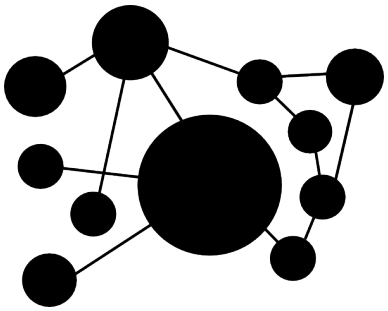
Seminararbeit

Prognose von Zeitreihen mit Hilfe von künstlichen neuronalen Netzen am Beispiel von Börsenprognosen



Sebastian Schötteler – Matrikelnummer 24 29 289
Benedikt Hofrichter – Matrikelnummer 22 72 198

22. Dezember 2015

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

1 Einleitung

1.1 Motivation

Die Untersuchung und Extrapolation von Zeitreihen ist ein bedeutendes Thema in zahlreichen Gebieten. Typische Anwendungsbereiche sind zum Beispiel die Prognose von Wetterdaten, von Therapieverläufen in der Medizin, von Arbeitslosenzahlen auf dem Arbeitsmarkt sowie von Börsenkursen. Um eine Zeitreihe möglichst genau zu extrapolieren, wird auf mehrere Hilfsmittel zurückgegriffen. Eins dieser Hilfsmittel können KNN (künstliche neuronale Netze)sein.

Bei KNN handelt es sich um Netzwerke mit künstlichen Neuronen als Knoten, die mittels gerichteter Verbindungen Eingaben einlesen, weiterverarbeiten und die daraus resultierenden Ergebnisse an weitere Neuronen weiterleiten oder als Ergebnis ausgeben. Bei der Terminologie von KNN wird bewusst auf Begriffe der Biologie zurückgegriffen, da KNN das biologische Gehirn als Vorbild nutzen und dessen Herangehensweise auf analoger Weise umzusetzen zu versuchen. Man nennt das Verfahren dieser Netze aus diesem Grunde auch *naturanaloge Verfahren*.

Warum sind diese Netze nun so interessant für Prognosen? Das Erstellen von zum Beispiel Börsenprognosen basiert in der Regel auf Auswertungen von Informationen verschiedener Quellen. Die Art von Auswertungen, wie Börsenexperten sie vornehmen, ist weder vollständig formalisierbar noch besonders exakt, da uneinheitlich und in weiten Zügen intuitiv. Besonders schwer ist hier das Ermitteln von *nichtlinearen Zusammenhängen*. Ein KNN ist jedoch in der Lage, diese Zusammenhänge zu finden und diese objektiv und vorurteilsfrei zu bewerten. Somit sind diese prinzipiell in der Lage, jedes beliebige Muster in jedem beliebigen Markt zu erkennen - auch solche, die noch nie zuvor von irgendjemand entdeckt wurden.

Ob und wie gut KNN zur Prognose geeignet sind, ist pauschal nicht zu beantworten. In manchen Gebieten mag die Prognosefähigkeit durchaus ausreichen. Je höher die geforderte Genauigkeit jedoch wird, desto diskutabler wird ein Einsatz von KNN. Eine typische Grauzone ist hier wieder die Prognose von Börsenkursen. Während Befürworter auf die Eigenschaft von KNN hinweisen, nichtlineare Muster zu erkennen und entsprechend zu behandeln, argumentieren Kritiker, dass ein System, das dem menschlichen Lernen nachempfunden wurde, die gleichen Fehler machen wird wie der Mensch. Generell ist jedoch zu sagen, dass die Prognosequalität von KNN über die Jahre stets angestiegen ist, da zum einen stets neue Fortschritte in diesem Themengebiet gemacht werden und zum anderen die Leistungsfähigkeit von Rechners stetig ansteigt.

1.2 Ziel der Arbeit

In dieser Seminararbeit sollen KNN erschaffen werden, die in der Lage sind, Börsenkurse zu prognostizieren. Konkret sollen drei verschiedene KNN konzeptioniert und umgesetzt werden. Ein KNN zur Prognose des Kurses vom DAX (Deutschen Aktienindex), eines zur Prognose des Kurses vom Nikkei sowie eines zur Prognose des Kurses vom Dow Jones. Diese KNN sollen anschließend in einer Webanwendung überführt werden. Diese soll die Prognosefähigkeit der KNN visualisieren und Vergleiche zwischen einzelne Prognosen ermöglichen. In dieser Seminararbeit liegt der Fokus auf das Erlangen eines Grundverständnisses über KNN und nicht auf das komplette Ausreizen der Prognosefähigkeit von KNN. Trotzdem spielt die Prognosequalität der erstellten KNN eine wichtige Rolle in dieser Seminararbeit.

2 Konzeption der Anwendung

In diesem Kapitel wird die Konzeption der Anwendung beschrieben. Zunächst wird die Grundidee vorgestellt. Anschließend wird ein genaues technologisches Konzept erstellt.

2.1 Grundidee

Bekannte Zusammenhänge zwischen prognostiziertem und tatsächlichem Wert sind meist durch einfache Rechenoperationen umsetzbar. So kann beispielsweise eine Erhöhung der Mehrwertsteuer für ein Produktpreis P mit einer Rechenoperation P = P * M errechnet werden. Für diese Zusammenhänge ist es meist nicht notwendig und sinnvoll, komplexe und vernetzte Softwaresysteme zu entwickeln.

Wie anfangs erwähnt wird bei Börsenprognosen versucht, *nicht-lineare Zusammenhänge* innerhalb von Daten zu finden, da lineare und bekannte Zusammenhänge, die zudem noch exakt sind quasi nicht vorhanden sind. Deshalb ist es für die Anwendung sinnvoll und notwendig einen komplexeren und vernetzeren Ansatz zu verfolgen, auch wenn dies mit einer Steigerung der Kosten einher geht.

Neben dem Hauptziel, die Zeitreihenextrapolation von Börsendaten, soll gezeigt werden, wie es konzeptionell und technisch möglich ist, eine derartige Softcomputing-Lösung zu realisieren und in einen Unternehmenskontext zu integrieren.

Weitere Faktoren, die bei der Entscheidung über die eingesetzten Frameworks und Werkzeuge mit eingeflossen sind, sind Skalierbarkeit, Wartbarkeit, Erreichbarkeit sowie Anschaulichkeit.

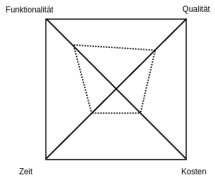


Abbildung 2.1: Abb - nicht lineare

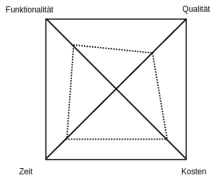


Abbildung 2.2: Abb - lineare

Quandl Data-Provider Für Börenprognosen werden quantitative Maßeinheiten benötigt, die im Zusammenhang mit Kurs stehen. Die Wohl aussagekräftigste sind Börsenschluss-Index-Werte von vergangenen Handelsperioden. Je mehr dieser Daten vorhanden, je realistischer diese Daten sind, desto warscheinlicher ist es, einen Schlusswert in der Zukunft vorherzusagen. Aus diesem Grund spricht die Anwendung, genauer der Rest-Controller der Stockmarket-Webapp den Rest-Service von Quandl an. Quandl versteht sich als Datenhändler / Datenanbieter mit integrierter Suchfunktion. Das Angebot umfasst die globale Finanzbranche, so z.B. sämtliche nationale und internationale Börsenkurse der vergangenen Dekaden. Die Rest-API bietet die Export-Formate *XML*, *JSON* und *CSV* an. ; Diese ermöglicht der Anwendung mit echten Börsendaten zu arbeiten.

[https://www.quandl.com/blog/getting-started-with-the-quandl-api].
;

Stockmarket-Webapp Bei der Stockmarket-Webapp handelt es sich um einen Restful-Service der als Schnittstelle zwischen der Quandl-Rest-API, der Visualisierung sowie dem Neuroph Werkzeug fungiert. Wie in Abbildung ABBILDUNG zu sehen ist gibt es keine Kommunikation zwischen Fremd-Services, wenn diese nicht über den Controller der Stockmarket-Webapp statt findet. Hierüber können gleich mehrere Vorteile realisiert werden. 1) Single-Point-Of-Failure: Wenn der Datenlieferant Quandl insolvent geht oder nicht ganz so drastisch, wenn die Anfrage-Mechanismen der Quandl-Rest-API geändert werden. Ein weiteres Szenario, könnte aus Sicht der Visualisierung einen aus Performance-Bedingten Wechsel der JavaScript-Bibliothek darstellen. Sofern für diese Szenarien keine SPOF-Lösung existiert, wird eine entsprechende Anpassung womöglich ein umfassendes Refactoring aller Anwendungskoponenten erfordern. 2) Skalierbarkeit: Den Fall angenommen, weitere Datensätze sollen an andere Visualisierungen bzw. die Visualisierung um Diagramme erweitert

werden, so ist bereits mit einer einseitigen Anpassung der Visualisierung die Anforderung erfüllt. Die durch die Rest-Spezifikation definierte Tatsache der einheitlichen Repräsentation von Anfragen spielt einem hier in die Hände.
Die Aufgabe der Stockmarket-Webapp ist es eine schnittstellen - und benutzerdefinierte Datenaufbereitung zu realisieren.

Visualisierung Die Grundidee die Visualisierung von der Anwendungslogik weitgehend zu trennen hat diverse Vorteile. Die Abhängigkeitsreduktion von anderen Komponenten, sowie Sicherheitsaspekte spielen dabei eine besondere Rolle. Die Visualisierung ist eine rein Client-seitig laufende Anwendung. Diese enthält alle notwendigen Logikroutinen, um Daten anzufragen und entsprechend zu verarbeiten. Die Implementierung dieser Logik wird mit der JavaScript-Bibliothek jQuery umgesetzt. Diese speißt und manipuliert das Document-Object-Model(DOM) je nach *Datenlage*. Das DOM kann auch als HTML-Gerüst bezeichnet werden. Für die visuellen Effekte, die eine bessere Veranschaulichung der Graphen ermöglichen soll, wird CSS eingesetzt. Um das Layout für Geräte mit unterschiedlicher Displaygröße gleichermaßen nutzbar zu machen, wird die CSS-Bibliothek Bootstrap verwendet.

2.2 Technologie

Rest-Kommunikation

Die Kommunikation zwischen den drei konzeptionell festgelegten Komponenten soll ausschließlich über das Programmierparadigma von ReST (Representational State Transfer) von statten gehen. Damit eine Kommunikation als *Restful* bezeichnet werden kann müssen einige Kriterien erfüllt werden. Ein Rest-Service ist adressierbar, jeder Rest-Endpunkt hat eine eindeutige URI. Das Format der zurückgelieferten Ressource kann variieren. So können beispielsweise *CSV*, *HTML*, *XML* und *JSON* gleichermaßen von einem Rest-Service geliefert werden. Hierbei ist das Prinzip der Zustandslosigkeit zu beachten, also egal in welchem Format ein Nachricht ankommt, der Informationsgehalt muss äquivalent für gleiche Anfragen sein. Der Kommunikationskanal ist zwar nicht festgeschrieben, ist aber ich aller Regel das HTTP. Hierbei gilt es die entsprechenden Empfehlungen für die HTTP-Verbs zu beachten. Dies ist vor allem im Punkt Sicherheit wichtig.

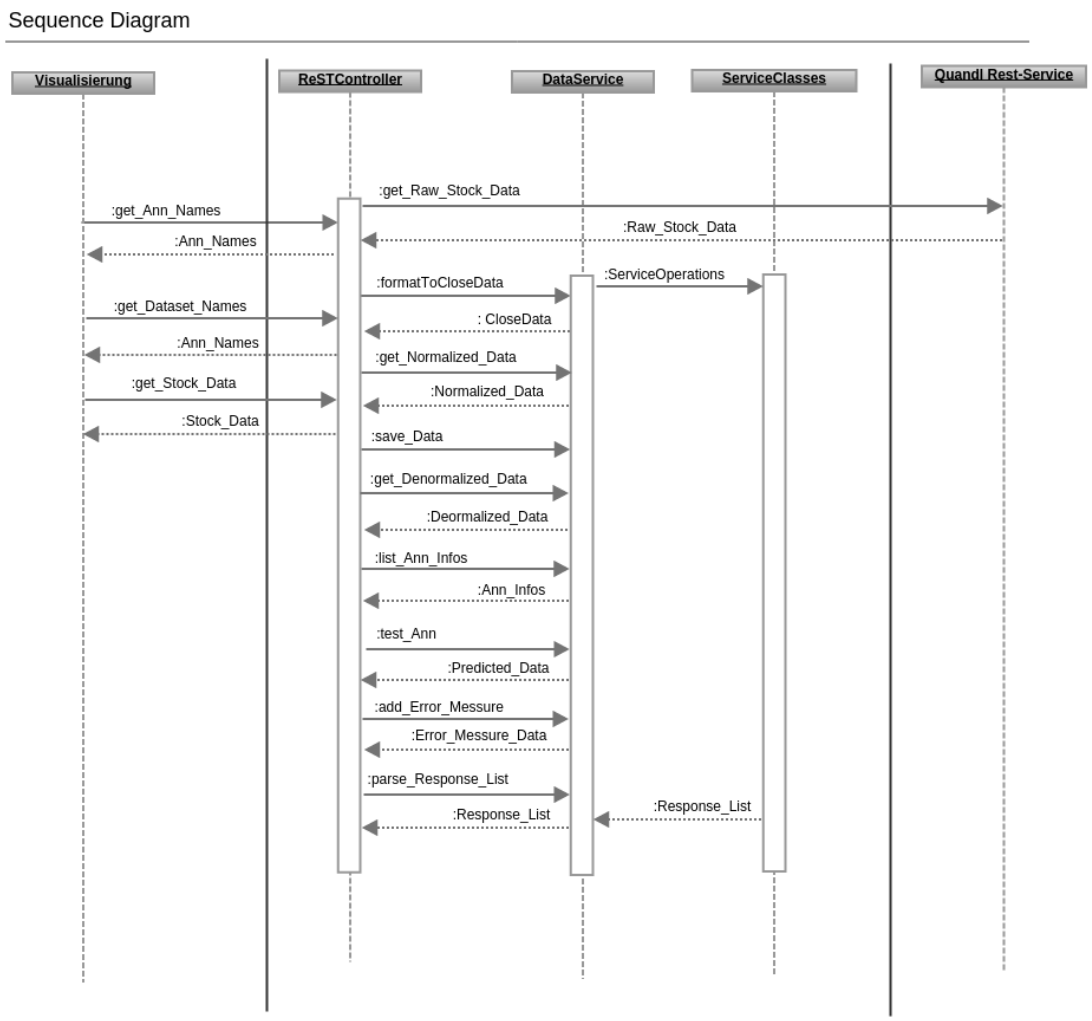


Abbildung 2.3: Sequenzdiagramm

2.2.1 Frameworks

Zu den nachfolgenden Frameworks und Bibliotheken werden hauptsächlich anwendungsrelevante Funktionalitäten erläutert, da es sonst den Rahmen dieser Seminararbeit sprengen würde.

Spring Boot Spring ist ein unter der Apache Lizenz veröffenlichtes, quelloffenes Java-Framework. Eine umfassende Übersicht würde den Rahmen dieser Seminararbeit sprengen. Daher wird im folgenden auf auf zwei besonders markante und wichtige Punkte kurz eingegangen. Beide Verfahren zielen darauf ab, Wiederholungen im Quellcode zu vermeiden sowie insgesamt prägnatere Anweisungen umzusetzen. Diese Thematik wird unter dem Begriff des *Boilerplate Code* oder besser die Vermeidung von Boilerplate Code geführt. Spring Boot bietet eine vereinfachte und schnellere Möglichkeit Anwendungen zu entwickeln, da noch vielfältigere Art und Weise Abhängigkeiten integriert werden können, so kann z.B. ein Tomcat-Server automatisch eingebunden werden.

Dependency Injection Die Abhängigkeitsinjektion zielt darauf ab, möglichst wenig Abhängigkeiten zwischen Java-Klassen zu konstruieren. Dependency Injection ist ein Entwurfsmuster (Software Pattern) das dieses umsetzt. Instanzen von Javaklassen müssen demnach ihre Abhängigkeitsinformationen durch einen Aufruf von Methoden einer externen Instanz zugewiesen bekommen. Deshalb wird dieser Vorgang als Injektion bezeichnet. Es werden drei verschiedene Arten von Abhängigkeitsinjektionen unterschieden, die *Inversion of Control* (IoC), die Konstruktorinjektion sowie die Setterinjektion. Spring implementiert beispielsweise einen sogenannten IoC-Container. Die darin enthaltenen Objekte verweisen explizit auf Abhängigkeiten, woraus sogenannte Java-Beans (Klassen die der Spezifikation LINK genügen) konstruiert werden. Spring bemüht sich im Grund darum *Best-Pratices* der Softwareentwicklung im Javaumfeld umzusetzen und den Entwickler hierbei zu unterstützen. Nähers zum Thema Spring kann z.B. unter dem LINK nachgelesen werden.

2.2.2 Annotationen

Annotationen sind äußerlich leicht zu erkennen. Sie beginnen mit einem @-Zeichen und können Methoden sowie Klassen gleichermaßen kennzeichnen. Annotationen sind Schnitt-

stellen (Interfaces), die grob in zwei Arten unterteilt werden können. Zum einen diejenigen die während der Kompilierzeit ausgeführt werden und anschließend nicht mehr benötigt werden wie beispielsweise die *@Override*-Annotation für die Implementierung von Methoden abstrakter Klassen. Die zweite Art ist auch während der Laufzeit noch von Bedeutung, so z.B. die *@Autowired*-Annotation. Diese gibt explizit an, dass eine Klasse per Dependency-Injection in die Programmlogik integriert werden soll.

Javascript Bibliothek - C3js Die Auswahl einer geeigneten Javascript-Bibliothek ist nicht leicht. Es gibt zahlreiche gut umgesetzte Lösungen im Umlauf, welche in unterschiedliche Bereichen jeweils Vor - und Nachteile bieten. *C3js* ist eine relativ schlank und verzichtet auf einige Zusatzkomponenten. Die Basis für C3 ist die *D3js* Bibliothek, deren Diagramme meist einen höheren Grad an visuellen Extravaganzen umsetzt oder auf spezielle Einsatzgebiete eingeht. D3js eignet sich daher eher für ausgefallenere Diagramme, C3 wurde aus Performance-Gründen gewählt und weil die angebotenen Funktionen den gewünschten Informationsgehalt entsprechend darstellen kann.

Layouts und CSS Bootstrap Das Bootstrap Framework ist eine freie und sehr umfangreiche Bibliothek, die Komponenten und Funktionalitäten bereitstellt, die sich an den neuesten Webdesign-Kriterien orientiert. Sie wird unter der MIT-Lizenz veröffentlicht und unterstützt die am bekanntesten Browser. Die Visualisierung profitiert start von der Möglichkeit, das HTML-Gerüst dynamisch an verschiedene Displaygrößen anzupassen (Responsive Design).

2.2.3 Apache Maven

Die benötigten Frameworks können auf verschiedene Wege eingebunden werden. Die Softwarepakete können per Hand heruntergeladen und in den entsprechenden Verzeichnispfad kopiert werden, was allerdings bei größeren Projekten hinsichtlicher der Übersicht und Verwaltbarkeit problematisch ist. Ein besserer Ansatz ist es ein Build-Tool wie Apache Maven zu verwenden, was auch für diese Anwendung eingesetzt wurde. Alternative Build-Tools sind z.B. Apache Ant und Gradle.
Zentraler Beschreibungspunkt, der das Projekt benötigter Abhängigkeiten und Build-Prozesse ist die POM.xml. POM steht dabei für *Projekt Object Model*. Die Softwarepakte heißten im

Maven-Jargon *Dependencies*, also Abhängigkeiten. Dependencies werden in *Repositories* verwaltet. Es gibt zwei Arten, *local* und *remote* Repositories. Die lokale Datenhaltungskomponente stellt eine exakte Kopie aller heruntergeladenen Dependencies sowie deren Verzeichnisstruktur, von rechnerfernen Repositories dar.

3 Konzeption des künstlichen neuronalen Netzes

In den Abschnitten 3.1, 3.2 sowie 3.3 wird ein KNN zur Prognose des DAX konzeptioniert. Dieses soll als Vorlage für die Erstellung der KNN zur Prognose des Dow Jones und des Nikkei verwendet werden.

3.1 Wahl des Netztyps

Zunächst ist zu ermitteln, welche Netztypen sich zur Prognose von Börsenkursen grundsätzlich eignen. Nicht jeder Netztyp ist gleichermaßen zur Prognose geeignet. Bestimmte Netztypen sind beispielsweise überhaupt nicht in der Lage, Prognosen zu erstellen. Grundsätzlich lassen sich alle Netztypen in eine von zwei Oberklassen einordnen. Es existiert die Klasse der hetero-assoziativen Netze sowie die Klasse der auto-assoziativen Netze. Hetero-assoziative Netze bilden einen Vektor A der Länge n auf einem Vektor B einer meist kürzeren Länge m $\{m \in \mathbb{N} | m \leq n\}$ ab. Auto-assoziative Netze wiederum bilden einen Eingabevektor A der Länge n auf einem Ausgabevektor der gleichen Länge n ab. Die Tabelle 3.1 liefert hierzu eine Übersicht.[1]

Hetero-assoziative Netzmodelle	**Auto-assoziative Netzmodelle**
(M)Adaline	Hopfield-Netze
Perzeptron	Boltzmann Maschinen
Multilayerperzeptron	-

Tabelle 3.1: Netzklassen & korrespondierende Netztypen

Aus diesen Netztypen ist nun der beste Netztyp zur Prognose von Börsenkursen zu wählen. Zunächst kann die richtige Klasse auf pragmatischer Weise ermittelt werden. Dazu

[1] Vgl. Klaus Peter Kratzer (1990), Seite 33 ff.

kann man sich die Grundidee des zu erstellenden KNN vorstellen. Dieses soll mithilfe von mehreren vorhergehenden Börsenkursen den zukünftigen Börsenkurs prognostizieren. Da es sich bei den zu prognostizierenden Börsenkurs um einen skalaren Wert handelt, ist die Anzahl der Eingabe-Neuronen (und damit die Anzahl der Elemente des Eingabevektors) höher als die Anzahl der Ausgabeneuronen (und damit höher als die Anzahl der Elemente des Ausgabevektors). Somit sind für diese Seminararbeit nur hetero-assoziative Netze von Relevanz.

Aus der Menge der hetero-assoziativen Netze ist nun der Netztyp zu ermitteln, der für die Anwendung am besten geeignet ist. Dafür kann man zunächst Definition 1 betrachten.

Definition 1. *Definition der linearen Separierbarkeit*
Seien X_0 und X_1 zwei Wertemengen im n-dimensionalen euklidischen Raum. Dann sind die Mengen X_0 and X_1 genau dann „linear separierbar“, wenn es $n+1$ Werte $w_1, w_2, .., w_n, k$, gibt, sodass jeder Punkt $x \in X_0$ die Bedingung $\sum_{i=1}^{n} w_i x_i < k$ erfüllt und jeder Punkt $y \in X_1$ die Bedingung $\sum_{i=1}^{n} w_i y_i > k$ erfüllt.

Um das Verständnis der Definition 1 zu erleichtern, kann die Abbildung 3.1 betrachtet werden. Im oberen Graph kann eine Gerade zur Unterteilung der möglichen Ergebnisse in Klassen gelegt werden. Somit ist die AND-Funktion linear separierbar. Im unteren Graph ist dies nicht möglich. Somit ist die XOR-Funktion nicht linear separierbar.

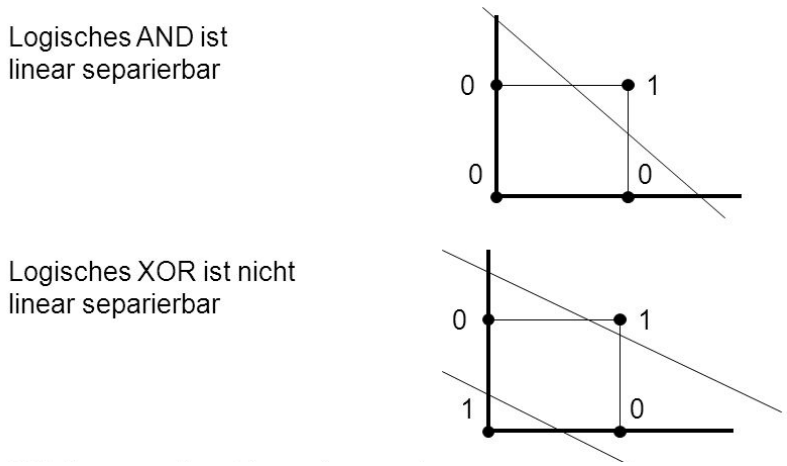


Abbildung 3.1: Bildliche Erläuterung der linearen Separierbarkeit

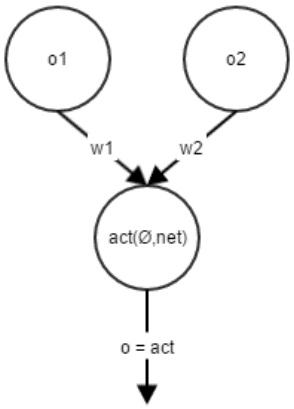
Analog setzt sich dies in Funktionen höherer Dimensionen fort. Ist die Funktion zum Beispiel dreidimensional, erfolgt die Separierung durch eine Ebene.

Nachdem der Begriff der linearen Separierbarkeit erläutert wurde, kann dies als Grundlage für den nächsten Schritt genutzt werden. Zur Auswahl des passenden Netztyps kann nun der Beweis 1 betrachtet werden. Dieser von Minski und Pappert erstellte Beweis belegt, dass einschichtige KNN nur in der Lage sind, linear separierbare Funktionen zu berechnen.[2]

[2] Vgl. Uwe Lämmel und Jürgen Cleve (2008), Seite 212 f.

Beweis 1. *Beweis der eingeschränkten Fähigkeit von KNN anhand des XOR-Problems*

Gegeben sind ein Perzeptron der folgenden Bauart



sowie folgende Rahmenbedingungen:
$w_1 \cdot o_1 + w_2 \cdot o_2 = net$
$f_{act}(o) = id$
$\emptyset = Schwellenwert$

Dann gilt:
$a) w_1 \cdot 0 + w_2 \cdot 0 < \emptyset$ *Bei einem Inputvektor (0,0) liefert der Output 0.*
$b) w_1 \cdot 0 + w_2 \cdot 1 \geq \emptyset$ *Bei einem Inputvektor (0,1) liefert der Output 1.*
$c) w_1 \cdot 1 + w_2 \cdot 0 \geq \emptyset$ *Bei einem Inputvektor (1,0) liefert der Output 1.*
$d) w_1 \cdot 1 + w_2 \cdot 1 < \emptyset$ *Bei einem Inputvektor (1,1) liefert der Output 0.*

Der Widerspruch ergibt sich wie folgt:
$(b + c) : w_1 + w_2 \geq \emptyset \wedge (d) : w_1 + w_2 < \emptyset$

Dieser Beweis kann ebenfalls auf andere nicht linear separierbare Funktionen angewandt werden. Somit steht fest, dass ein einschichtiges Perzeptron nicht in der Lage sein kann, nicht linear separierbare Funktionen zu approximieren.

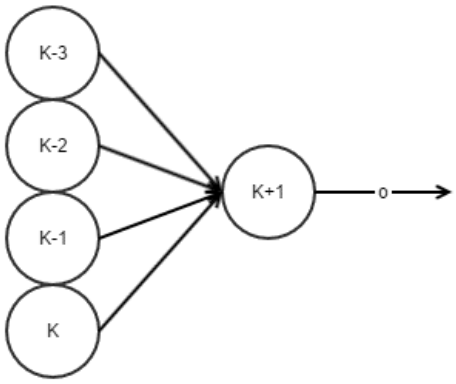
Um festzustellen, ob eine Funktion linear separierbar ist, kann das Konvergenztheorem von Rosenblatt (Theorem 1) hinzugezogen werden[3].

Theorem 1. *Konvergentheorem von Rosenblatt*
Der Lernalgorithmus des Perzeptrons konvergiert in endlicher Zeit, d.h. das Perzeptron kann in endlicher Zeit alles lernen, was es repräsentieren kann.

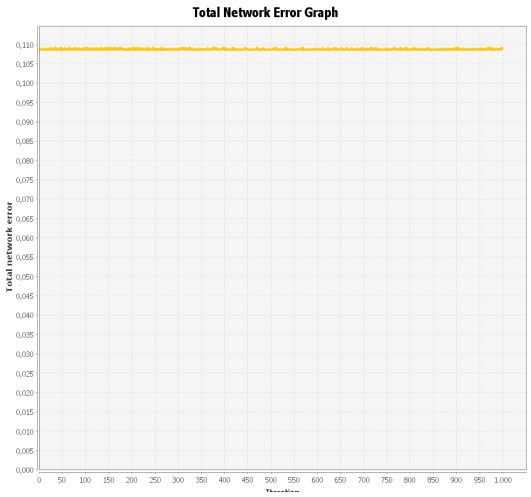
Es ergibt sich also folgende Relation:

Perzeptron konvergiert → Funktion linear separierbar → Perzeptron geeignet. Und analog: Perzeptron konvergiert nicht → Funktion nicht linear separierbar → Perzeptron nicht geeignet.

Auf Basis der oben genannten Relation kann ermittelt werden, ob ein einschichtiges Perzeptron zur Approximation von Börsenkursen geeignet ist. Dafür wurde ein Perzeptron wie in Abbildung 3.2 entwickelt und untersucht, ob dieses Konvergiert.



(a) Test-Perzeptron



(b) MSE des Perzeptrons

Abbildung 3.2: Test-Perzeptron sowie der dazugehörige MSE

Bei Betrachtung des Netzwerkfehlers des Perzeptrons erkennt man, dass das Perzeptron nicht konvergiert. Der Netzwerkfehler des Perzeptrons bleibt über alle Iterationen konstant

[3] Vgl. Uwe Lämmel und Jürgen Cleve (2008), Seite 209

auf einem Niveau von circa $0,10$. Somit steht fest, dass der Börsenkurs eine nicht linear separierbare Funktion darstellt und durch einen Perzeptron nicht approximiert werden kann.

Folglich bleibt nur noch das Multilayerperzeptron als Mögliche Auswahl übrig. Das dieses KNN tatsächlich zur Prognose geeignet ist, belegt das Theorem der universellen Approximation (Theorem 2)[4].

Theorem 2. *Theorem der universellen Approximation*
Jede stetige Funktion kann mittels eines künstlichen neuronalen Netzes mit mindestens einer versteckten Schicht beliebig genau approximiert werden.

Ein Börsenkurs kann prinzipiell jede beliebige (stetige) Funktion annehmen. Durch Theorem 2 ist jedoch sichergestellt, dass das Multilayerperzeptron in der Lage ist, diese Funktion zu approximieren, da ein Multilayerperzeptron als universeller Approximator fungiert.

3.2 Wahl der Topologie

Zur Prognose des Börsenkurses sollen die letzten vier Börsenkurse als Input dienen. Durch diesen Input soll der Börsenkurs am nächsten Tag prognostiziert werden. Folglich gestaltet sich Auswahl der Anzahl an Input-Neuronen (4) sowie Output-Neuronen (1) trivial. Etwas komplexer gestaltet sich jedoch die richtige Dimensionierung der inneren Schicht. Hierbei können aber einige Richtlinien hinzugezogen werden, um die Dimensionierung zu erleichtern:

- Die Anzahl der versteckten Neuronen in der inneren Schicht sollte nicht zu groß gewählt werden, damit das Netz das antrainierte Verhalten nicht „auswendig“ lernt. Sonst kann es nur das bereits trainierte Muster entsprechend verarbeiten. Dies bedeutet ein Verlust der Generalisierungsfähigkeit. Man spricht in diesem Fall von einem Overfitting.
- Die Anzahl der versteckten Neuronen in der inneren Schicht sollte nicht zu klein gewählt werden, da eine gewisse Menge an Neuronen wichtig ist, um sich Regelsätze merken zu können.

[4] Vgl. TU Cottbus (2014)

- Eine grobe Annäherung zur Bestimmung der Obergrenze der Anzahl von Neuronen in der versteckten Schicht liefert die folgende Formel[5]:

$$N_h = \frac{N_d}{10 * (N_i + N_o)} \tag{3.1}$$

N_h ist hierbei die Obergrenze, N_i die Anzahl der Input-Neuronen und N_o die Anzahl der Output-Neuronen. Da 450 Trainingsdaten verwendet werden sollen, bedeutet das für diese Seminararbeit konkret:

$$N_h = \frac{450}{10 * (4 + 1) = 9} \tag{3.2}$$

Somit ergibt sich insgesamt die folgende Topologie aus Abbildung 3.3.

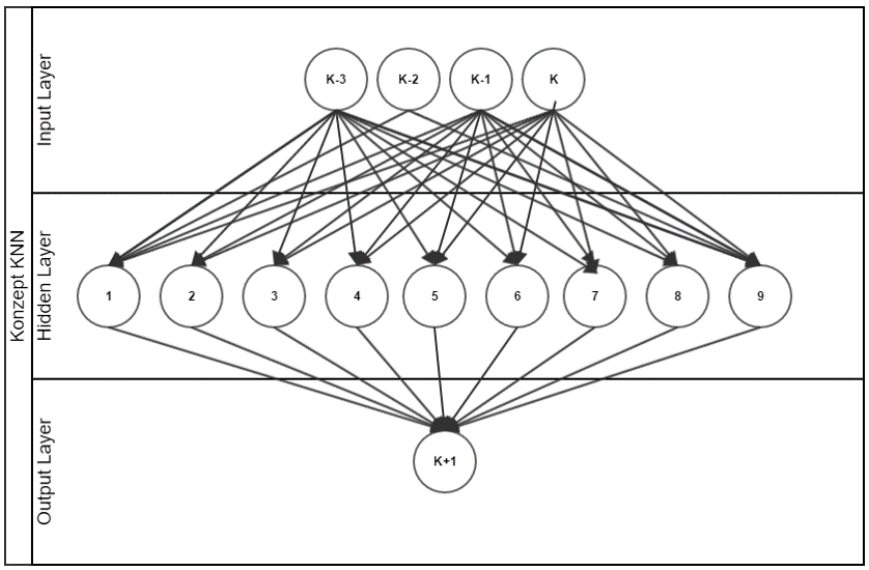


Abbildung 3.3: Grundlegendes Konzept des KNN

Dieses KNN stellt ein solides Grundkonstrukt dar, das in der Umsetzungsphase noch weiter optimiert werden kann.

[5] Vgl. TU Illmenau (2014)

3.3 Wahl des Lernverfahrens

Grundsätzlich existieren drei Lernverfahren, wie ein KNN trainiert werden kann. In diesem Abschnitt werden alle drei Lernverfahren näher vorgestellt und anschließend eine begründete Auswahl des gewählten Verfahrens getroffen[6].

- **Überwachtes Lernen**:
  Beim überwachten Lernen sind sowohl die Eingabedaten als auch die dazugehörigen Ausgabedaten bekannt. Zunächst berechnet das KNN bestimmte Ausgabedaten zu den Eingabedaten. Diese berechneten Ausgabedaten können anschließend mit den tatsächlichen Ausgabedaten verglichen werden. Dieser Fehler wird dann genutzt, um die Verbindungsgewichte des KNN anzupassen. Typische Vertreter dieses Lernverfahrens sind die sogenannten Backpropagation-Lernverfahren.

- **Bestärkendes Lernen**:
  Ähnlich wie das überwachte Lernen, jedoch biologisch motivierter ist das sogenannte bestärkende Lernen. Hier sind dem KNN die Eingabewerte zwar bekannt, aber die dazugehörigen Ausgabewerte nur zum Teil oder gar nicht. Das KNN wird lediglich darüber informiert, ob das Ergebnis richtig bzw. falsch war. Es ist ein sehr zeitaufwändiges Lernverfahren, da es die Gewichte auf Grund der spärlichen Information nur sehr langsam anpassen kann. Dieses Verfahren kann als Mischung aus überwachtes Lernen und nicht überwachtes Lernen gesehen werden.

- **Nicht überwachtes Lernen**:
  Das nicht überwachte Lernen ist biologisch gesehen am plausibelsten. Bei diesem Lernverfahren existieren nur Eingabemuster, jedoch keine erwünschten Ausgaben oder Angaben, ob das Netz die Eingaben richtig oder falsch klassifiziert hat. Stattdessen versucht der Lernalgorithmus selbständig, Gruppen ähnlicher Eingabevektoren zu identifizieren und diese auf Gruppen ähnlicher oder benachbarter Neuronen abzubilden.

Da sowohl die Eingabewerte als auch die Ausgabewerte der zu verwendenden Datensätze bekannt sind, bietet sich das überwachte Lernen an. Verglichen mit den anderen Lernverfahren ist dies die effizienteste Lernmethode. Sie verfügt zwar über kein biologisches Vorbild, dieser Umstand hat aber für diese Seminararbeit keine Relevanz.

[6] Vgl. Uwe Lämmel und Jürgen Cleve (2008)

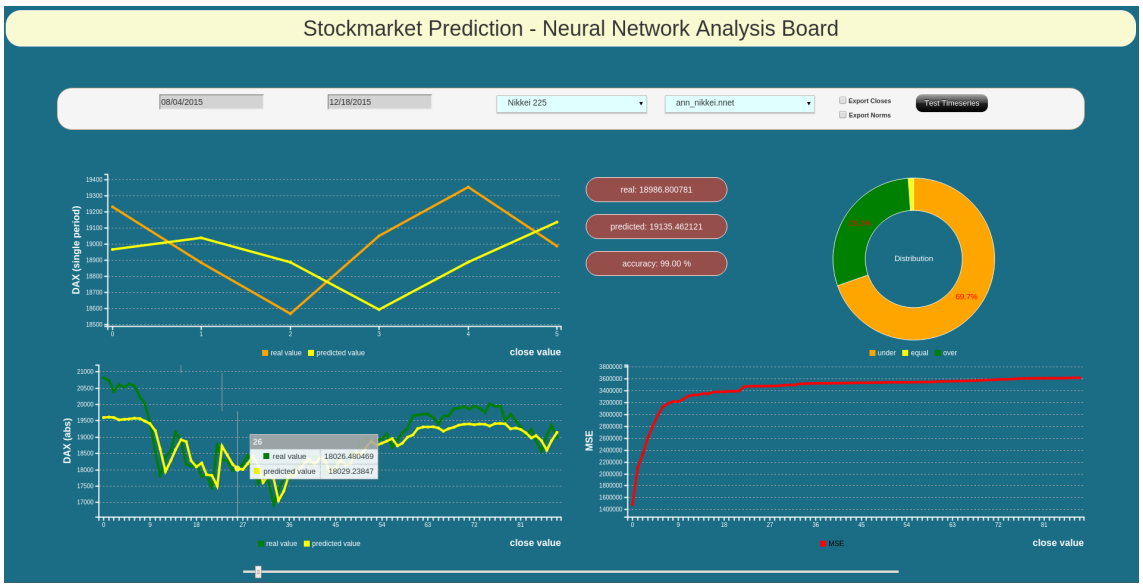
4 Umsetzung

In den folgenden Abschnitten wird auf die Umsetzung der Anwendung und des KNN eingegangen.

4.1 Umsetzung der Anwendung

4.1.1 Oberfläche

Die Oberfläche ist grob eingeteilt in zwei funktional getrennte Bereiche. Der weiße Container beinhaltet ein Formular das der Anfragestellung dient. Hierin werden die Rahmenbedingung der Vorhersage eingegeben. Zum einen Start - und Enddatum, sowie welche Datenmengen, die von Quandl bezogen werden mit welchem Neuronalen Netz getestet werden sollen. Sofern die davon rechts gelegenen Checkboxen angeklickt abgeschickt werden, so werden CSV-Datein mit den entsprechenden Rahmenbedingung in einem konfigurierbarem Verzeichnis gespeichert. *Export Closes* steht hierbei für denormalisierte Börsenschlusswerte, *Export Norms* für die gleichen normalisierten Werte. Mit *Test Timeseries* werden alle Informationen serialisiert und die Anfrage durchgeführt.
Das Ergebnis der Anfrage wird, sofern sie die Fehlerprüfung übersteht, unterhalb des Formulars angezeigt. Die Ausgewählten Werte bleiben hierbei dem Formular hinterlegt. Die Antwort wird in Form von Diagrammen und Displays angezeigt. Alle Informationen werden bzgl. eines zeitlichen Intervalls aktualisiert, somit wird eine Art Echtzeit-Darstellung realisiert.
Die Javascript Bibliothek *C3js* wird bei der Realisierung von vier Diagrammen eingesetzt. Das erste links oben angeordnete visualisiert diskrete *tatsächliche* Werte (real value) und *vorhergesagte Werte* (predicted value) für einen definierten Zeitraum *single period*. Dies dient vor allem der Übersicht, so wird eine Art *Zoom*-Modus realisiert. Das zweite unten links angeordnete Diagramm visualisiert die gleichen Werte, jedoch über den gesamten



Zeitraum *absolute-graph* dar, was einem Resümee bzgl. der Vorhersage zuträglich ist.
Das dritte und letzte Diagramm stellt den Mean-Square-Error-Graph (MSE) der gesamten Zeitreihe dar. Die erste drei Diagramme sind vom gleichen Typ. Eine weitere statistische Auswertung, die oben rechts also c3-donut-Diagramm umgesetzt ist, ist die Klassen-Zuordnung von vorhergesagten Werten in die Klassen *under* (unterhalb vom tatsächlichen Wert), *equal* und *over*. Da in der Realität die Werte fast nie exakt übereinstimmen, wurde ein Konfidenz-Intervall definiert, dass eine gewisse Fehlertoleranz gegenüber der Differenz beider Werte zulässt und equal-Klasse somit faktisch keine leere Menge bildet.
Des weiteren wurden Displays mit eigens geschriebenem CSS-Klassen definiert, die den tatsächliche, den vorhergesagten sowie die *accurancy* (Exaktheit) Anzeigt. Die Accuracy lässt sich als Betrag der Differenz zweier Werte verstehen.
Am unteren Rand ist ein Schieberegler integriert um die Echtzeit-Intervalllänge zu verändern.

Im folgenden wird ein Anfrage-Antwort-Zyklus (Request-Response-Lifecycle) der Umsetzung detailliert beschrieben und erläutert, welche Rolle die bereits vorgestellten Technologien und Überlegungen in die Implementierung eingeflossen sind, sowie wie diese miteinander verknüpft sind.

4.1.2 Dataset und AnnNames ajax

Die Auswahl der Quandl-Datensätze, sowie der .nnet-Datei werden ausschließlich mit Hilfe der Stockmarket-Webapp befüllt. Hierbei werden die Informationen mit AJAX-Aufrufen abgefragt.

4.1.3 Anfrage

4.1.4 Anfragestellung

Das Auslösen einer Anfrage wurde bereits oben geschildert. Der Button-Click wird mit einem jQuery-Event-Listener behandelt. Dieser führt eine Fehlerprüfung der Formulareingabe durch. Wenn ein Enddatum eingegeben wird, dass kleiner dem Startdatum ist, so wird eine entsprechende Fehlermeldung in einem *div*-Tag geladen. Gleiches passiert das, wenn unvollständige Angaben gemacht werden.
Beim Bestehen der Prüfung wird ein AJAX-Aufruf (Asynchronous JavaScript and XML) für den ReSTController der Stockmarket-Webapp ausgeführt. Hierbei wird eine URI konstruiert, die zum einen die Basis-URL des Webservice anspricht, zum anderen die Formulareingabedaten als Parameter enthält. Wenn der Aufruf erfolgreich durchgeführt wird, werden die Hilfsvariablen im Javascript der Visualisierung erneut initialisiert, damit es bei mehreren hintereinander folgenden Ausführungen ohne Seitenaktualisierung keine veränderte Verarbeitungsroutine der Antwortdaten gibt.

4.1.5 Antwort-Erstellung

Die Antwort-Erstellung obliegt der Stockmarket-Webapp, die Visualisierung liefert die Eingabedaten hierzu. Der ReSTController ist das Verbindungsglied im Anfrage-Antwort-Lebenszyklus, aber auch die Komponente, die die Programmablauflogik der Antworterstellung implementiert. Dieser enthält drei Rest-Endpunkte, entsprechend der drei Anfragen, die die Visualisierung per AJAX realisiert.

Listing 4.1: ReSTController Snippet

```
@RestController
@RequestMapping(value = "/stock")
public class ReSTController implements ServletContextAware, ServletConfigAware {

    @Autowired
    private EnvService envService;

    @Autowired
    private DataService dataService;

    \vdots

    @RequestMapping(value= "/data", method = RequestMethod.GET)
    public String data(@RequestParam String startDate,
                       @RequestParam String endDate,
                       @RequestParam String collapse,
                       @RequestParam String stock,
                       @RequestParam String ann,
                       @RequestParam Boolean saveNorm,
                       @RequestParam Boolean saveClose )
                       {
        // Programmablauflogik
        \vdots
        }

   @RequestMapping(value= "/ann_names", method = RequestMethod.GET)
   public List<String> annNames()
   {
   \vdots
   }

   @RequestMapping(value= "/datasets", method = RequestMethod.GET)
   public String datasets()
   {
   \vdots
   }
}
```

Die ReSTController-Klasse ist mit *@RestController* annotiert, was eine Implementierung der *ServletContextAware* und *ServletConfigAware* erfordert. Diese stellen sicher, dass der ReSTController auf den Servlet-Kontext zurückgreifen kann sowie die Servlet-Konfiguration beachtet wird. Diese wird unter anderem dazu verwendet, die Basis-URL zu spezifizieren. Die Rest-Endpunkte werden relativ zu dieser Basis-URL abgebildet. Die Annotation *@RequestMapping* erlaubt die Kennzeichnung von Klassen und Methoden als Zielpunkte gleichermaßen. Entsprechend der HTTP-Verbs können Anfrage-Methoden (RequestMethod)

Quandl-API URL-Parameter	Bedeutung
• order	Reihenfolge der Ergebnismenge
• exclude_column_names	Schließt Header-Information aus
• start_date	Startdatum
• end_date	Enddatum
• collapse	Datenfequenz

Tabelle 4.1: verwendete URL-Parameter - Quandl API

spezifiziert werden. Für den ReSTController wird ein Basis-Endpunkt */stock* definiert.
Das *data*-Methode implementiert die Programmlogik für die Antworterstellung, die annNames-Methode liest die Bezeichnungen der .nnet-Dateien, die *datasets*-Methode die Rest-Endpunkte der Quandl-Datasets zu den entsprechenden Börsenkursen aus der *app.properties* Datei.
Die URI des AJAX-Aufruf des Eingabeformulars wird in der Methodensignatur auf Vollständigkeit und Datentyp-Korrektheit mit der *@RequestParam*-Annotation überprüft. *@RequestParam* fordert einen URL-Parameter gleichnamig zum Variablennamen zu übergeben, alternativ könnte man eine *@Param*-Annoation verwenden, die eine unterschiedliche Namensgebung erlaubt, und sich an der Reihenfolge der Parameter orientiert.
Die Antworterstellung ist auf sechs Serviceklassen aufgeteilt, die wiederum gekapselt in einer Serviceklasse sind. Der ReSTController bindet ausschließlich den diese Klasse, den *Data-Service* ein.
Erster Schritt ist das Einlesen einiger Konfigurationseigenschaften aus der *app.properties*-Datei. Unter anderem das gewünschte Rückgabeformat des Quandl-Datasets sowie die Basis-URL der Quandl-API. Anschließend wird wird aus den zur Vefügung stehenden Informationen eine URI konstruiert und eine Restabfrage der Quandl-Api durchgeführt. Das Ergebnis wird als Liste von Zeichenketten gespeichert. Dabei wird die Daten mit folgenden Spezifikationen angefragt:

Der zweite von zehn Methodenaufrufen, die der *data-Service* organisiert ist das Formatieren der Ergebnismenge in eine für die Neuroph-API passende Struktur. Diese Aufgabe übernimmt der FormService.

Die Konversion beachtet zwei Kriterien, zum einen die Anzahl der Input-und Output-Neuronen, die in der Konfigurationsdatei unter der Eigenschaft *format.period.length* also Summe angegeben ist, sowie die Spaltenposition desjenigen Wertes, der vorhergesagt werden soll, also der Börsenschlusswert. Es werden ausschließlich Börsenschlusswerte und deren Datum

berücksichtigt. Eine Zeile in der konvertierten Liste besteht aus dem Datum des letzten (spätesten) betrachteten Wert innerhalb einer Periode, sowie die Werte der gesamten Periode. Das Ergebnis wird als Listen von Listen von Zeichenketten gespeichert, was einen bessere Verarbeitung für die folgenden Schritte ermöglicht.

Als zweiter Schritt steht die Normalisierung an. Diese Aufgabe wird im *NormDenorm-Service* durchgeführt. Wie in Kapitel 3.3 beschrieben gibt es unterschiedliche Ansätze für die Normalisierungsfunktion, die von der Wahl der Aktivierungsfunktion abhängt. Somit ist es sinnvoll die Formeln für die Normalisierung und Denomalisierung auszulagern um keinen Flaschenhals für künftige Anpassungen oder Experimente zu bilden. Die *normalize*-Methode ließt zuerst die entsprechende Formel, die maximale Anzahl gewünschter Nachkommstellen sowie die Periodenlänge aus der *app.properties* Datei. Die Erfüllbarkeit der Funktion in Kapitel 3.3 hängt von der Möglichkeit ab, das globale Maximum und Minimum zu bestimmen. Damit wird gewährleistet, dass aussschießlich Werte zwischen Null und Eins angenommen werden können. Da unsere Datenmenge eine feste Größe besitzt ist dies kein Problem. Eine überaus nützliches Softwarepaket ist der *ScriptEngineManager*, diese die Art der Modularität erlaubt. Dieser kann einen als Javascript initialisierten *ScriptEngine* verwenden um die eingelesene Funktion (Zeichenkette) entsprechen zu interpretieren. Die private Methode *processFormula* wird für Normalisierung und Denormalisierung verwendet. Diese konvertiert schließlich jeden Wert der Liste über die übergebene Funktion. Anschließend wird die Liste eine Variable im ReSTController zurückgegeben.

Die nächsten zwei Schritte setzt die Anfrage der Checkboxen in die Tat um und speichern, sofern im Formular angeklickt, transformierte Börsensschlussdaten sowie normalisierte Daten als CSV-Dateien in ein Verzeichnis, dessen Pfad in der *app.properties*-Datei festgelegt ist. Der Dateinamen ist autogeneriert und besteht aus dem angefragten Zeitraum sowie dem Namen des Datasets. Notwendig war diese Funktionalität vor allem für das Training der Neuronalen Netze, das im Neuroph Studio durchgeführt wurde, also außerhalb der Anwendung.

Im fünften Schritt wird das Neuronale Netz mit den normalisierten Daten getestet. Der *Ann-Service* implementiert hierzu eine Methode *testAnn*. Um die Erweiterbarkeit auch für künftige Umsetzungen von Neuronalen Netzen möglichst einfach zu gewährleisten, so z.B. eine andere Topologie zu verwenden, die auf mehr Eingangsneuronen setzt, wird die Periodenlänge aus der *app.properties*-Datei gelesen. Der Testalgorithmus wird allgemeingültig gehalten, und funktioniert für alle Neuronalen Netztypen, die ein Ausgangsneuron besitzen. An dieser Stelle wird das erste mal die Neuroph-Core-Bibliothek innerhalb der Stockmarket-Webapp verwendet. Zuerst wird eine *Dataset*-Instanz mit der Anzahl der Input- und Outputneuronen initialisiert. Anschließend wird die normalisierte Liste zeilenweise iteriert. Jede Zeile wird in zwei Arrays von *double*-Werten geteilt, ein Input- und Output-Array und anschließend dem DataSet-Objekt hinzugefügt. Als nächstes wird das Multi-Layer-Perzeptron über den hinterlegten Dateipfad geladen. Hierzu bietet Neuroph die *createFromFile*-Methode. Anschließend wird das DataSet-Objekt in eine Liste von *DataSetRow*-Instanzen gesplittet. Eine DataSetRow ist das wichtigste Attribut der DataSet-Klasse und repräsentiert einen Testschritt. Für jedes Input-Array in einer solche DataSetRow wird ein Ausgabewert errechnet, welcher zu einer Liste aus Zeichenketten hinzugefügt wird. Nachdem die Iteration abgeschlossen ist und die Menge der vorhergesagten Werte vollständig kalkuliert wurde, wird die Liste mit der Testdatenliste erweitert. Diese Aufgabe übernimmt der entsprechend *Form-Service*.

Der sechste Schritt ist die Denormalisierung der gewonnen Ergebnisliste. Wie oben bereits erwähnt wird die *processFormula*-Methode des *NormDenorm-Serive* lediglich mit anderen Parameterwerten ausgeführt. Es wird die Denormalisierungsfunktion, die maximale Anzahl der Nachkommastellen sowie die Periodenlänge aus der *app.properties*-Datei geladen und übergeben. Eine Funktionalität die derzeit nicht benötigt wird, aber dennoch implementiert ist, stellt die Möglichkeit dar Spalten der List von den Operationen auszuschließen. Wären zu diesem Zeitpunkt bereits denormalisierte Werte vorhanden, wäre dies deshalb kein Problem.

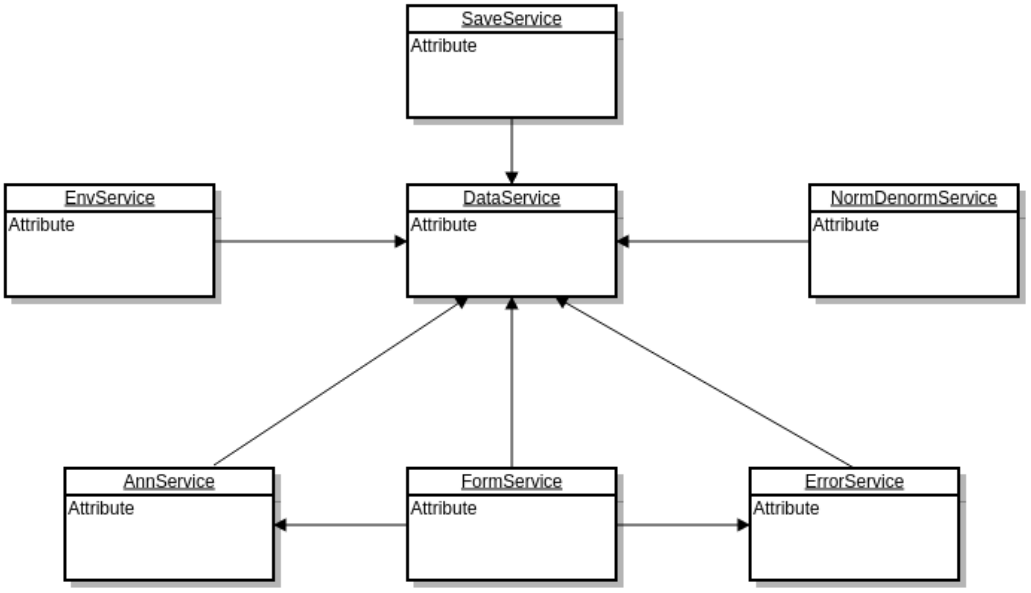
Der siebte und letzte Schritt, der eine Modifikation der Liste vollführt ist das hinzufügen von Fehlermaßen. Hierfür ist der *Error-Service* zuständig. Grundidee des Error-Service ist, eine Methode pro Fehlermaß umzusetzen, wobei jede Methode eine Liste von Zeichenketten zurückgibt. Der *Data-Service* greift ausschließlich auf eine im zugedacht Methode zu, die für alle diese Fehlermethoden zuständig ist. Hier ist das die *getAllErrorMessures*-Methode.

Als erstes wird geprüft ob ein Fehlermaß überhaupt hinzugefügt werden soll. Diese Informationen sind in der *app.properties*-Datei gelistet. Für jedes Fehlermaß gibt es einen Eintrag mit dem Präfix *error.avail*, der als *Boolean*-Wert interpretiert wird. Je nach dem ob diese wahr sind, werden die entsprechenden Fehlermaße berechnet. Die Ergebnislisten dieser Methoden werden zu einer großen Ergebnisliste hinzugefügt und anschließend mit der *addCols*-Methode des *FormService* hinzugefügt. Für jedes weitere Fehlermaß muss also lediglich ein Eintrag in der app.properties sowie eine Methoden Implementierung umgesetzt werden, welche in der Hauptmethode entsprechend ausgeführt wird. Somit können die Ziele von Funktionsumfang, Flexibilität und Performance harmonieren.
Der Mean-Square-Error (MSE) wird bereits in ..... erklärt. Des Weiteren wird hier die *Accuracy* (Exaktheit) sowie die *Distribution* (Klassenzuteilung) berechnet. Das hierfür benötigt Konfidenzintervall bleibt ebenfalls als Eigenschaft in der Konfigurationsdatei hinterlegt.

Abschließender Schritt zur Antworterstellung ist die Konversion der generierten Liste in einen geeignete Struktur, damit die Visualisierung eine möglichst einfache Handhabe bei dem Verarbeiten der Daten hat. Die Aufgabe übernimmt sinngemäß der *FormService*. Die Methode *parseResponseList* gibt ein *JSONArray*-Objekt zurück. Dieses wird iterativ befüllt, indem jede Liste innerhalb der Ergebnisliste in einer fest definierte Struktur innerhalb einer jeweils neuen JSONObject-Instanz abgebildet wird. Zu jedem Wert gibt es einen textuellen Schlüssel, somit kann auf einen Schleifendurchlauf eines JSONObjekt innerhalb des Arrays in der Visualisierung verzichtet werden und der entsprechende Wert unmittelbar angesprochen werden. Der *Data-Service* wandelt das JSONArray-Objekt in einen String. Dieser wird der Visualisierung nun als Antwortnachricht übergeben.

Listing 4.2: Beispiel - JSONArray

```
[
   {
      "date":"2015-12-07",
      "acc":"100.00",
      "input":[
         "11261.240234",
         "11190.019531",
         "10789.240234",
         "10752.099609"
      ],
      "dist":{
         "equal":"0",
         "over":"0",
         "under":"1"
      },
```




```
        "ro":"10886.089844",
        "mse":"454.036617",
        "po":"10864.781709"
    }
]
```


4.1.6 Antwort-Verarbeitung

Die Zeichenkette des JSON-Arrarys wird in einer globalen Variable zwischengespeichert. Da alle Diagramme und Displays einem einheitlichen Takt folgen sollen, wird die Initialisierung der Graphikkomponenten, sowie deren gesamter Aktualisierungszyklen von einer Funktion *setDeceleratingTimeout* aufgerufen. Nach einer Verzögerung, die durch eine globale Intervalllängendefinition beeinflusst wird und über die JavaScript-spezifische Methode *window.setTimeout()* realisiert wird, wird jeweils die *processValues*-Funktion abgearbeitet. In dieser wird jeweils eine JSONObject-Repräsentation verarbeitet und für die Anzeige für das kommende Intervall vorbereitet sowie anschließend an die Darstellungsfunktionen (Rendering-Funktionen) übergeben. Grob gesagt werden entsprechende temporäre Arrays mit den Werten der Repräsentation befüllt und übergeben. Neben den Darstellungen wird

der Schieberegler, der den Wert der globalen Variable *Interval* in Echtzeit verändert, in die Visualisierungslandschaft integriert.

4.1.7 Konfigurationsdatei der Stockmarket-Webapp

Listing 4.3: app.properties - Konfigurationsdatei der Stockmarket-Webapp

```
## filepath where ann files are stored
media.source.base=/home/bthofrichter/Schreibtisch
media.source.ann=/source/ann
media.source.analyzed=/source/analyzed/

quandl.api.baseurl=https://www.quandl.com/api/v3/datasets

quandl.api.key=KsHDYzZK6uyyynwQNS7p

## response format by quandl API
quandl.data.dataset.format=csv
quandl.data.dataset.order=asc
quandl.data.dataset.header=true
quandl.data.dataset.close.pos=4

##
format.period.length=5
format.data.included=true
format.data.norm.precision=12
format.data.denorm.precision=6
format.data.error.acc.precision=2

dist.conf=10

## quandl datasets
quandl.dataset.dax=/YAHOO/INDEX_GDAXI
quandl.dataset.nikkei_225=/YAHOO/INDEX_N225
quandl.dataset.djia=/YAHOO/INDEX_DJI


## quandl dataset names for visualisation
quandl.dataset.name.dax=DAX,dax
quandl.dataset.name.nikkei_225=Nikkei 225,nikkei_225
quandl.dataset.name.djia=Dow Jones (DJIA),djia

## X := current value
## MIN := global minimum
## MAX := global maximum
normalize.formula=((X-MIN)/(MAX-MIN))*0.8+0.1
```

```
denormalize.formula=((X-0.1)/0.8)*(MAX-MIN)+MIN

## ERROR-AVAILABILITY
error.avail.dist=true
error.avail.mse=true
error.avail.acc=true
```

4.1.8 Maven POM der Anwendung

Die in ListingPOM zeigt die Grundstruktur der POM.xml der Anwendung. Diese enthält alle wichtigen Informationen um die Anwendung in der Beschriebenen Systemlandschaft zu bauen.
Kurze Erläuterung zum Aufbau der POM.xml:
Project-Knoten: Enthält alle anderen Knoten. Die Deklaration in Zeile 1–4 verweist auf eine entsprechende Namensraum und Schema-Defintion. modelVersion-Knoten: Die Versionsnummer muss mit den Versionsinformationen im Proect-Knoten übereinstimmen. In Zeile 7-9 sind die typischen 3 Knoten *groupId*, *artifactId* und *version* zur Beschreibung einer Dependency. Da diese Knoten nicht mit dem dependency-Knoten eingeschlossen ist, bezieht sich die Dependency-Information auf sich selbst. Sofern die Stockmarket-Webapp also in einem anderen Projekt als Softwarepakte hinzugefügt werden soll, müsste man diese Knoten einfach in den dependency-Knoten aufführen.

In Zeile 11 beschreibt der packaging-Knoten das Zielformat der kompilierten Anwendung. Für eine Spring-Webapp eignet sich ein Webarchiv (war).

In Zeile 13-17 werden genaue Angaben zur Systemlandschaft, genauer zum Tomcat-Server und zum JDK (Java Development Kit) gemacht. Die Stockmarket-Webapp setzt auf Java 8 und einen Tomcat 7.0.64. Diese Properties sind zur Absicherung gedacht. Prinzipiell ist es auch möglich ein Java 7 oder eine andere Tomcat Version einzusetzen. Da aber keine vollständige Prüfung jeglicher Versionen durchgeführt wurde und auch wirtschaftlich unsinnig wäre einigt man sich auf eine spezielle Version, mit der alle Funktionalität der Anwendungen getestet werden.

Der Parent-Knoten in den Zeilen 19-23 kann als *higher-lever*-Dependency gesehen werden. Diese bindet das Spring-Boot-Start-Parent Paket ein.

In den Zeilen 25–31 werden Dependencies nach bereits beschriebenem Format aufgeführt.

Zeile 35-40 listet Remote-Repositories. Sinnvoll ist eine solche Definition vorallem dann, wenn mehrere Abhängigkeiten hieraus benötigt werden.

Der build-Knoten Knoten der in Zeile 44-68 beschrieben ist, ist optional und nimmt auf den Maven-Build-Zyklus einfluss. Für die Stockmarket-Webapp ist es notwenig eine Dependency, die sich nicht per Standard-Deklaration in Kontext einfügen lässt, auf diesem Weg zu integrieren. Hierbei handelt sich das Neuroph-Core-2.9.jar Softwarepaket. Da die Neuronalen Netze mit dem Neuroph-Studio, das mit dieser Version arbeitet, erstellt und trainiert wurde, war es notwendig die gleiche Version in der Anwendung zu verwenden. Da das offizielle Remote-Repository allerdings diese Version noch nicht bereitstellt musste die JAR-Datei während der *install*-Phase des Maven-Builds berücksichtigt wird. In dieser Phase werden die Abhängigkeiten in das lakale Repository kopiert, was für das automatische Deployment im Tomcat-Server führt (bei entsprechender Konfiguration).
Dank Maven kann der Build-Prozess der Stockmarket-Webapp dynamisch, übersichtlich und effizient gestaltet werden.

Listing 4.4: POM.xml Snippet - Stockmarket-Webapp

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4                               http://maven.apache.org/maven-v4_0_0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6      <groupId>de.soco.stockmarket</groupId>
7      <artifactId>stockmarket-app</artifactId>
8      <version>1.0</version>
9      <packaging>war</packaging>
10     <properties>
11         <maven.compiler.source>1.8</maven.compiler.source>
12         <maven.compiler.target>1.8</maven.compiler.target>
13         <tomcat.version>7.0.64</tomcat.version>
14     </properties>
15     <parent>
16         <groupId>org.springframework.boot</groupId>
17         <artifactId>spring-boot-starter-parent</artifactId>
18         <version>1.3.0.M5</version>
```

```
19      </parent>
20      <dependencies>
21          <dependency>
22              <groupId>org.springframework.boot</groupId>
23              <artifactId>spring-boot-starter-web</artifactId>
24          </dependency>
25              ...
26      </dependencies>
27      <repositories>
28          <repository>
29              <id>spring-releases</id>
30              <url>https://repo.spring.io/libs-release</url>
31          </repository>
32          ...
33      </repositories>
34  \vdots
35      <build>
36          <finalName>Stockmarket-Webapp</finalName>
37          <plugins>
38              <plugin>
39                  <groupId>org.apache.maven.plugins</groupId>
40                  <artifactId>maven-dependency-plugin</artifactId>
41                  <version>2.10</version>
42                  <executions>
43                      <execution>
44                          <id>copy-installed</id>
45                          <phase>install</phase>
46                          <goals>
47                              <goal>copy-dependencies</goal>
48                          </goals>
49                          <configuration>
50                              <overWriteReleases>false</overWriteReleases>
51                              <overWriteSnapshots>false</overWriteSnapshots>
52                              <overWriteIfNewer>true</overWriteIfNewer>
53                              <outputDirectory>${project.build.directory}/Stockmarket-Webapp/WEB-INF/lib</outp
54                          </configuration>
55                      </execution>
56                  </executions>
57              </plugin>
58          </plugins>
59      </build>
60  </project>
```

4.2 Umsetzung des künstlichen neuronalen Netzes

In diesem Abschnitt wird beschrieben, wie das KNN aus der Konzeptionsphase (siehe Abbildung 2.3) umgesetzt wurde. Zur Umsetzung wurde die Anwendung „Neurophstudio“ verwendet. Diese ist ein Teil des Neuroph-Frameworks und erlaubt das Erstellen, Trainieren und Testen von KNN mittels einer graphischen Oberfläche. Das erstellte KNN kann anschließend mittels einer Library in einer Java-Anwendung eingebunden werden.

Nachdem das grundlegende KNN in der Anwendung angelegt wurde, musste dieses noch trainiert und anschließend getestet werden. Für diesen Vorgang sind Trainings- sowie Testdaten nötig. Die benötigten Daten konnten als Excel-Datei von der nachfolgenden Webseite bezogen werden: *http://www.quandl.com*. Es wurden die letzten 600 Börsenkurse des DAX extrahiert und anschließend in 2 Datensätze aufgeteilt: In einem Trainingsdatensatz bestehend aus 450 Trainingsdaten sowie in einem Testdatensatz bestehend aus 150 Testdaten. Da diese Datensätze noch nicht normalisiert waren, die Daten jedoch in normalisierter Form für das KNN zur Verfügung stehen müssen, wurden diese mit der folgenden Formel normalisiert:

$$N_h = \frac{A) - min(A}{max(A) - min(A)} \cdot 0,8 + 01 \tag{4.1}$$

Wobei A den Datensatz als Matrix repräsentiert.

Damit wurde sichergestellt, dass sich alle Werte der Datensätze im Intervall $[0, 1]$ befinden. Die Multiplikation mit $0,8$ sowie die Addition mit $0,1$ soll Extremwerte abmildern.

Nachdem alle Komponenten für die Erstellung eines fertigen KNN vorhanden waren, konnte mit dem Training begonnen werden. Dafür wurden 200.000 Trainingszyklen gestartet. Als Lernverfahren wurde das Backpropagation-Verfahren mit einer Lernrate von $0,7$ benutzt und als Aktivierungsfunktion eine Sigmoide Funktion. Nachdem das Training abgeschlossen war, wurde das KNN noch entsprechend mit dem Testdatensatz getestet. Dabei haben sich jeweils die folgenden Werte ergeben:

Durchlauf	MSE
Trainingszyklus	0,001048
Testzyklus	0,002134

Tabelle 4.2: Die Trainings- und Testergebnisse des Grundnetzes

Dieses KNN bildet nun die Grundlage für weitere Optimierungsmaßnahmen.

4.3 Optimierung des künstlichen neuronalen Netzes

Nachdem das Grundmodell des KNN erstellt wurde, ist dieses noch weiter optimiert worden. Darauf wird nun in den Unterabschnitten 4.3.1, 4.3.2 sowie 4.3.3 genauer eingegangen.

4.3.1 Optimierung der Topologie

Das im Abschnitt **??** erstellte KNN wird in diesem Abschnitt hinsichtlich der verwendeten Topologie optimiert. Dabei werden sukzessive Neuronen in der Zwischenschicht hinzugefügt bzw. entfernt und für jeden Trainings- und Testverlauf der MSE (Mean Squared Error) notiert. Auch wird jede Topologie einmal mit und einmal ohne ein Bias-Neuron trainiert und getestet. Die Topologie mit dem geringsten MSE im Testverlauf wird dann übernommen. Die Ergebnisse dieser Optimierung können aus der Tabelle 4.3 entnommen werden. Der Buchstabe (B) steht dabei für das Bias-Neuron.

Topologie	Training-MSE	Test-MSE	Training-MSE (B)	Test-MSE (B
4-03-1 (B)	0.0011562	0.002569	$9.449 \cdot^{-4}$	0.001788
4-05-1 (B)	0.001062	0.002879	$9.598 \cdot^{-4}$	0.001799
4-07-1 (B)	0.001090	0.001784	$9.407 \cdot^{-4}$	0.001781
4-09-1 (B)	0.001048	0.002134	$9.488 \cdot^{-4}$	0.0024436
4-11-1 (B)	0.001022	0.001785	$9.760 \cdot^{-4}$	0.0033215
4-13-1 (B)	0.001002	0.001787	$9.906 \cdot^{-4}$	0.004067

Tabelle 4.3: Jeweilige Topologien & korrespondierende MSE

Wie aus der Tabelle 4.3 zu erkennen, liefert eine Topologie mit 4 Input-Neuronen, 7 versteckten Neuronen, ein Bias-Neuron sowie ein Output-Neuron die besten Testergebnisse. Die Start-Topologie aus der primären Umsetzung wird nun durch diese Topologie ausgetauscht.

4.3.2 Wahl der optimalen Transferfunktion

Nachdem die Topologie des KNN optimiert wurde, ist noch die Transferfunktion optimiert worden. Hierbei wurde das Netz einmal mittels einer sigmoiden Funktion und anschließend nochmals mit der Tanh-Funktion trainiert und getestet. Dabei ist anzumerken, dass die Tanh-Funktion lediglich einen Sonderfall einer sigmoiden Funktion darstellt (Das wird klarer, wenn man bedenkt, dass „Sigmoid“ mit „S-Förmig“ übersetzt werden kann). Die Abbildung 4.1 zeigt nochmals die Bauart der beiden Funktionen auf.

(a) Sigmoid (b) Tanh

Abbildung 4.1: Die Sigmoide Funktion und die Tanh Funktion im Vergleich

$$(a)\ f(x) = \frac{1}{1+e^{-cx}} \qquad (b)\ f(x) = tanh(x) \tag{4.2}$$

Aus der Tabelle 4.4 können die Ergebnisse dieses Optimierungsschrittes entnommen werden.

Transferfunktion	Training-MSE	Test-MSE
Sigmoid	$9.406 \cdot 10^{-4}$	0.001767
Tanh	0.0103333	0.044330

Tabelle 4.4: Jeweilige Transferfunktionen & korrespondierende MSE

Man erkennt, dass es sich bei der bisher genutzten sigmoiden Funktion bereits um die beste Lösung handelt. Folglich wurde das KNN in dieser Hinsicht nicht weiter optimiert und die ursprüngliche Funktion wurde belassen.

4.3.3 Wahl der optimalen Lernregel

Als letzten Schritt wurde die Lernregel des KNN optimiert. Innerhalb des Verfahrens der überwachten Lernens existieren mehrere Lernregeln, um das Netz zu trainieren. Die bekannteste Lernregel ist die Backpropagation-Lernregel. Diese Regel gibt es in mehreren Variationen. In dieser Seminararbeit werden zum einen das Grundverfahren sowie einige Variationen, namentlich das „Momentum Backpropagation“ sowie das „Resilient Backpropagation“ beschrieben und untersucht. Anschließend wird das für die Anwendung am besten geeignete Verfahren ausgewählt[7].

- **Backpropagation**:
  Dies ist das klassische Fehlerrückführungsverfahren zum Anpassen der Verbindungsgewichte. Die Gewichtsveränderung erfolgt durch ein Fehlersignal, dass aus der Abweichung von tatsächlicher und prognostizierter Ausgabe berechnet wird. Die Gewichtsveränderung erfolgt hierbei schichtweise von den Ausgangs-Neuronen bis zu den Eingangs-Neuronen.
- **Momentum Backpropagation**:
  Dieses Verfahren fügt dem klassischen Verfahren einen Trägheitsterm hinzu, indem die Gewichtsveränderung zum Zeitpunkt $t-1$ berücksichtigt wird. Dieser Term kann einen Wert zwischen 0 und 1 annehmen. Umso größer dieser Term ist, umso stärker

[7] Vgl. Uwe Lämmel und Jürgen Cleve (2008), S. 225 f.

wir die vorhergehende Gewichtsveränderung berücksichtigt. Durch diesen Trägheitsterm wird die Wahrscheinlichkeit verringert, dass das KNN beim Training in ein lokales Minimum oszilliert und sich somit nicht weiter dem Idealwert approximieren kann. Auch die Wahl der Lernrate gestaltet sich hier weniger kritisch.

- **Resilient Propagation**:
  Resilient heißt Federnd. Dieses Verfahren nutzt das Vorzeichen das Gradienten zum Zeitpunkt t und entscheidet anhand dessen, ob das Gewicht vergrößert oder verkleinert werden muss. Der Betrag der Gewichtsveränderung wird jedoch unabhängig von der Richtung der Gewichtsänderung ermittelt. Dadurch werden die typischen Probleme klassischer Gradientenabstiegsverfahren, wie sie beim klassischen Backpropagation sowie beim Momentum Backpropagation genutzt werden, gemindert.[8].

  Die Formeln für Resilient Propagation lauten wie folgt:

$$\Delta w_{ij} = \begin{cases} -\Delta_{ij} & falls S(t) > 0 \\ +\Delta_{ij} & falls S(t) < 0 \\ \pm 0 & sonst \end{cases} \tag{4.3}$$

$$\Delta_{ij} = \begin{cases} \Delta_{ij}(t-1) \cdot n^{+} & falls S(t-1) \cdot S(t) > 0 \\ \Delta_{ij}(t-1) \cdot n^{-} & falls S(t-1) \cdot S(t) < 0 \\ \Delta_{ij}(t-1) & sonst \end{cases} \tag{4.4}$$

  Solange das Vorzeichen des Gradienten negativ ist, wird das Vorzeichen des Gewichtes ebenfalls beibehalten und die Schrittweite und somit das Gewicht um einen konstanten Wert n^{+} vergrößert. Somit können Plateaus besser überwunden werden. Ändert sich das Vorzeichen des Gradienten von negativ auf positiv(was bedeutet, dass ein Minimum übersprungen wurde), so wird das Vorzeichen geändert und die Schrittweite um einen fixen Faktor n^{-} verringert. Somit werden Oszillationen verhindert.

  Die Abbildung 4.2 stellt das Vorgehen des Resilient Propagation Algorithmus grafisch dar[9].

[8] Vgl. Valentina Stellwag (2009), S. 71
[9] Vgl. Geith (2006), S. 71

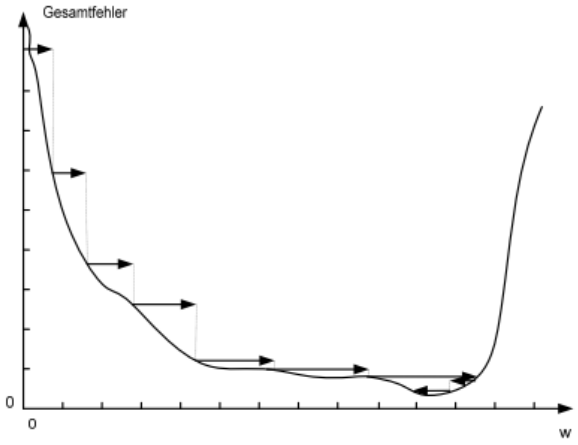


Abbildung 4.2: Visualisierung des Resilient Propagation Algorithmus

Allen drei Lernregeln ist gemein, dass zur Bestimmung des Fehlers zwischen der prognostizierten und tatsächlichen Ausgabe der MSE benutzt werden kann. Die MSE-Formel würde in dem konkreten Fall der Anwendung wie folgt lauten:

$$\frac{1}{2}\sum_{i=1}^{n}(KT_i - KV_i)^2 \tag{4.5}$$

Wobei n für die Anzahl der Daten im Datensatz steht, KT_i für den tatsächlichen Ausgabewert eines Datum i steht und KV_i für den korrespondierenden prognostizierten Ausgabewert eines Datums i steht.

In der Tabelle 4.5 kann das Ergebnis der Trainings- und Testdurchläufe mit den jeweiligen Lernregeln betrachtet werden.

Lernregel	Training-MSE	Test-MSE
Backpropagation	$9.325 \cdot 10^{-4}$	0.001636
Momentum Backpropagation	$9.109 \cdot 10^{-4}$	0.001608
Resilient Propagation	$8.89 \cdot 10^{-4}$	$9.406 \cdot 10^{-4}$

Tabelle 4.5: Lernregeln & jeweilige MSE

In der Regel liefert Resilient Propagation sehr gute Ergebnisse, dies ist auch hier der Fall. Wie man erkennen kann, ist das Resilient Propagation Verfahren hier den anderen überlegen. Folglich wurde das KNN entsprechend optimiert und Resilient Propagation als Lernregel eingesetzt. Da es sich bei Resilient Propagation um eine adaptive Lernregel handelt und bei der Berechnung keine Lernrate benutzt handelt, muss diese auch nicht angegeben werden.

4.4 Die endgültigen künstlichen neuronalen Netze

Nachdem das KNN zur Prognose des DAX erstellt und optimiert wurde, wurden diese Schritte in analoger weise für die KNN zur Prognose des Nikkei sowie zur Prognose des Dow Jones wiederholt. Es stellte sich heraus, dass das optimale KNN für den DAX ebenfalls das Optimale KNN für den Nikkei und den Dow Jones darstellt. Das Endgültige Netz sowie dessen Parameter können aus der Tabelle 4.6 sowie aus der Abbildung 4.3 entnommen werden.

Topologie	4-7-1 mit Bias
Lernregel	Resilient Propagation

Tabelle 4.6: Die jeweiligen Börsenkurse & deren Endwerte

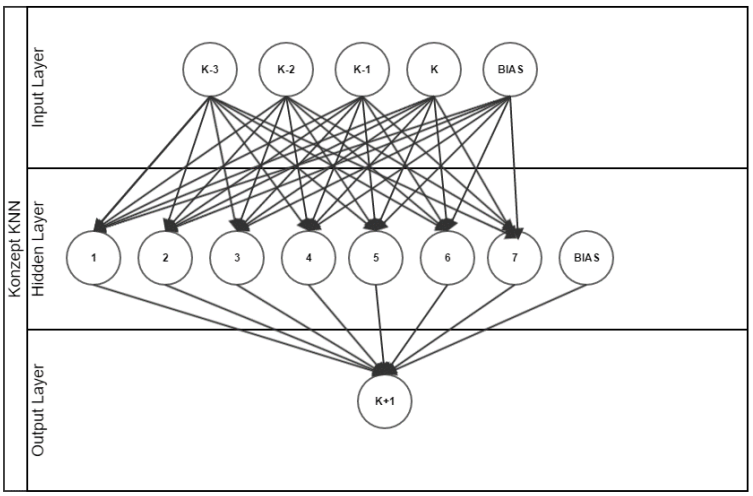


Abbildung 4.3: Das endgültige KNN für alle Börsenkurse

Die Tabelle 4.7 zeigt die Trainings- sowie Testergebnisse der implementierten und optimierten KNN nach jeweils 200.000 Trainingszyklen.

Börsenkurs	Training-MSE	Test-MSE
DAX	$4.252 \cdot 10^{-5}$	$4.820 \cdot 10^{-5}$
Nikkei	$1.350 \cdot 10^{-5}$	$4.520 \cdot 10^{-5}$
Dow Jones	$6.672 \cdot 10^{-5}$	$2.820 \cdot 10^{-4}$

Tabelle 4.7: Die endgültigen Parameter für alle KNN

5 Analyse der künstlichen neuronalen Netze

Die Banken - und Finanzkrise der letzten Jahre beherrschte die Gedanken von Anlegern, Sparern, Investoren, Volk und Politikern. Als Startzeitpunkt kann der 9. August 2007 gesehen werden, wobei bis heute Auswirkungen im Finanzsystem zu spüren sind und kein klarer Endzeitpunkt, auch auf Grund der globalen Reichweite, festgemacht werden. Eines ist klar, lineare Zusammenhänge zwischen Börsenkursen können in einer solch einer Extremsituation ausgeschlossen werden. Dieses Kapitel hat nicht nur entscheidende Zeiträume dieser Finanzkrise im Bezug auf die Vorhersagegenauigkeit von Börsenkursschlusswerten untersucht, sondern auch den Zeitraum in dem der Vorfall des Kernkraftwerks in Fukushima statt fand, betrachtet.

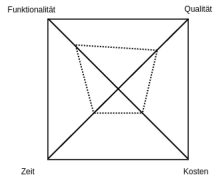


Abbildung 5.1: Abb - nicht lineare

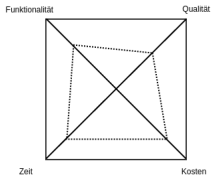


Abbildung 5.2: Abb - lineare

6 Zusammenfassung und Fazit

Die KNN dieser Seminararbeit liefern zwar gute Approximationen, jedoch sind diese für praktische Zwecke noch nicht ausreichend. Dafür sind primär zwei Ursachen verantwortlich. Zunächst ist zu berücksichtigen, dass die KNN in dieser Anwendung ein abgeschottetes System bilden. Das bedeutet, dass diese nicht in der Lage sind, auf einschneidende Ereignisse (wie z.B. Terroranschläge) angemessen zu reagieren, obwohl solche einen großen Einfluss auf den Börsenkurs haben können. Eine Erweiterung um diese Eingaben wäre prinzipiell möglich, jedoch sehr aufwändig. Ein weiteres Manko der in dieser Seminararbeit erstellten KNN ist das Fehlen von nichtlinearen Zusammenhängen. Es wurden lediglich die letzten vier Börsenkurse zur Prognose des darauffolgenden Kurses verwendet. Erweiterungen wie z.B. durch den Leitzins oder Kurse anderer Börsen als Input würden die Prognosefähigkeit wahrscheinlich stark steigern, denn genau hier erweisen sich KNN als besonders effektiv.

Auf dem Markt befinden sich bereits zahlreiche Anbieter von sehr ausgefeilten Anwendungen auf Basis von KNN, die Börsenkurse prognostizieren. Diese liefern tatsächlich recht genaue Ergebnisse, die auch in der Praxis vom Nutzen sein können. Der Preis zur Nutzung dieser Anwendungen ist jedoch recht hoch. So stellt sich die Frage, ob der Nutzen tatsächlich höher ist als die Kosten zu Nutzung eines KNN dieser Anbieter.

Zusammenfassend kann festgehalten werden, dass die Prognose von Börsenkursen mittels eines KNN möglich ist, jedoch mit sehr viel Aufwand verbinden ist, wenn man praxistaugliche Ergebnisse erzielen möchte. Auch sollte ein KNN nie als alleiniges Prognoseinstrument, sondern immer nur als Ergänzung zu anderen Prognoseinstrumenten eingesetzt werden.

Literatur- und Quellenverzeichnis

[1] Klaus Peter Kratzer, *Neuronale Netze – Grundlagen und Anwendungen*. Hanser Verlag in München, 1990.

[2] Uwe Lämmel und Jürgen Cleve, *Künstliche Intelligenz, 3. Auflage*. Hanser Verlag in München, 2008.

[3] TU Cottbus, *Backpropagation-Netze*. http://vieta.math.tu-cottbus.de/ kolb/ml-nn/node5.html, 2014.

[4] TU Illmenau, *Experimentelle Modellbildung mit einem Multilayer Perceptron*. https://www.tu-ilmenau.de/fileadmin/public/systemanalyse/fnc-3.pdf, 2014.

[5] Valentina Stellwag, *Steuerung eines mobilen Roboters mit Hilfe von Neuronalen Netzen*. http://fbim.fh-regensburg.de/ saj39122/Diplomarbeiten/ValentinaStellwag/diplomarbeit.pdf, 2009.

[6] A. Geith, *Künstliche neuronale Netze zur Missbrauchserkennung in Mobilfunknetzen auf Basis von Verbindungsdaten*. http://michael.hahsler.net/stud/done/geith/geith-dipl.pdf, 2006.