Diplomarbeit

Entwicklung eines Verfahrens zur Identifikation von Spam-Mail mit künstlichen neuronalen Netzen

ausgeführt zum Zweck der Erlangung des akademischen Grades eines
„*Diplom-Ingenieurs für technisch-wissenschaftliche Berufe*“
am Masterstudiengang Telekommunikation und Medien
der Fachhochschule St. Pölten

unter der Erstbetreuung von
Dipl. Ing. (FH) Fritz Grabo

Zweitbegutachtung von
Dipl. Ing. Grischa Schmiedl

ausgeführt von
Gerald Weinberger, B.Sc.
tm0710262067

Ort, Datum Unterschrift

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter oder einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der von den Begutachter/inne/n beurteilten Arbeit überein.

Ort, Datum Unterschrift

Kurzfassung

Jährlich verursachen etwa 100 Milliarden Spam-Mails einen enormen wirtschaftlichen Schaden. Verschiedene, oft statische Filtermethoden versuchen Mails zu klassifizieren, um eine Spam-Filterung durchzuführen.
Das in dieser Arbeit entwickelte Verfahren nutzt einen lernenden, dem menschlichen Gehirn nachempfundenen Mechanismus: Als Teilgebiet der Neuroinformatik erhält der Softwareentwickler mit einem *künstlichen neuronalen Netz* ein Werkzeug zur Lösung verschiedenster Aufgaben der Informatik. Für eine Spam-Identifikation wird die Datenmenge durch einen Preprozessor aufbereitet und dem neuronalen Netz zur Klassifikation übergeben, wobei unbekannte Daten einem Trainingsset zugeführt werden. Das neuronale Netz wächst durch erneutes Lernen dieser Trainingsmenge.
Netze mit wenigen Neuronen erlauben eine eindeutige Klassifizierung von Mails. Bei größeren Netzen kommt es zu einem „Verwässern“ des Wissens und eine eindeutige Erkennung des Eingabemusters ist nicht mehr möglich. Es bedarf weiterer Maßnahmen um das Verfahren zu optimieren.

Abstract

Annually more than 100 billion spam e-mails cause an enormous economic damage. Several existing static filter methods try to recognise spam by classifying mails.
A procedure developed for this thesis uses a mechanism, which is similar to the human brain having the ability of learning. As part of the neuro-computer science, an *artificial neural network* became a tool for developers to solve various tasks of computer science. To identify spam, the input data is passed through a preprocessor before the neuronal network receives the mail for classification. Unknown data is stored in a trainingset. The neural network grows by re-learning from this trainingset.
Networks consisting of a few neurons provide a explicit classification of mails. For larger networks, a dilution of knowledge occurs and consequently a explicit recognition of input patterns is no longer possible. Further tasks are required in order to optimize the procedure.

Das Gehirn ist nicht nur ein Gefäß, das gefüllt werden muss,
sondern ein Feuer, das gezündet werden will.

(Plutarch, griechischer Philosoph, 45-125 n. Chr.)

Vorwort und Motivation

Die Welt der künstlichen neuronalen Netze übt schon seit Langem eine magische Anziehungskraft auf mich aus. Es klingt faszinierend ein System zu schaffen, das nicht der klassischen, logischen Programmierung der Informatik gehorcht und fernab von strukturierten *If-Else* Bedingungen Entscheidungen trifft, die den Verhaltensmustern eines organischen Gehirns ähneln. Denkbar viele Anwendungsgebiete und Szenarien eröffnet diese Art der Simulation. Forscher erschufen sogar ein neuronales Netz, das die menschliche Sprache wie ein Kleinkind Stück für Stück erlernte. (vgl. [DCPP08] Plaut et al. 01.08.2008)

So faszinierend die Thematik der künstlichen neuronalen Netze auch ist - so basiert sie doch nur auf mathematischen Rechenmodellen steriler Computerelektronik. Gefühle, Geist oder Seele bleiben wohl nur Lebewesen vorbehalten.
Und das ist auch gut so.

In Zitaten wurde der übliche Sprachstand beibehalten. Im Sinne der Lesbarkeit beziehen sich geschlechtsneutrale Bezeichnungen auf weibliche und männliche Vertreter/Personen.

Für die Unterstützung bei dieser Arbeit bedanke ich mich bei meinem Betreuer Dipl. Ing. (FH) Fritz Grabo, bei meinem Freund Ing. Reinhard Drexel für das Korrekturlesen, sowie bei der Fachhochschule St. Pölten. Mein ganz besonderer Dank gilt Kerstin für die aufopfernde Unterstützung, sowie all meinen Freunden die mir zur Seite standen.

Diese Arbeit wurde erst durch das umfassende Studium an der Fachhochschule St. Pölten ermöglicht. Daher möchte ich mich nun am Ende des Studiums vor allem bei den Lektoren Dipl. Ing. Grischa Schmiedl, Dipl. Inform. Joachim Baumert und Dipl. Ing. (FH) Fritz Grabo für ihre Kollegialität und ihre Bemühungen bedanken.

Inhaltsverzeichnis

1 Einleitung **8**

1.1 Forschungsfeld 8

1.2 Problemstellung 8

1.3 Ziel und Aufbau der Arbeit 9

2 Spam Grundlagen **10**

2.1 Herkunft und Arten von Spam 10

2.2 Rechtliche Relevanz von Spam 12

2.3 Ökonomische Auswirkung von Spam 14

2.4 Maßnahmen zur Spam-Bekämpfung 15

2.5 Text-Klassifizierung 16

2.5.1 Grundlagen 16

2.5.2 Algorithmen 17

3 Künstliche Neuronale Netze **21**

3.1 Einleitung 21

3.2 Abgrenzung zur künstlichen Intelligenz 23

3.3 Historie und Grundlagen 23

3.4 Biologische Analogie 27

3.5 Modellaufbau 30

3.6 Units (Neuronen) 32

3.6.1 Aktivitätsfunktionen 35

3.6.1.1 Lineare Funktion 35

3.6.1.2 Linear mit Schwelle 36

3.6.1.3 Binäre Schwellenwertfunktion (threshold) 36

3.6.1.4 Sigmoide Funktion 37

3.7 Netztypen 38

3.7.1 Feedforward Netze 38

3.7.2 Feedback Netze 39

3.8 Lernregeln 41
3.8.1 Supervised learning 41
3.8.1.1 Hebb-Regel 42
3.8.1.2 Delta-Regel 42
3.8.1.3 Backpropagation 43
3.8.2 Unsupervised learning 45
3.8.2.1 Competetive learning 45

4 Entwicklung eines KNN-Klassifizierers 46
4.1 Methodik des Verfahrens 46
4.1.1 Struktureller Aufbau 48
4.1.2 Einsatz-Szenarien 51
4.2 Softwareentwicklung: nAntiSpam 52
4.2.1 MailConnector Layer 57
4.2.2 Core Layer 59
4.2.3 Executor Layer 65
4.3 Analyse des Prototypen 66

5 Zusammenfassung 74
5.1 Zusammenfassung der Arbeit 74
5.2 Optimierung des Verfahrens 77
5.3 Persönliche Meinung 80

Literaturverzeichnis 81

Abbildungsverzeichnis 85

Tabellenverzeichnis 87

Stichwortverzeichnis 88

Glossar 90

Anhang: AForge.Neuro Klassendiagramm 93

Kapitel 1

Einleitung

1.1 Forschungsfeld

Die Informationsübertragung mittels E-Mail ist heutzutage ein fundamentaler Bestandteil der Geschäftswelt sowie der privaten Kommunikation über das Internet. 80-85% der E-Mails werden mißbräuchlich verwendet und als Spam unerwünscht zugestellt. ([AS08] Stiller, 03.08.2008)
Diese Spam-Mails verursachen einen erheblichen ökonomischen Schaden, der sich nur durch Einsatz moderner Filtersysteme eindämmen läßt. Hierzu wird der Inhalt der Nachricht mit Text-Klassifizierungs-Algorithmen untersucht um eine mögliche Spam-Gefahr zu erkennen. Aktuelle Filter arbeiten mit statischen-, oder halbintelligenten Filtern wie dem Naive-Bayes-Algorithums (siehe Kaptiel 2.5).

Forschungen der Neuroinformatik befassen sich mit der Simulation neuronaler Netze und eröffnen neue Möglichkeiten einer intelligenten, lernenden Text-Klassifizierung. Durch mathematische Beschreibungen wird ein künstliches neuronales Netz modelliert und mit einer Lernmenge trainiert. Dem Netz ist es möglich, das erlernte Wissen auf neue Eingaben anzuwenden und sich durch wiederholtes Training an veränderte Bedingungen anzupassen.

1.2 Problemstellung

Viele der derzeit eingesetzten Verfahren der Mail-Klassifikation beruhen auf den Naive-Bayes-Algorithmus, Blacklists oder anderen mehr oder weniger statischen Prozeduren. Es liegt auf der Hand sich zu fragen, ob nicht maschinengelerntes

Wissen zur automatischen Erkennung von Spam-Mails eingesetzt werden kann, die der Entscheidung eines Menschen ähnelt - beispielsweise mit genetischen Algorithmen oder künstlichen neuronalen Netzen (KNN bzw. ANN - artificial neural network). Wie schon die wissenschaftliche Arbeit „A LVQ-based neural network anti-spam email approach“ ([CXMX05] Chuan et al. 25.04.2008)) feststellt, könnte ein selbstlernendes System eine eindeutige Erkennung von Spam-Mails zu etwa 98% ermöglichen.
Ist es möglich, ein pragmatisches, technisch realisierbares Verfahren zu entwickeln, das eine E-Mail Klassifikation durch Einsatz von ein- oder mehreren künstlichen neuronalen Netzen ermöglicht?
Wie lässt sich eine derartige Software entwickeln und ist diese performant und wirkungsvoll?
Können die errechneten Zahlen der Arbeit [CXMX05] bestätigt werden?

1.3 Ziel und Aufbau der Arbeit

Mit dieser Arbeit wird die Spam-Mail-Problematik mit Hilfe eines selbstlernenden Algorithmus in Form von künstlichen neuronalen Netzen erforscht. Zu Beginn wird die aktuelle Situation der Spams beleuchtet, gefolgt von den unterschiedlichen Arten der Textklassifizierung. Der inhaltsanalytische Teil der Arbeit schließt mit der grundlegenden Vorstellung der künstlichen neuronalen Netze ab.
Der zweite Teil der Arbeit widmet sich der Verfahrens- und Softwareentwicklung. Mittels Experimenten soll eine Möglichkeit gefunden werden, eine softwarebasierende Mail-Klassifikation mit künstlichen neuronalen Netzen zu entwickeln. Ferner wird verifiziert, ob sich die errechneten Werte des ACM-Artikels [CXMX05] mit Praxistests bestätigen lassen.
Künstliche neuronale Netze sind bereits seit den 1940 Jahren (siehe Kaptiel 3.3) bekannte mathematische Modelle zur Simulation von biologisch vernetzten Neuronen (wie dem menschlichen Gehirn). Durch Lerntechniken kann ein künstliches Netz trainiert werden, um komplexe Beurteilungen zu fällen.

Kapitel 2

Spam Grundlagen

2.1 Herkunft und Arten von Spam

Der Begriff Spam wurde ursprünglich von der Firma Hormel Foods[1] für das Produkt „Spiced Pork and Ham“ eingeführt. Aus rechtlicher Sicht darf dieses Markenzeichen auch als Bezeichnung für unerwünschte Mails verwendet werden, sofern dieses in Versalien geschrieben wird. (vgl. [Foo08] Hormel Foods, 31.07.2008)

Allen Spam-Mails liegt zu Grunde, dass sie ohne Einwilligung des Empfängers - also unerwünscht - zugestellt werden. Der Inhalt reicht von erotischen Angeboten über Medikamente und Raubkopien bis hin zu Phishing[2] -Versuchen.
Wie in ([Egg07] Eggendorfer 2007, S. 13ff) beschrieben, unterteilt die Wissenschaft Spam-Mails in folgende Kategorien:

- **Unsolicited Bulk E-Mail**
  - Kettenbriefe, Hoaxes
    Ziel der Kettenbriefe ist es, sich sinnlos weiter zu vermehren; der Nutzer wird in den Mails meist darauf hingewiesen, die Nachricht an Freunde weiterzuleiten - ansonsten würde einem Schlimmes widerfahren. Hoaxes[3] ähneln Kettenbriefen; der Inhalt suggeriert dem Empfänger beispielsweise Sicherheitsprobleme, wie einen neu entdeckten Virus. Für eine Verbreitung der Nachrichten sorgt oft auch das schlechte Gewissen

[1] Firmenwebsite: http://www.hormelfoods.com
[2] Weitere Informationen zum Thema: http://www.symantec.com/de/de/norton/products/library/article.jsp?aid=phishing_for_dollars
[3] „Schabernack“, Übers. d. Verf.

bei Mails über Organ- oder Geldspenden für erkrankte oder in Armut lebende Personen.

- Scam
  Beim Vorschussbetrug werden die Empfänger unter Vorspiegelung falscher Tatsachen dazu gebracht, an einem Schneeballsystem teilzunehmen oder in Erwartung zugesagter Vermittlungsprovisionen gegenüber den Absendern (den Scammern) finanziell in Vorleistung zu treten. Bekannte Vertreter sind *Nigeria-Scam* und *Internet Love Scam*. (vgl. [Wie07] BKA, 29.09.2008)

- Phishing
  Mittels Phishing („password fishing“) wird versucht, dem Nutzer vertrauliche Daten wie PIN-Codes, TAN-Nummern, etc. für z.B. Internetbanking Anwendungen zu entlocken um diese dann für eigene Zwecke zu missbrauchen (z.B. Überweisungen). Phishing und Scam zählen zu den Methoden des *Social Engineerings*. ([Mit03] Mitnick 2003)

- **Unsolicited Commercial E-Mail**
  Sind gekennzeichnet durch kommerziellen Inhalt wie Glücksspiele, Aktienhandel, Verkauf von Software, etc.

Das lukrative Geschäft mit Spam ist gut organisiert. Das Spammer-Geschäftsmodell unterscheidet zwischen dem Versand von Spam, dem Verkauf des Produktes und der Mail-Adressen Beschaffung. (vgl. [Egg07] Eggendorfer 2007, S. 28, 29) Offiziell agierende Firmen wie beispielsweise das *Russian Business Network* bieten verschiedenste Dienste zum Versenden von Spam-Mails an. (vgl. [Fab08] Faber 2008, S. 92ff)

Warum ist Spam überhaupt möglich?
Der Versand von E-Mails erfordert aus technischer Sicht keine direkte Verbindung zwischen den beiden Endpunkten der Kommunikation. Das E-Mail Programm des Absenders sendet die Nachricht an den SMTP[4]-Server seines Internet Service Providers. Dieser SMTP-Server erkennt an der Empfänger-Adresse, für welche Domain die Nachricht bestimmt ist und befragt ggf. einen DNS-Server nach der Adresse des zuständigen SMTP-Servers über den „MX-Record“, um die E-Mail ein weiteres

[4]Informationen zum Thema: http://www.ietf.org/rfc/rfc2554.txt

mal per SMTP an diesen zu senden (ggf. wird die Mail über mehrere SMTP-Server weitergereicht). Trifft die Mail letztendlich beim zuständigen Server ein, stellt dieser die Nachricht dem lokalen Mailsystem-Benutzer zu. Von dort wird sie vom Empfänger in der Regel über POP3[5] oder IMAP[6] übermittelt und im E-Mail Programm dargestellt.
Richtig konfigurierte SMTP-Server akzeptieren E-Mails nur dann, wenn ihnen entweder der Empfänger, oder der Absender bekannt ist. Da das SMTP-Protokoll per se keine Authentifizierung kennt und der Absender ohne große Umstände gefälscht werden kann, sind zusätzlich mehrere Mechanismen zur Sicherung[7] wie beispielsweise SMTP-after-POP3 und SMTP-AUTH bei einem korrekt konfigurierten Mailserver möglich.

2.2 Rechtliche Relevanz von Spam

Am Beispiel des Rechtsstaates Österreich ist lt. Telekommunikationsrecht § 107 das Versenden von Spam in Österreich nicht zulässig (in anderen europäischen Ländern gelten ähnliche Bestimmungen):
„(1) Anrufe - einschließlich das Senden von Fernkopien - zu Werbezwecken ohne vorherige Einwilligung des Teilnehmers sind unzulässig. Der Einwilligung des Teilnehmers steht die Einwilligung einer Person, die vom Teilnehmer zur Benützung seines Anschlusses ermächtigt wurde, gleich. Die erteilte Einwilligung kann jederzeit widerrufen werden; der Widerruf der Einwilligung hat auf ein Vertragsverhältnis mit dem Adressaten der Einwilligung keinen Einfluss.
(2) Die Zusendung einer elektronischen Post - einschließlich SMS - ist ohne vorherige Einwilligung des Empfängers unzulässig, wenn
1.die Zusendung zu Zwecken der Direktwerbung erfolgt oder 2.an mehr als 50 Empfänger gerichtet ist.
(3) Eine vorherige Zustimmung für die Zusendung elektronischer Post gemäß Abs. 2 ist dann nicht notwendig, wenn
1.der Absender die Kontaktinformation für die Nachricht im Zusammenhang mit dem Verkauf oder einer Dienstleistung an seine Kunden erhalten hat und

[5]Informationen zum Thema: http://www.ietf.org/rfc/rfc1939.txt
[6]Informationen zum Thema: http://tools.ietf.org/html/rfc3501
[7]Informationen zum Thema: http://www.ietf.org/rfc/rfc2195.txt

2.diese Nachricht zur Direktwerbung für eigene ähnliche Produkte oder Dienstleistungen erfolgt und
3.der Empfänger klar und deutlich die Möglichkeit erhalten hat, eine solche Nutzung der elektronischen Kontaktinformation bei deren Erhebung und zusätzlich bei jeder Übertragung kostenfrei und problemlos abzulehnen und
4.der Empfänger die Zusendung nicht von vornherein, insbesondere nicht durch Eintragung in die in § 7 Abs. 2 E-Commerce-Gesetz genannte Liste, abgelehnt hat.
(4) (Anm.: aufgehoben durch BGBl. I Nr. 133/2005)
(5) Die Zusendung elektronischer Post zu Zwecken der Direktwerbung ist jedenfalls unzulässig, wenn die Identität des Absenders, in dessen Auftrag die Nachricht übermittelt wird, verschleiert oder verheimlicht wird oder bei der keine authentische Adresse vorhanden ist, an die der Empfänger eine Aufforderung zur Einstellung solcher Nachrichten richten kann.“
([HPL06] Lehofer et al. 2006, S. 164)

2.3 Ökonomische Auswirkung von Spam

Spam-Mails bedeuten nicht nur lästige Werbung, sondern verursachen auch einen erheblichen wirtschaftlichen Schaden. *„Heutzutage geht man von etwa 100 Milliarden Spam- oder Junk-E-Mails pro Jahr aus, rund 80 bis 85 Prozent des E-Mail-Verkehrs seien verspamt.“* ([AS08] Stiller, 03.08.2008)

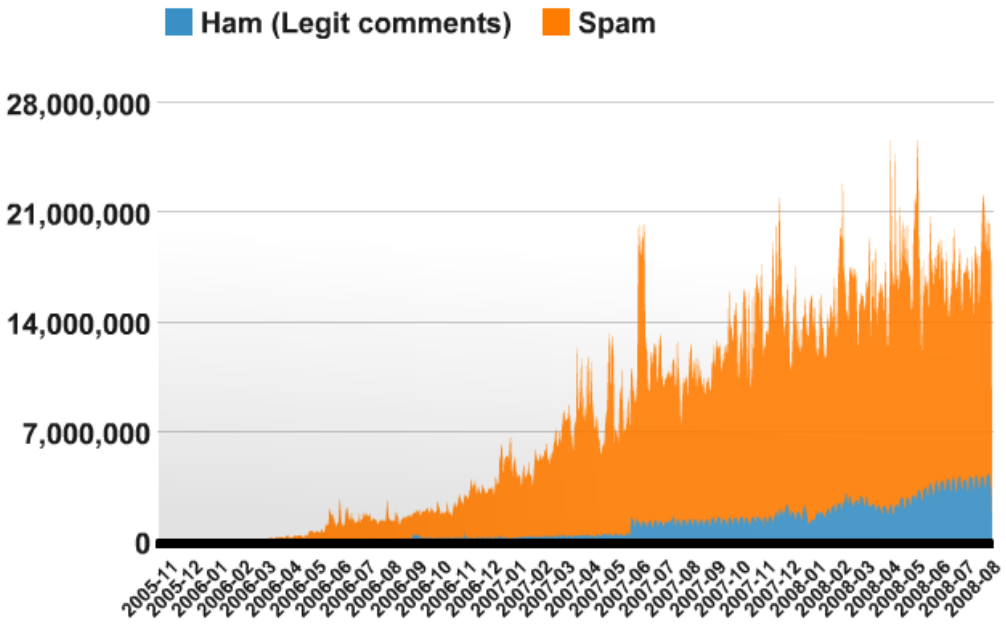


Abbildung 2.1: Spam-Statistik von Akismet
Quelle: [Aki08] Akismet

In einem österreichischen Kleinbetrieb mit 15 öffentlichen E-Mail-Adressen werden derzeit (Stand August 2008) wöchentlich etwa 4000 Spam-Mails identifiziert. Eine Studie des deutschen Bundesamt für Sicherheit in der Informationstechnik (BSI) errechnet bei einem kleineren Provider (ca. 50.000 Kunden) Kosten pro Spam-Mail von 0,20 €-Cent. In einem Unternehmen hingegen müssen die Mitarbeiter die Mails sortieren, um Spam zu filtern. Das BSI geht davon aus, dass bei einem Unternehmen mit 5000 Mitarbeitern Kosten von 0,22 €-Cent pro Spam-Mail anfallen. In einem Kleinunternehmen mit nur fünf Mitarbeitern steigen die Kosten auf bis zu 0,70 €-Cent pro Spam-Mail. (vgl. [Top05] Topf, 03.08.2008)
Es liegt auf der Hand, dass eine zuverlässige automatisierte Mail-Klassifizierung unumgänglich geworden ist.

2.4 Maßnahmen zur Spam-Bekämpfung

Die derzeit meist eingesetzten Maßnahmen zur Spam-Abwehr untersuchen die Mails nach verschiedensten Kriterien um die Mails als unerwünscht (Spam) oder erwünscht (Ham) zu klassifizieren. Diese Maßnahmen erreichen eine Erkennungsrate von etwa 80-97%[8]. Wird eine Spam-Mail fälschlicherweise als harmloses Ham klassifiziert, spricht man von einem *false negative*. Für den Benutzer weit aus dramatischer ist der Fall, wenn das System eine erwünschte Ham-Mail als Spam identifiziert (*false positive*).
Zu den aktuellen Anti-Spam Technologien zählen (vgl. [Egg07] Eggendorfer, 2007, S. 87ff):

- **Blacklists, Whitelists**
  Auf Blacklists werden Domains, IP- oder E-Mail Adressen von Spammern gespeichert. Der Spamfilter kann aufgrund dieser Liste relativ einfach entscheiden, ob eine E-Mail Spam oder Ham ist. Bei Whitelists hingegen werden jene Adressen eingetragen, die bedenkenlos akzeptiert werden können. Solche Listen können lokal, oder zentral auf Servern (*Realtime Blackhole List* - RBL) gespeichert sein.
  Spammer umgehen solche Listen indem sie mit Bot-Netzen[9] ständig die IP-Adressen und die Domains ändern. Um über eine Whitelist-Adresse gefahrlos zu spammen, werden die lokalen Adressbücher eines Zombie[10] durchforstet (meist werden die darin gespeicherten Adressen in Whitelists eingetragen).

- **Header-Überprüfung**
  Bei dieser Art der Analyse werden die Header-Felder einer Mail überprüft um potentielle Spam-Mails zu erkennen. Ungerer ([Ung03], 04.08.2008) beschreibt im Computermagazin iX in einem Artikel unter anderem folgende Kriterien:
  Zeichensatzprüfung, Zahl der Received-Header, Zeichen/Zahlenkombination der Absenderadresse, Zeitzonen, korrekter MIME-Type (base64-Encoding und UUencode schalten Content-Filter aus), etc.

[8] Informationen zum Thema: http://spamassassin.apache.org/
[9] Anm. d. Verf.: Vernetzte Schadprogramme auf meist privatenten Rechnern.
[10] Anm. d. Verf.: Als Zombie wird ein Rechner bezeichnet, der unter nicht erwünschter Kontrolle eines Fremden steht.

- **Contentfilter**
  Eine einfache Art des Contentfilters ist, die einzelnen Wörter des Bodies einer Mail zu überprüfen und festzustellen, ob deren Aussage auf Spam schließen lässt (Wörter wie z.B. *Sex* oder *Adult*). Leider versagen solche einfachen Filter wenn beispielsweise eine Einladung zur Erwachsenenbildung (adult education) versendet wird, die im Anmeldeformular auch das Geschlecht (sex) auswählen lässt. Dieses Problem lässt sich durch Gewichtung der einzelnen Wörter oder auch durch Berechnung der räumlichen Nähe der Wörter zueinander relativieren.
  In den Contentfiltern steckt wohl das meiste Entwicklungspotential - wie die Möglichkeit von selbstlernenden Algorithmen, welche später genauer behandelt werden.

2.5 Text-Klassifizierung

2.5.1 Grundlagen

Um einen kontextbezogenen Spam-Filter zu entwickeln, muss der Inhalt (Body) der Nachricht klassifiziert werden - wobei sich die Themengebiete (Klassen) aus den sg. Merkmalsvektoren ergeben. Der Klassifizierer selbst muss dabei mit einem Sample an Nachrichten und den bereits bekannten Klassen trainiert werden, ehe dieser neue Nachrichten auf das konstruierte Modell anwenden kann.
Zur Illustration wird die einfachste Methodik zur Klassifizierung von Texten vorgestellt - die *Wort-Merkmale* (bag of words model) (vgl. [Seb02] Sebastiani, 2002). Jedes Wort der Nachricht entspricht dabei einem Merkmal. Die Häufigkeit des Wortes wird zusammen mit dem Merkmal gespeichert. All diese Merkmale bilden dann den Merkmalsvektor.

Der Text: *Das ist ein Testtext. Und das ist noch ein Testtext.* liefert den Merkmalsvektor aus Tabelle 2.1:

das	2
ist	2
ein	2
testtext	2
und	1
noch	1

Tabelle 2.1: Merkmalsvektor eines *bag of words models*

Die Algorithmen der Mail-Klassifizierung müssen einige zusätzliche Herausforderungen beherrschen:

- Verschiedene Sprachen
- Viele tausende Merkmale
- Lernfähigkeit
- Schnelle Verarbeitung
- Rasche Lernphase

2.5.2 Algorithmen

Die folgenden Algorithmen werden eingesetzt, um diesen Herausforderungen beim Klassifizieren von Textem zu begegnen:

- **Entscheidungsbäume (Decision Trees)** Mitchel ([Mit97] Mitchell 1997, S. 52ff) beschreibt Decision Trees als Entscheidungsbäume, die anhand der Lernmenge einen Baum mit vielen Knoten darstellen. Jeder dieser Knoten entspricht einer Entscheidung. Die Anzahl der Kindsknoten eines Elternknotens entspricht dabei den möglichen Werten des Elternknotens. Jeder Kindsknoten erhält die gesamte Instanzmenge des Vorgängers, sodass sich die Instanzmenge immer weiter unterteilt bis letztendlich eine bestimmte Ende-Bedingung erfüllt ist.
- **k-Nearest-Neighbour**
  Dieser sehr einfache Algorithmus speichert alle Merkmalsvektoren in einem n-dimensionalen Raum ab (n entspricht der Anzahl der Merkmale). Wird

eine neue Nachricht klassifizert, wird diese in den Raum eingeordnet und ihre k naheliegensten Nachbarn ermittelt. (vgl. [Bla02] Blankenhorn, 27.04.2008)

Die Abbildung 2.2 zeigt eine Klassifizierung des roten Kreises mit k=3. Da innerhalb dieser Distanz die Dreiecke überwiegen, wird der Kreis den Dreiecken zugeordnet.

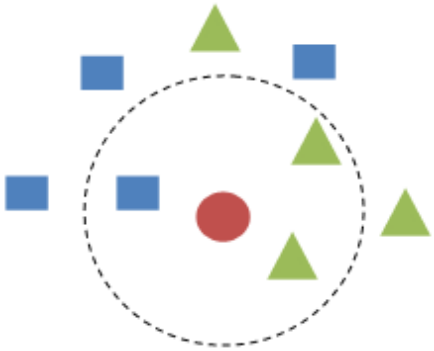


Abbildung 2.2: k-Nearest-Neighbour

- **Regel-Lerner (Rule Induction)**
  Aus der Lernmenge generiert der Regel-Lerner einfache, verständliche, und lesbare Regeln. Ein verbreiteter Algorithmus ist RIPPER (Repeated Incremental Pruning to Produce Error Reduction). (vgl. [Coh95] Cohen, 06.08.2008) Eine solche automatisch generierte Regel kann wie folgt aussehen:

$$Spam \leftarrow \text{"}\mathbf{viagra}\text{"} \in subject \wedge \text{"}\mathbf{improve}\text{"} \in body \tag{2.1}$$

  Die Erstellung solcher Regeln ähnelt den Entscheidungsbäumen, führt aber zu einem genaueren Ergebnis. Rule Induction wird heutzutage auch häufig im Zusammenhang von Data Mining[11] erwähnt.

- **Naive Bayes**
  Der englische Pastor Thomas Bayes[12] entwickelte das Bayes-Theorem ([Bay63] Bayes, 06.08.2008) auf dem der Naive-Bayes-Algorithums basiert. Das Theorem liefert eine Lösung für Entscheidungen, wenn sich mehrere Behaup-

[11] Informationen zum Thema: http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm

[12] Informationen zum Thema: http://www.morris.umn.edu/~sungurea/introstat/history/w98/Bayes.html

tungen gegenseitig ausschließen. Jeder Behauptung ist eine Häufigkeit zugewiesen - sie bilden das „Vorwissen“, die A-Priori-Wahrscheinlichkeiten. Anhand der auftretenden Daten - wenn eine bestimmte Bedingung zutrifft - und anhand derer Wahrscheinlichkeiten kann zusammen mit den A-Priori-Wahrscheinlichkeiten die Wahrscheinlichkeit berechnet werden, dass eine bestimmte Behauptung für ein neues beobachtetes Beispiel zutrifft.
Für die Textklassifizierung kommt vor allem eine Umsetzung des Theorems als Naive-Bayes-Algorithums zum Einsatz. Dieser nimmt an, dass alle Merkmale konditional unabhängig voneinander auftreten (also eine „*naive*“ Annahme). (vgl. [Bla02] Blankenhorn, 27.04.2008)

- **Künstliche neuronale Netze**
  Das menschliche Gehirn ist aufgrund der Möglichkeit der Parallelverarbeitung zu enormen Leistungen fähig. Das Modell der künstlichen neuronalen Netze versucht den physischen Aufbau des Gehirns in einem Rechenmodel nachzubilden. Beispielsweise werden beim *feed-forward* Netztyp Neuronen in verschiedenen Schichten (Input-Layer, Hidden-Layer, Output-Layer) angeordnet und verbunden. Verschiedene Lernalgorithmen gewichten die Verbindungen der Neuronen, um das benötigte Wissen zu *erlernen*. Für die Textklassifizierung bedeutet dies, dass durch die Vernetzung der Neuronen automatisch eine Abhängigkeit der Merkmale zueinander besteht und so ein komplexe Bewertung möglich ist.

Die vorgestellten Algorithmen können mit wenig Aufwand implementiert werden. Für eine Spam-Identifikation zeigen statische Verfahren wie Rule Induction oder große und deshalb unflexible Decision Trees Schwächen. Aktuelle Spamfilter wie *Spamassassin*[13] setzen für die Bewertung auf den Naive Bayes Algorithmus und erreichen damit gute Erkennungsraten bis zu 97%.
Künstliche neuronal Netze hingegen heben sich durch folgende Eigenschaften von den anderen Klassifizierungs-Algorithmen ab: Sie können sich an verändernde Umgebungen (wechselnde Lernmengen) durch wiederholtes Training anpassen und bewerten auch die Merkmale zueinander. Nachteilig gegenüber den statischen Verfahren ist jedoch die zeitintensivere Lernphase. Da sich die Spam-Mail-Signaturen aber nicht permanent ändern, beeinflussen die Lernzyklen einen KNN-Spam-Filter

[13]Informationen zum Thema: http://spamassassin.apache.org/

nur marginal. Im folgenden Kapitel werden die Grundlagen der künstlichen neuronalen Netze vorgestellt.

Kapitel 3

Künstliche Neuronale Netze

3.1 Einleitung

Das menschliche Gehirn ist - verglichen mit einem Computer - zu enormen Leistungen fähig. Computer eignen sich zur Verarbeitung von logischen Abfolgen, während das menschliche Gehirn die Fähigkeiten besitzt, komplexe Sachverhalte zu interpretieren und sich einer veränderten Sachlage anzupassen. Es ist besonders zur Parallelverarbeitung von Informationen im hohen Maße fähig. Obwohl die Schaltzeiten des Gehirns viel geringer als die des Computers sind, ist das Lesen eines Textes auf einem stark verschmutzten Papier für den Menschen kein Problem. Ein Computer hingegen würde für diese Aufgabe viele Lese- und Korrekturzyklen benötigen.
Manche Problemstellungen lassen sich nur schwer mit Computerlogik abbilden. Um beim Radfahren das Gleichgewicht zu halten würde man programmatisch folgendes festlegen:

Listing 3.1: Beispielcode Grundlagen neuronaler Netze

```
1 If Gefahr_links_umfallen Then Gewichtsverlagerung_rechts
2 If Gefahr_rechts_umfallen Then Gewichtsverlagerung_links
```

Es ist sofort erkennbar, dass dieses Modell nicht funktionieren kann. Denkt man darüber nach welche der Bedingungen zutrifft, wird man bereits stürzen. Außerdem würden diese Regeln zu einem sich selbst verstärkenden Schwanken führen.
Biologische und künstliche neuronale Netze (KNN[1]) haben gemein, dass sie die

[1] engl. ANN - „artificial neural network“

Fähigkeit des „Lernens“ inne haben. Das Wissen über das Reagieren auf Eingangsinformationen wird implizit aus Beispielen durch wiederholtes Training erlernt. Neue Eingangsreize kann ein künstliches neuronales Netz verarbeiten, indem es sich an das erlernte Wissen „erinnert“. Ein KNN kann beispielsweise ein zuvor erlerntes Bild eines Gesichts trotz veränderten Aufnahmebedingungen erkennen.

Für die enorme biologische Leistung, ist das menschliche Gehirn aus etwa $10^{\mathbf{11}}$ Nervenzellen - den so genannten *Neuronen* aufgebaut. Die eigentliche Basis der „Intelligenz“ wird jedoch erst durch die etwa $10^{\mathbf{15}}$ Verbindungen der Neuronen untereinander ermöglicht. Dieses sehr komplexe Konstrukt ist metabolisch sehr aktiv. Das menschliche Nervensystem verbraucht etwa 25% der Gesamtenergie des Körpers obwohl es nur 1%-2% des Körpergewichts ausmacht. (vgl. [And95] Anderson 1995, S. 5)

Künstliche Neuronale Netze versuchen den Aufbau des menschlichen Gehirns durch mathematische Modelle nachzubilden. Analog zum biologischen Netz werden idealisierte Neuronen durch gewichtete Verbindungen zu einem virtuellen Netz verdrahtet. Das Netz ist in der Lage, sich durch Lern-Algorithmen Wissen anzutrainieren, um so verschiedenste komplexe Aufgaben zu bewältigen. Je nach Anwendungsfall kommen unterschiedliche Netztypen und Eigenschaften der Neuronen zum Einsatz. Verschiedene Lern-Algorithmen trainieren das Netz *supervised* (beaufsichtigt), oder *unsupervised* (unbeaufsichtig) (siehe Kaptiel 3.8). Ein künstliches neuronales Netz ist kein starres Modell, sondern kann durch Hinzufügen von weiteren Neuronen und Verbindung natürlich wachsen. Durch wiederholtes Training kann der Output[2] eines KNN laufend an sich ändernde Bedingungen angepasst und somit ständig optimiert werden.
Nachfolgend wird auf die Geschichte, das biologischen Vorbild und die grundlegende Funktionsweise der neuronalen Netze eingegangen.

[2] Anm. d. Verf.: entspricht dem „Ergebnis“ eines KNN

3.2 Abgrenzung zur künstlichen Intelligenz

Künstliche neuronale Netze sind nicht gleichzusetzen mit dem Begriff *Künstlicher Intelligenz* (KI[3]). Künstliche neuronale Netze sind ein Zweig der künstlichen Intelligenz und prinzipieller Forschungsgegenstand der Neuroinformatik. Die KI ist ein „*Teilgebiet der Informatik, welches versucht, menschliche Vorgehensweisen der Problemlösung auf Computern nachzubilden, um auf diesen Wege neue oder effizientere Aufgabenlösungen zu erreichen.*“ ([UL04] Lämmel et al. 2004, S. 13)

Die klassische KI, auch „Symbol verarbeitende KI“[4] genannt, geht von einer expliziten Darstellung des Wissens unter Nutzung logischer Konzepte aus. Auch Eigenschaften, Beziehungen und Ereignisse werden durch Symbole dargestellt. Das Wissen wird vom Menschen in das KI-System eingearbeitet, damit das Programm durch die Verarbeitung dieses Wissens (Symbolverarbeitung) neue Aussagen treffen kann. Künstliche neuronale Netze hingegen wenden eine konnektionistische[5] Herangehensweise an. (vlg. [UL04] Lämmel et al. 2004, S. 13-14)

3.3 Historie und Grundlagen

Die Grundlagen künstlicher neuronaler Netze sind keiner visionären Idee des 21. Jahrhunderts entsprungen, sondern bedienen sich an Forschungen, die bereits Ende des zweiten Weltkriegs durchgeführt wurden.

1943 beschrieben Warren McCulloch (1898-1972) und Walter Pitts (1924-1969) in ihrem Aufsatz *A logical calculus of the ideas immanent in nervous activity*[6] bereits neurologische Netzwerke, die auf dem *McCulloch-Pitts-Neuron* (siehe Abbildung 3.1) basieren. (vgl. [Roj96] Rojas 1996, S. 3)
Solche Neuronen sind binäre Elemente und kennen nur die zwei möglichen Eingangszustände 0 und 1. Sie arbeiten mit festen Schwellen und empfangen Eingaben von erregenden sowie von hemmenden Synapsen. Eine einzige hemmende Eingabe deaktiviert das Neuron, alle erregenden Eingaben werden addiert bis die Schwelle

[3] engl. „AI - artificial intelligence“
[4] Objekte und Subjekte werden benannt. Anm. d. Verfassers
[5] subsymbolisch, keine zentrale Kontrolle, gehirnartig
[6] Informationen zum Thema: http://portal.acm.org/citation.cfm?id=104377

erreicht ist. (vgl. [And95] Anderson 1995, S. 49-51)
Dieses sehr einfache Modell kann zwar jede arithmetische oder logische Funktion berechnen, ein „selbständiges Lernen“ ist jedoch nicht möglich.

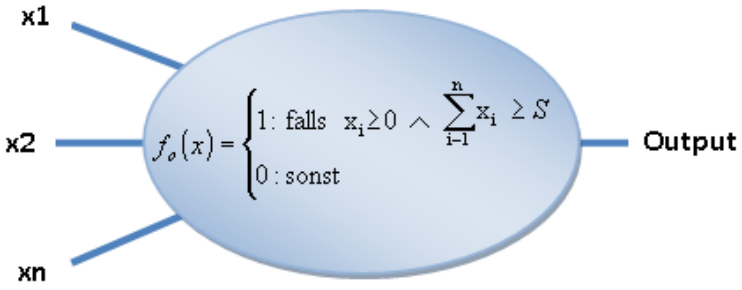


Abbildung 3.1: McCulloch-Pitts-Neuron

Donald O. Hebb (1904-1985) verfasste 1949 in seinem Buch *The Organization of Behaviour* die Hebb'sche Lernregel als einfaches universelles Lernkonzept individueller Neuronen (siehe Kaptiel 3.8.1.1).
Diese Regel besagt für künstliche neuronale Netze: wenn Zelle j eine Eingabe von Zelle i erhält und beide gleich stark aktiviert sind, erhöht sich das Gewicht (= die Verbindung) $w_{i,j}$. Allerdings ist die Hebb'schen Lernregel nicht unproblematisch: Bei anhaltenden Aktivitäten beider Zellen i und j wachsen die Gewichte in das Unendliche. Die Hebb'sche Lernregel ist somit nicht realistisch und muss durch Modifikationen einem realen Verhalten angepasst werden.
In ihrer allgemeineren Form ist die Hebb'sche Lernregel bis heute die Basis der meisten neuronalen Lernverfahren.

1951 entwickelte Marvin Minsky den ersten bekannten Neurocomputer *Snarc* (Stochastic Neural Analog Reinforcement Calculator), der bereits die Gewichte automatisch justierte.

1957-1958 wurde der erste erfolgreiche Neurocomputer *Mark I perceptron* von Frank Rosenblatt, Charles Wightman und Mitarbeitern am MIT[7] entwickelt und

[7]Massachusetts Institute of Technology in Cambridge (Massachusetts)

für Mustererkennungsprobleme eingesetzt. Dieser Neurocomputer konnte mit einem 20x20 Pixel großen Bildsensor einfache Ziffern erkennen. Frank Rosenblatt ist insbesonders durch sein 1959 erschienenes Buch *Principles of Neurodynamics* bekannt geworden in dem er verschiedenste Varianten des Perzeptrons beschreibt. (vgl. [WU08] 18.08.2008)
Perzeptrons (siehe Abbildung 3.2) bestehen aus zwei oder drei Schichten, einer Input-, ggf. einer Hidden-, und einer Output-Schicht. Dieses Netzwerk erlaubt mittels eines Lernverfahrens Muster zu klassifizieren. (vgl. [Roj96] Rojas 1996, S. 55)

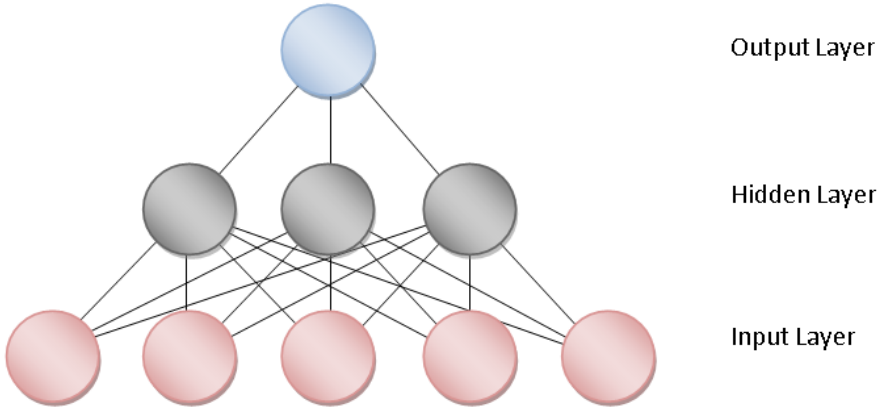


Abbildung 3.2: Perzeptron

1969 führten Marvin Minsky und Seymour Papert genaue mathematische Analysen des Perzeptrons durch und erkannten, dass das Modell des Perzeptrons viele wichtige Probleme nicht repräsentieren konnte. Sie definierten diese Forschung als *research dead-end*. Dies hatte zur Folge, dass in den kommenden 15 Jahren keine Gelder für Forschungen im Bereich der künstlichen neuoronalen Netze von der DARPA (Defense Advanced Research Projects Agency)[8] vergeben wurden. (vgl. [WU08] 18.08.2008)

1972 beschrieb Teuvo Kohonen in seiner Arbeit *Correlation matrix memories* (vgl. [Koh72] Kohonen 1972, S. 353-359) ein Modell des linearen Assoziierers, eines speziellen Assoziativspeichers. Dieses Modell arbeitet mit linearen Aktivierungsfunktionen und kontinuierlichen Werten für Gewichte, Aktivierungen und

[8]Informationen zum Thema: http://www.darpa.mil

Ausgaben. (Auf das Kohonen-Netz wird gesondert in den folgenden Kapiteln eingegangen.)

Paul Werbos entwickelte 1974 an der Harvard-Universität das *Backpropagation*-Verfahren. Wirklich bekannt wurde dieses Verfahren aber erst etwa 10 Jahre später durch David E. Rumelhart, Geoffrey E. Hinton und Ronald J.Williams, deren Verfahren (1986) eine Verallgemeinerung der Delta-Regel (siehe Kaptiel 3.8.1.2) darstellt. Heutzutage wird der Backpropagation-Algorithmus in sehr vielen Anwendungsfällen der künstlichen neuronalen Netze eingesetzt, die konkrete Anwendungsprobleme lösen sollen. (vgl. [WU08] 18.08.2008)

Mittlerweile sind die Forschungen rund um künstliche neuronale Netze zu einem festen Bestandteil der Wissenschaft avanciert. Einschlägige Magazine wie *Neural Networks*, *Neural Computation*, *Neurocomputing*, *IEEE Trans. on Neural Networks* beleben diese Disziplin ebenso wie große anerkannte wissenschaftliche Gesellschaften wie die *INNS*[9] (International Neural Network Society), die *ENNS*[10] (European Neural Network Society), sowie IEEE Fachgruppen.
Die Anwendungsgebiete reichen von der Text-, Bild-, und Gesichtserkennung über die Regelungstechnik bis hin zu Prognosen und Frühwarnsysteme.

[9]Informationen zum Thema: http://www.inns.org
[10]Informationen zum Thema: http://www.snn.ru.nl/enns

3.4 Biologische Analogie

Als Vorbild der künstlichen neuronalen Netze dienen die biologischen Nervenzellen (Neuronen), die aus einem bis zu 0,25mm großen Zellkörper (Soma) bestehen (siehe Abbildung 3.3). „*Neuron [...] bildet nach der Neuronenlehre eine genetische, morpol., funkt., trophische u. regenerative Einheit.*“ ([Psc02] Pschyrembel 2002, S. 1167) Aus diesem Zellkörper gehen kurze verästelte Fortsätze hervor - die Dendriten. „*[...] kurzer, zelllaibnah verzweigter Fortsatz einer Nervenzelle; ein Neuron (Nervenzelle) hat viele Dendriten, Der D. leitet die Erregung zentripital.*“ ([Psc02] Pschyrembel 2002, S. 64)

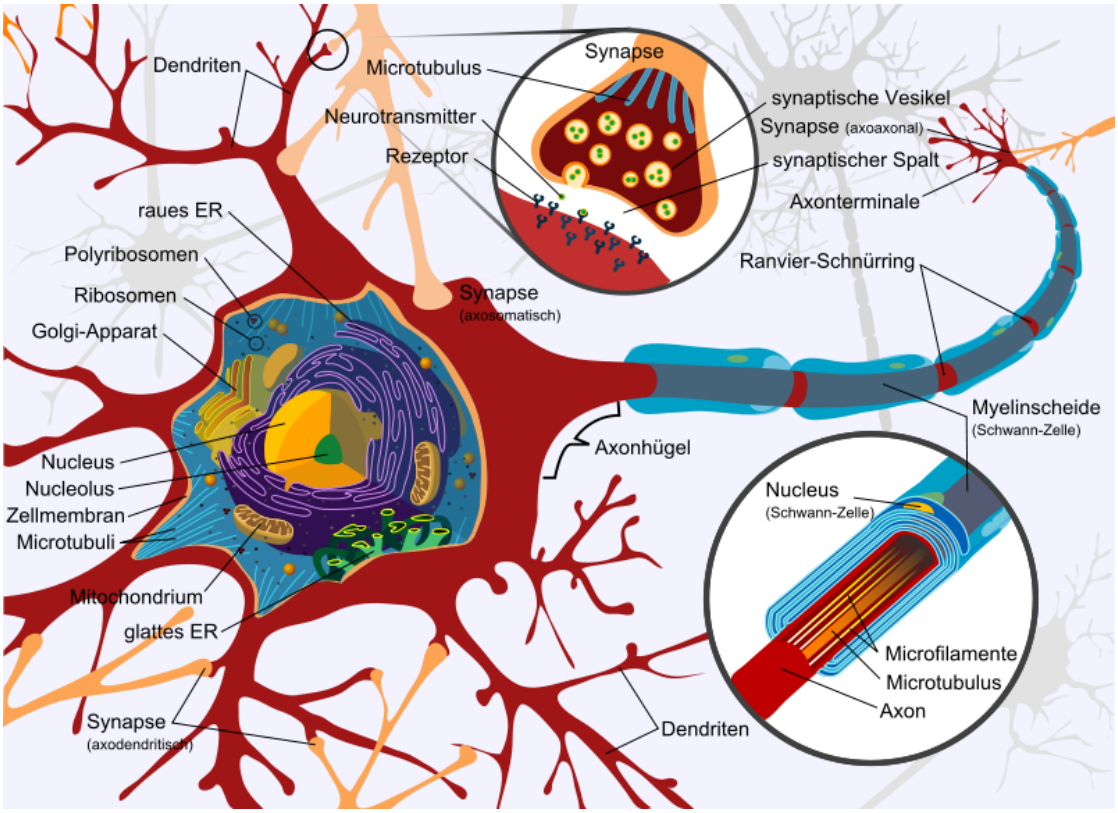


Abbildung 3.3: Biologisches Neuron
Quelle: [Vil08] Mariana Ruiz Villarreal

Die feinen plasmatischen Dendriten stellen über Synapsen den Kontakt zu anderen Nervenzellen her (siehe Abbildung 3.3). Die Dendriten nehmen die Eingangssignale auf, während das *Axon* für das Weiterleitung von Nervenreizen an andere Zellen dient. (vgl. [WU08] 19.08.2008)

Die Signalübertragung erfolgt als kodierte Erregung durch Ausschüttung eines Neurotransmitters (chemischer Botenstoff) von der sendenden Zelle. Die Erregung wird über die Dendriten an das Soma (Zellkörper) der Nervenzelle und von dort zum Axonhügel weitergeleitet. Jede der eingehenden Information an den Rezeptoren der Nervenzelle verändert dabei das elektrische Membranpotential an der axonalen Membran. Bei Überschreiten eines Schwellenwertes wird ein Aktionspotential ausgelöst. Je näher eine Synapse am Soma ansetzt, desto stärker ist ihr Einfluss auf die Nervenzelle. (vgl. [And95] Anderson 1995, S. 37-48)

Abbildung 3.4 zeigt ein grün fluoreszierendes biologisches Neuron eines Lappens in der Großhirnrinde (cerebralis cortex).

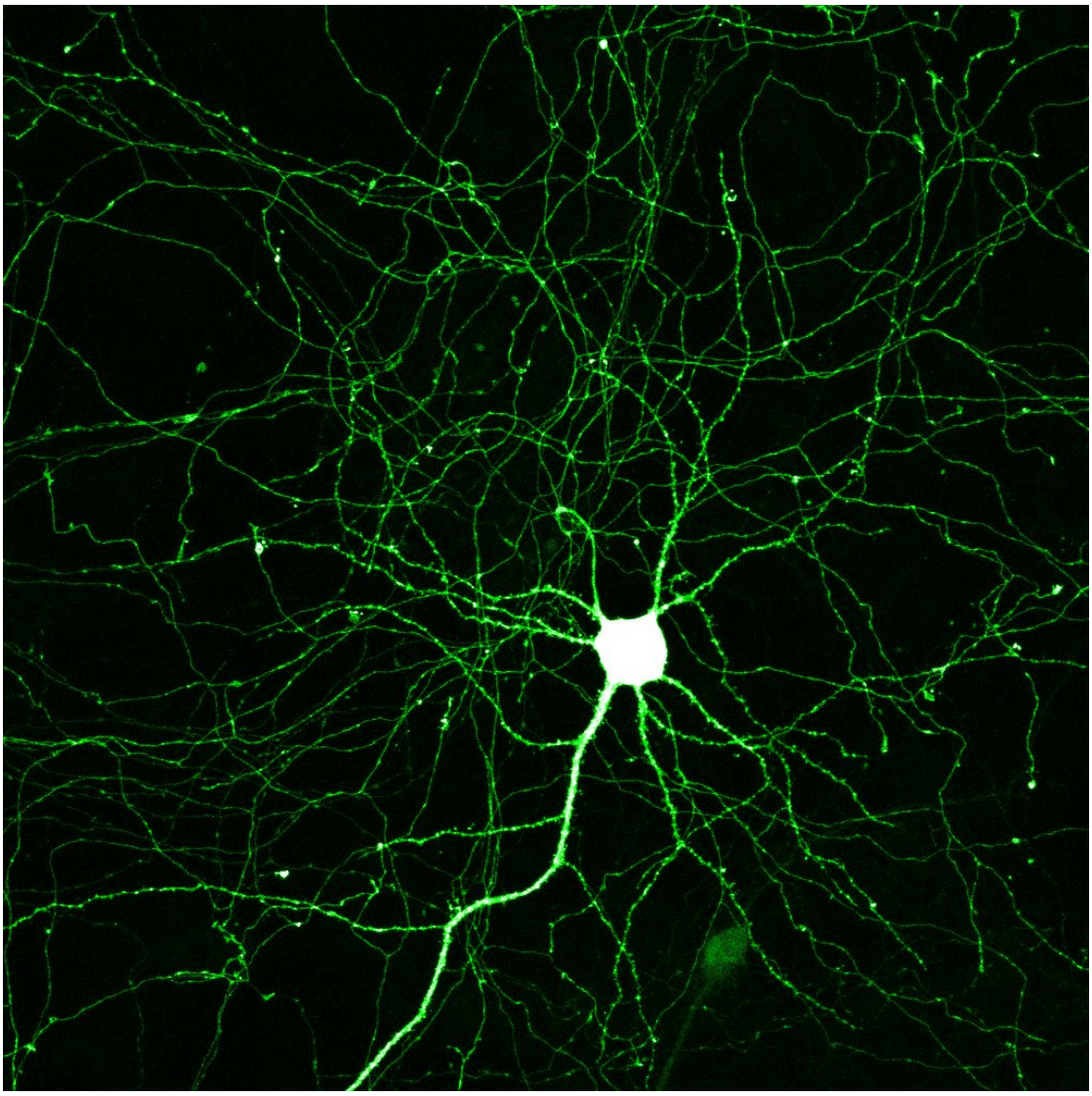


Abbildung 3.4: Hippocampal neuron expressing monomeric Green Fluorescent Protein (GFP)
Quelle: [Lab08] Paul De Koninck Laboratory

3.5 Modellaufbau

Künstliche neuronale Netze orientieren sich zwar an biologischen neuronalen Netzen, unterscheiden sich aber in einigen wesentlichen Punkten:

KÜNSTLICHES NEURONALES NETZ	BIOLOGISCHES NEURONALES NETZ
viel geringere Anzahl der Neuronen (10^2-10^4)	ca. 10^{11} Neuronen
viel geringere Anzahl von Verbindungen	höhere Anzahl an Verbindungen zwischen den Neuronen
Stärke einer Synapse wird ausschließlich durch das Gewicht bestimmt	Einfluß verschiedener Neurotransmitter auf die Stärke einer Synapse
zeitliche Vorgänge der Nervenleitung werden vernachlässigt	verzögerte Aktivierung
Schaltzeit 10^{-2}s	Schaltzeit 10^{-9}s

Tabelle 3.1: Unterschied künstliches und biologisches neuronales Netz

Wie in Abbildung 3.3 illustriert (vgl. [PJS08] Sauer 2008, S. 17), besteht ein künstlichs neuronales Netz aus folgenden grundlegenden Elementen (diese werden in den folgenden Kapiteln genauer erläutert):

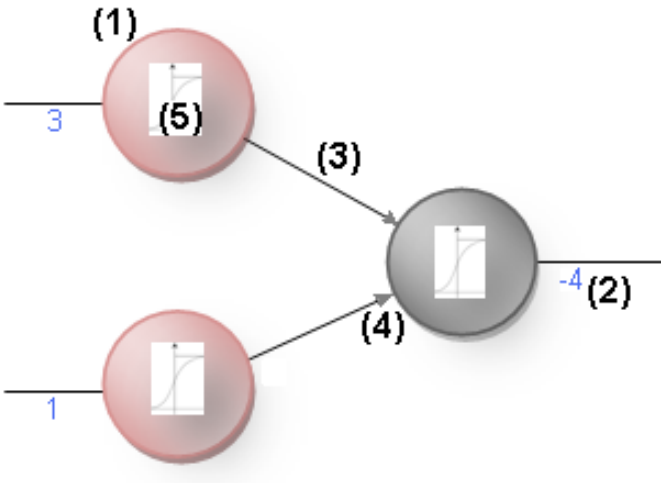


Abbildung 3.5: Modell eines künstlichen neuronalen Netzes

- eine Menge von Verarbeitungselementen (Neuronen). Diese Neuronen (Units) sind ein idealisiertes Modell eines biologischen Neurons (1).

- eine Menge von Aktivitätszuständen (2)
- eine Verbindungshierarchie (3). Diese wird durch den Netztypus bestimmt (Kohonennetze, Rekurrente Netze, etc.)
- eine Regel zur Fortschaltung der Aktivitätsmuster von einem Element zum nächsten (4)
- eine Regel zur Aktivierung eines Elements durch anliegende Signale und Erzeugung eines neuen Aktivitätszustands (5) (entspricht der Übertragungsfunktion innerhalb einer Unit)
- einer Lernregel zur Modifizierung der Verbindungsgewichte
- eine Umgebung, in der das Netz arbeitet

3.6 Units (Neuronen)

Künstliche Neuronen - die s.g. Units - orientieren sich an ihren biologischen Vorbildern, stellen aber eine idealiserte Form derer dar.

Biologisches Neuron	Künstliche Unit
Dendrit	Input
Axon	Output
Zellkörper	Unit mit Funktion

Tabelle 3.2: Vergleich biologisches Neuron und künstliche Unit

In Abhängigkeit der Schichten in denen sich die Units befinden unterscheidet man:

- Input Unit: Empfängt Signale von der Außenwelt (Reize, Muster)
- Hidden Unit: Befinden sich zwischen Input- und Output-Unit und beinhalten eine interne Repräsentation der Außenwelt.
- Output Unit: Diese geben Signale an die Außenwelt ab.
- Bias Unit: Diese Unit erhält keinen Input; das Aktivitätslevel ist immer +1. Solche Units werden verwendet, wenn z.B. eine Unit häufiger feuern soll, oder man Schwellen benötigt (negatives Bias).

(vgl. [Roj96] Rojas 1996, S. 124ff, [PJS08] Sauer 2008, S. 17,18)

Eine Unit erhält ein oder mehrere Eingangssignale; diese werden innerhalb der Unit durch eine Summenfunktion aufaddiert und variieren so den Output der Unit. Dabei entscheidet die der Unit zugrunde liegende Aktivitätsfunktion, wann der Schwellenwert erreicht ist, bzw. in welcher Form das Output-Signal von den summierten Inputsignalen beeinflusst wird (auf die Aktivitätsfunktionen wird später genauer eingegangen).

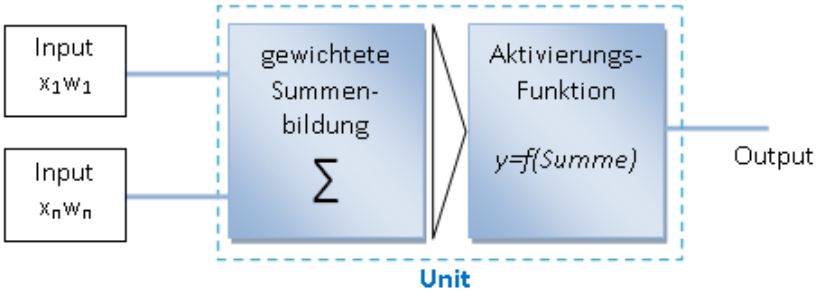


Abbildung 3.6: Modell einer Unit

Um ein künstliches neuronales Netz zu bilden, werden die einzelnen Units in einer strukturierten Weise mit anderen Units verbunden (siehe Kapitel 3.7). Diese Verbindungen sind entscheidend für die Arbeitsweise und Lernfähigkeit des Netzes. Jeder Verbindung wird ein Wert zugewiesen - sie wird *gewichtet*. Der Signalwert eines Eingangssignals einer Unit wird mit diesem Gewichtungswert multipliziert. Je größer der Absolutbetrag des Ergebnisses ist, desto größer ist der Einfluss einer Unit auf eine andere Unit.

- Ein positives Gewicht bewirkt, dass eine Unit auf eine andere Unit einen exzitatorischen, erregenden Einfluss ausübt.
- Ein negatives Gewicht bedeutet, dass der Einfluss inhibitorisch, also hemmender Natur ist.
- Ein Gewicht mit dem Wert Null besagt, dass eine Unit auf ein andere Unit keinen Einfluss ausübt.

(vgl. [GDR08] 09.07.2007, [And95] Anderson 1995, S. 216)

Das eigentliche „Wissen“ eines neuronalen Netzes ist in diesen Gewichtungen gespeichert. Wobei man das Lernen bei künstlichen neuronalen Netzen als Gewichtsveränderungen zwischen den Units definieren kann. (vgl. [GDR08] 09.07.2007)

Der Inputwert, den eine Unit von einer anderen Unit empfängt, hängt vom Aktivitätslevel der sendenden Einheit, und vom Gewicht zwischen den beiden Units ab. Diese sind multiplikativ miteinander verknüpft. Je stärker der Aktivitätslevel der sendenden Einheit und je höher das Gewicht zwischen den beiden Units, desto größer ist der Input der empfangenden Einheit. Ist einer der beiden Terme gleich

Null, so ist kein Einfluss vorhanden.
Der gesamte Input - der Netz-Input - setzt sich dabei folgendermaßen zusammen:

$$netinput_i = \sum_j a_j \cdot w_{ji} \tag{3.1}$$

a_j = Aktivitätslevel der sendenden Unit j
w_{ji} = Gewicht zwischen der sendenden (j) und der empfangenden Unit i

Die Abbildung 3.7 und die Formel 3.2 zeigen die Berechnung des Netz-Inputs eines einfachen Perzeptrons:

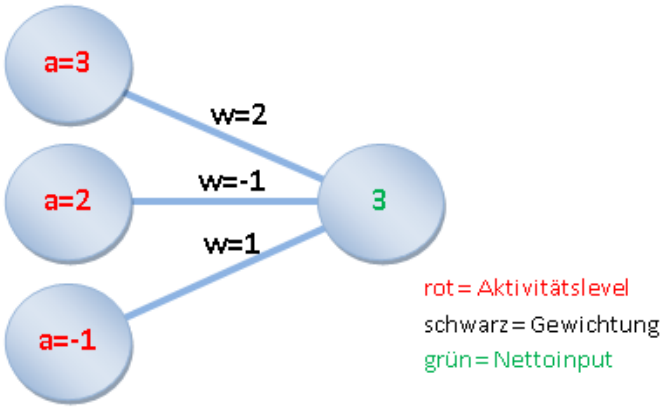


Abbildung 3.7: Netz-Input eines Perzeptrons

$$netinput = \sum_j a_j w_{ji} = (3 \cdot 2) + (2 \cdot -1) + (-1 \cdot 1) = 3 \tag{3.2}$$

3.6.1 Aktivitätsfunktionen

Den eigentlichen Output einer Unit bestimmt eine Übertragungsfunktion - die Aktivitätsfunktion. Diese stellt den Zusammenhang zwischen Netz-Input und dem Aktivitätslevel einer Unit dar. Die Funktionen werden in einem zweidimensionalen Diagramm dargestellt, wobei die Abszisse dem Netz-Input entspricht und die Ordinate dem Aktivitätslevel zugeordnet ist. Somit berechnet sich der Output einer Unit durch:

$$output = f(u) = f\left(\sum_j a_j \cdot w_{ji}\right) \tag{3.3}$$

(vgl. [And95] Anderson 1995, S. 57)

Man unterscheidet folgende Aktivitätsfunktionen:

3.6.1.1 Lineare Funktion

Bei der linearen Aktivitätsfunktion (Abb. 3.8) verhalten sich der Netz-Input und die Aktivitätsfunktion zueinander linear. Da diese Funktion keine oberen und unteren Grenzen kennt, können für die biologische Plausibilität obere und untere Grenzwerte festgelegt werden. Man spricht dann von *linear threshold*.

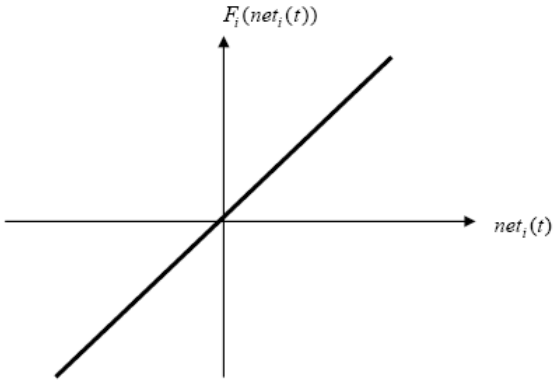


Abbildung 3.8: Lineare Aktivitätsfunktion

3.6.1.2 Linear mit Schwelle

Die Aktivitätsfunktion *Linear mit Schwelle* (Abb. 3.9) wird für Units verwendet, die zuerst einen bestimmten Schwellwert überschreiten müssen damit ein linearer Zusammenhang zwischen Aktivitätsfunktion und Netz-Input eintritt. Mit Hilfe solcher Schwellen kann beispielsweise ein Rauschen unterdrückt werden. Sauer (vgl. [PJS08] Sauer 2008, S. 21) spricht hier auch von einer *halblinearen Funktion*.

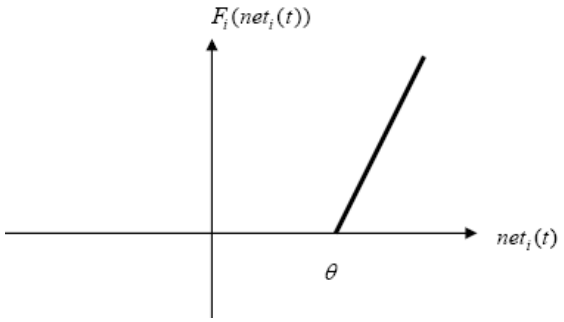


Abbildung 3.9: Lineare Aktivitätsfunktion mit Schwelle

3.6.1.3 Binäre Schwellenwertfunktion (threshold)

Diese Funktion (Abb. 3.10) kennt nur die Zustände 0, 1. Je nach Anwendung kann diese auch mit einem Schwellenwert definiert werden.

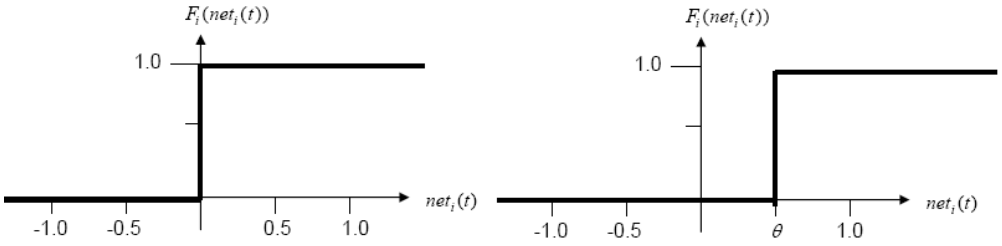


Abbildung 3.10: Binäre Schwellenwertfunktion

3.6.1.4 Sigmoide Funktion

Diese Aktivitätsfunktion (Abb. 3.11) wird in jenen Modellen verwendet, die kognitive Prozesse simulieren. Dies hat den Vorteil, dass die Funktion natürlich Grenzen kennt (bei einem hohen Netz-Input nähert sich der Wert asymptotisch der 1).

$$f(u) = \frac{1}{1 + e^{-u}} \tag{3.4}$$

(vgl. [And95] Anderson 1995, S. 57, [GDR08] 09.07.2007)

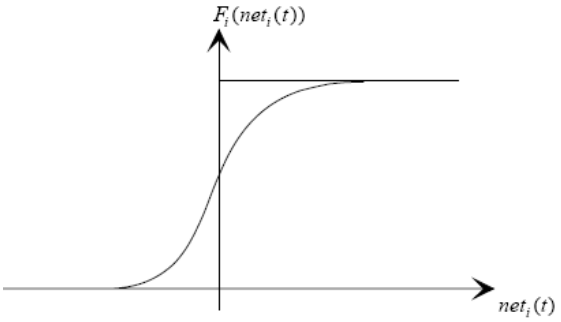


Abbildung 3.11: Sigmoide Aktivitätsfunktion

3.7 Netztypen

Die Topologie eines künstlichen neuronalen Netzes ist von verschiedensten Merkmalen abhängig. Die Netzwerktypen wurden aufgrund unterschiedlicher Motivationen und Zielsetzungen entwickelt. Ein Teil der Netze wurde bspw. durch physikalische Modelle inspiriert, während andere Netzwerke biologische Wurzeln haben. Sauer (vgl. [PJS08] Sauer 2008, S. 28) beschreibt folgende Merkmale:

- Art der Lernregel
- Mit oder ohne Zwischenschicht (Hidden-Layer)
- Feedforward oder feedback
- Annahme binärer oder stetiger Ereignisse

Mittlerweile existieren mehr als 20 in der Wissenschaft eingesetzte Modelle. Prinzipiell unterscheidet man *Feedforward* und *Feedback* (rekurrente) Netze.

3.7.1 Feedforward Netze

Die Information breitet sich von den Input-Units beginnend über einen möglichen Hidden-Layer bis zu den Output-Units aus, ohne dass eine Rückkopplung stattfindet (z.B.: Perzeptron). Folgende Varianten unterscheidet man:

- **Pattern Associator**
  Diese Netze haben keine Hidden-Units; sie dienen der Mustererkennung, indem sie Assoziationen zwischen verschiedenen Reizpaaren bilden. Als Lernmethoden werden die Hebb-Regel oder die Delta-Regel eingesetzt (siehe Kaptiel 3.8). (vgl. [GDR08] 09.07.2007)
- **Kompetitive Netze**
  Haben wie das Pattern Associator-Netz keinen Hidden Layer. Das Lernen erfolgt *unsupervised* und gliedert sich in drei Schritte:
  1. Erregung
  2. Wettbewerb
  3. Adjustierung der Gewichte

(vgl. [PJS08] Sauer 2008, S. 168-170, [GDR08] 09.07.2007)

- **Kohonennetze**
  - auch *Self-Organizing Maps* (SOM) (vgl. [PJS08] Sauer 2008, S. 171ff) genannt, wurden von Teuvo Kohonen 1982 entwickelt und stellen eine Erweiterung kompetitiver Netze dar. Es ist kein definierter korrekter Output vorhanden, sodass das Lernen ebenfalls unsupervised stattfindet. (Selbstorganisiertes Lernen über eine topologische Nachbarschaftsbeziehung.) Ein Vorteil von Kohonennetzen liegt in der biologischen Plausibilität, da das menschliche Gehirn meist Probleme ohne externen Lehrer löst. Anwendung finden Kohonennetze bei Inverser Kinematik (Robotik), dem Traveling Salesman Problem[11], und der Musterkennung (Spracherkennung, Gesichtererkennung). (vgl. [GDR08] 09.07.2007, [And95] Anderson 1995, S. 464-490)

3.7.2 Feedback Netze

Bei Feedback Netzen (rekurrente Netze) exisitieren Rückkopplungen von Units einer Schicht zu anderen Units derselben, oder einer vorangegangenen Schicht. Solche Netze werden eingesetzt, um Zeitabhängigkeiten bei Daten, wie z.B. die Struktur einer Schwingung, modellieren zu können. Über die Rückkopplung erhält man als Netzeingabe sowohl die neuen Daten, als auch die bereits verarbeiteten „alten“ Daten. (vgl. [WU08] 19.08.2008)

Man unterscheidet zwischen folgenden rekurrenten Netzen:

- **Netze mit direkten Rückkopplungen (direct feedback)**
  Entspricht einer direkten Verbindung von Output zu Input ein und derselben Unit. Damit wird die eigene Aktivierung vom Output abhängig abgeschwächt oder verstärkt.
- **Netze mit indirekten Rückkopplungen (indirect feedback)**
  Dies sind Rückkopplungen von Units höherer Ebenen zu Units niederer Ebenen. Dadurch erreicht man eine Aufmerksamkeitssteuerung auf bestimmte Bereiche von Eingabe-Units oder auf bestimmte Eingabemerkmale durch das Netz.

[11] Informationen zum Thema: http://www.tsp.gatech.edu

- **Netze mit Rückkoppelungen innerhalb einer Schicht (lateral feedback)**
  Rückkopplungen innerhalb derselben Schicht werden als lateral feedback bezeichnet. Diese Netz werden oft für Aufgaben eingesetzt, bei denen nur eine Unit einer Gruppe aktiv werden soll. Jede Unit hat dann hemmende Verbindungen zu den anderen Units und meist eine aktivierende direkte Rückkoppelung zu sich selbst. Die Unit mit der stärksten Aktivierung, der „Gewinner", hemmt dann die anderen Neuronen. Man spricht auch von *winner-takes-it-all*-Netzwerk. (vgl. [WU08] 19.08.2008)
- **vollständig verbundene Netze**
  Jede Unit ist mit allen anderen Units verbunden. Diese Topologie wird auch Hopfield-Netz[12] genannt.

(vgl. [PJS08] Sauer 2008, S. 30)

[12] Informationen zum Thema: http://www.comp.nus.edu.sg/~pris/AssociativeMemory/HopfieldModel.html

3.8 Lernregeln

Das Lernen ist ein weitläufiger Begriff - für künstliche neuronale Netze ist es immer eine Frage der Implementierbarkeit. Kriesel ([Kri08] Kiesel: 43) nennt folgende Vorgänge beim Lernen eines KNN:

- Neue Verbindungen entwickeln
- Vorhandene Verbindungen löschen
- Verbindungsgewicht ändern
- Schwellenwerte von Neuronen ändern
- Neue Neuronen entwickeln
- Neuronen löschen

Die Gewichtsveränderung ist die gängigste Methode. Das Löschen wird sehr einfach realisiert indem die Verbindung auf 0 gesetzt wird (es wird daher nicht mehr trainiert). Das Erstellen und Löschen von Units ist bei speziellen Anwendungsfällen nötig und wird von der eigentlichen Implementierung des KNNs durchgeführt. Während der Trainingsphasen erlernt das künstliche neuronale Netz sukzessiv das Wissen, um die dem Netz aufgetragene Problemstellung lösen zu können. Die Lernregeln bilden die Basis zur Veränderung der Gewichte zwischen den Units. In der Regel werden Gewichte solange verändert, bis die Zuordnung der Eingabemuster zu einer gewünschten Ausgabe erreicht wird. Das Lernen kann *supervised* (beaufsichtigt), oder *unsupervised* (unbeaufsichtigt) erfolgen.

3.8.1 Supervised learning

Supervised learning - überwachtes Lernen - setzt eine existierende Trainingsmenge voraus, die aus bekannten Eingabemustern und den entsprechenden Ausgabewerten besteht. Für das eingegebene Trainingsmuster kann so beispielsweise die Ausgabe direkt mit der korrekten Lösung verglichen werden und anhand der Differenz die Netzgewichtungen modifiziert werden. Ziel ist eine Veränderung der Gewichte, sodass das Netz nach dem Training nicht nur selbständig Ein- und Ausgabemuster assoziieren, sondern bis dato unbekannte, ähnliche Eingabemuster einem plausiblen Ergebnis zuführen kann. (vgl. [Kri08] Kiesel 2005, S. 44)

3.8.1.1 Hebb-Regel

Eine der einfachsten Lernregeln mit großer biologischer Plausibilität wurde von Donald Olding Hebb 1949 formuliert (siehe Kapitel 3.3). „*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*“ ([Heb02] Hebb 2002, S. 62)
Diese Regel besagt: Falls zwei Einheiten gleichzeitig aktiv sind, dann wächst die Stärke ihrer Verbindung. Die Größe der Gewichtsveränderung bemisst sich aufgrund dreier Werte:

- dem Aktivitätslevel der sendenden Unit $\mathbf{a_j}$
- dem Aktivitätslevel der empfangenden Unit $\mathbf{a_i}$
- Lernrate ε ($0 \leq \varepsilon \leq 1$; 0=nicht lernen, 1=bereits gelernte Muster werden vermutlich zerstört)

Die Hebb-Regel ist nur auf Netze ohne Hidden-Layer anwendbar und ist folgendermaßen definiert:

$$\triangle w_{ij} = \varepsilon \cdot a_i \cdot a_j \tag{3.5}$$

(vgl. [PJS08] Sauer 2008, S. 42)

3.8.1.2 Delta-Regel

Die Delta-Regel (Widrow & Hoff-Regel) beruht auf einem Vergleich zwischen dem gewünschten und dem tatsächlich beobachteten Output einer Ausgabe-Unit. Aus der Differenz von gewünschtem und erzieltem Ausgabeverhalten berechnet sich ein Fehlerkorrekturwert, der die Gewichte in Abhängigkeit von diesem Fehlerkorrekturwert ändert. Diese Regel ist nur für Netzwerke mit zwei Layern (Input-Layer und Output-Layer) anwendbar. Für mehrschichtige Modelle muss das Backpropagation-Lernverfahren eingesetzt werden. (vgl. [PJS08] Sauer 2008, S. 25, 49)

Man kann drei Möglichkeiten unterscheiden:

- Die beobachtete Aktivität ist zu niedrig. Um die Aktivität zu steigern werden die Gewichte zwischen den sendenden und der empfangenden Unit erhöht, sofern von den sendenden Einheiten ein positiver Input ausgeht. Die Gewichte zu den sendenden Units mit negativem Input werden hingegen gesenkt.
- Die beobachtete Aktivität ist zu groß. Da die Aktivität hier reduziert werden soll, schwächt man alle Verbindungen bei denen der Input positiv ist und stärkt diejenigen Verbindungen mit negativem Input.
- Die beobachtete und gewünschte Aktivität sind gleich groß (= gewünschtes Resultat). In diesem Fall erfolgen keinerlei Veränderungen.

([GDR08] 09.07.2007)

Diese drei Möglichkeiten sind in der Delta-Formel berücksichtigt:

$$\triangle w_{ij} = \varepsilon \cdot \delta_i \cdot a_j \tag{3.6}$$

$\delta_i = a_i(soll) - a_i(ist)$: Entspricht dem Fehlersignal der Unit i

3.8.1.3 Backpropagation

Die bisher dargestellten Lernregeln gelten nur für neuronale Netze ohne Hidden-Layer. Für Netze mit verborgenen Schichten wird daher der Backpropagation-Algorithmus verwendet.
„*Das Backpropagation-Verfahren ist ein Verfahren des überwachten Lernens zum Anpassen der Verbindungsgewichte. Die Gewichtsänderung erfolgt ausgehend von einem aus der Abweichung der berechneten Ausgabe von der erwarteten Ausgabe bestimmten Fehlersignals. Die Gewichtsänderung wird schichtweise, beginnend mit den Verbindungen zur Ausgabeschicht rückwärts in Richtung Eingabeschicht, vorgenommen.*“ ([UL04] Lämmel et al. 2004, S. 192)
Heutzutage greifen KNNs, die konkrete Anwendungsprobleme lösen sollen, typischerweise auf das Backpropagation-Verfahren zurück.

In der Trainingsphase wird jede Gewichtsveränderung in drei Schritte unterteilt:

1. **Forward-Pass:** Zunächst werden den Input-Units Reize präsentiert und der Output des neuronalen Netzes berechnet.

2. **Fehlerbestimmung für die Output-Units:** Die gewünschten Output-Werte werden mit den im forward-pass tatsächlich ermittelten Werten verglichen. Überschreitet der Fehler die vorgegebene Güteschwelle, erfolgt der dritte Schritt. Sind die Fehler klein genug und überschreiten die Güteschwelle nicht, kann die Trainingsphase abgebrochen werden.

3. **Backward-Pass:** Die Fehlerterme breiten sich nun in entgegengesetzter Richtung bis zur Input-Schicht aus. Mit Hilfe dieser Fehlerterme werden nach und nach die Gewichte der Units jedes Layers modifiziert, um die Fehlerterme zu reduzieren.

(vgl. [GDR08] 09.07.2007)

Die Gewichte werden üblicherweise mit dem *Gradientenabstiegsverfahren* modifiziert. Das Gradientenabstiegsverfahren startet mit einer zufällig gewählten Gewichtskombination. Für diese wird der Gradient[13] bestimmt und um eine vorgegebene Länge - der Lernrate - hinabgestiegen (Gewichte werden verändert). Für die neu erhaltene Gewichtskombination wird wiederum der Gradient bestimmt und abermals eine Modifikation der Gewichte vorgenommen. Dieses Verfahren wiederholt sich so lange, bis ein lokales Minimum erreicht ist. (vgl. [GDR08] 09.07.2007)

Für die Berechnungen ist der Fehlerwert über folgende Formel definiert:

$$E = \frac{1}{2} \cdot \sum_{k} (t_k - o_k)^2 \tag{3.7}$$

$t_k - o_k$: Lernfehler eines Ausgabeneurons k für ein vorgegebenes Muster.

Der Backpropagation-Algorithmus ist bestimmt durch:
Ausgabeschicht:

$$\delta_k = (t_k - o_k) \cdot o_k \cdot (1 - o_k) \tag{3.8}$$

Verborgene Schicht:

$$\delta_j = o_j \cdot (1 - o_j) \cdot \sum_{k} \delta_k \cdot w_{jk} \tag{3.9}$$

(vgl. [PJS08] Sauer 2008, S. 86-89)

[13] Differentialoperator, der auf ein Skalarfeld angewandt wird

3.8.2 Unsupervised learning

Unüberwachtes Lernen ist die biologisch plausibelste Methode, die aber nicht für alle Fragestellungen geeignet ist. Beim unsupervised learning ist nur ein Eingabemuster gegeben; das Netz versucht Muster zu identifizieren und ähnliche Muster in passenden Kategorien zu klassifizieren. (vgl. [Kri08] Kiesel 2005, S. 44)

3.8.2.1 Competetive learning

Das kompetitive Lernen (Wettbewerbslernen) arbeite ohne Vorgabe eines korrekten, bekannten Outputs. Das neuronale Netz versucht anhand der Ähnlichkeit der Input-Muster eine Kategorisierung durchzuführen. Beim Competitive Learning können drei Phasen unterschieden werden:

1. **Erregung (excitation):** Es wird über die Formel 3.1 des Net-Inputs der Output berechnet.
2. **Wettbewerb (Competition):** Die Netzinputs sämtlicher Output-Units werden miteinander verglichen. Die Unit mit dem höchsten Netzinput ist der Gewinner.
3. **Adjustierung der Gewichte (weight adjustment):** Im letzten Schritt werden die Gewichte bei allen Verbindungen verändert, die zur Gewinner-Unit führen. Alle anderen Gewichte werden nicht verändert („The Winner takes it all“ - Prinzip; siehe Kaptiel 3.7.2).

Die Verbindungen zum Gewinner werden so modifiziert, dass sich die Gewichte dem Input annähern (a_j - w_{ij}).
Das Wettbewerbslernen kann ausgedrückt werden durch:

$$\triangle w_{ij} = \varepsilon \cdot (a_j - w_{ij}) \tag{3.10}$$

(vgl. [GDR08] 09.07.2007)

Kapitel 4

Entwicklung eines KNN-Klassifizierers

4.1 Methodik des Verfahrens

Dieses Kapitel befasst sich mit der Erstellung eines Verfahrens zur Spam-Klassifizierung mittels KNN sowie der Umsetzung als Software-Projekt und der anschließenden Analyse mittels einer Teststellung.

Überlegung

Für das künstliche neuronale Netz eines Mail-Klassifizierers ist sowohl die Lernmenge (Sample), als auch das Ergebnis jedes Samples (Spam oder Ham) bekannt. Es gilt lediglich eine Klassifizierung - also ob eine Mail Spam oder Ham ist - durchzuführen. In Artikel (vgl. [Dir06] Dirmeikis, 2006, S. 73-76) wird beschrieben, dass ein Backpropagation-Lernalgorithmus für diese Problemstellung zielführend ist, da sowohl der Input-Reiz, als auch der Output bekannt sind. (Kohonennetze und Kompetetive Netze erlernen das Wissen *unsupervised* und sind daher nicht geeignet; siehe auch Kapitel 3.)

Backpropagation ist für Netztopologien mit Hidden-Layer konzipiert, sodass für das KNN ein feedforward multilayer Perzeptron (MLP) zum Einsatz kommt. Der Merkmalsvektor (bestehend aus Wort-Merkmalen in Form einer *bag of words*-Darstellung - siehe Kapitel 2.5) wird an den Units des Input-Layers angelegt. Das Ergebnis des KNNs ist bekannt: 0 entspricht Ham, 1 bedeutet Spam. Dieses Ergebnis wird durch zwei Units im Output-Layer repräsentiert (Ham-Wahrscheinlichkeit und Spam-Wahrscheinlichkeit).

Durch das Backpropagation-Verfahren werden die Verbindungen der Units vom Output-Layer zurück zum Input-Layer gewichtet. Dieser Vorgang wird solange

wiederholt, bis die Fehlerrate unter einen definierten Wert sinkt. Dies ermöglicht auch einfache, nachträgliche Lernphasen durch neue, bis dato unklassifizierte *false negatives* und *false positives*.

Die Größe des Hidden-Layer

Herausfordernd gestaltet sich die Festlegung der richtigen Größe des Hidden-Layers für eine optimale Effizienz des KNN. Laut Lämmel (vgl. [UL04] Lämmel et al. 2004, S. 217-219) lässt sich die Speicherkapazität - und damit die Größe des Hidden-Layers eines künstlichen neuronalen Netzes - mathematisch nicht berechnen. Dennoch muss eine passende Größe für den Hidden-Layer gefunden werden. Eine zu große innere Schicht führt zu einem Auswendiglernen der Trainingsdaten und damit zu hohem Speicherbedarf. Das erlernte Wissen lässt sich nur schlecht auf neue Problemstellungen anwenden - es ist daher kaum generalisierungsfähig.
Die optimale Größe lässt sich näherungsweise bestimmen, indem man das Netz mit einer Musterdatenmenge (zufälliges Rauschen) trainiert. Mit jedem neuen Training wird die Anzahl der Muster erhöht bis der Trainingsfehler größer Null ist. (Das Netz beginnt zu abstrahieren da keine Zusammenhänge mehr erkennbar sind.) Somit lässt sich die Speicherbedarf des Netzes ausloten. (vgl. [UL04] Lämmel et al. 2004, S. 217-219)

Vorbereitende Prozesse

Die Daten einer E-Mail sind in ihrer ursprünglichen Form nur marginal für die Klassifizierung geeignet da sich Wörter oft wiederholen, oder in Steigerungsformen vorkommen. Um die empfangene E-Mail zu parsen und die resultierende Datenmenge für das künstliche neuronale Netz vorzubereiten, bedarf es daher mehrerer vorbereitender Verarbeitungsschritte.
Wie schon der Artikel *A neural network based approach to automated e-mail classification* ([CKP03] Clark et al. 2003, S. 73-76) beschreibt, muss die zu prüfende Nachricht mehrere Prozesse passieren bevor sie das MLP durchläuft (siehe Kapitel 4.2.2):

- Entfernen von unnötigen Delimitern <>[]{}|#\%-_&*,.:@ #'+=
- Aussagekräftige Delimiter speichern (!, $, ?)
- Stemming words (Reduzierung auf den Wortstamm: *finally* wird zu *final*)

- Entfernen von Pronomen und Konjunktionen
- Rauschunterdrückung und Bestimmung der Entropie[1] (Alternativen: *Information Gain*, *LVQ* oder *Mutal Information*)

4.1.1 Struktureller Aufbau

Das beschriebende Procedere ist in der im Rahmen dieser Diplomarbeit erstellten Software „nAntiSpam“ eingearbeitet. Das Verfahren wurde so konzipiert, dass die Software in Schichten (Layer) mit definierten Schnittstellen (Interface) entwickelt werden kann. Ziel dieser Struktur ist eine hohe Abstraktion und Kapselung der einzelnen Layer. Durch den Tausch von einzelnen Layern kann das System an verschiedene Umgebungen und Rahmenbedingungen angepasst werden (siehe Abb. 4.1). Ebenso ist ein API[2] vorgesehen, um einerseits externen Filtern die zu prüfende Nachricht zur Verfügung zu stellen, und andererseits das Ergebnis dieses Filters nAntiSpam zur weiteren Verarbeitung zu übergeben (die einzelnen Schichten werden in Kapitel 4.2 im Detail erklärt).

- Der Layer *MailConnector* verarbeitet den Daten-Input und kann je nach Implementierung als AddIn für Mailclients (Outlook, Notes, etc.), oder als SMTP-Relay eingesetzt werden. Dieser Layer ist durch das Assembly *NAntiSpamMailConnector.dll* realisiert.
- Der Kern des Verfahrens findet sich im *Core-Layer* (Assembly *NAntiSpamMailCore.dll*) wieder. Hier erfolgt die eigentliche Aufbereitung der Daten und die Klassifizierung.
- Am Ende der Prozesskette übernimmt der *Executor* (Assembly *NAntiSpamMailExecuter.dll*) die ihm aufgetragenen Vorgänge um die analysierte Mail zu „taggen“[3], zu löschen oder - wenn es sich um einem SMTP-MailConnecter handelt - per SMTP weiterzuleiten.

[1] Entropie ist ein Maß für den mittleren Informationsgehalt oder auch Informationsdichte; (vgl. [Sha63] Shannon 1963)
[2] „Application Programming Interface“ Anm. d. Verfassers
[3] Anm. d. Verf.: Kennzeichnung durch Veränderung des Betreffs, des Inhalts, oder der Header-Daten

Tabelle 4.1 zeigt die daraus resultieren nAntiSpam-Systemkomponenten. In der linken Spalte werden die Dateien, in der rechten Spalte die jeweiligen Aufgaben aufgelistet.

Komponente	Aufgabe
nAntiSpam.exe	ausführbare Applikation
NAntiSpamMailConnector.dll	Mail-Connector-Layer
NAntiSpamCore.dll	Core-Layer
NAntiSpamExecutor.dll	Executor-Layer
executor.xml	Settings des Executors
setting.xml	Settings des Cores
stoppwords-en.xml	Wort-Ausnahmeliste (englisch)
learningset.dat	Lernmenge (Samples)
unknownset.dat	neu zu lernende Daten
inputset.dat	Wortmenge der Input-Units
ann.dat	serialisiertes KNN

Tabelle 4.1: nAntiSpam-Systemkomponenten

Die Grafik 4.1 illustriert die Struktur von nAntiSpam. Die einzelnen Verarbeitungsschritte sind nach Layer gegliedert und beginnen mit dem MailConnector (Mail-Connector Layer), führen über den Preprozessor, die Normalisierung und Klassifizierung (Core-Layer), bis hin zum Executor (Executor-Layer). Die jeweiligen Schnittstellen zwischen den Schritten werden am Rand der Prozesse beschrieben (grüne Rechtecke).
In Kapitel 4.2 werden die einzelnen Schritte im Detail erklärt.

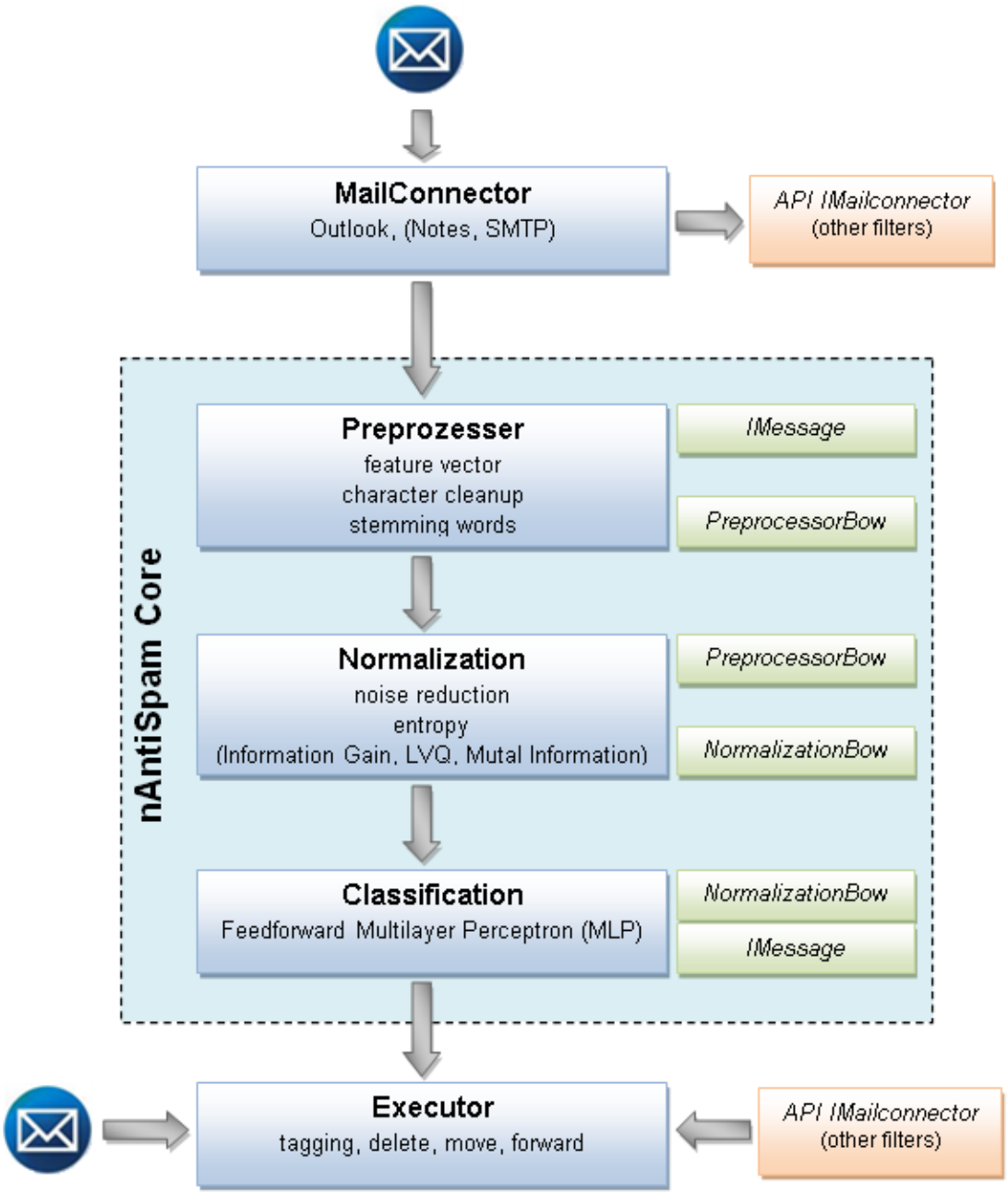


Abbildung 4.1: nAntiSpam-Struktur

4.1.2 Einsatz-Szenarien

Durch das Konzept der Schichten kann nAntiSpam in verschiedenen Umgebungen zum Einsatz kommen. Während der Core selbst unverändert bleibt, kann durch Modifizierung resp. Tausch des MailConnector-Layers und des Executor-Layers die Software die zwei Betriebsmodi *Add-In* und *SMTP-Relay* annehmen. Die Abbildung 4.2 zeigt die verarbeitenden Komponenten eines Outlook-AddIns während Abbildung 4.3 ein SMTP-Szenario illustriert.

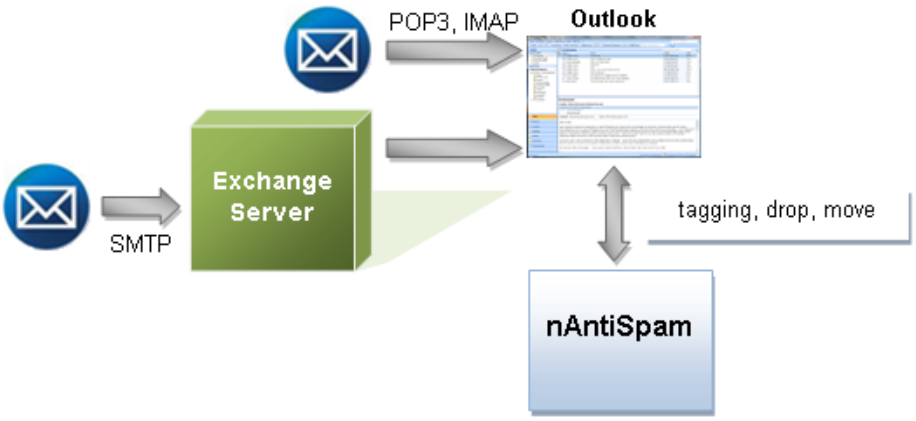


Abbildung 4.2: nAntiSpam als Add-In

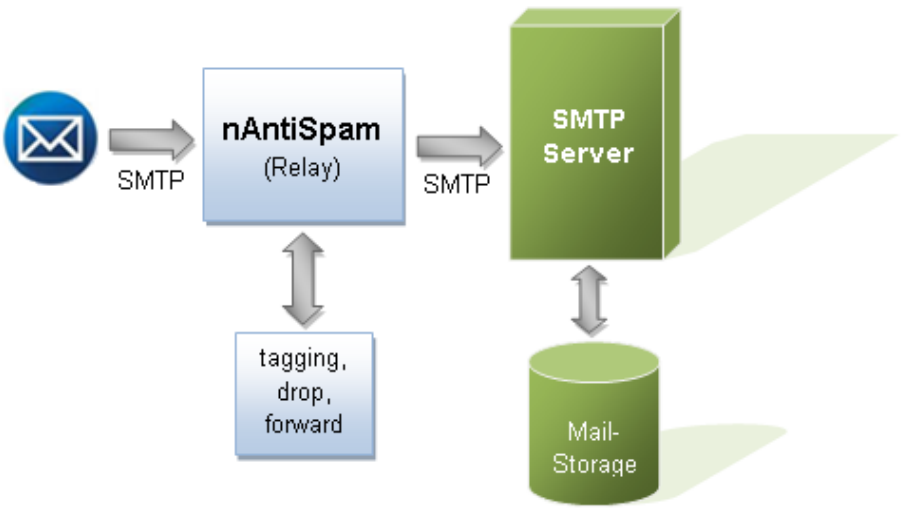


Abbildung 4.3: nAntiSpam als SMTP-Relay

4.2 Softwareentwicklung: nAntiSpam

Im Folgenden werden die einzelnen Schichten der Software *nAntiSpam* sowie deren Implementierung beschrieben. Im Rahmen dieser Diplomarbeit wurde eine Machbarkeitsstudie in Form eines Prototypen entwickelt der nicht den kompletten Funktionsumfang abdeckt, aber dennoch eine aussagekräftige Analyse zulässt. Da die gewählte Library für die Implementierung des neuronales Netzes als C# Quelltext vorliegt, wird die Software auf Basis des .NET-Frameworks[4] 3.5 mittels der Hochsprache C# erstellt.

Für die KNN-Implementierung von nAntiSpam wurden drei OpenSource-Frameworks evaluiert. Als Kriterien für die Auswahl der Frameworks wurden definiert:

1. Es muss sich um ein freies (Open-Source) Framework handeln
2. Der Quelltext soll vorhanden und modifizierbar sein
3. Eine aktive Weiterentwicklung und die Community müssen gegeben sein
4. Optional soll das Framework in der Programmiersprache C# vorliegen, um eine homogene Softwarearchitektur zu gewährleisten

- **NeuronDotNet**[5] enthält eine durchdachte Klassenstruktur und unterstützt Backpropagation und Kohonen-Netztypen (SOM). In der Praxis ist aber die Erstellung eines KNNs durch den stark verschachtelten Objektgraphen sehr langsam und zeitintensiv.
  Für die Initalisierung eines KNN mit 8.653 Input-Units und 865 Hidden-Units benötigt das Netz bereits 480MB RAM bei einer Intialisierungszeit von 13 Sekunden. Das Hinzufügen von Trainigsdaten ist durch den begrenzten physischen Speicher nur mehr eingeschränkt möglich.
- **BPSimple**[6] von Murat Firat bietet lediglich ein Backpropagation-Netz, arbeitet aber schneller und speicherschonender als NeuronDotNet. Es stellte sich bei der Evaluierung heraus, dass BPSimple konstruktionsbedingt nicht für nAntiSpam einsetzbar ist. Jedes Trainingsmuster kann nur **einem** Output-Wert zugeordnert werden. Dies ist beispielsweise bei Anwendungen

[4] Weitere Informationen: http://msdn.microsoft.com/de-de/netframework/default.aspx
[5] Weitere Informationen und Download: http://neurondotnet.freehostia.com
[6] Weitere Informationen und Download: http://www.codeproject.com/KB/cs/BackPropagationNeuralNet.aspx

der Bilderkennung sinnvoll - nAntiSpam kennt hingegen nur zwei Ausgangswerte: 0 für kein Spam und 1 für Spam. Alle Eingangsereize werden beim Training diesen beiden Werten zugeordnet. Daher scheiterte das experimentelle Setting mit diesem KNN-Framework.

- **AForge.Neuro**[7] als Teil vom AForge.NET-Framework bietet sowohl die nötigen Trainingsmechanismen (Backpropagation und sigmoide Aktivitätsfunktion) als auch ein hohe Geschwindigkeit und moderate Speichernutzung und wurde deshalb als Library für den Prototypen gewählt. Es unterliegt einem Dualem-Lizenzsystem LGPL[8] und GPL[9]. „*AForge.NET is a C# framework designed for developers and researchers in the fields of Computer Vision and Artificial Intelligence - image processing, neural networks, genetic algorithms, machine learning, etc.*“ ([Tea08] AForge.Net, 29.11.2008)

Wie Buschmann (vgl. [Bus98] Buschmann 1998, S. 32-53) beschreibt, lässt sich mit einem Layer-Architekturmuster die Software strukturieren, in Teilaufgaben zerlegen und somit in verschiedene Abstraktionsebenen gliedern. nAntiSpam isoliert grundsätzlich drei Hauptschichten: Den *Connector*, den *Core* und den *Executor* (siehe Abbildung 4.1). Die zugrundeliegende Business-Logik basiert auf zwei Design-Patterns (Design-Patterns sind bewährte, generalisierte Entwurfsmuster für wiederkehrende Entwurfsprobleme (vgl. [HH05] Herold et al. 2005, S. 641)). Die Objekte der Klassen werden durch ihren Constructor erzeugt - es werden keine *factory methods* angeboten. Das Grundgerüst (siehe Abb. 4.4) von nAntiSpam ist als klassenbasiertes *stuctural pattern* (Strukturmuster) in Form eines *composite* (Kompositum) angelegt (vgl. [HH05] Herold et al. 2005, S. 678ff). Die Abbildung der *bag of words*-Klassen (siehe Abb. 4.5) entspricht einem *behavioural pattern* (Verhaltensmuster) und ist der *template method* (Schablonenmethode) zuzuordnen (vgl. [HH05] Herold et al. 2005, S. 721ff).

Die visualisierte UML (Abbildung 4.4) zeigt den Kern von nAntiSpam. Eine neue Nachricht wird über eine Instanz der Klasse *NAntiSpamMailConnector* mittels des Event-Handlers *NewMail* an den Core übergeben. Als Schnittstelle dient die Klasse *NAntiSpamMessage* die das Interface *IMessage* implementiert. Diese

[7] Weitere Informationen und Download: http://code.google.com/p/aforge
[8] Weitere Informationen: http://www.gnu.org/licenses/lgpl.html
[9] Weitere Informationen: http://www.gnu.org/licenses/gpl.html

Schnittstelle ist auch für das API vorgesehen.
Die Klasse *NAntiSpamCore* wird von der eigentlichen Anti-Spam-Applikation instanziert und führt über die Methode *PerformCheck* die Prüfung der Nachricht durch.
Die Methode *Execute* der Klasse *NAntiSpamExecutor* übt - je nach Einstellung (executor.xml) - die über den Enumerator *ExecutorAction* definierte Aktion aus.

Die Abbildung 4.5 illustriert das UML-Diagramm der *bag of words* Implementierung von nAntiSpam. Die Basis bildet die abstrakte Klasse *Bow*[10] mit der generischen Liste der abstrakten Klasse *BowItem*.
Je nach Verarbeitungsschritt (siehe Abbildung 4.1), werden einerseits die abstrakten Klassen von *PreprocessorBow* und *PreprocessorBowItem*, andererseits von *NormalizationBow* und *NormalizationBowItem* implementiert. Zusätzliche Eigenschaften und Methoden - wie das Berechnen der Entropie oder die Rauschunterdrückung - sind in den jeweiligen zum Verarbeitungsschritt passenden Klassen eingearbeitet.

[10] Abkürzung für *bag of words*

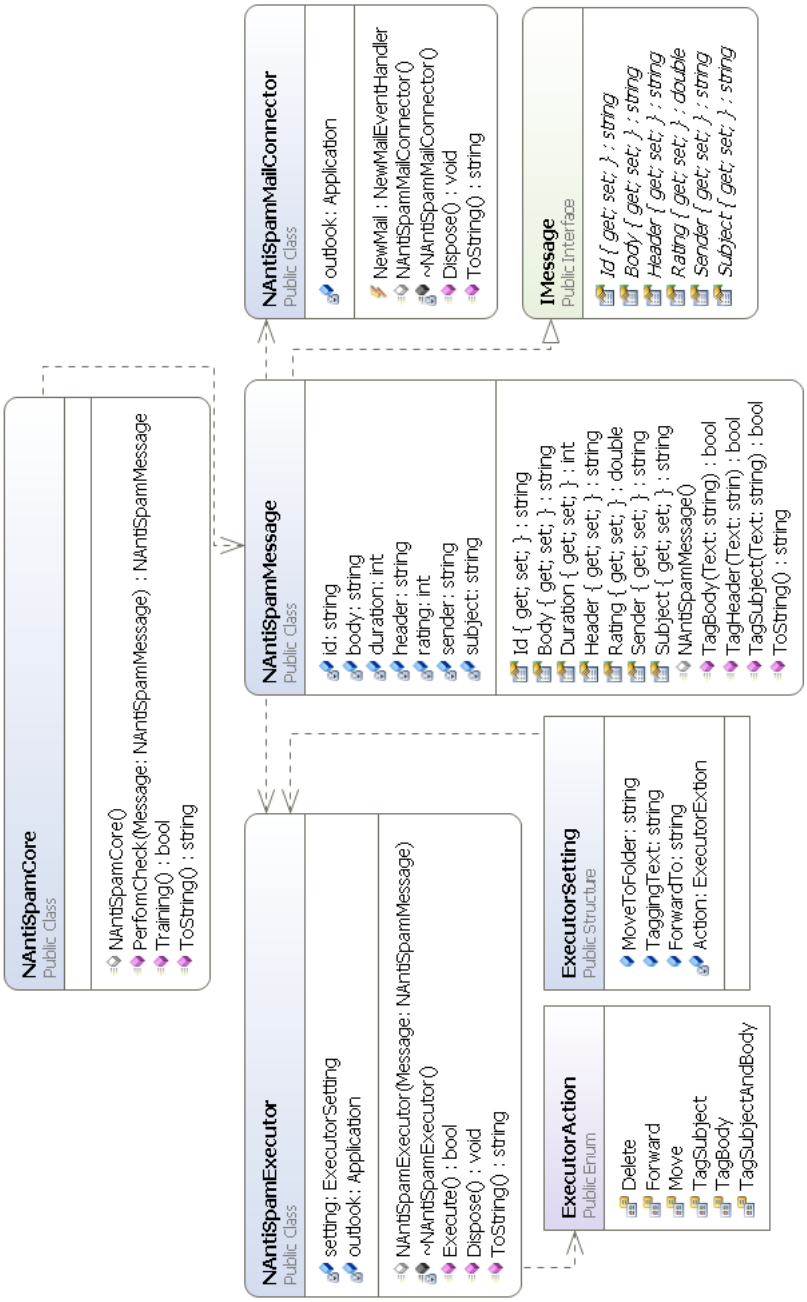


Abbildung 4.4: nAntiSpam Core (structural pattern)

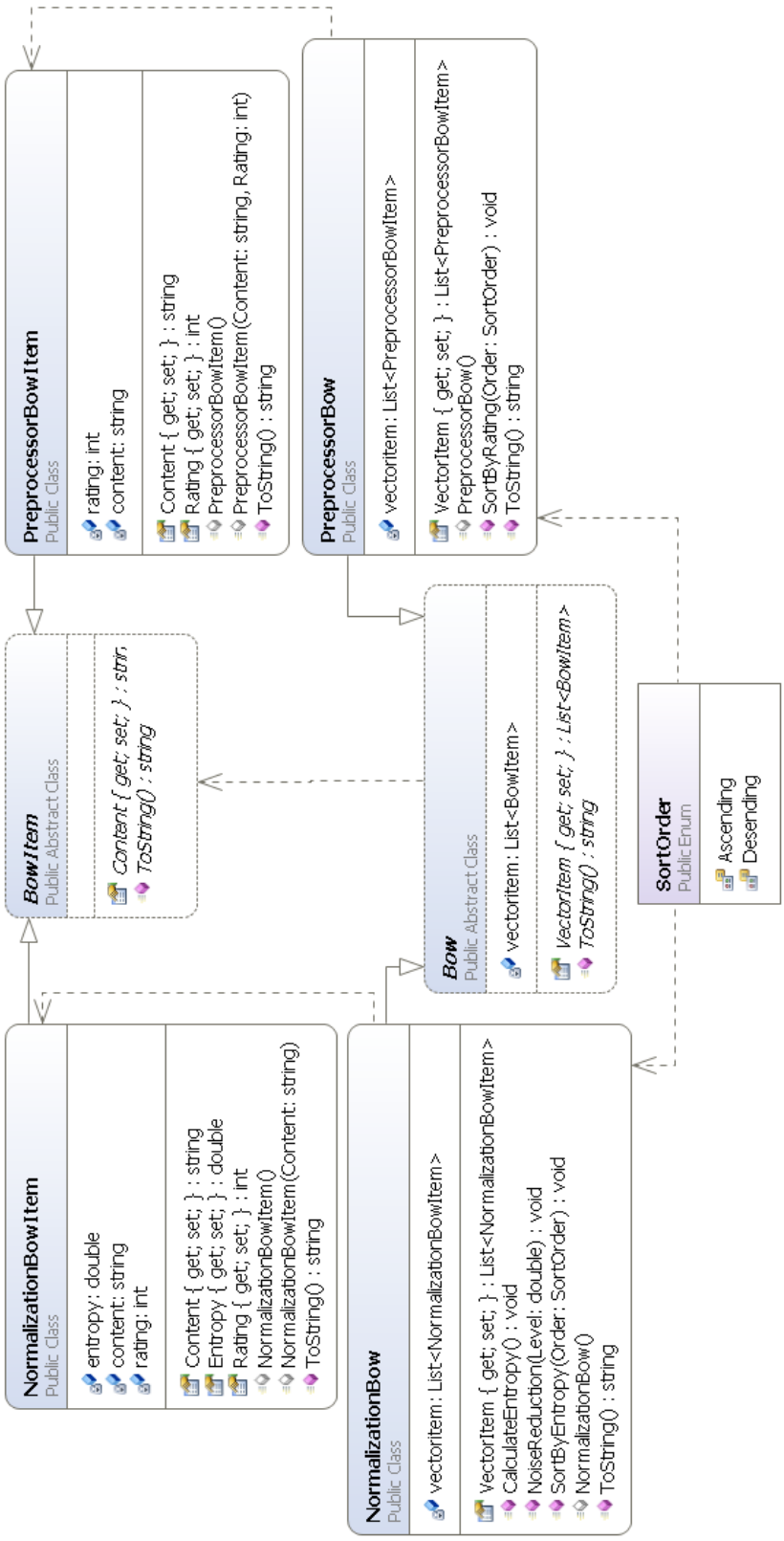


Abbildung 4.5: nAntiSpam Bag of words (behavioural pattern)

4.2.1 MailConnector Layer

Der MailConnector-Layer kann je nach Implementierung als SMTP-Relay, oder als Add-In für gängige Mail-Clients eingesetzt werden. Im Rahmen dieser Diplomarbeit wurde der Layer für Outlook 2007 von Microsoft realisiert.

Als Basis für die Implementierung wurde der MSDN-Artikel *An Introduction to Programming Outlook 2003 Using C#*[11] herangezogen.

Grundsätzlich ist diese Klasse so konzipiert, dass ein Event getriggert wird, wenn eine neue Mail vom Mail-Client empfangen wird. Die Schnittstelle zum Core-Layer - und somit das *EventArgument* des Ereignisses - ist ein Objekt der Klasse *NAntiSpamMessage*, das der neu empfangenen Mail entspricht. (Wird diese Schicht als SMTP-Relay implementiert, so bleibt diese Schnittstelle gleich.)

Outlook kann mit externen Applikationen über *COM-based in-process server* - (msoutl.olb) interagieren. Damit C# mit Outlook kommunizieren kann, muss der *interoperability layer* des globalen Assembly-Caches als Wrapper verwendet werden (*Microsoft.Office.Interop.Outlook.dll*).

Als Besonderheit ist hervorzuheben, dass das .NET Memory Management Model *non-deterministic* ist und demnach nicht genau definiert ist, wann der *Garbage Collecter* Objekte freigibt. Das COM Memory Model hingegen ist *deterministic*; daher muss der Entwickler dafür Sorge tragen, mittels der *Marshal*-Klasse (System.Runtime.InteropServices) die COM-Objekte zu zerstören (vgl. [Tro04] Troelsen, 14.10.2008).
Das folgende Listing zeigt exemplarisch die Interaktion mittels Event über den Interop-Wrapper und den Destroy-Mechanismus im Destructor.

[11] http://msdn.microsoft.com/en-us/library/aa289167.aspx

Listing 4.1: Exemplarische C# Interaktion mit Outlook

```
using System.Runtime.InteropServices;
using Microsoft.Office.Interop.Outlook;
namespace nAntiSpam
{
  public class MailConnector
  {
    public delegate void NewMailEventHandler(object sender,
        NAntiSpamMessage e);
    public event NewMailEventHandler NewMail;
    private Application outlook;
    public MailConnector()
    {
      outlook = new Application();
      outlook.NewMailEx += new ApplicationEvents_11_
          NewMailExEventHandler(outlook_NewMailEx);
    }
    ~MailConnector()
    {
      Marshal.ReleaseComObject(outlook);
    }
    private void outlook_NewMailEx(string EntryIDCollection)
    {
      //...
    }
  }
}
```

4.2.2 Core Layer

Dem Core-Layer wird ein Objekt der Klasse *NAntiSpamMessage* zur Prüfung übergeben. Als Ergebnis liefert diese Schicht einen Spam-Wahrscheinlichkeitswert von 0 bis 1. Somit kann mittels Threshold die Applikation die nAntiSpam referenziert entscheiden, ab welchem Level der Executor-Layer die Mail als Spam behandeln soll.
Der Core-Layer führt dabei folgende Schritte zur Klassifizierung der Nachricht durch:

1. **Entfernen von Trennzeichen und bilden einer Wortliste:**
   Im ersten Schritt werden bestimmte Zeichen (aus einem Set an konfigurierbaren Zeichen) aus dem Text entfernt. (Beispiel: <>[]{}|#\%-_&*,.:@ #'+=)
   Die Zeichen ! $? bleiben hingegen erhalten - sie besitzen eine wertbare Aussagekraft, da sie in unmittelbarem Zusammenhang mit der Aussage eines Satzes stehen.
   Das Ergebnis dieses Schrittes resultiert in einer generischen string-Liste deren Elemente Wörter sind, die durch Trennung der einzelnen Wörter mittels der Zeichen *space*, *linefeed*, und *carriage return* getrennt sind. Ein Filter eliminiert zusätzlich Wörter mit nur einem Zeichen sowie Wörter mit mehr als 30 Zeichen und Links (diese Wörter werden für die weitere Verarbeitung als nutzlos angesehen).

2. **Stemming words (Normalformenreduktion), Entfernen von Konjunktion und Pronomen:**
   Da gleichbedeutende Wörter in verschiedenen Ausprägungen vorkommen können, wird jedes Wort der generischen Wörterliste zum Wortstamm zurückgeführt. So haben beispielsweise die Wörter *control*, *controls*, *controlled*, *controller*, *controlling* alle den Stamm *control*. Für dieses Verfahren wird der Porter-Algorithums[12] (nach Martin Porter) eingesetzt. Dieser Algorithmus basiert auf Verkürzungsregeln, die so lange auf ein Wort angewandt werden, bis dieses eine Minimalanzahl von Silben aufweist. (vgl. [Por97] Porter, 1997, S. 313-316)
   Die Verkürzungsregeln des Porter-Algorithmus' sind so konzipiert, dass in

[12] Weitere Informationen: http://tartarus.org/~martin/PorterStemmer

Abhängigkeit der Buchstaben am Wortende, eine Reihe von Ersetzungen nach einem bestimmen Muster durchgeführt werden. Beispielsweise:

a) „sses“ ⇒ „s“, „ies“+„ied“ ⇒ „i“

b) Löschen von „ed“ „edly“ „ing“ „ingly“, usw

c) „y“ ⇒ „i“

d) „tional“ ⇒ „tion“, „entli“ ⇒ „ent“, usw

e) Löschen von „al“ „ance“ „ence“ „er“, usw

Der Porter-Algorithmus ist sprachenabhängig; nAntiSpam verwendet eine Referenzimplementierung[13] für C# für die englische Sprache. Um weitere Sprachen zu unterstützen muss einerseits die Klasse *Porter2*, als auch die Liste der Pronomen und Konjunkionen (*stoppwords-en.xml*) adaptiert werden.
Die resultierende Wortliste wird - wie in Abb. 4.5 zu sehen ist - in ein Objekt der Klasse *PreprocessorBow*, die von der abstrakten Klasse *Bow* abgeleitet wird, transformiert. Die einzelnen Wörter werden durch Objekte der Klasse *PreprocessorBowEntity* (abgeleitet von *BowEntity*) dargestellt, wobei die Häufigkeit des Wortes im Feld *Rating* gespeichert ist.

Die Stoppwörter werden von der Datei *stoppwords-en.xml* über den *Xml-Serializer* des *System.Xml.Serialization-Namespaces* geladen und ebenfalls aus dem Text der E-Mail entfernt. Diese XML-Datei beinhaltet nur Stoppwörter der englischen Sprache - für zusätzliche Sprachen ist der Einsatz von mehreren stoppwords-Dateien sinnvoll.

Listing 4.2: Auszug aus stoppwords-en.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <structConjunctionList xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/
       XMLSchema">
3    <words>
4      <string>their</string>
5      <string>themselves</string>
6      ...
7    </words>
```

[13] Weitere Informationen: http://snowball.tartarus.org/algorithms/english/stemmer.html

```
8   </structConjunctionList>
```

3. **Rauschunterdrückung und Normalisierung:**
   Bei der Klassifizierung kann die Dimension des Merkmalsvektors (Anzahl der Wörter) sehr groß werden. Damit man diese Dimensionalität niedrig hält und das inhärente Rauschen im Featureraum unterdrückt, nutzt man informationstheoretische Maße.
   nAntiSpam berechnet hierzu die Entropie jedes Wortes. Die Entropie ist ein Maß für den mittleren Informationsgehalt eines Zeichens/Wortes, das in einer Informationsfolge steckt. „*Sobald die Zeichen nicht mehr mit derselben Häufigkeit auftreten, muss deren Auftretenswahrscheinlichkeit mit berücksichtigt werden. Der Informationsgehalt wird auch mit Entropie, einem aus der Thermodynamik entlehnten Begriff, bezeichnet. Je seltener ein Ereignis (Zeichen, Meldung) ist, desto höher ist sein Entropiegrad (Informationsgehalt).*“ ([TM05] Müller et al. 2005, S. 10)
   Der Informationsgehalt eines Ereignisses H_i wird in Bit gemessen und kann aus der Auftretenswahrscheinlichkeit p_i folgendermaßen bestimmt werden:

$$H_i = ld(1/p_i) \tag{4.1}$$

   Lanquillon (vgl. [Lan01] Lanquillon, 2001, S. 40-41) zeigt noch weitere einsetzbare informationstheoretische Maße (nAntiSpam verwendet aufgrund der einfacheren Berechnung die Entropie):

   Mutual Information (MI-Maß) ist die relative Entropie zwischen Termen und Klassen.

   Information Gain wird oft als Feature-Selector für Entscheidungsbäume (siehe Kapitel 2.5.2) verwendet. Es misst die statistische Abhängigkeit zwischen einem Begriff und der Klasse und basiert ebenfalls auf der Berechnung der Entropie.

   Der letzte Teil dieses Schrittes ist Rauschunterdrückung. Sie ist so implementiert, dass durch setzen eines Tresholds (in Prozent) jene Wörter entfernt werden, deren Entropie unterhalb dieses Tresholds liegen.

4. **Klassifizierung:**
   Nachdem die Datenmenge durch den Preprozessor aufbereitet wurde, erfolgt der letzte Schritt zur Klassifizierung (siehe Abbildung 4.6).
   Die einzelnen Wörter des *NAntiSpamNormalizationBow*-Objekts werden auf Vorhandensein in der Datei *inputset.dat* geprüft. (Diese Datei entspricht der Reihe nach den Wörtern der Input-Units.) Finden sich alle Wörter in dieser Datei wieder, wird für jedes Wort die entsprechende Unit des KNNs mit dem errechneten Wert der Entropie gereizt. Das Netz errechnet dann den Level der Output-Units über die Funktion *PerformCheck*. Diese Werte zeigen von 0-1 die Ham-, und Spam-Wahrscheinlichkeit einer Mail an.
   Werden nicht alle Wörter in der Datei *inputset.dat* gefunden, so erfragt die Anwendung die Spam-Einschätzung der Mail vom Benutzer. Das Ergebnis wird mit den Wörtern in der Datei *unknownset.dat* gespeichert und so für spätere Trainingsphasen persistent gehalten.

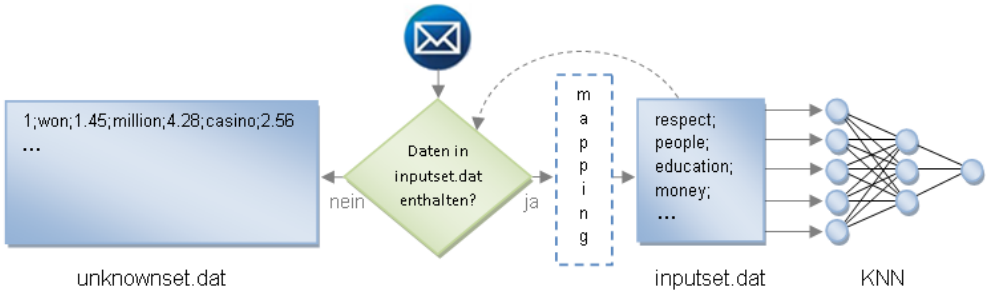


Abbildung 4.6: nAntiSpam-Klassifikationsphase

Das Format der Datei *unknownset.dat* und *learningset.dat* setzt sich wie folgt zusammen (wobei jede Zeile einem Sample=Mail entspricht):

Listing 4.3: unknownset.dat

```
1 Output[0/1][Delimiter]Wort[string][Delimiter]Entropie[double]
```

Folgend ein Beispiel der Datei *unknownset.dat*:

Listing 4.4: unknownset.dat Beispiel

```
1;hardwar;6.07;dear;6.07;profession;6.07;solarisexpert;5.07;
    offer;6.07;microsystem;5.07
1;pleas;4.41;help;4.41;alien;5.41;disguis;6.41;human;6.41;
    technolog;5.41;physic;5.41
0;re;6.25;scan;6.25;bug;5.25;updat;6.25;ftoc;4.66;run;6.25;show
    ;6.25;consid;6.25;prevent;6.25
```

Die Einstellungen des Core-Layers sind in der Struktur *structSetting* als XML-Datei *setting.xml* serialiert gepeichert und beinhalten die in Tabelle 4.2 dargestellten Daten:

Eigenschaft	Aufgabe	Default
debug	Detailierte Informationen anzeigen	false
learningrate	Lernrate (max. Fehlerwert)	0.3
hiddenlayer_size	Hiddenlayer-Größe in % vom Input-Layers	0.1
sigmoidalphavalue	Schrittweite der Sigmoiden Funktion	0.1
inputset_treshold	Min. Vorhandensein eines Wortes	3
training_end	Fehlerwert wann Training beendet wird	0.3
file_ann	Dateiname des KNN	ann.dat
file_stopwords	Dateiname der Stoppwörter	stoppwords-en.xml
file_inputset	Dateiname der Wörter der Input-Neuronen	inputset.xml
file_unknownset	Dateiname der untrainierten Daten	unknownset.xml
file_learningset	Dateiname der Trainingsdaten	learningset.xml

Tabelle 4.2: nAntiSpam-setting.xml

Das Training

Die Datei *learningset.dat* beinhaltet die bereits durch den Preprozessor verarbeiteten Daten von 1000 Spam und 1000 Ham Mail. Das Trainingsprocedere (siehe Abbildung 4.7) erstellt aus diesen Daten die Datei *inputset.dat* mit eindeutigen Wörtern, die mindestens mit der Häufigkeit des Wertes *inputset_ treshold* vorkommen. Das Mapping erfolgt der Reihe nach: Das erste Wort der Datei *inputset.dat* entspricht der ersten Unit des MLPs, das zweite Wort der zweiten Unit und so weiter.

Für den Aufbau der Datei *inputset.dat* wird im Speziellen die Klasse *Dictionary*

des Namespaces *System.Collections.Generic* verwendet, die einen erheblichen Geschwindigkeitsvorteil[14] gegenüber einer generischen Liste bei der Suche aufweist.

Jedes einzelne Sample der Datei *learningset.dat* wird zum Trainingsset des KNNs hinzugefügt. Das Training selbst wird über die Methode *RunEpoch* der Klasse *AForge.Neuro.Learning.BackpropagationLearning* so oft wiederholt, bis der Wert *training_ end* erreicht wird.

Sind untrainierte Daten in der Datei *unknownset.dat* vorhanden, können diese nach Benutzeraufforderung zum Learningset hinzugefügt werden. Das neuronale Netz wächst und lernt somit automatisch die neuen Daten. Ein wiederholtes Lernen ist daher nötig, macht aber im Kontext einer Mail-Klassifizierung Sinn: Der textuelle Aufbau der Spam-Mails ändert sich im Laufe der Zeit. Aktuelle Themen finden sich in neuen Mails wieder, während andere Texte veralten.

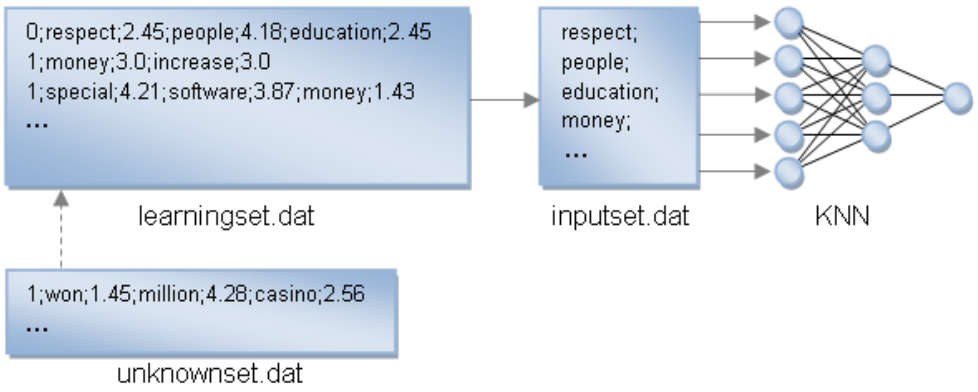


Abbildung 4.7: nAntiSpam-Trainingsphase

[14] Weitere Informationen: http://dotnetperls.com/Content/Dictionary-Time.aspx

4.2.3 Executor Layer

Die Executor-Schicht kann passend zum MailConnector-Layer (siehe Kapitel 4.2.1) als SMTP-Relay oder als Add-In für gängige Mail-Clients implementiert werden. Im Rahmen dieser Diplomarbeit wurde der Layer für Outlook 2007 realisiert. Als Basis für die Implementierung diente der MSDN-Artikel *An Introduction to Programming Outlook 2003 Using C#*[15] herangezogen.

Die Tabelle 4.3 zeigt die möglichen Aktionen des Executor-Layers die im Public Enumerator *ExecutorAction* definiert sind:

Aktion	Aufgabe
Delete	Mail wird sofort gelöscht
Forward	Mail wird an eine definierte Adresse weitergeleitet
Move	Mail wird in einen definierten Ordner verschoben
TagSubject	Betreff der Mail wird mit definiertem Text ergänzt
TagBody	Text der Mail wird mit definiertem Text ergänzt
TagSubjectAndBody	Betreff und Text der Mail werden mit definiertem Text ergänzt

Tabelle 4.3: nAntiSpam-ExecutorAction

Die Einstellungen des Executors werden über die *Public Structure ExecutorSetting* gespeichert. Um die Einstellungen persistent zu halten, wird die Struktur über den *XmlSerializer* des *System.Xml.Serialization-Namespaces* in eine XML-Datei serialisert und von dort beim Initialisieren eines Objekts der Executor-Klasse geladen.

Listing 4.5: executor.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <ExecutorSetting xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
      " xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <MoveToFolder>junk-e-mail</MoveToFolder>
4   <TaggingText>*** nAntiSpam ***</TaggingText>
5   <ForwardTo>admin@mycompany.com</ForwardTo>
6   <Action>TagSubjectAndBody</Action>
7 </ExecutorSetting>
```

[15] http://msdn.microsoft.com/en-us/library/aa289167.aspx

4.3 Analyse des Prototypen

Der entwickelte Prototyp von nAntiSpam wurde mittels Testsamples[16] vom SpamAssassin-Projekt[17] getestet. Im Folgenden werden die Ergebnisse dargestellt und analysiert.

Teststellung

Die experimentelle Umgebung bildete ein zur Zeit (Jahr 2008) üblicher Personal Computer mit einem Dual-Core Intel Pentium D Prozessor mit 3.2GHz Taktfrequenz. Der Hauptspeicher ist mit 2GB ausgelegt. Ein RAID0[18] Verbund von zwei 300GB Serial-ATA Festplatten diente als persistentes Speichermedium.
Die Leistung des Computers wurde mittels einer Java-Implementierung[19] vom *LINPACK-Benchmark*[20] mit 2.14 GFLOPS[21] ermittelt.
Der Prototyp selbst ist als ausführbares Kompilat mit normaler Prozessorpriorität getestet worden.

Preprocessing

Die Analyse der Mail-Samples zeigt, dass eine englische Nachricht aus durchschnittlich 290 Wörtern besteht. Wie schon der Artikel *A neural network based approach to automated e-mail classification* ([CKP03] Clark et al. 2003, S. 73-76) beschreibt, verringert der Preprozessor die Anzahl der Wörter mit den Mechanismen des Stemmming, der Entfernung von Stopp-Wörtern und der Rauschunterdrückung (siehe Kapitel 4.2.2).

Tabelle 4.4 und Abbildung 4.8 zeigen die durchschnittliche Wortanzahl der einzelnen Schritte des Preprozessors, gemessen an 1000 Mails. Durch dieses Prozedere kann die Wortmenge - und damit der Merkmalsvektor - auf 28% der ursprünglichen Größe reduziert werden.

[16] Downloads: http://spamassassin.apache.org/publiccorpus
[17] Weitere Informationen http://spamassassin.apache.org
[18] Weitere Informationen: http://www.tecchannel.de/storage/extra/401665/raid_sicherheit_level_server_storage_performance_festplatten_controller/index.html
[19] Weitere Informationen: http://www.netlib.org/benchmark/linpackjava
[20] Weitere Informationen: http://www.netlib.org/linpack
[21] Anm. d. Verf.: Gleitkommaoperationen pro Sekunde, Abk.: FLOPS

Um den Merkmalsvektor klein zu halten und damit die Effektivität des künstlichen neuronalen Netzes zu erhöhen, ist dieser Prozess für das Verfahren sehr wichtig. Hier kann durch Veränderung des Wertes der Rauschunterdrückung und durch Erweiterung der Stopp-Wörter der Merkmalsvektor in seiner Größe optimiert werden.

Gesamt-Wörter	nach Stemming	nach Entfernung der Stoppwörter	nach Rauschunterdrückung
290	121	84	81

Tabelle 4.4: Analyse Preprozessor

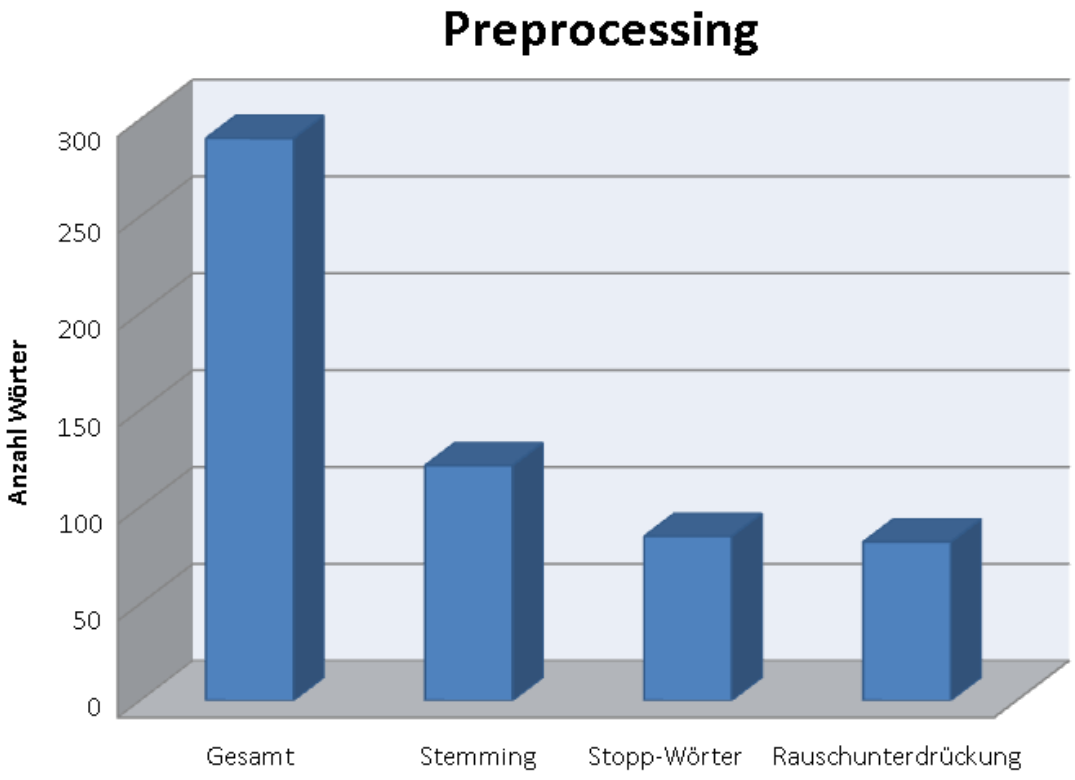


Abbildung 4.8: Analyse Preprocessing

Training

Das Trainingsset beinhaltet Daten von Nachrichten, die bereits durch den Preprozessor aufbereitet wurden (Stemming, Entfernen von Stopp-Wörtern und 30% Rauschunterdrückung). Die Trainingsmengen der Experimente umfassen einen Bereich von 289 bis 2080 Units (Neuronen). Das Training des künstlichen neuronalen Netzes erfolgte mit folgenden Einstellungen:

Eigenschaft	Wert
debug	true
learningrate	0.3
hiddenlayer_size	0.08
sigmoidalphavalue	0.1
inputset_treshold	3
training_end	0.3

Tabelle 4.5: Experimentelle Einstellungen des KNN

Tabelle 4.6 beinhaltet die ermittelten Daten der Experimente vom Preprozessor und dem künstlichen neuronalen Netz:

Wörter gesamt	Input Units	Hidden Units	KNN Grösse	Lernschritte	Zeit
1583	266	21	46KB	2880	0:00:26
2973	753	60	359KB	2665	0:04:28
4204	1038	83	681KB	2465	0:06:47
4974	1513	121	1442KB	2605	0:25:34
5466	1712	137	1846KB	3185	0:54:00
6063	1924	154	2330KB	4460	1:51:38

Tabelle 4.6: Ergebnisse der Experimente im Überblick

Die Lerndauer (siehe Abbildung 4.9) erhöht sich signifikant bei steigernder Anzahl an Units. Ein neuronales Netz mit 2000 Units benötigt bereits knapp zwei Stunden zum Erlernen der Trainingsmuster. Die Trainigszeit ist demnach ein diskutabler Aspekt (siehe Kapitel 5.2), da im Kontext der Mail-Klassifizierung das neuronale Netz auf viele tausende Units anwachsen kann.

Die Größe des serialisierten KNNs (siehe Abbildung 4.10) verhält sich hingegen proportional zur Anzahl der Units. Ein neuronales Netz mit 2000 Units benötigt etwa 2.3MB Speicherplatz und findet somit problemlos auf gängigen Speicherme-

dien Platz. Auch bei einer angenommenen Netzgröße von 50000 Units wäre die Dateigröße von etwa 60MB akzeptabel. Anzumerken ist, dass der Speicherbedarf des serialisierten Netzes von der Implementierung des neuronalen Netzes abhängig ist. Die Experimente beziehen sich auf das in nAntiSpam verwendete KNN-Framework von *AForge* (siehe Kapitel 4.2).

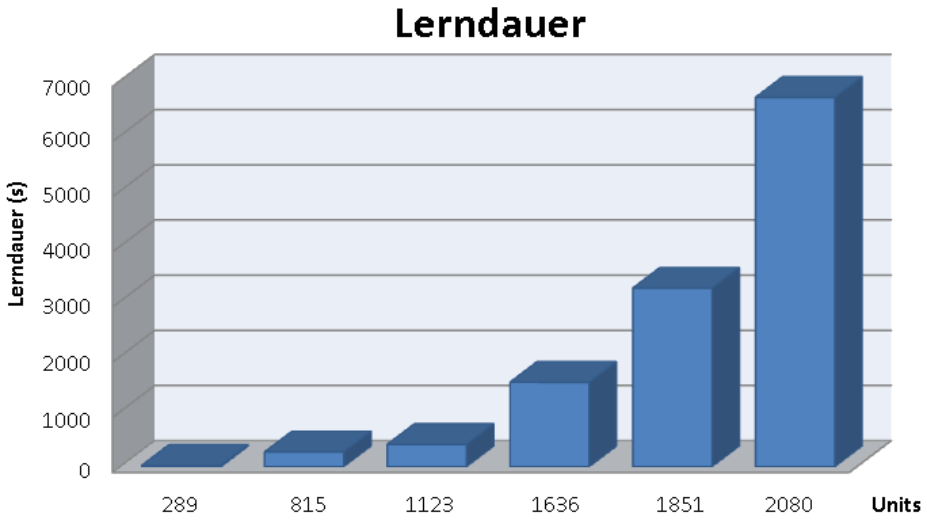


Abbildung 4.9: Analyse Lerndauer

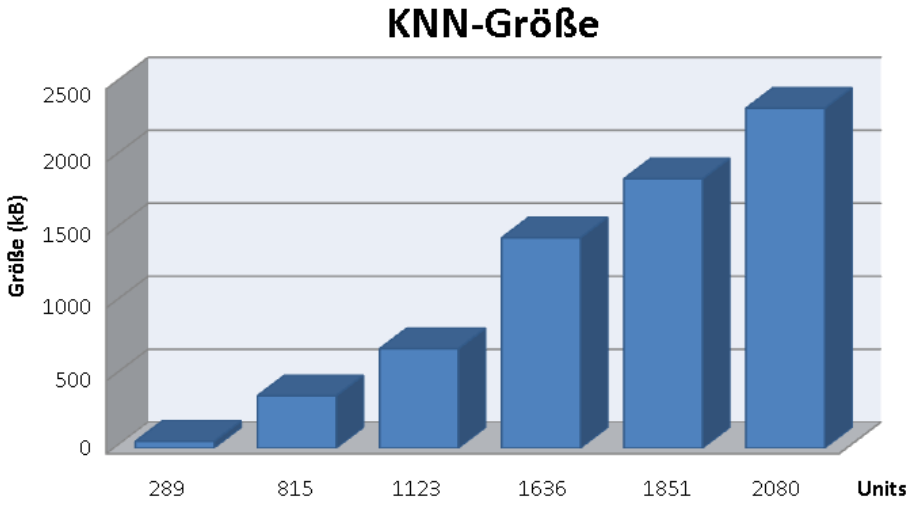


Abbildung 4.10: Analyse KNN-Größe

Die Experimente zeigten folgende Besonderheiten: Beinhaltet die Trainingsmenge alle Wörter - auch jene, die nur ein einziges Mal vorkommen - dann ist es dem Netz bei einer größeren Anzahl an Units nicht mehr möglich zu lernen. Der Merkmalsvektor ist zu breit - das Netz kann keine eindeutigen Muster identifizieren und der Netz-Fehler kann nicht durch das Backpropagation-Training reduziert werden. nAntiSpam orientiert sich daher am Wert *inputset_ treshold* der Datei *setting.xml*, der das minimale Vorhandensein eines Worts definiert.

Herausfordernd gestaltete sich die Ermittelung der Größe des Hidden-Layers. Die Anzahl der Units im Hidden-Layer wirken sich unmittelbar auf die Trainingsgeschwindigkeit aus. Zu viele Hidden-Units resultieren in zu langen Trainingszeiten und führen dazu, dass das Wissen im Hidden-Layer „zerstreut“ wird und eine Verringerung des Netz-Fehlers auch durch wiederholtes Lernen nicht möglich ist.
Ein zu kleiner Hidden-Layer kann die Fülle an Informationen nicht speichern. Ein Lernen ist ebenfalls nicht möglich. Es wurde experimentell eine optimale Hidden-Layer Größe von 5%-10% der Anzahl der Input-Units ermittelt.

Ebenfalls auf die Trainingsgeschwindigkeit wirken sich die Werte *learningrate* und *sigmoidalphavalue* aus. Eine geringe *learningrate* führt zu längeren Trainingszeiten, während ein höherer Wert die Geschwindigkeit steigert, aber der Netzfehler nicht weiter verringert werden kann. Für die Experimente wurde eine Einstellung von 0.3 gewählt. Der Wert *sigmoidalphavalue* gibt die Schrittweite der sigmoiden Funktion an. Ein Wert zwischen 0.05 und 0.1 zeigte sich für nAntiSpam als sinnvolle Einstellung.

Die Lernrate der einzelnen Experimente ist in Abbildung 4.11 visualisiert. Der beginnende Fehlerwert erhöht sich verständlicherweise mit der Anzahl der Units. Mit jedem erneuten Durchlauf des Backpropagation-Trainings-Prozedere verringert sich der Netz-Fehler bis der Wert *training_ end* (in der Datei *setting.xml*) erreicht ist.

Bis zu etwa 800 Lernzyklen wirken sich die Gewichtsveränderungen der Verbindungen zwischen den Units nur marginal auf den Netz-Fehler aus. Sobald die Gewichtungen den Aktivitätslevel auf die steile Flanke der sigmoiden Aktivitätsfunktion verschieben, beginnt das neuronale Netz massiv zu lernen. Das Netz kann die Eingangsreize bereits dem korrekten Output zuordnen und abstrahiert daher sukzessiv weniger.

Ab etwa 1200 Lernzyklen wirken sich die Gewichtsänderungen nicht mehr gravierend aus - der Netz-Fehler nähert sich asymptotisch 0. Ab diesem Bereich hat das neuronale Netz die Muster erlernt. Durch Veränderung des Wertes *learningrate* und *training_ end* kann der Netz-Fehler weiter minimiert werden. Zu Beachten ist, dass sich der Wert *learningrate* unmittelbar auf die Geschwindigkeit des neuroanlen Netzes auswirkt, da ein einzelner Trainingszyklus erst dann abgeschlossen wird, wenn dieser Wert erreicht wurde.

Das Netz ist zu Beginn mit zufälligen Gewichtungen initialisiert. Dies führt dazu, dass die selbe Trainingsmenge unterschiedlich lange zum Erreichen des Wertes *training_ end* benötigen kann. Dies ist im Diagramm an den verschiedenen Punkten zu erkennen, bei denen das „massive“ Lernen beginnt.

Zusammenfassend kann festgehalten werden, dass ab etwa 2000 Lernzyklen nAntiSpam die Trainingsdaten ausreichend erlernt hat, um eine Klassifizierung von Mails durchzuführen.

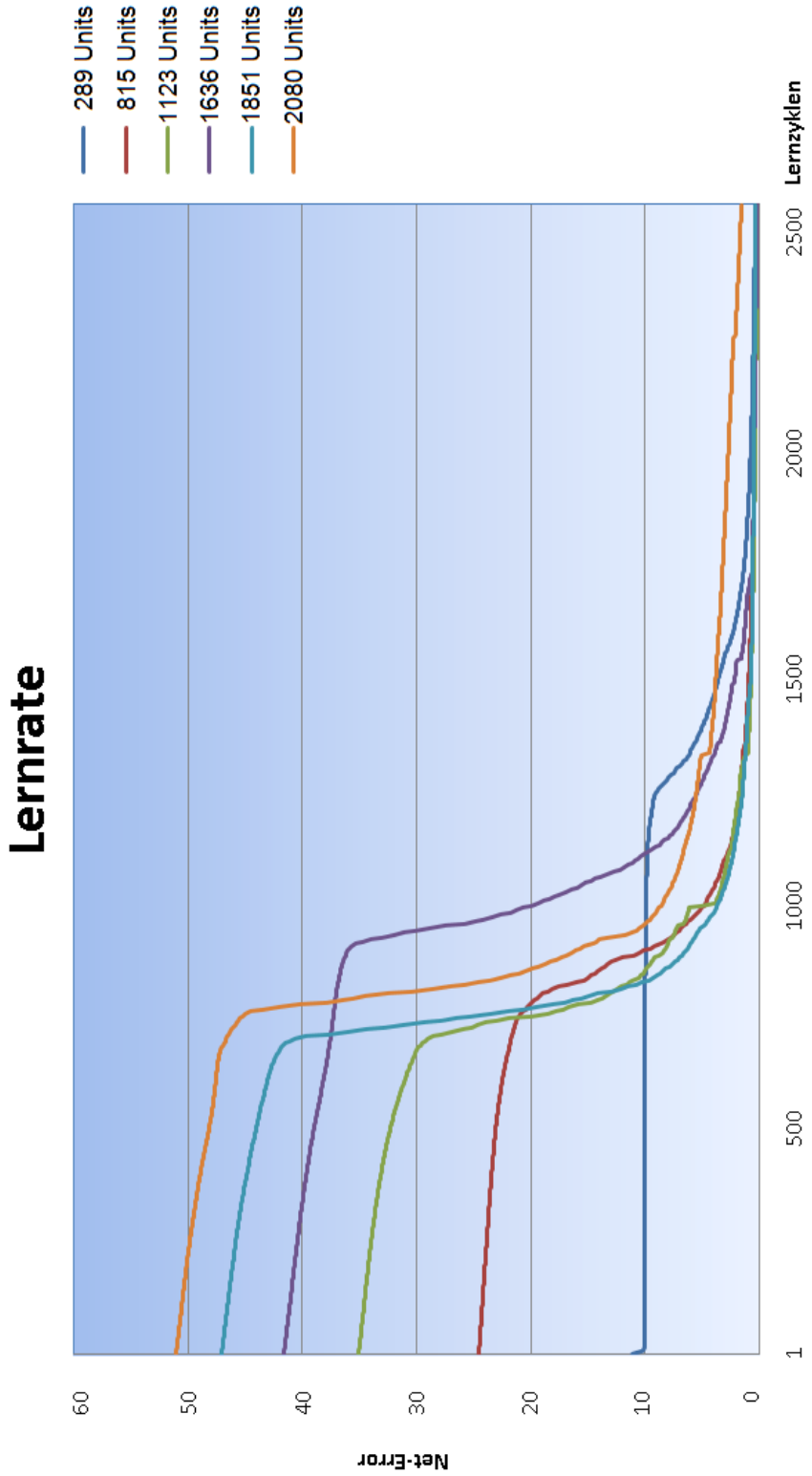


Abbildung 4.11: Analyse Lernrate

Klassifikation

Der Preprozessor verarbeitet die eingehenden Mails in durchschnittlich 30 Millisekunden. Die Zeit zur Berechnung des Outputs einer an den Input-Units anliegenden Nachricht beträgt bei 2000 Units nur drei Millisekunden.
Diese geringe Zeitspanne von unter 40 Millisekunden für die gesamte Klassifizierung ermöglicht demnach einen optimalen Einsatz von nAntiSpam als Add-In für Mailclients sowie auch als STMP-Relay. Auch bei größeren Netzen steigt die Dauer für die Berechnung des Outputs nicht wesentlich an. Eine Klassifizierung in „Echtzeit“ ist problemlos möglich.

Die Experimente zeigten jedoch eine klare Restriktion für den Einsatz von nAntiSpam. Das Ergebnis der Klassifikation spiegelt sich in den beiden Output-Units wider. Eine Unit repräsentiert die Ham-, die andere die Spam-Wahrscheinlichkeit. Bei kleinen Netzen mit weniger als 500 Input-Neuronen kann noch relativ eindeutig entschieden werden, ob eine Nachricht als Ham oder als Spam zu klassifizieren ist. Bei Experimenten mit Nachrichten aus der Lernmenge errechnet das Netz für die Spam-Unit einen Wert von 0.89, sowie 0.11 für die Ham-Unit. nAntiSpam kann mit hoher Wahrscheinlichkeit feststellen, dass es sich um eine Spam-Mail handelt.

Bei einer größeren Zahl an Input-Units (>500) hingegen verbreitert sich der Merkmalsvektor zu sehr. Das neuronale Netz kann nur mehr bedingt eindeutige Muster erkennen - das Wissen „verwässert“ zunehmends.
Die Experimente zeigten, dass eine Spam-Mail aus dem Trainingssample den Wert 0.41 für die Ham-Unit, und 0.58 für die Spam-Unit errechnete. Es wird nAntiSpam demnach mit steigender Anzahl an Daten in der Datei *learningset.dat* immer schwieriger, eine konkrete Aussage über die Klasse der Mail zu tätigen. Die Zahlen der Arbeit „A LVQ-based neural network anti-spam email approach“ ([CXMX05] Chuan et al. 25.04.2008) können nur im experimentellen Umfeld von wenigen hundert Input-Units bestätigt werden.
Mögliche Optimierungen und Verbesserung des Verfahrens werden im Kapitel 5.2 illustriert.

Kapitel 5

Zusammenfassung

5.1 Zusammenfassung der Arbeit

Die Informationsübertragung mittels E-Mail ist heutzutage ein fundamentaler Bestandteil der Geschäftswelt sowie der privaten Kommunikation über das Internet. 80-85% der Mails (entspricht etwa 100 Milliarden Mails) werden unerwünscht als Spam zugestellt und verursachen einen enormen wirtschaftlichen Schaden. Für ein kleines Unternehmen mit fünf Mitarbeitern belaufen sich die Kosten pro Spam-Mail auf bis zu 0,70 €-Cent. Aus rechtlicher Sicht ist das Versenden von Spam lt. Telekommunikationsrecht § 107 in Österreich nicht zulässig.

Die Spam-Mails werden mit Black-, und Whitelists, Header-Überprüfungen und Contextfilter bekämpft. Bei den weit verbreiteten Contenfilter wird der Inhalt der Nachricht bestimmten Verfahren unterzogen, die eine Spam-Identifizierung ermöglichen. Textklassifierungsmethoden wie *Decision Trees*, *k-Nearest-Neighbour*, *Rule Induction* und der sehr bekannte *Naive Bayes*-Algorithums sind größtenteils statische Filter. Sie versuchen Wörter zu gruppieren oder mittels Regelsätzen eine Klassifikation durchzuführen. Aufgrund ihrer Konzepte sind sie nicht in der Lage, die Beziehung der Wörter zueinander zu analysieren und können nur bedingt neue Daten „lernen“.

Forschungen der Neuroinformatik befassen sich mit der Simulation neuronaler Netze und eröffnen neue Möglichkeiten einer intelligenten, lernenden Text- Klassifizierung. Es wird dabei ein Abbild eines biologischen neuronalen Netzes (wie das eines menschlichen Gehirns) mittels mathematischer Modelle als künstliches

neuronales Netz modelliert. Dem künstlichen neuronalen Netz ist es möglich, das Wissen mittels Trainingsdaten zu erlernen und anzuwenden. Das Netz kann sich durch neue Eingangsmuster mit wiederholtem Training an veränderte Bedingungen anpassen.

Für die Modellierung künstlicher neuronaler Netze ermöglichen *feedworward* und *feeback* Netztopologien, sowie verschiedene Aktivitätsfunktionen der Neuronen (sigmoide Funktion, lineare Funktion, etc.) eine Fülle von Einsatzszenarien. Das Training kann *unsupervised* oder *supervised* - mittels der Hebb-Regel, Delta-Regel oder Backpropagation erfolgen.

Die Experimente mit dem entwickelten Verfahren dieser Diplomarbeit erfolgten mit einem feedforward multilayer Perzeptron (MLP) des *AForge*-KNN-Framework. Die Schichten-Architektur des nAntiSpam-Prototypen ermöglicht den Einsatz als SMTP-Relay und als Mail-Client-AddIn. Um ein performantes Training und eine schnelle Klassifizierung sicherzustellen, bedarf es vorbereitender Schritte, bevor die Mail dem künstlichen neuronalen Netz übergeben wird. Zu Beginn werden überflüssige Sonder- und Trennzeichen entfernt, sowie aussagekräftige Zeichen (!?$) gespeichert. Der *Preprozessor* erstellt aus der resultierenden Wortliste eine *bag of words*-Darstellung. Jedes einzelne Wort dieses Merkmalsvektors wird mittels Stemming auf den Wortstamm zurückgeführt (control, controls, controlled, controller, controlling haben alle den Stamm control). Im nächsten Schritt errechnet die Software den mittleren Informationsgehalt (Entropie) jedes Wortes und führt eine Rauschunterdrückung durch. Das nun vorliegende *bag of word* wird dem künstlichen neuronalen Netz - das zuvor mit Beispielsmustern trainiert wurde - zur Klassifizierung vorgelegt. Das Ergebnis repräsentieren zwei Ausgangsneuronen mit der errechneten Ham- und Spam-Wahrscheinlichkeit.

Die Analyse des Prototypen erfolgte mit Testsamples des *Spam-Assassin*-Projekts. Es stellte sich heraus, dass eine englische Nachricht aus durchschnittlich 290 Wörtern besteht. Die einzelnen Schritte des Preprozessors reduzieren diese Wortanzahl auf 28% der ursprünglichen Größe. Die Anzahl der Wörter wird durch das Stemming von 290 auf 121 reduziert; die Entfernung der Stopp-Wörter verringert

die Anzahl auf 84 und die Rauschunterdrückung auf 81 Wörter. Die Durchführungszeit des Preprozessors liegt bei durchschnittlich 30 Millisekunden. Um den Merkmalsvektor klein zu halten und damit die Effektivität des künstlichen neuronalen Netzes zu erhöhen, ist dieser Prozess für das Verfahren sehr wichtig.

Bei den Trainingsphasen zeigten die Experimente, dass sich die Lerndauer signifikant mit steigender Anzahl an Units erhöht. So benötigt ein Netz mit 300 Units 26 Sekunden um die Trainingsdaten zu erlernen. Besteht das künstliche neuronale Netz hingegen aus 2000 Units, erhöht sich die Lerndauer auf bereits 2 Stunden.
Die Dateigröße des serialisierten neuronalen Netzes verhält sich etwa proportional zur Anzahl der Units und ist mit 2.3MB für 2000 Units problemlos verwaltbar.
Herausfordernd gestaltete sich die Ermittelung der Größe des Hidden-Layers. Die Anzahl der Units im Hidden-Layer wirken sich unmittelbar auf die Trainingsgeschwindigkeit aus. Zu viele Hidden-Units resultieren in zu langen Trainingszeiten und führen dazu, dass das Wissen im Hidden-Layer zerstreut wird und eine Verringerung des Netz-Fehlers auch durch wiederholtes Lernen nicht mehr möglich ist. Ein zu kleiner Hidden-Layer kann die Fülle an Informationen nicht speichern. Für nAntiSpam wurde experimentell eine optimale Hidden-Layer Größe von 5%-10% der Anzahl der Input-Units ermittelt.
Zu erkennen ist, dass ab etwa 800 Lernzyklen das neuronale Netz beginnt, den korrekten Output zu den Input-Mustern zu errechnen - es abstrahiert daher sukzessiv weniger.
Für die Klassifikation benötigt das neuronale Netz im Durchschnitt nur drei Millisekunden was einen „Echtzeitbetrieb“ von nAntiSpam als Mail-Client-AddIn oder als SMTP-Relay durchaus zulässt. Die Experimente zeigten jedoch eine klare Restriktion für den Einsatz von nAntiSpam: Bei einer größeren Zahl an Input-Units ($>$500) verbreitert sich der Merkmalsvektor zu sehr. Das neuronale Netz kann nur mehr bedingt eindeutige Muster erkennen - das Wissen „verwässert“ zunehmends. So kann bei einer Anzahl von 200 Input-Unit eine Spam-Mail eindeutig erkannt werden (Spam-Wert: 0.89, Ham-Wert: 0.11), während bei 2000 Input-Units eine genaue Klassifizierung nicht mehr eindeutig durchführbar ist (Spam-Wert: 0.58, Ham-Wer: 0.41).

Zusammenfassend kann festgehalten werden, dass es mit moderatem Mitteleinsatz durchaus möglich ist, eine Implementierung eines Spam-Filters mit einem

künstlichen neuronalen Netz zu entwickeln. Das in dieser Diplomarbeit erarbeitete Verfahren ist modular und parametrisierbar aufgebaut und lässt sich für Optimierungen erweitern und anpassen. Die Zahlen der Arbeit „A LVQ-based neural network anti-spam email approach“ ([CXMX05] Chuan et al. 25.04.2008) können jedoch nur im experimentellen Umfeld von wenigen hundert Input-Units bestätigt werden. Für den Produktiveinsatz von nAntiSpam bedarf es weiterer Optimierungen (siehe im folgenden Kaptiel 5.2).

5.2 Optimierung des Verfahrens

Um einen Produktiveinsatz von nAntiSpam zu ermöglichen, sind weitere Optimierungen der Software nötig. Das Schichtenmodell, sowie das Klassendiagramm können weiterhin verwendet werden.

Optimierung des Preprozessors

Um den Merkmalsvektor klein zu halten, muss der Preprozessor die Wortmenge verringern. Dies kann einerseits durch Erweiterung der Stopp-Wörter-Liste, andererseits durch Veränderung des Wertes der Rauschunterdrückung ermöglicht werden.
Ebenso ist denkbar einen Algorithmus zu implementieren, der Namen von Personen ausfiltert, da diese keine unmittelbare Aussagekraft besitzen. Weiters gilt es ein Procedere zu entwerfen, das die mit Absicht verstümmelten Wörter (z.B.: vi@ggr@) über intelligente Verfahren zum ursprünglichen Wort überführt.

Optimierung des künstlichen neuronalen Netzes

Der mit zunehmender Anzahl an Units immer größer werdenden Lerndauer kann mit einem relativ einfachen Szenario entgegengewirkt werden: Während die Software ein aktuelles neuronales Netz für die Mail-Klassifizierung im Speicher hält, kann ein weiterer Thread parallel im Hintergrund das Lernen mit einem temporären Netz durchführen. Dies erfordert zwar den doppelten Speicher, ist aber aufgrund des moderaten Speicherverbrauchs durchaus praktikabel. Ist das Trainingsende erreicht, wird dieses neu trainierte Netz geladen.

Etwas umfangreicher gestaltet sich die Lösung des Problems des zu breiten Merkmalsvektors:
Das künstliche neuronale Netz klassifiziert eindeutig bis etwa 500 Input-Neuronen. Die mögliche Lösung wäre eine Vor-Klassifizierung in Kategorien (z.B: fraud, medicines, etc). Jede Kategorie besteht somit aus reduzierten Units, die von einem eigenen künstlichen neuronalen Netz klassifiziert werden (siehe Abbildung 5.1).
Der Pre-Classifier kann wiederrum aus einem neuronalen Netz aufgebaut werden - erfordert dann aber eine spezielle Filterung der Daten, um nur die für die Vorklassifizierung relevanten Inhalte auszufiltern. (Die Implementierung eines Descision-Trees zur Vorklassifizierung wäre ebenfalls eine denkbare Möglichkeit.)
Findet der Pre-Classifier keine passende Kategorie, kann automatisch ein neues neuronales Netz für dieses Thema erzeugt werden. Die grundlegende Implementierung von nAntiSpam kann durch dieses Optimierungsverfahren dennoch weiterverwendet werden.

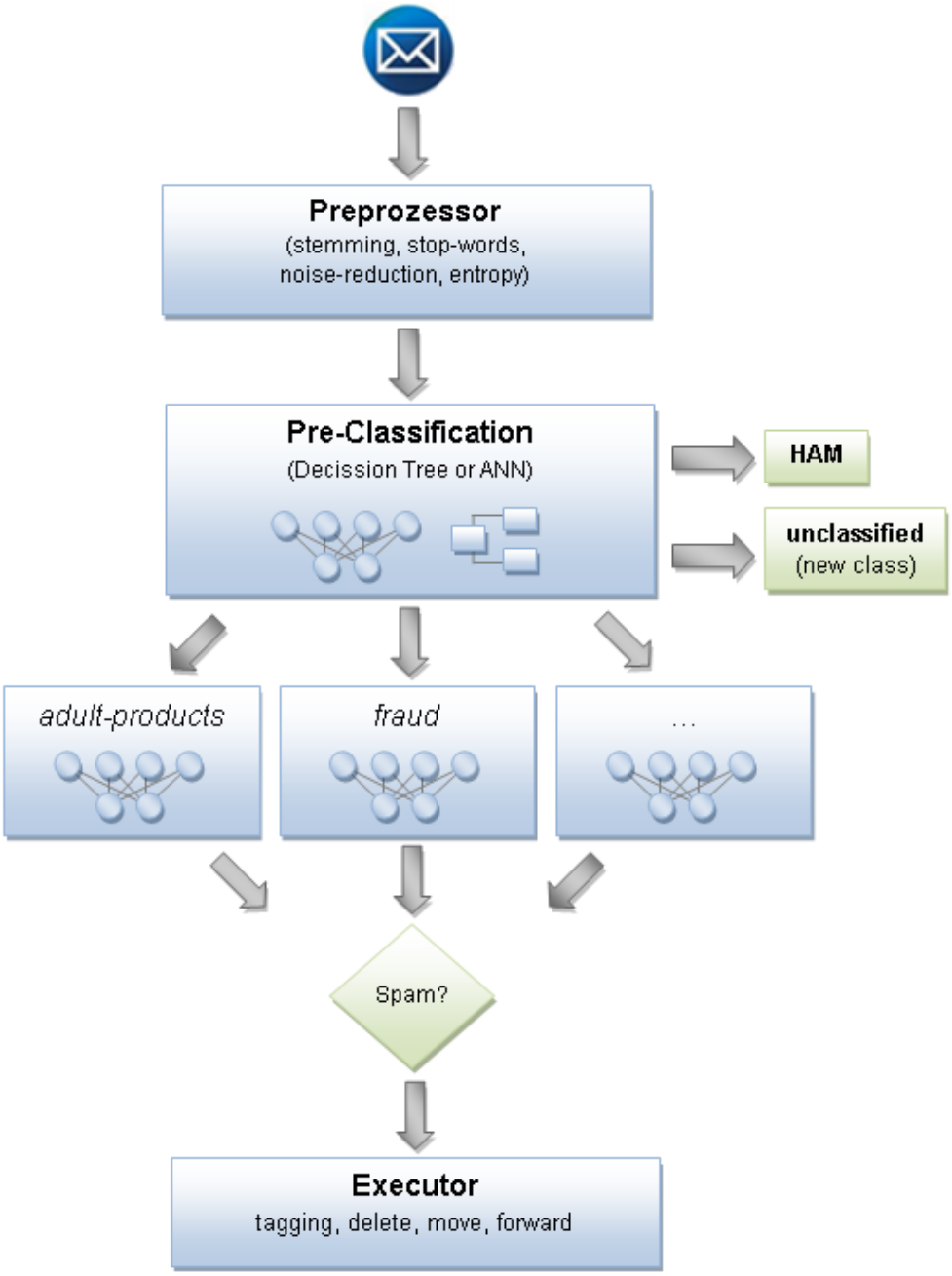


Abbildung 5.1: nAntiSpam Optimierung

5.3 Persönliche Meinung

Der Forschungsgegenstand der künstlichen neuronalen Netze fasziniert durch die Fülle an möglichen Einsatzszenarien. Abstrahiert man die zugrundeliegenden mathematischen Zusammenhänge, erhält man mit einem aktuellen KNN-Framework ein universell nutzbares Werkzeug für Problemlösungen in der Informatik.

Für eine Mail-Klassifierung mit einem künstlichen neuronalen Netz bedarf es durchdachter Vorbereitungen um die Datenmenge effizient aufzubereiten. Diesem Teil der Software muss mindestens ebensoviel Zeit gewidmet werden wie der Justage des neuronalen Netzes. Die Implementierung selbst ist mit wenigen tausend *lines of codes* vergleichbar einfach durchzuführen.

Nüchtern betrachtet wird schnell erkannt, dass das neuronale Netz mit der derzeit verfügbaren Hardware ab einer gewissen Anzahl an Neuronen zu lange für die Trainingsphasen benötigt. Auch das „Verwässern“ des Wissens bei zu breitem Merkmalsvektor trübt die ansonsten guten Ergebnisse der Experimente. Diese Arbeit zeigt auf, dass eine Implementierung eines KNN-basierenden Spam-Mail-Filters mit moderatem Mitteleinsatz möglich ist, aber dennoch eine Optimierung des Verfahrens nötig macht.

Grundlegend wäre die beste Strategie zur Spam-Bekämpfung, einen unerwünschten Datenverkehr erst gar nicht zustande kommen zu lassen. Es gleicht dem Kampf gegen Windmühlen, am Ende der Nachrichtenkette die Krankheit heilen zu wollen; sinnvoller erscheint, die Erreger der Krankheit erst gar nicht entstehen zu lassen.

Literaturverzeichnis

[Aki08] Akismet. *Akismet Stats Page.* http://akismet.com/stats/, 2008. 03.08.2008.

[And95] James A. Anderson. *Introduction to Neural Networks.* MIT Press, 1995.

[AS08] ct Andreas Stiller. *Spam feiert 30. Geburtstag.* http://www.heise.de/newsticker/Spam-feiert-30-Geburtstag--/meldung/107347, 2008. 03.08.2008.

[Bay63] Thomas Bayes. *An Essay towards solving a Problem in the Dcotrine of Chances. In: Philosophical Transactions of the Royal Society of London 53, London, 1763, S. 370-418.* 1763.

[Bla02] Kai Blankenhorn. *Spam-Filterung mittels Maschinellem Lernen.* http://www.bitfolge.de/spamcleaner/Spam-Filterung%20mittels%20Maschinellem%20Lernen.pdf, 2002. 27.04.2008.

[Bus98] Frank Buschmann. *Pattern-orientierte Software-Architektur.* Addison-Wesley, 1998.

[CKP03] James Clark, Irena Koprinska, and Josiah Poon. A neural network based approach to automated e-mail classification. In *WI '03: Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence*, page 702, Washington, DC, USA, 2003. IEEE Computer Society.

[Coh95] William W. Cohen. *Fast Effective Rule Induction.* http://www2.cs.uh.edu/~vilalta/courses/advancedai/GudlaP.ppt, 1995. 06.08.2008.

[CXMX05] Zhan Chuan, Lu Xianliang, Hou Mengshu, and Zhou Xu. A lvq-based neural network anti-spam email approach. *SIGOPS Oper. Syst. Rev.*, 39(1):34–39, 2005.

[DCPP08] M. S. Seidenberg D. C. Plaut, J. L. McClelland and K. Patterson. *Understanding normal and impaired word reading.* http://www.cnbc.cmu.edu/~plaut/papers/pdf/PlautETAL96PsyRev.wordReading.pdf, 2008. 01.08.2008.

[Dir06] R. Dirmeikis. An artificial neural nets for spam e-mail recognition. *Electronics and Electrical Engineering*, 5(69):73–76, 2006.

[Egg07] Tobias Eggendorfer. *Methoden der Spamekämpfung und -vermeidung.* Books on Demand, Norderstedt, 2007.

[Fab08] F. Faber. *Unter Verdacht. Eine russische Bande professionalisiert das Cybercrime-Business. In: c't 11/2008.* Heise Verlag, 2008.

[Foo08] Hormel Foods. *Spam and the Internet.* http://www.spam.com/legal/spam, 2008. 31.07.2008.

[GDR08] Fabian Beck Günter Daniel Rey. *Neuronale Netze - Ein Einführung.* http://www.neuronalesnetz.de/downloads/neuronalesnetz_de.pdf, 2008. 09.07.2007.

[Heb02] Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Lawrence Erlbaum, 2002.

[HH05] Michael Klar Helmut Herold. *C++, UML und Design Patterns.* Addison-Wesley, 2005.

[HPL06] Dragana Damjanovic Hans Peter Lehofer, Michael Holoubek. *Grundzüge des Telekommunikationsrechts.* Springer, 2006.

[Koh72] T. Kohonen. Dcorrelation matrix memories. *IEEE Transactions on Computers*, C-21(4), 1972.

[Kri08] David Kriesel. *Neuronale Netze.* http://www.dkriesel.com/index.php?id=neuronalenetze, 2008. 19.08.2008.

[Lab08] Paul De Koninck Laboratory. *Hippocampal neuron expressing monomeric Green Fluorescent Protein (GFP).* http://www.greenspine.ca/en/mGFP_neuron2.html, 2008. 19.08.2008.

[Lan01] Carsten Lanquillon. Enhancing text classification to improve information filtering. Master's thesis, Otto-von-Guericke-Universität Magdeburg, 2001.

[Mit97] Thomas Mitchell. *Machine Learning.* Mcgraw-Hill Higher Education, 1997.

[Mit03] Kevin D. Mitnick. *Die Kunst der Täuschung.* mitp-Verlag, 2003.

[PJS08] Fachhochschule Regensburg Prof. Jürgen Sauer. *Neuronale Netze, Fuzzy Control-Systeme und Genetische Algorithmen.* http://fbim.fh-regensburg.de/~saj39122/nn/skript/NFG_Skript_WS0708.pdf, 2008. 25.04.2008.

[Por97] M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1997.

[Psc02] Willibald Pschyrembel. *Klinisches Wörterbuch.* Walter de Gruyter Verlag, 2002.

[Roj96] Raúl Rojas. *Neural Networks - A Systematic Introduction.* Springer, 1996.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[Sha63] Claude E. Shannon. *Mathematical Theory of Communication.* B&T, 1963.

[Tea08] AForge.NET Project Team. *AForge.NET.* http://code.google.com/p/aforge, 2008. 29.11.2008.

[TM05] et al. Thomas Müller, Hans Käser. *Technische Informatik 1: Grundlagen der Informatik und Assemblerprogrammierung.* vdf Hochschulverlag AG, 2005.

[Top05] Jochen Topf. *Antispam - Strategien, Unerwünschte E-Mails erkennen und abwehren.* http://www.bsi.bund.de/literat/studien/antispam/antispam.pdf, 2005. 03.08.2008.

[Tro04] Andrew W. Troelsen. *An Introduction to Programming Outlook 2003 Using C#.* http://msdn.microsoft.com/en-us/library/aa289167.aspx, 2004. 14.10.2008.

[UL04] Jürgen Cleve Uwe Lämmel. *Künstliche Intelligenz, Lehr- und Übungsbuch*. Carl Hanser Verlag, 2004.

[Ung03] Andreas Ungerer. *NiX Spam. In: iX 5/2003*. Heise Verlag, Hannover, Germany, 2003.

[Vil08] Mariana Ruiz Villarreal. *neuron cell diagram*. http://www.dentalarticles.com/visual/d/neuron-cell-diagram.php, 2008. 19.08.2008.

[Wie07] Bundeskriminalamt Wiesbaden. *Nigerianische Angebote*. http://bka.de/profil/faq/hinweise/angebot.html, 2007. 01.08.2008.

[WU08] Westfälische Wilhelms-Universität. *Einführung in Neuronale Netze - Geschichtlicher Überblick*. http://wwwmath.uni-muenster.de/SoftComputing/lehre/material/wwwnnscript/startseite.html, 2008. 18.08.2008.

Abbildungsverzeichnis

2.1 Spam-Statistik von Akismet 14
2.2 k-Nearest-Neighbour 18

3.1 McCulloch-Pitts-Neuron 24
3.2 Perzeptron 25
3.3 Biologisches Neuron 27
3.4 Hippocampal neuron expressing monomeric Green Fluorescent Protein (GFP) 29
3.5 Modell eines künstlichen neuronalen Netzes 30
3.6 Modell einer Unit 33
3.7 Netz-Input eines Perzeptrons 34
3.8 Lineare Aktivitätsfunktion 35
3.9 Lineare Aktivitätsfunktion mit Schwelle 36
3.10 Binäre Schwellenwertfunktion 36
3.11 Sigmoide Aktivitätsfunktion 37

4.1 nAntiSpam-Struktur 50
4.2 nAntiSpam als Add-In 51
4.3 nAntiSpam als SMTP-Relay 51
4.4 nAntiSpam Core (structural pattern) 55
4.5 nAntiSpam Bag of words (behavioural pattern) 56
4.6 nAntiSpam-Klassifikationsphase 62
4.7 nAntiSpam-Trainingsphase 64
4.8 Analyse Preprocessing 67
4.9 Analyse Lerndauer 69
4.10 Analyse KNN-Größe 69
4.11 Analyse Lernrate 72

5.1 nAntiSpam Optimierung 79

5.2 AForge.Neuro Klassendiagramm (Teil 1) 93
5.3 AForge.Neuro Klassendiagramm (Teil 2) 94

Tabellenverzeichnis

2.1 Merkmalsvektor eines *bag of words models* 17

3.1 Unterschied künstliches und biologisches neuronales Netz 30
3.2 Vergleich biologisches Neuron und künstliche Unit 32

4.1 nAntiSpam-Systemkomponenten 49
4.2 nAntiSpam-setting.xml 63
4.3 nAntiSpam-ExecutorAction 65
4.4 Analyse Preprozessor 67
4.5 Experimentelle Einstellungen des KNN 68
4.6 Ergebnisse der Experimente im Überblick 68

Stichwortverzeichnis

Add-In, 48, 51
Aktivitätsfunktion, 32, 35
Aktivitätslevel, 33
Aktivitätsmuster, 31
Architekturmuster, 53
Axon, 27

Backpropagation, 26, 43, 46
Bag of words, 16, 46
Bayes-Theorem, 18
Behaviourial Pattern, 53
Binäre Schwellenwertfunktion, 36
Biologische Analogie, 27
Blacklist, 15
Bot-Netze, 15

C#, 52, 57, 60
COM, 57
Contentfilter, 16
Core-Layer, 48, 59

Decision Trees, 17
Delta-Regel, 42
Dendriten, 27
Design-Pattern, 53
DNS-Server, 11

Entropie, 48, 61
Executor, 48, 65

Feedback Netze, 39
Feedforward, 46
Feedforward Netze, 38

Gehirn, 21, 22
Geschichte, 23

Hebb'sche Lernregel, 24, 42
Hidden-Layer, 25, 32, 38, 46
Historie, 23
Hoax, 10
Hormel Food, 10

IMAP, 12
Information Gain, 61

k-Nearest-Neighbour, 17
Künstliche Intelligenz, 23
Künstliche neuronale Netze, 19, 21, 46, 62
KI, 23
Klassifizierung, 62
Kohonen-Netz, 26, 39
Kompetetive Netze, 38

Lern-Algorithmus, 22
Lernregel, 41
Lineare Aktivitätsfunktion, 35

MailConnector, 48, 57
Mark I perceptron, 24
Marshal-Klasse, 57

McCulloch-Pitts-Neuron, 23
Merkmalsvektor, 16, 46
MLP, 46
Mutal Information, 61

Naive Bayes, 18
nAntiSpam, 48, 52, 61
Netztypen, 38
Neuroinformatik, 23
Neuron, 22, 23, 27
Neurotransmitter, 28
Normalisierung, 61

Pattern-Associator, 38
Perzeptron, 25, 46
Phishing, 10, 11
POP3, 12
Porter-Algorithmus, 59

Rauschunterdrückung, 61
Rule Induction, 18

Scam, 11
Sigmoide Funktion, 37
SMTP-after-POP3, 12
SMTP-AUTH, 12
SMTP-Relay, 48, 51
SMTP-Server, 11
Snarc, 24
SOM, 39
Soma, 27
Stemming, 47, 59
Structural Pattern, 53
Summenfunktion, 32
supervised, 22, 41
Synapsen, 23, 27

Telekommunikationsrecht, 12
Template Method, 53
Text-Klassifizierung, 16

Unsolicited Commercial E-Mail, 11
unsupervised, 22, 45

Whitelist, 15

Glossar

Aktivitätsfunktionen	Stellt den Zusammenhang zwischen Netz-Input und dem Aktivitätslevel einer Unit in einem zweidimensionalen Diagramm dar, 35
Aktivitätslevel	Ausgangswert eines Neurons, 33
API	Application Programming Interface; Schnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird, 48
Backpropagation	Netztyp mit Hidden-Layer; die Gewichtsänderung wird mittels Fehlersignal schichtweise, beginnend mit den Verbindungen zur Ausgabeschicht rückwärts in Richtung Eingabeschicht, vorgenommen, 43
Bag of words	Speicherung von Wörtern und deren Häufigkeit als Merkmalsvektor, 16
Blacklist	Liste mit unerwünschten Adressen, 15
Decision Trees	Eine Lernmenge formt den Entscheidungsbaum mit vielen Knoten wobei die Information immer weiter zu den Kindsknoten vererbt werden, 17
DNS	Domain Name System: Dient zur Namensauflösung im Internet, 11
Hidden-Unit	Befinden sich zwischen Input- und Output-Unit und beinhalten eine interne Repräsentation der Außenwelt, 32

Hoax	Kettenbriefe mit dem Ziel, sich sinnlos weiter zu vermehren, 10
IMAP	Internet Message Access Protocol: Erlaubt den Zugriff und das Verwalten von entfernten E-Mails, 11
Input-Unit	Empfängt Signale von der Außenwelt, 32
k-Nearest-Neighbour	Speichert Merkmalsvektoren in einem Raum mit n Dimensionen, 17
KNN	Künstliches neuronales Netz: Mathematische Abbildung eines Neuronenverbundes, 21
Kohonennetz	Netztyp mit unbeaufsichtigtem Lernen für Inverser Kinematik und dem Traveling Salesman Problem, 39
MLP	Multilayer Perzeptron; Vorwärtsgerichtetes, künstliches neuronales Netz mit mehreren Schichten, 46
Naive Bayes	Vom Bayestheorem hergeleiteter Klassifikator. Er ordnet jedes Objekt der Klasse zu, zu der es mit der größten Wahrscheinlichkeit gehört, 18
Neuron	Nervenzelle; genetische, morpol., funkt., trophische u. regenerative Einheit, 27
Output-Unit	Gibt Signale an die Außenwelt ab, 32
Pattern Associator	Feedforward Netz ohne Hidden-Layer zur Mustererkennung, 38
Perzeptron	Ein Perzeptron besteht aus einer Input-, ggf. einer Hidden-, und einer Output-Schicht. Dieses Netzwerk erlaubt es Muster zu klassifizieren, 24
Phishing	Abkürzung für „password fishing“. Mit dieser Technik wird versucht, dem Benutzer vertrauliche Daten zu entlocken, 11

POP3	Post Office Protocol: Dient zum Abholen von E-Mails von einem Server, 11
Rule Induction	Der Regellerner generiert aus einer Lernmenge einfache, lesbare Regeln, 18
SCAM	Durch den Vorschussbetrug werden die Empfänger unter Vorspiegelung falscher Tatsachen dazu gebracht, an einem Schneeballsystem teilzunehmen, 11
SMTP	Simple Mail Transfer Protocol: Zum Austausch von E-Mails in Computernetzen, 11
SOM	Self-Organizing Maps; siehe Kohonennetz, 39
SPAM	Unerwünschte Mail; zusätzlich Abkürzung für „Spiced Pork and Ham“, 10
Stemming	Rückführung eines Wortes auf dessen Wortstamm, 59
Supervised learning	Setzt eine existierende Trainingsmenge voraus, die aus bekannten Eingabemustern und den entsprechenden Ausgabewerten besteht, 41
Unit (KNN)	Künstliche Neuronen; orientieren sich an ihren biologischen Vorbildern, 32
Unsupervised learning	Unüberwachtes Lernen; nur Eingabemuster sind gegeben; das Netz versucht Muster zu identifizieren und ähnliche Muster in passenden Kategorien zu klassifizieren, 44
Whitelist	Liste mit vertrauenswürdigen Adressen, 15

Anhang: AForge.Neuro Klassendiagramm

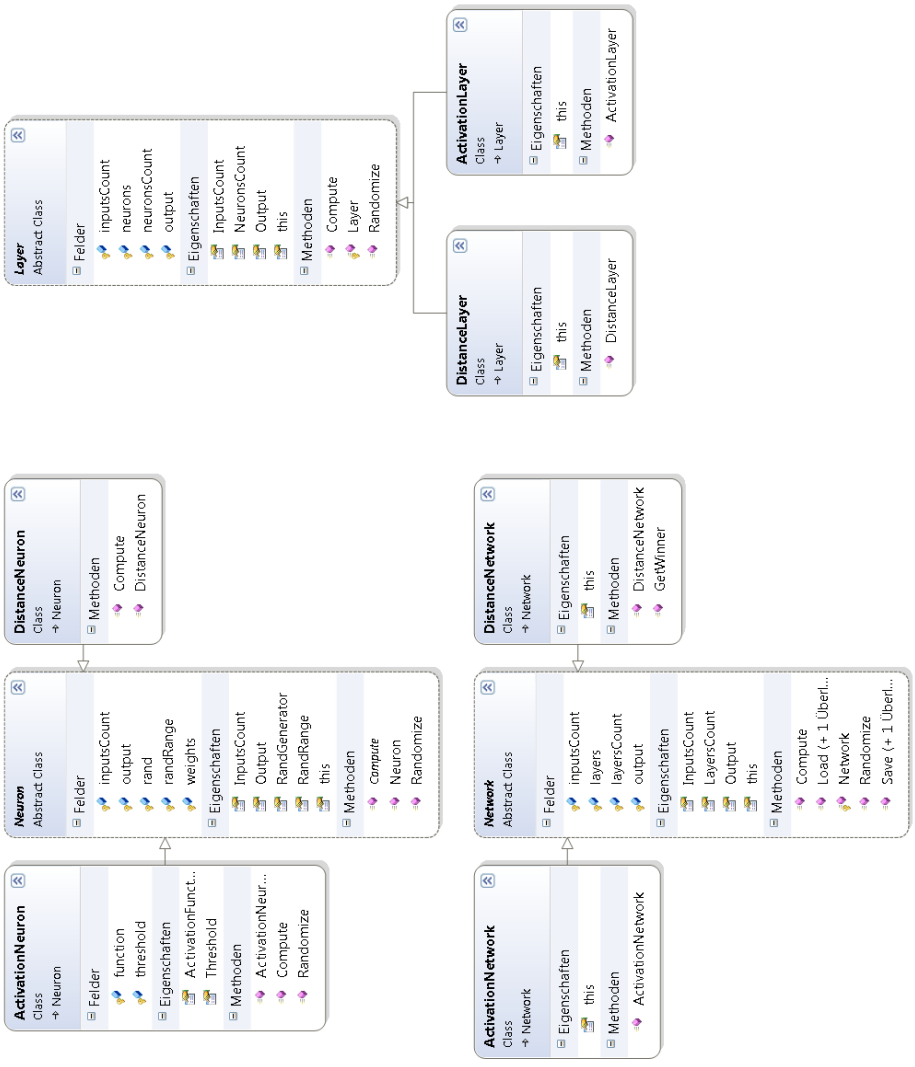


Abbildung 5.2: AForge.Neuro Klassendiagramm (Teil 1)



Abbildung 5.3: AForge.Neuro Klassendiagramm (Teil 2)