger für KNN eignen.



(a)



(b)



(c)

Abbildung 4.6: Vergleich der Werteverteilungen einiger Parameter für Untergrund und Signal. Auch wenn 4.6(b) und 4.6(c) der hier abgebildeten Beispiele vielversprechend aussehen, stellte sich heraus, dass sie nicht als Informationsträger für die Klassifizierung durch KNN geeignet sind. Im Fall von 4.6(a) lässt bereits die Verteilung wegen ihrer Übereinstimmung bereits annehmen, dass ein KNN daraus keine Rückschlüsse ziehen kann.

nisse noch nicht besonders zufriedenstellend. Durch oben beschriebene Auswahlkriterien konnte das Verhalten der KNN aber, wie man sieht, deutlich verbessert werden und die Effizienz und die Untergrundunterdrückung gesteigert werden. In Kapitel 4.3 wird die Berechnung der Parameter beschrieben, die die genannten Auswahlkriterien erfüllten.

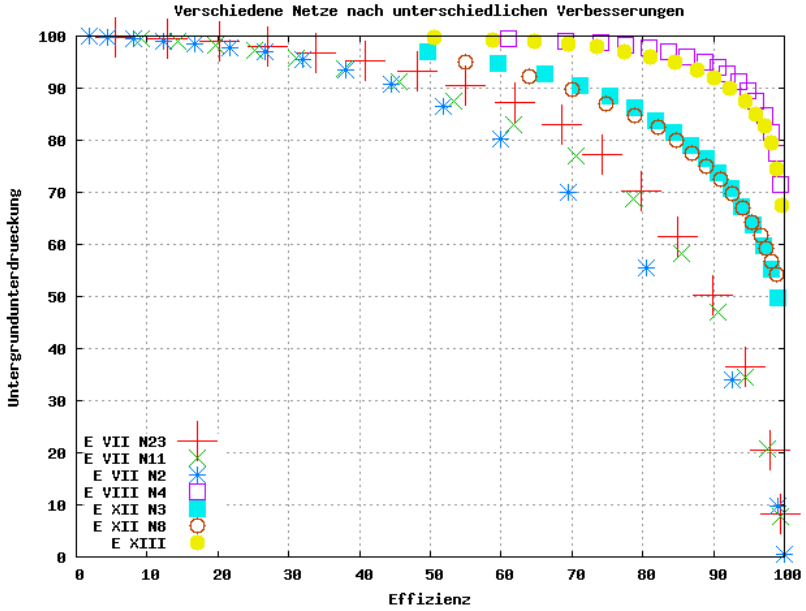


Abbildung 4.7: Verschiedene Netze zu verschiedenen Zeitpunkten der Optimierung. Durch Änderung der Topologie und verwendeter Parameter, konnte das Ergebnis weiter verbessert werden. Je größer die Fläche unter den Messpunkten, desto besser konnte das Netz die Aufgabe der Klassifizierung erlernen.

4.3 Ausgewählte Parameter

Die Suche nach den passenden Parametern war ein wesentlicher Bestandteil dieser Arbeit. Aus diesem Grund werden die Variablen, die durch die Verfahren in Kapitel 4.2 ausgewählt wurden, hier beschrieben. Einige Parameter setzen sich aus einer Kette von Berechnungen zusammen.

- P_1 (chargePerPulse): Die Summe der Amplituden aller getriggerten Hits A_{all} geteilt durch die Anzahl der getriggerten Hits N_{all}

$$P_1 = \frac{A_{all}}{N_{all}} \tag{4.3}$$

- P_2 (Lines): Die Anzahl der Strings, die wenigstens einen getriggerten Hit aufweisen.

- P_3 (storeyMaxFrac): Die Dauer des Events t_{event}, berechnet aus dem ersten und letzten getriggerten Hit, wird durch 100ns geteilt. Somit entsteht eine bestimmte Anzahl an Klassen. Alle getriggerten Hits werden nun entsprechend ihrer Zeitpunkte in diese Klassen eingeteilt und wie in einem Histogramm gezählt. Die Anzahl der Hits in der Klasse mit den meisten Hits N_{tmax} wird durch die Anzahl der im gesamten Detektor aktivierten Stockwerke $N_{storeys}$ geteilt:

$$P_3 = \frac{N_{tmax}}{N_{storeys}} \tag{4.4}$$

- P_4 (maxCDPSFrac): Der Detektor wird in 20 Einzelteile zerlegt. Für jedes dieser Teile wird der Wert *cd* berechnet:

$$cd_{sub} = \frac{\sum_{hit}^{allhits} A_{sub,hit}}{N_{sub} \cdot t_{event} \cdot 20} \tag{4.5}$$

  Wobei $A_{sub,hit}$ die Amplituden der Hits in den jeweiligen Einzelteilen des Detektors, N_{sub} die Anzahl der aktivierten optischen Module jedes Teils und t_{event} die Dauer des Events ist. Das Maximum aller berechneter Werte *cd* wird anschließend durch die Summe aller *cd* geteilt:

$$P_4 = \frac{{cd_{sub}}^{max}}{\sum_{sub} cd_{sub}} \tag{4.6}$$

- P_5 (relMeanCD12SubVol): Wie bei maxCDPSFrac wird auch hier der Wert *cd* für insgesamt 20 Einzelteile berechnet. In diesem Fall wird der zweitgrößte Wert der *cd*-Werte durch den größten *cd*-Wert geteilt. Um dies zu verdeutlichen wird die Menge C eingeführt, die die Werte *cd* aller Einzelteile des Detektors enthält. Die Menge C hat demnach 20 Elemente. Des weiteren soll entsprechend zu maxCDPSFrac gelten: ${cd_{sub}}^{max} = max\{C\}$ und ${cd2_{sub}}^{max} = max\{C \backslash \{{cd_{sub}}^{max}\}\}$. Dann ergibt sich dieser Parameter zu:

$$P_5 = \frac{{cd_{sub}}^{max}}{{cd2_{sub}}^{max}} \tag{4.7}$$

- P_6 (relPulseSize12SubVol): Die Berechnung dieses Parameters ist identisch zu relMeanCD12SubVol, nur die *cd*-Werte werden ersetzt durch:

$$cg_{sub} = \frac{\sum_{hit}^{allhits} A_{sub,hit}}{N_{sub}} \tag{4.8}$$

- P_7 (orientation_T_UW): Dieser Wert besitzt die Werte +1 oder -1 entsprechend der vertikalen Orientierung des Vektors $\vec{o}$:

$$\vec{o} = \vec{v}_t - \vec{v}_0 \tag{4.9}$$

$\vec{v}_0$ ist der Schwerpunkt aller getriggerter Hits:

$$\vec{v}_0 = \frac{\sum_{hit}^{allhits} \vec{r}_{hit}}{N_{all}} \tag{4.10}$$

wobei $\vec{r}_{hit}$ die Positionen der optischen Module sind, von denen ein getriggertes Signal stammt. $\vec{v}_t$ berechnet sich wie $\vec{v}_0$, es werden lediglich die Positionen $\vec{r}_{hit}$ der Hits im Detektor vor der Summation durch den Zeitfaktor: $t_v = t_{hit} - t_0$ gewichtet, wobei t_0 der Zeitpunkt des ersten getriggerten Hits in dem Event ist:

$$\vec{v}_t = \frac{\sum_{hit}^{allhits} t_v \cdot \vec{r}_{hit}}{N_{all}} \tag{4.11}$$

- P_8 (calcZenith): Mit Hilfe des Trägheitstensors wird hier die Richtung eines Vektors berechnet, der die Amplituden der getriggerten Hits berücksichtigt. Identisch zu $\vec{v}_t$ bei der Berechnung von orientation_T_UW wird hier $\vec{v}_a$ berechnet:

$$\vec{v}_a = \frac{\sum_{hit}^{allhits} a_{hit} \cdot \vec{r}_{hit}}{N_{all}} \tag{4.12}$$

Wobei a_{hit} die Amplituden der einzelnen Hits sind. Aus dem Vektor $\vec{V}_{hit} = \vec{r}_{hit} - \vec{v}_a \cdot A_{all}$ wird der Trägheitstensor I über alle Hits berechnet:

$$I = \sum_{hit}^{allhits} A_{hit} \cdot \begin{pmatrix} y_{hit}^2 + z_{hit}^2 & -x_{hit}y_{hit} & -x_{hit}z_{hit} \\ -y_{hit}x_{hit} & x_{hit}^2 + z_{hit}^2 & -y_{hit}z_{hit} \\ -z_{hit}x_{hit} & -z_{hit}y_{hit} & x_{hit}^2 + y_{hit}^2 \end{pmatrix} \tag{4.13}$$

Wobei x, y und z die Komponenten von $\vec{V}_{hit}$ sind. Der Eigenvektor, der zum kleinsten Eigenwert gehört, wird normiert. Sein Winkel zur z-Achse ergibt calcZenith.

- P_9 (numOfCluster0): Diese Variable entspricht der Anzahl der Cluster, die erstellt wurden. Ausschlaggebend ist die Bedingung unter der Hits zu einem Cluster zusammengeschlossen werden. In diesem Fall müssen die Hits i und j folgender Bedingung genügen, um zu einem Cluster zu gehören:

$$|t_i - t_j| \leqq 1.38 \frac{|\vec{r}_i - \vec{r}_j|}{c} \tag{4.14}$$

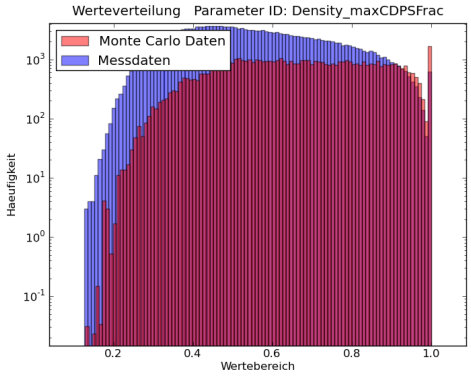
Wobei $\vec{r_x}$ die Position der aktivierten optischen Module und t_x die Zeitpunkte der Hits sind und c die Lichtgeschwindigkeit in Vakuum ist. Ein Hit wird zu einem bestehenden Cluster hinzugefügt, wenn er diese Bedingung zu allen Hits, die sich bereits in dem Cluster befinden, erfüllt.

4.4 Vergleich Mess- und MC-Daten

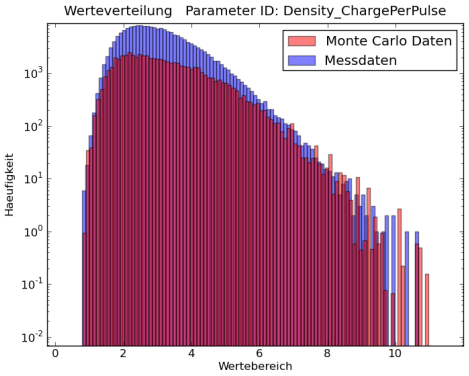
Nach den in Kapitel 4.2 beschriebenen Schritten und den darauf folgenden Trainingszyklen konnte ein Set von insgesamt 9 Parametern ausgemacht werden, mit denen die Netze das Problem am besten adaptierten. Da alle genannten Auswahlverfahren aber immer auf Monte Carlo Daten zurückgreifen und das KNN natürlich mit Monte Carlo Daten angelernt und getestet werden musste, ist es notwendig, die Parameterräume der simulierten Daten mit den Messdaten zu vergleichen. Denn nur bei möglichst guter Übereinstimmung lässt sich das Verhalten des KNN, auf Messdaten anwenden. Die Histogramme in 4.8 bis 4.11 vergleichen die Werteverteilungen der verwendeten Parameter. Wie man, sieht stimmen die simulierten Daten nicht immer mit den Messdaten überein. Es wäre demnach notwendig, die Simulation zu verbessern. Zum Zeitpunkt dieser Arbeit waren dies aber die neuesten Simulationsdaten, die zur Verfügung standen. Ein weiterer Vergleich der Wertebereiche wird notwendig, wenn die Simulation verbessert wurde. Die Basis der Monte Carlo Daten, die für diesen Vergleich verwendet wurden, bildeten ausschließlich Untergrundereignisse (unterlegt mit einem Rauschen von 60kHz), in der Annahme, dass auch in den Messdaten, in erster Linie, Untergrund aufgenommen wurde. Dabei wurde beachtet, dass die Ereignisse aus den Monte Carlo Daten mit einem Faktor versehen sind, der diese entsprechend ihrer Verteilung in den Messdaten gewichtet. Die Zeitspanne, die in dieser Gewichtung mit berücksichtigt wird, ergab sich aus der Aufnahmedauer der Messdaten. Bei den Messdaten handelt es sich um alle 'Golden Runs' aus Dezember 2008 (eine genaue Liste findet sich im Anhang B). Bei den gezeigten Verteilungen ist in erster Linie die Form und der Wertebereich von entscheidender Bedeutung. Diese sollten für Monte-Carlo- und Messdaten nicht zu sehr voneinander abweichen, da das Netz sonst an falsche Gegebenheiten herangeführt würde. Die Differenz der Höhe der Histogramme deutet zwar auf einen Unterschied zwischen Monte-Carlo Daten und Messdaten hin, doch ist das für das Training des KNN von geringerer Bedeutung, da jedes Ereignis für sich selbst trainiert und getestet wird.

4.5 Zusammenstellung der Lern- und Testdaten

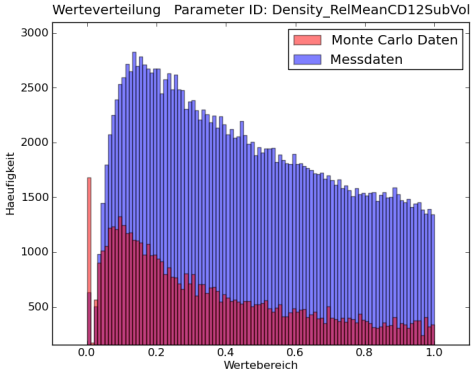
Die Trainingsdaten sind ein wesentlicher Bestandteil der Kette bei der Erstellung eines KNN. Dabei spielt besonders die Vollständigkeit eine wichtige Rolle. Denn nur was das KNN gelernt hat, kann es auch wieder anwenden. Zwar besitzen KNN die Fähigkeit auch auf ungelernte Gegebenheiten zu reagieren, doch diese Fähigkeit ist eher begrenzt und das tatsächliche Verhalten ist kaum vorhersagbar. Daher sollte sichergestellt sein, dass möglichst alle Fälle bereits in den Lerndaten vorhanden sind. Das gewinnt besonders dann an Bedeutung, wenn das Netz tatsächlich eingesetzt wird, nachdem seine Fähig-



(a)

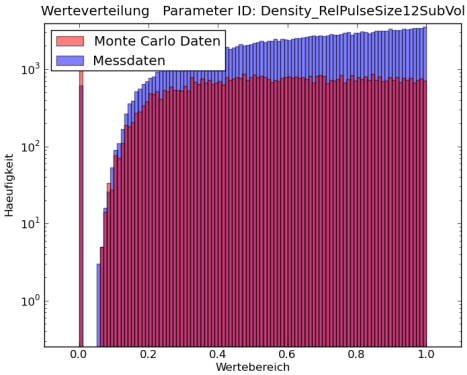


(b)

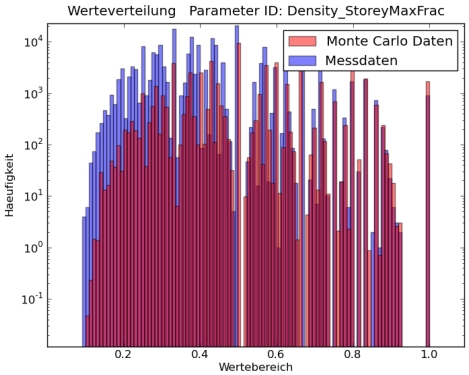


(c)

Abbildung 4.8: Werteverteilungen der verwendeten Parameter im Vergleich von Monte-Carlo-Simulation und Messdaten.

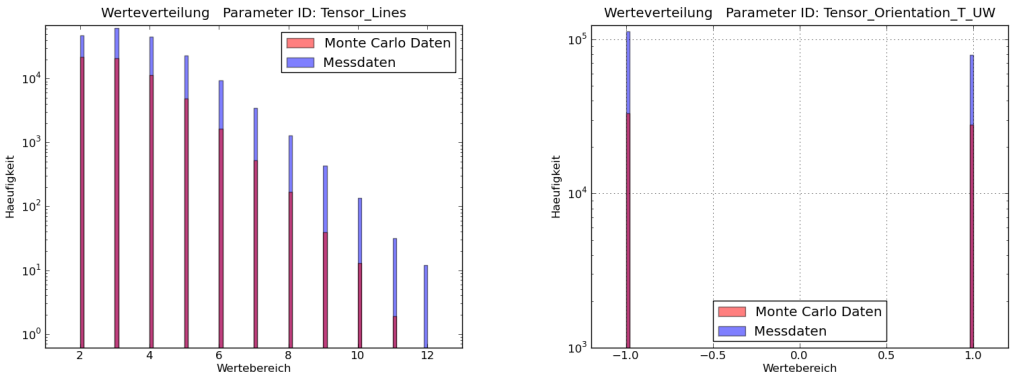


(a)



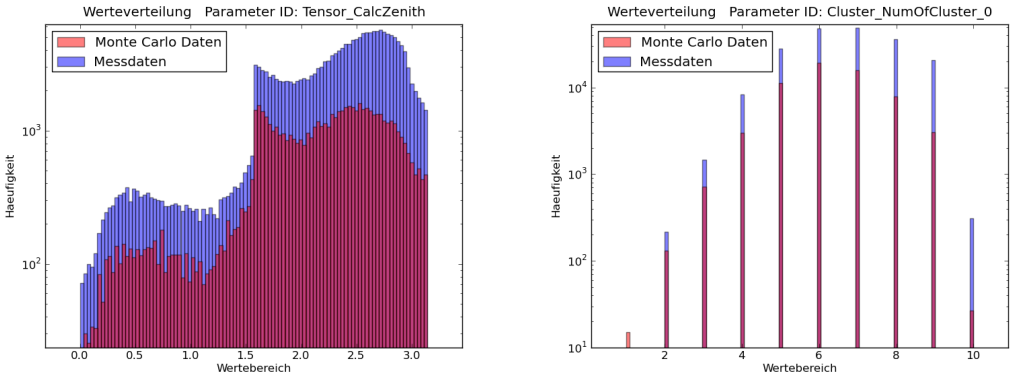
(b)

Abbildung 4.9: Fortsetzung: Werteverteilungen der verwendeten Parameter im Vergleich von Monte-Carlo-Simulation und Messdaten (Fortsetzung).



(a) (b)

Abbildung 4.10: Fortsetzung: Werteverteilungen der verwendeten Parameter im Vergleich von Monte-Carlo-Simulation und Messdaten (Fortsetzung).



(a) (b)

Abbildung 4.11: Fortsetzung: Werteverteilungen der verwendeten Parameter im Vergleich von Monte-Carlo-Simulation und Messdaten (Fortsetzung).

keit, das gegebene Problem zu erlernen, ausreichend getestet wurde. Als Lern- und Testdaten wurden MC-Daten verwendet, die die Information für den richtigen Ausgangswert enthalten, was für die Lern- und Testphase nötig ist. Die verwendeten Lerndaten sind nach unterschiedlichen Winkeln, Energien und Primärteilchen (Fe, He, p, N, Mg) aufgeteilt. Das Gleiche gilt für die simulierten Myon-Neutrinos. Auch hier wurden Daten mit unterschiedlicher Energie, mit Neutrino- und Antineutrino-Ereignissen verwendet. Dabei wurde darauf geachtet, dass die Anzahl der Ereignisse für die einzelnen Parameter wie Primärteilchen, Richtung und Energie in ihrer Summe möglichst gut übereinstimmen, was nicht immer möglich war, da es für manche Energien und Teilchen deutlich weniger Daten gab als für andere. Aus diesen Datensätzen entstand anschließend ein großer Datensatz, mit einer Verteilung der atmosphärischen Myonen zu Neutrinos, von etwa 50:50. Dieses Verhältnis entspricht zwar in keiner Weise der Realität, doch sollte das KNN auf keinen Fall an ein Übergewicht eines bestimmten Ereignisses gewöhnt werden. Nur so ist sicherzustellen, dass das KNN jedes Ereignis möglichst unabhängig betrachtet. Dieser Datensatz wurde anschließend aufgeteilt, um einen Lern- und einen Test-Datensatz, im Größenverhältnis von etwa 1:5, zu bekommen. Abhängig von der Art und Weise wie der Datensatz aus den Monte Carlo Daten zusammengestellt wird, ist es unter Umständen nötig, die Reihenfolge der Events im Datensatz von einem Programm zufällig mischen zu lassen (siehe dazu auch Kapitel 3.3.1). Aus diesen Daten entstanden ca. 2 Millionen Ereignisse. Eine zweite Testdatei beinhaltete dieselbe Verteilung wie die erste, mit dem Unterschied, dass den Monte-Carlo-Daten ein Rauschen mit 60kHz aufmoduliert wurde, um der Realität näher zu kommen.

Bevor die Daten an das KNN weiter gegeben werden können, müssen deren Wertebereiche noch auf [-1, 1] umgerechnet werden. Das gilt natürlich auch für den Ausgang, der im Falle eines atmosphärischen Myons -1 (klassifiziert als Untergrund), und für ein (Anti-)Neutrino-Event +1 (klassifiziert als Signal) ausgeben sollte. Im Laufe der Arbeit wurde versucht die Werte der Eingänge und des Ausgangs unabhängig voneinander, einmal auf den Wertebereich [0, 1] und [-1, 1] einzuschränken. Die sich daraus ergebenden 4 Möglichkeiten, wurden alle getestet. Die Abbildungen 4.12 zeigen wie unterschiedlich KNN trotz gleicher Topologie auf diese Änderungen reagieren. Versuche mit verschiedenen Aktivierungsfunktionen brachten keine weiteren Vorteile. Wie man an den Kennlinien sieht, ist der Wertebereich [-1, 1] die geeignetste Lösung.

4.6 Variationen

Ebenso wichtig wie die Auswahl der geeigneten Parameter ist die Wahl der Topologie des KNN. Wie in Kap. 3.2.2 erwähnt, sind zwei versteckte Schichten ausreichend, es sollten aber auch nicht weniger sein. Es bleibt die Anzahl der Neuronen, in den ver-

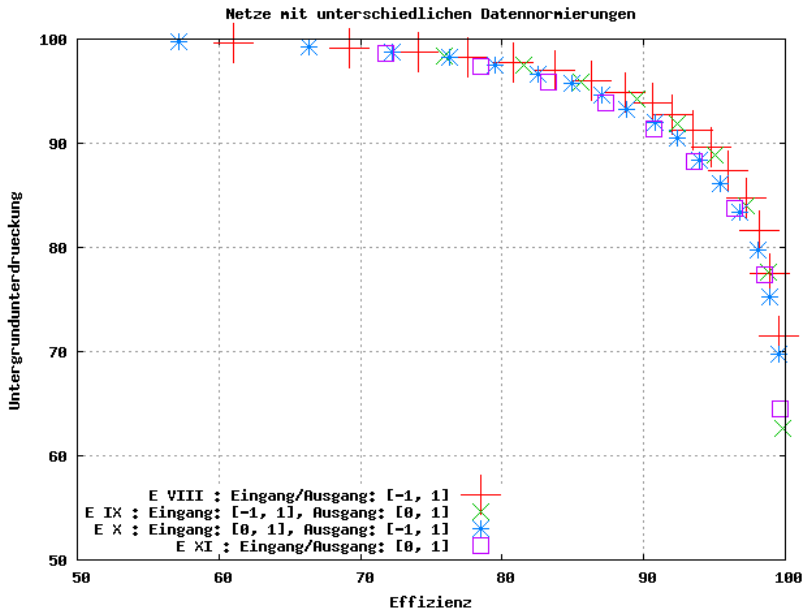


Abbildung 4.12: Hier wurden nur die Wertebereiche der Lern- und Testdaten wie in 4.5 beschrieben geändert. Die Topologie der Netze und die verwendeten Parameter blieben gleich, nur die Aktivierungsfunktionen wurden zu Gunsten des besten Ergebnisses variiert.

steckten Schichten, als variable Größe. In der Literatur sind diesbezüglich keine genauen Angaben zu finden. Letztendlich können nur Tests mit unterschiedlichen Topologien Aufschluss darüber geben, welche Anzahl der Neuronen in den versteckten Schichten am besten geeignet ist. Prinzipiell sollten es umso mehr versteckte Neuronen sein, je mehr Eingänge das KNN besitzt. Im Rahmen dieser Arbeit wurden mehrere hundert Netze trainiert und getestet, um das jeweils beste Netz zu finden. Dabei wurde unter anderem auch die Netztopologie geändert. Nach vielen Tests zeigte sich, dass in diesem Fall ein Netz mit 11 Neuronen in der ersten und 5 Neuronen in der zweiten versteckten Schicht am besten geeignet zu sein scheint. Größere Netze konnten das Problem nicht besser adaptieren, kleinere Netze schnitten dagegen schlechter ab. Die Lernkurven in 4.13 zeigen das Lernverhalten von zwei Netzen mit unterschiedlicher Topologie. Wie man sieht, ist mit dem größeren Netz keine wesentliche Verbesserung im Lernprozess zu beobachten. Es erreicht sein Minimum zwar schon nach weniger Lernschritten, im Endeffekt erreicht das kleine Netz aber nach etwa 4000 Lernzyklen denselben Wert und kann diesen sogar unterschreiten. Die Kurven in 4.13 sind als Beispiele zu betrachten und spiegeln im Allgemeinen die Erfahrung wieder, die im Rahmen dieser Arbeit bei Versuchen mit unterschiedlich großen KNN gemacht wurde.

Die Kette von der Erzeugung, über die Gewinnung der Daten, bis hin zum Training der KNN, erlaubt eine Vielzahl von Variationen. Neben den bereits genannten, ist es möglich die Aktivierungsfunktionen der einzelnen Neuronen zu ändern. Zwar

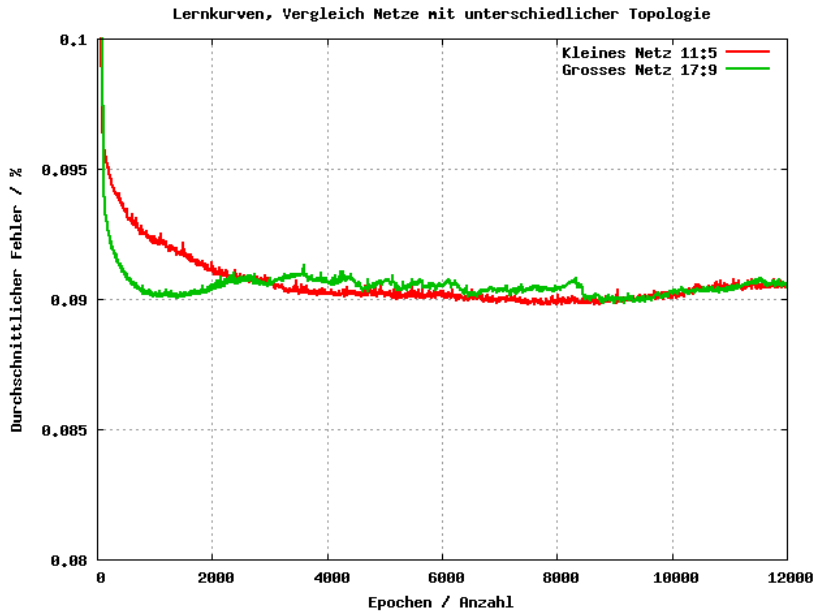


Abbildung 4.13: Lernkurven von zwei verschieden großen Netzen. Die Parameter und die Lerndaten sind gleich. Offenbar ist das kleinere Netz ausreichend dimensioniert, so dass es nicht nötig ist noch mehr Neuronen in die versteckte Schicht einzubauen.

wurde nicht an jeder sich bietenden Stelle dieser Arbeit der Einfluss unterschiedlicher Aktivierungsfunktionen getestet, aber insbesondere als sich die Wertebereiche der Eingangsdaten änderten, wurden verschiedene Neuronentypen getestet. Neben der Tangens-Hyperbolicus-Funktion wurden auch die nicht symmetrische Sigmoidfunktion (siehe 3.4 aus Kapitel 3.2.1) und Gaussfunktion getestet. In allen Tests schnitt das Netz mit der tanh-Funktion am besten ab und wurde letztendlich beibehalten.

Kapitel 5

Zusammenfassung und Endergebnis

Um das beste Ergebnis zu erhalten, wurden im Laufe der Arbeit viele verschiedene Konfigurationen getestet. Die Anzahl der Neuronen in den versteckten Schichten wurde durch zahlreiche Tests optimiert. Um sicher zu stellen, dass das geeignetste KNN verwendet wird, wurden die Aktivierungsfunktionen der Neuronen jedesmal geändert (siehe Kapitel 3.2.1), wenn sich beispielsweise die verwendeten Eingangsparameter oder ihr Wertebereich geändert haben.

Zur Auswahl geeigneter Parameter wurden verschiedene Verfahren angewendet (siehe Kapitel 4.2). Die Werteverteilungen zwischen Signal und Untergrund wurden verglichen, um Unterschiede in deren Formen zu erkennen, die darauf hindeuten, dass in ihnen Informationen entahlten sind, die zur Lösung des Problems beitragen können.

Einige Parameter konnten entfernt werden, da sie aufgrund ihrer Berechnung voneinander Abhängig sind und deswegen das Netz belasten würden. Zusätzlich wurden die Gewichte an den Neuronen der Eingangsschicht betrachtet, um Parameter mit geringem Einfluss auf das Ergebnis zu identifizieren.

Für jede Konfiguration wurden 5 bis 15 Netze trainiert und getestet, um sicher zu stellen, dass ein einzelnes, zufällig schlechtes Lernergebnis nicht zu einer falschen Annahme und letzten Endes zum Verwerfen, wie z.B. einer bestimmten Netztopologie, führt. Nur das Netz mit dem besten Ergebnis wurde weiter betrachtet, die anderen wurden verworfen. Übertraining wurde wie in Kap. 3.3 beschrieben vermieden.

Am Ende dieser Arbeit konnten neun, aus den ursprünglich 66 Parametern, ausgewählt werden, die die Klassifikation zwischen von oben und von unten kommenden Ereignissen durch ein künstliches Neuronales Netz ermöglichen. In Kapitel 4.3 werden die verwendeten Parameter und ihre Berechnung beschrieben. Die Werte der Effizienz und Untergrundunterdrückung können den Abbildungen 5.1 und 5.2 entnommen werden.

Das KNN, das am besten abschnitt und welches in 5.1 zu sehen ist, besitzt zwei

versteckte Schichten. Die erste mit 11, die zweite mit 5 Neuronen. Zusätzlich wurden dem Netz Biasneuronen hinzugefügt. Die Netze wurden mit ca. 1000 Epochen trainiert. Tests zeigten keine Verbesserung bei mehr Epochen. Als geeignetste Aktivierungsfunktion stellte sich der Tangens Hyperbolicus heraus.

Wenn die MC-Simulation verbessert wird und die verwendeten Parameter im Vergleich zu den Messdaten besser abschneiden wie die in Kapitel 4.4, sollte das Netz neu trainiert und getestet werden, um von den Verbesserungen profitieren zu können.

Um das Netz optimal an die realen Gegebenheiten anzupassen, wurde den Trainingsdaten, in einem letzten Schritt der Verbesserung, ein Rauschen von 60kHz auf moduliert, da es unmöglich ist Messdaten ohne Rauschen aufzunehmen.

Die Kennlinien in Abbildung 5.1 zeigen die Erkennungsleistung des finalen Netzes, welches im Fall der Kurve 'E XV' ohne Rauschen und für 'Ep XVI' mit Rauschen trainiert wurde. Die Unterschiede, zu sehen in 5.2, fallen zwar nicht überraschend groß aus, doch konnte durch diesen einfachen Schritt das Verhalten des KNN, wieder um einige Prozent verbessert werden. Im direkten Vergleich konnte die Untergrundunterdrückung von 61,1% bei 97,5% Effizienz für das erste Netz, auf 63,4% Untergrundunterdrückung bei 97.9% Effizienz für das mit Rauschen trainierte Netz, verbessert werden.

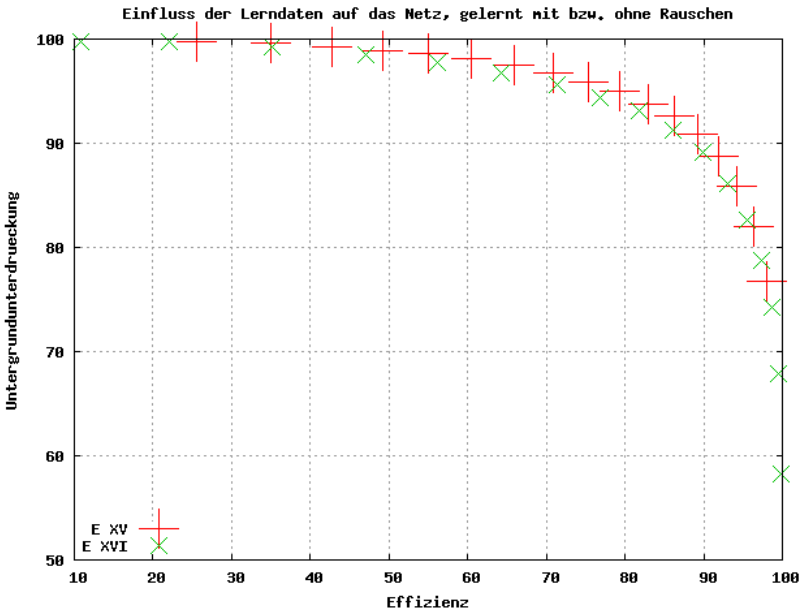


Abbildung 5.1: Zwei Netze, trainiert mit (E XVI) und ohne (E XV) Rauschen, getestet an Daten ohne Rauschen. Hier schneidet das Netz (E XVI) sogar ein wenig schlechter ab, doch spiegelt dieser Fall nicht die Wirklichkeit wieder.

Die Klassifikation in von oben kommende atmosphärische Myonen und von unten kommende Neutrino-Myonen konnte mit Hilfe eines künstlichen Neuronalen Netzes, wie

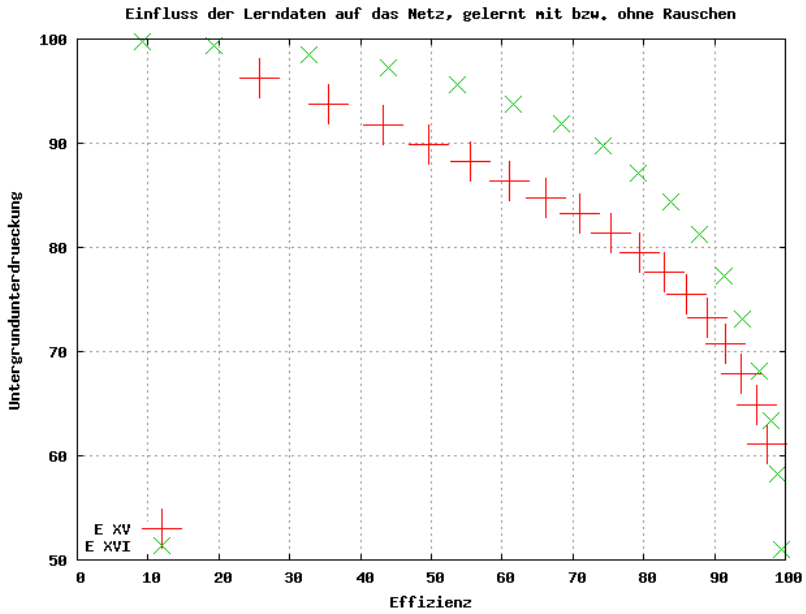


Abbildung 5.2: Zwei Netze, trainiert mit (E XVI) und ohne (E XV) Rauschen, getestet an Daten mit Rauschen. Hier ist das Netz (E XVI) klar besser.

oben beschrieben durchgeführt werden. In Anbetracht der hohen Anzahl atmosphärischer Myonen, ist die Untergrundunterdrückung von ca. 60% nicht ausreichend. Ein nächster Schritt könnte weitere multivariate Datenanalysemethoden mit in die Klassifizierung einbeziehen und diese durch Kombination weiter verbessern.

Literaturverzeichnis

[1] John A. Peacock. *Cosmological Physics*. Cambridge University Press, 1999.

[2] Thomas K. Gaisser. *Cosmic Rays and Particle Physics*. Cambridge University Press, 1990.

[3] Carlo Guinti, Chung W. Kim. *Fundamentals of Neutrino Physics and Astrophysics*. Oxford University Press, 2007.

[4] Berger. *Elementarteilchenphysik*. Springer-Verlag Berlin Heidelberg New York, 2 edition, 2006.

[5] K. Bethge, G. Walter, B. Wiedemann. *Kernphysik*. Springer-Verlag Berlin Heidelberg, 3 edition, 2007.

[6] T. Chiarusi, M. Spurio. High-Energy Astrophysics with Neutrino Telescopes. arxiv.org, 2009. arXiv:0906.2634v2 [astro-ph.HE].

[7] Kevin McFarland. Neutrino Interactions. Technical Report arXiv:0804.3899v1, arxiv.org, Apr 2008.

[8] Bettina Diane Hartmann. *Reconstruction of Neutrino-Induced Hadronic and Electromagnetic Showers with the ANTARES Experiment*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2006.

[9] J. Brunner. Antares Simulation Tools, Workshop on Technical Aspects of a Very Large Volume Neutrino Telescope in the Mediterrean Sear. Amsterdam, Netherlands, Oct 2003. http://www.vlvnt.nl/proceedings.

[10] Steffen Nissens. FANN. http://leenissen.dk/fann/index.php, 2003. Department of Computer Science University of Copenhagen (DIKU).

[11] Antares. Software Framework Seatray. http://wiki.km3net.physik.uni-erlangen.de/index.php.

[12] CORSIKA, an Air Shower Simulation Program. http://www-ik.fzk.de/corsika/.

[13] Haykin, Simon. *Neural Networks. A Comprehensive Foundation*. Prentice-Hall, 1999.

[14] Thorsten Kolb. Anwendung Neuronaler Netze. http://vieta.math.tu-cottbus.de/ kolb/ml-nn/node10.html.

[15] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[16] Warren S. Sarle. How many hidden units should i use?

[17] Alexander Pichler, Martin Brunninger, Stefan Wasserl. Simulator für Backpropagation-Netze. http://fbim.fh-regensburg.de/ saj39122/wabrpi/index.html.

Anhang A

Verwendetes Pythonscript

Die Daten zum Trainieren und Testen der KNN wurden mit dem Framework Seatray des ANTARES-Experiments aus Mess- und MC-Daten gewonnen. Das unten stehende Pythonscript zeigt die Reihenfolge der verwendeten Module.

```
#!/usr/bin/env python
from I3Tray import *
from os.path import expandvars

import os
import sys

load("libdataclasses")
load("libphys-services")
load("libdataio")
load("libantares-reader")
load("libhit-selector")
load("libantares-tools")
load("libclassification")
load("libicetray")


#in case of Real data use the following lines to load the files:
##############################################################################################################
##############################################################################################################
##################################### Reading REAL data   Start ##########################################

tasks.antares.exp.addReadAntExpData(tray, evtfile=RawDataFileName, calibrationversionstring=calibrationversionstri

#=========================================================================
#Filter Minimum Bias
#=========================================================================
# PYTHON MODULE: filter minimum bias events from Antares raw data
tasks.antares.exp.addFilterMinimumBiasEvents(tray)



#=========================================================================
#Calibration
```

```
#============================================================================
# PYTHON MODULE: Antares raw data calibration
tasks.antares.exp.addCalibration(tray)



################################### Reading REAL data   STOP ########################################
#####################################################################################################
#####################################################################################################

#in case of MC data use the following lines to load the files:
#####################################################################################################
#####################################################################################################
################################### Reading MC data   START ##########################################


evtfile = "EventFile"
numberOfGeneratedEvents = 123456 #change this to the number of events within the files you are going to process

seed=958
noise = 60.
irradiationTime = 475846.*I3Units.s #based on the durationtime of the golden Runs used


# these are the values used by the ANTARES tool "TriggerEfficiency"
coincidenceWindow = 20.*I3Units.ns
bigHitAmp = 2.5
ARSThresholdAmp = 0.3
ARSIntegrationTime = 40.*I3Units.ns
ARSDeadTime = 250.*I3Units.ns

geofile = expandvars("/pi1/common/antsoft/DETECTOR/r12_c00_s01.det") # new prodction 11-2009
tray = I3Tray()

tray.AddService("I3SPRNGRandomServiceFactory","random")(
    ("Seed",seed),
    ("NStreams",seed+1),
    ("StreamNum",1))

# Fill the geometry stream with data from the ANTARES detector file
tray.AddService("I3AntTextFileGeometryServiceFactory","geometry")(
    ("OMAngularParametrization", "Spring09"),
    ("AntaresGeoFile",geofile)
    )

#Reads the Antares text file
tray.AddService("I3AntTxtReaderServiceFactory","anttxtreader")(
    #("FilenameList",evtfile),
("Filename",evtfile),
    ("Year",2007),
    ("DAQTime",173356670000000001),
    ("MCSeriesName","EvtMCHitSeries"),
    ("RawSeriesName","EvtRawHitSeries")
    )
    #the name of the MC Tree : AntMCTree
    # name of the additional Parameters: AntMCAdditionalParameters
```

```
# Add empty streams where necessary
tray.AddService("I3EmptyStreamsFactory","empty_streams")(
    ("InstallGeometry",False),
    ("InstallEvent",False),
    ("InstallStatus",True),
    ("InstallCalibration",True),
    )

# mux everything together
tray.AddModule("I3Muxer","muxer")

#add some noise or delete this module
tray.AddModule("I3NoiseHitsAdder","addnoise")(
    ("WhiteNoiseRatePerPMT",noise*I3Units.kilohertz),
    ("NoiseOffsetTime",500.*I3Units.ns),
    ("RemoveOldNoise",True),
    ("InputMCHits","EvtMCHitSeries"),
    ("OutputMCHits","EvtMCHitSeries_WithNoise")
    )



#THIS IS THE L0 TRIGGER:

# first, do a PM simulation (including the ARS chip)
tray.AddModule("I3AntPMSimulator","simpm")(
("InputMCHits","EvtMCHitSeries_WithNoise"),
("OutputRecoPulses","RecoPulseSeriesAfterARS"),
("OutputHitRelationMap","RecoPulseToMCHitRelation"),
("NumARS",2),
("IntegrationTimeARS",ARSIntegrationTime),
("DeadTimeARS",ARSDeadTime),
("AmplitudeThresholdARS",ARSThresholdAmp),
("DoTrigEffTTS", True),# use the distributions from "TriggerEfficiency"
("DoTrigEffGainRandomization", True)
    )
##################################### Reading MC data   STOP ############################################
#########################################################################################################
#########################################################################################################

#common modules for Real- and MC-Data

#NOW FOLLOWING IS THE L1 TRIGGER (4 modules):

# we now have L0 hits. Apply two hit selections and merge
# all selected hits. The first selection chooses all hits
# with an amplitude larger than 2.5p.e. . The second one
# selects all hits on a LCM within a 20ns coincidence
# window.
tray.AddModule("I3HitSelectorModule<I3LowHighCutOff>","HighThreshSelection")(
("INmap", "RecoPulseSeriesAfterARS"),
("OUTmap", "RecoPulseSeriesAfterARS_high_thresh"),
("MinNPE", bigHitAmp),
("MaxNPE", 1.e7) # essentially no upper bound
)

# find coincidences within the remaining hits
```

```
tray.AddModule("I3HitSelectorModule<I3LocalCoincidences>","LocalCoincidenceSelection")(
("INmap", "RecoPulseSeriesAfterARS_high_thresh_deselected"), # use all "non-big" hits as the input
("OUTmap", "RecoPulseSeriesAfterARS_coincidences"),
("Coincidence", coincidenceWindow),
("OnlyOneHitPerCoincidence", True), # we only need one hit per floor coincidence, throw away all others
("NoFrameNamingPostfix", True),
("WriteSelectedHits", True),
("WriteDeselectedHits", False)
)

# build a combined list of big hits and floor coincidences
tray.AddModule("I3HitMerger","MergeSelections")(
("INmap1", "RecoPulseSeriesAfterARS_high_thresh_selected"),
("INmap2", "RecoPulseSeriesAfterARS_coincidences"),
("OUTmap", "RecoPulseSeriesAfterARS_L1_with_redundancies")
)

# clean up this list by removing redundant coincidences (they could be
# caused by a big hit that is part of a coincidence)
tray.AddModule("I3HitSelectorModule<I3RemoveRedundantCoincidences>","RemoveRedundantCoincidences")(
("INmap", "RecoPulseSeriesAfterARS_L1_with_redundancies"),
("OUTmap", "RecoPulseSeriesAfterARS_L1"),
("NoFrameNamingPostfix", True),
("WriteSelectedHits", True),
("WriteDeselectedHits", False),
("Coincidence", coincidenceWindow)
)

# For each floor coincidence, retrieve the full list of coincident hits
tray.AddModule("I3HitSelectorModule<I3LocalCoincidencesWithHits>","RebuildTriggeredHitsL1")(
("INmap", "RecoPulseSeriesAfterARS"),# use the full list of hits as input
("ReferenceHitSeriesMap", "RecoPulseSeriesAfterARS_L1"),   # compare them to the hits returned by the trigger
("OUTmap", "RecoPulseSeriesTriggeredHits_AfterL1"),
("CoincidenceGate", coincidenceWindow),
("NoFrameNamingPostfix", True),
("WriteSelectedHits", True),
("WriteDeselectedHits", False)
)




#NOW FOLLOWING IS THE L2 TRIGGER (2 modules):

# let's call the trigger the list of "L1" hits. Like in the ANTARES trigger
# code, the floor position is used for triggering instead of the OM positions.
# This makes sense, as the L1 selection leaves only one hit per floor coincidence.
tray.AddModule("I3AntTriggerSimulator","simtrigger")(
("InputRecoPulses", "RecoPulseSeriesAfterARS_L1"),
("OutputRecoPulses", "RecoPulseSeriesAfterTrigger"),
("OutputTriggeredBool", "Triggered"),
("OutputTriggeredNum", "NumTriggers"),
("TriggerHierarchyName", "SimulatedTriggers"),
("WriteOutUntriggeredEvents", False),
("UseFloorPositions", True), # use this for L1 triggering
("DoTrigger3D", False),
("DoTrigger3N", True),
```

```
("DoTrigger3S", False),
("DoTrigger3T", False),
("DoTrigger1D", False)
)

# For each floor coincidence, retrieve the full list of coincident hits
tray.AddModule("I3HitSelectorModule<I3LocalCoincidencesWithHits>","RebuildTriggeredHits")(
("INmap", "RecoPulseSeriesAfterARS"),# use the full list of hits as input
("ReferenceHitSeriesMap", "RecoPulseSeriesAfterTrigger"),   # compare them to the hits returned by the trigger
("OUTmap", "RecoPulseSeriesTriggeredHits_AfterL2"),
("CoincidenceGate", coincidenceWindow),
("NoFrameNamingPostfix", True),
("WriteSelectedHits", True),
("WriteDeselectedHits", False)
)

tray.AddModule("AntGetMCInfo","getmcinfo")(
   ("InputMCHits","EvtMCHitSeries_WithNoise"),
   ("InputRecoPulses","RecoPulseSeriesAfterARS"), # full hit list as input
   ("InputRelationMap","RecoPulseToMCHitRelation"),
   ("InputMCTree","AntMCTree"),
   ("InputAdditionalParams","AntMCAdditionalParameters"),
   ("NumberOfGeneratedEvents", numberOfGeneratedEvents),
   ("IrradiationTime", irradiationTime),
   ("OutputMCInfo","ClassificationMCInfoOutput"),
  )



#the mc amplitudes might be bigger than the pmts can handle.
#so let's cut them to a more realistic value

tray.AddModule("I3PulseHighAmpCutoff", "clip")(
    ("InputRecoPulses", "RecoPulseSeriesAfterARS"),
    ("OutputRecoPulses", "AllRecoPulsesClipped"),
    ("HighAmplitudeCutoff", 20.)
    )


tray.AddModule("I3PulseHighAmpCutoff", "cliptriggeredL1")(
    ("InputRecoPulses", "RecoPulseSeriesTriggeredHits_AfterL1"),
    ("OutputRecoPulses", "TriggeredRecoPulsesClippedL1"),
    ("HighAmplitudeCutoff", 20.)
    )


tray.AddModule("I3PulseHighAmpCutoff", "cliptriggeredL2")(
    ("InputRecoPulses", "RecoPulseSeriesTriggeredHits_AfterL2"),
    ("OutputRecoPulses", "TriggeredRecoPulsesClippedL2"),
    ("HighAmplitudeCutoff", 20.)
    )


#============================================================================
# The following modules calculate the meta data for the neural net

tray.AddModule("I3ClassificationCalcTensors", "calctensors")(
```

```
   ("InputRecoPulses","TriggeredRecoPulsesClippedL2"),
   ("InputTriggeredHits","TriggeredRecoPulsesClippedL2"),
   ("EventHeader","I3EventHeader"),
   ("OutputTensorParameters","TensorOutput"),
   )

tray.AddModule("I3ClassificationCalcDensity","calcdensity")(
   ("InputRecoPulses", "TriggeredRecoPulsesClippedL1"),
   ("InputTriggeredPulses", ""),
   ("DetectorLevels", 5),
   ("OutputDensityParameters","DensityOutput"),
   )

tray.AddModule("I3ClassificationCalcShape","calcshape")(
   ("InputRecoPulses","TriggeredRecoPulsesClippedL2"),
   ("OutputShapeParameters", "ShapeOutput"),
   )

tray.AddModule("I3ClassificationCalcCluster","calccluster")(
   ("InputRecoPulses","TriggeredRecoPulsesClippedL1"),
   ("NumberOfPulses", 10),
   ("OutputClusterParams", "ClusterOutput"),
   )



#and thats it for calculating the Metadata.

#following modules could write the meta data to a file for training and testing neural networks or
#apply an allready trained neural network to the event represented in this frame.









tray.AddModule("I3Writer","writer")(
("filename", i3outfile),
 )

tray.AddModule("TrashCan", "the can")

tray.Execute()
tray.Finish()
```

Anhang B

Verwendete Messdaten

Die folgende Liste enthält die Messdaten, die für den Vergleich in Kapitel 4.4 verwendet wurden. Es handelt sich dabei um alle 'Golden Runs' von Dezember 2008.

Antares-037603.root	Antares-037605.root	Antares-037619.root
Antares-037620.root	Antares-037622.root	Antares-037634.root
Antares-037635.root	Antares-037659.root	Antares-037662.root
Antares-037664.root	Antares-037675.root	Antares-037678.root
Antares-037679.root	Antares-037682.root	Antares-037689.root
Antares-037701.root	Antares-037712.root	Antares-037714.root
Antares-037716.root	Antares-037717.root	Antares-037720.root
Antares-037722.root	Antares-037723.root	Antares-037731.root
Antares-037732.root	Antares-037736.root	Antares-037738.root
Antares-037739.root	Antares-037741.root	Antares-037744.root
Antares-037746.root	Antares-037751.root	Antares-037752.root
Antares-037753.root	Antares-037755.root	Antares-037757.root
Antares-037759.root	Antares-037768.root	Antares-037770.root
Antares-037771.root	Antares-037783.root	Antares-038004.root
Antares-038007.root	Antares-038009.root	Antares-038045.root
Antares-038052.root	Antares-038054.root	Antares-038056.root
Antares-038058.root	Antares-038060.root	Antares-038063.root
Antares-038065.root	Antares-038067.root	Antares-038070.root
Antares-038072.root	Antares-038074.root	Antares-038075.root
Antares-038076.root	Antares-038078.root	Antares-038080.root
Antares-038082.root	Antares-038084.root	Antares-038086.root
Antares-038088.root	Antares-038090.root	Antares-038093.root
Antares-038106.root	Antares-038108.root	Antares-038183.root
Antares-038185.root	Antares-038212.root	Antares-038213.root
Antares-038215.root	Antares-038216.root	

Danksagung

Abschließend möchte ich mich bei allen Personen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.
Besonders bedanken möchte ich mich bei:

Prof. Dr. Gisela Anton für die Vergabe dieses Themas.

Dr. Thomas Eberl, dessen Tür für alle Fragen und Probleme offen stand.

Friederike Schöck, die mich bei der Lösung vieler Probleme unterstützt hat.

Allen Mitgliedern der Erlanger Antaresgruppe für ihre Hilfe und eine ausgezeichnete Arbeitsatmosphäre.

Und natürlich meinen Eltern und Freunden, die mich während des Studiums moralisch unterstützt haben.

Erklärung

Ich versichere, diese Diplomarbeit selbstständig verfasst zu haben und keine anderen als die angegeben Hilfsmittel und Quellen benutzt zu haben.

Erlangen, März 2010

Klaus Geyer