



~~Bachelor in Medien- und Kommunikationsinformatik~~

SAT

WS2018

Prof. Dr. Christian Kücherer

Erstfassung

~~Wie können neuronale Netze in der
Softwaretechnik eingesetzt werden und wo
sind die aktuell bekannten Grenzen~~

vorgelegt von:

**Konstantin
Rosenberg**

MKI, 6. Semester

**Robin
Schramm**

MKI, 6. Semester

**Ravell
Heerdegen**

MKI, 6. Semester

Kontaktadressen:

konstantin_nils.rosenberg
@Student.reutlingen-
university.de

robin_connor.schramm
@Student.reutlingen-
university.de

ravell.heerdegen
@Student.reutlingen-
university.de

Eingereicht am: 20.11.2018

Abstract (Alle)

Künstliche neuronale Netze haben in den letzten zwanzig Jahren ein exorbitantes Interesse für vielerlei Bereiche erweckt. Ihre Entwicklung geht bis in die fünfziger Jahre zurück und erfährt seitdem immer wieder neue Fortschritte. Dies liegt vor allem daran, dass sie sich weiterentwickeln und dazulernen können und somit auch für zukünftige Themen und Aufgaben geeignet sind. Besonders im Bereich der Softwaretechnik sind künstliche neuronale Netze heute nicht mehr wegzudenken. In dieser Arbeit wird näher auf die Disziplinen Mustererkennung, Kosten- und Aufwandsschätzung sowie Qualitätsmanagement eingegangen und Beispiele aufgezeigt, um die Frage zu beantworten, wie neuronale Netze in der Softwaretechnik eingesetzt werden können und wo die aktuell bekannten Grenzen sind. Dafür werden relevante Begriffe und Methoden erläutert sowie handfeste Beweise referenziert. Auch vergleichen wir in unserer Arbeit die verschiedenen Arten von neuronalen Netzen und zeigen Vor- sowie Nachteile auf. Das Ergebnis unserer Arbeit ist, dass sich neuronale Netze fest in der Softwaretechnik etabliert haben, und jede aufgeführte Art in jeder der beschriebenen Disziplinen sowohl eindeutige Verbesserungen als auch Defizite gegenüber klassischen Methoden aufweist. Daraus resultiert, dass es noch Verbesserungsbedarf sowie -möglichkeiten für die Entwicklung von künstlichen neuronalen Netzen gibt. In Zukunft könnten sich noch mehr Gebiete erschließen, welche mit neuronalen Netzen gekoppelt werden, um schnellere und bessere Ergebnisse zu erzielen. Vorstellbar ist auch, dass sich komplett neue Arten von neuronalen Netzen bilden, da selbst heute schon diverse Typen kombiniert werden, um zu neuen Erkenntnissen zu gelangen.

Inhaltsverzeichnis

1	Einleitung (Alle)	3
2	Pattern recognition (Ravell Heerdegen)	4
2.1	Speech recognition	4
2.1.1	Speech recognition und convolutional neural networks .	5
2.1.2	Speech recognition und residual neural networks	6
2.2	Emotion recognition	6
2.3	Fazit	8
3	Cost and effort estimation (Robin Schramm)	8
3.1	Stand der Technik	9
3.2	COCOMO	9
3.3	Soft Computing Ansätze	10
3.4	Leistungsbeurteilung	11
3.5	Vergleich	11
3.6	Fazit	11
4	Software quality (Konstantin Rosenberg)	12
4.1	Automatisiertes Testen durch neural networks	12
4.2	Evaluation von Softwarequalität durch neural networks	13
4.3	Vorhersage von SQ durch neural networks	14
4.4	Fazit	14
5	Fazit und Ausblick (Alle)	14



1 Einleitung (Alle)

Diese Arbeit soll beantworten, ob neuronale Netze (NNs) in der Softwaretechnik (SWT) eingesetzt werden können und wo die Grenzen der Anwendungen liegen. Die Arbeit soll keine Umfrage sein, ob NNs in der Realität eingesetzt werden. Vielmehr werden drei Themengebiete vorgestellt, in denen Forscher NNs testeten und einsetzten. Unser Ziel ist es, anhand der Themengebiete und konkreter Beispiele herauszufinden, ob und wie NNs in der SWT eingesetzt werden können, welche Probleme sie lösen und an welche Grenzen sie stoßen. Sobald NNs bessere Ergebnisse als herkömmliche Methoden erzielen, ohne den Aufwand oder die Kosten signifikant zu erhöhen, gehen wir davon aus, dass ihr Einsatz in einem bestimmten Gebiet möglich ist. Künstliche neuronale Netzwerke, auch artificial neural networks(ANNs) oder nur neural networks(NNs) basieren auf der Funktion des biologischen Nervensystems. Informationen werden durch eine Vielzahl miteinander verknüpfter parallel laufender Einheiten verarbeitet. Die daraus entstehenden Systeme trainieren sich teilweise selbst, um schnellere Entscheidungen treffen und Fehler vermeiden zu können.[1] NNs lassen sich in Kategorien unterteilen. Bekannte sind convolutional neural networks (CNNs), deep neural networks (DNNs) und multilayer perception(MLP)-NNs. In dieser Arbeit wird auf verschiedene NN-Arten eingegangen. Convolutional neural networks (CNNs) sind feed-forward NNs. Das bedeutet, dass Signale durch die verschiedenen Schichten (Layer) des NNs gegeben werden..[2] Residual neural networks (Resnets) gehören zur Kategorie der deep neural networks (DNNs). Jeder Layer des Resnets besitzt mehrere trainierbare Neuronen. Das Besondere ist der Dropout, welcher durch zufälliges Ausschalten von Neuronen Überanpassungen vermeidet.[3] Multilayer perceptrons (mehrschichtige neuronale Netze) benutzen u.a. Back propagation, um Parameter für künstliche neuronale Netze zu trainieren. Back propagation dient dem Trainieren von Layern, genauer dessen Neuronen, um auf entstandene Fehler aufmerksam zu machen, welche rückwärts durch das NN gegeben werden, um die sogenannten shared weights neu auszurichten.[2] Das Hidden Markov Model (HMM) ist bekannt für seinen Nutzen im Bereich der speech recognition. HMM ist in der Lage Wörter zu modellieren, welche daraufhin einem bestimmten Kontext zugeordnet werden können. Der HMM-Algorithmus gehört zu den unsupervised learning techniques, da er Parameter unkontrolliert schätzt ohne zusätzliche Kennzeichnungen durch Menschen zu benötigen. [4] Ein MLP-NN weist während des Trainings jedem Neuron eine Gewichtung zu, um den Einfluss zwischen Input und Output zu maximieren.[5] Radial Basis Function-NNs (RBF) werden in den Gebieten der pattern recognition und Funktionsapproximation eingesetzt. Im ersten der folgenden Kapitel wird die

Disziplin pattern recognition und dazugehörige Verfahren und Möglichkeiten näher beschrieben. Anschließend wird der Einsatz von NNs im Gebiet cost and effort estimation erläutert und damit verbundene Methoden und Praktiken aufgezeigt. ~~Das letzte~~ Kapitel befasst sich mit neuronalen Netzen im Bereich quality management und gibt einen Einblick auf verwendete Mittel und Vorgehensweisen, welche in dem genannten Bereich notwendig und nützlich sind. Letztendlich folgt ein Fazit, welches die Ergebnisse aus allen Kapiteln vereint, vergleicht und bewertet.



2 Pattern recognition (Ravell Heerdegen)

Pattern recognition zeichnet sich durch die Möglichkeit aus, Klassifizierungen durchzuführen, welche entweder überwacht oder unüberwacht, also automatisch und ohne menschliche Hilfe ablaufen [6]. Im Bereich pattern recognition haben sich in den letzten Jahren besonders neuronale Netze als enormer Fortschritt herausgestellt. Mit neuronalen Netzen werden u.a. Muster erkannt und zugeordnet, Umgebungen analysiert, z.B. im Bereich emotion recognition, Merkmale erkannt und spezifischen Klassen zugeordnet oder auch Bewertungen von Performanz durchgeführt.[7] Im folgenden Kapitel werden zwei Disziplinen bezüglich pattern recognition beschrieben. Der erste Abschnitt befasst sich mit der Disziplin speech recognition. Dabei gehe ich auf die Definition von speech recognition ein und zeige anschließend diverse Beispiele für den Einsatz von neuronalen Netzen in selbiger Disziplin auf. Im zweiten Abschnitt folgt die Disziplin emotion recognition sowie einige Beispiele für den Einsatz von neuronalen Netzen in der selbigen Disziplin. Abschließend folgt ein Fazit, welches den Inhalt des Kapitels kurz zusammenfasst und die Ergebnisse reflektiert.

2.1 Speech recognition

Speech recognition, auch automatic speech recognition (ASR) genannt, macht es Maschinen möglich, Sprache in Text oder Kommandos zu übersetzen. Dies geschieht sowohl durch Identifikation als auch durch Verstehen der eingegebenen Sprachsignale. Die Sprachsignale werden auf erlernte Muster angewendet, wobei Merkmale entnommen werden, um die Signale zuordnen zu können. Das Ziel ist, die menschliche Sprache zu verstehen, um mit dem Menschen kommunizieren zu können.[1]

2.1.1 Speech recognition und convolutional neural networks

In [2] beschreibt Du Guiming et. al. einen Versuch, bei dem Speech recognition in Verbindung mit einem CNN getestet wird. Die dafür verwendeten Trainingsdaten stammen von 30 Personen. Für die automatische Erkennung der Sprachsignale und ihrer Tonhöhen wird der Mel Frequency Cepstral Coefficient (MFCC) [8] eingesetzt, welcher das Frequenzspektrum kompakt darstellen kann. Für die Merkmalextraktion kommen **hamming windows** zum Einsatz. Mit dieser Technik können Signale gefiltert und Varianzen geglättet werden, bevor sie das neuronale Netz erreichen. Nach der Verarbeitung der Signale, wird das neuronale Netz mit fünf Personen getestet. Beim ersten Durchlauf der Signale durch das neuronale Netz, sind die erwarteten und vorhergesehenen Werte alle gleich. Der Unterschied zwischen erwarteten und echten Werten wird als cost function bezeichnet. Ist die cost function niedrig, sind die Parameter hochwertiger. Es zeigt sich, dass mit steigender Durchlaufzahl die cost function niedriger wird. Somit kann bestätigt werden, dass sich CNNs sowohl für die Erkennung von isolierten Wörtern eignen, als auch für die Reduzierung von Varianzen.[2]

Rafael M. Santos et. al. zeigen in [9] auf, wie sich ein CNN in einer geräuschvollen Umgebung verhält. Da sich die sogenannten shared weights in einem CNN über den gesamten Eingaberaum erstrecken, können Merkmale selbst dann gefunden werden, wenn sie sich aufgrund von Störungen z.B. durch Lautstärke an unterschiedlichen Positionen befinden. In dem Experiment wird das CNN so trainiert, dass es die HMM-Verteilungswahrscheinlichkeit schätzen kann. Außerdem klassifiziert das CNN nicht nur die aktuelle Aufnahme, sondern bezieht vorherige und nachkommende Aufnahmen mit ein, um Zusammenhänge zwischen den Informationen herstellen zu können. Als Eingabe werden einzelne Wörter mit ansteigender Lautstärke eingesetzt. Die Eingabedaten bestehen aus brasilianischen und portugiesischen Wörtern, welche zehn mal von jeweils sechs Männern und zwei Frauen vorgetragen wurden. Die Aufnahmen wurden in einer unkontrollierten Umgebung mit einem Mobiltelefon aufgezeichnet, von welchem 8000 Samples in der Sekunde entnommen wurden. Bei dem Versuch hat sich ergeben, dass die Kombination von CNN und HMM, sowohl **feature extraction** als auch signal detection mit großem Erfolg durchführt. Auch wurde herausgefunden, dass sich die Kombination CNN-HMM am wenigsten von Lautstärke und Geräuschen beeinflussen lässt, im Vergleich zu anderen getesteten Kombinationen wie GMM [6] oder SVM [6] in Verbindung mit HMM.

2.1.2 Speech recognition und residual neural networks

In der Studie [3] wurde ein Resnet mit mehreren Res in Kombination mit Speech recognition getestet. Es hat sich zunächst herausgestellt, dass mit einer Anzahl von acht Res, welche hintereinander geschaltet werden, die beste Performanz in Bezug auf die word error rate (WER) erreicht wird. Ab einem Wert von mehr als acht Res, hat die Performanz abgenommen. Verglichen wurde das sogenannte 8Res dabei mit einem 8-DNN, welches aus sechs hidden-layers bestand. Dabei hat sich ergeben, dass das 8Res bessere Ergebnisse als das 8-DNN erzielt. Auch wird aufgezeigt, wie sich wide residual networks (WResnets) auf die Validierung von Daten auswirkt. Wurde die Anzahl der Layer im getesteten Resnet von vier auf zehn erhöht, wurde eine noch bessere Performanz erreicht. Ab einem Wert von zwölf nahm die Performanz wiederum ab, da die Neuronen ab diesem Wert überangepasst waren. Schließlich wurde festgestellt, dass HMM-Resnets mit steigender Anzahl an Res und Layern eine um 0,4% bessere Performanz als ein HMM-DNN [3] wie z.B. CNN mit HMM in Bezug auf die WER erreichten.



2.2 Emotion recognition

Emotion recognition, auch automatic emotion recognition (AER) genannt, ist ein interdisziplinäres Forschungsgebiet, welches u.a. die Bereiche Computer Sciences, Cognitive Science sowie Psychology vereint. Das Erforschen von emotionalen Zuständen ist hilfreich, um menschliche Aktionen zu verstehen, als auch um menschliche Faktoren in künstliche Systeme zu integrieren.[10] Generell wird das Erfassen von Emotionen über die Muskelbewegungen im menschlichen Gesicht geregelt. Die Muskelbewegungen im Gesicht werden auch action units (AUs) genannt. Nachdem ein Gesicht erfasst und isoliert wurde, kann man dem Gesicht eine Emotion zuordnen mithilfe von action units analysis.[11] Beim Arbeiten im Bereich Emotion recognition und facial expression recognition, unterscheidet man zwischen sechs generellen, kulturunabhängigen Emotionen, welche 1971 von Ekman und Friesen als universell festgelegt wurden: Wut, Ekel, Angst, Freude, Trauer und Überraschung.[12] Emotion recognition findet aber auch Anwendung in Verbindung mit Sprache, bzw. Körpersprache. Mit Emotion recognition möchte man natürliche Kommunikation zwischen Mensch und Maschine herstellen und verbessern, jedoch auch Zustände eines Menschen erkennen, um anschließend passende Werbung oder zugeschnittene Angebote zu erstellen. Methodisch wird zwischen static approaches und dynamic approaches unterschieden. Das bedeutet, einerseits mit statischen Bildern zu arbeiten, andererseits mit Sequenzen von Bildern, u.a. auch Videos. Emotion expression recognition wird grundsätzlich in die

Schritte face image acquisition, feature extraction und facial emotion expression recognition unterteilt. [13]

Emotion recognition und convolutional neural networks

In [13] wird ein Versuch mit einem CNN in Verbindung mit automatic facial expression recognition (AFER) beschrieben. Bei dem Versuch wurde das Facial Action Coding System (FACS) verwendet, um die Genauigkeit der Erkennung von Emotionen zu verbessern. Das geschieht aufgrund der Möglichkeit des FACS, die AUs im Gesicht zu erkennen und zu interpretieren. Auch kann mit dem FACS die Intensität einer Emotion definiert werden. Als Trainingsdaten fungierte der Cohn-Kanade (CK+) Datensatz. Der CK+ besteht aus 123 Individuen, welche verschiedene Folgen von emotionalen Ausdrücken aufgenommen haben. Die daraus entstandenen Bilder wurden auf eine einheitliche Größe skaliert und verpixelt. Zusätzlich kamen pre-processing techniques wie z.B. Rotations- oder Farbkorrektur zum Einsatz, bevor die Daten in das CNN gegeben wurden. Im ersten convolutional layer des eingesetzten CNN wurden visuelle Merkmale wie Kanten, Lippenformen, Falten, Augen und Augenbrauen extrahiert. Durch die berechneten shared weights zwischen den Schichten im CNN, kann ein hohes oder ein geringes Vorkommen von AUs im output layer festgestellt werden, was auf verschiedene Klassifizierungen von Emotionen schließen lässt. Ergebnis des Versuches war, dass das trainierte CNN Parallelen zwischen den Emotionen Wut und Neutralität gefunden hat, was mit den ähnlichen Gesichtszügen von Wut und Neutralität zusammenhängt. Auch hat sich ergeben, dass Freude am besten erkannt wurde. Dadurch wurde bestätigt, dass sich CNNs für die Erkennung von Emotionen in einem menschlichen Gesicht eignen.

Das Experiment [11] benutzt ebenfalls ein CNN, jedoch mit anderen Techniken. Zum Einsatz kommen zwei Methoden, welche sich für das Erfassen von sogenannten lokalen sowie globalen Informationen eignen. Die ~~eine, genannt~~ bottom-up, eignet sich zum Isolieren von Gesichtern aus Bildern, um diese anschließend in das bereits trainierte CNN als Input einzugeben. Die ~~andere nennt sich~~ top-down ~~und~~ bezeichnet sich als labeling algorithm. Beim labeling werden ~~label, auch descriptors~~ genannt, einer Szene zugeordnet, um den Kontext der Szene zu erfassen. Die descriptors werden daraufhin in ein bayesian network (BN) [11] gegeben, welches die descriptors verarbeitet, um Beziehungen und Abhängigkeiten zwischen den Begriffen herzustellen. Die verarbeiteten descriptors werden als Input in das CNN geleitet. Der output layer des CNN besteht dadurch aus drei Ausgängen, welche die Szene

als negativ, positiv oder neutral einordnen. Zum Einsatz kommt außerdem die GAF Datenbank, welche 6470 Bilder beinhaltet. Diese Menge wurde aufgeteilt auf die Trainingsdaten, Validierungsdaten und auf die Testdaten. Zusätzlich wurde ein Dropout eingesetzt, um die Überanpassung der Neuronen zu vermeiden. Endlich hat sich ergeben, dass das CNN in Kombination mit einem BN die besten Ergebnisse liefert, im Gegensatz zum Versuch ohne BN bzw. ohne CNN. Ein interessanter Ausbau des Versuches wird am Ende gegeben. Dabei wird erwähnt, dass die Klassifizierungen negativ, positiv und neutral, auf drei CNNs aufgeteilt werden könnten, um jedes zu spezialisieren und somit noch bessere Ergebnisse erzielen zu können.

2.3 Fazit

Die aufgeführten Versuche haben gezeigt, dass neuronale Netze im Bereich der Mustererkennung vielerlei Anwendung finden. Vorallem die CNN werden in beiderlei Disziplinen in Verbindung mit diversen Methoden und Vorgehensweisen eingesetzt. Aktuelle Grenzen liegen momentan in der Skalierbarkeit der Schichten, sowie in der Qualität der Trainings- und Testdaten. Zukünftig wären neue Kombinationen oder sogar neue Arten von neuronalen Netzen möglich, um noch bessere und vorallem schnellere Ergebnisse zu erzielen, ohne immenses vorheriges Training oder Konfigurieren.

3 Cost and effort estimation (Robin Schramm)

In den sechziger und siebziger Jahren stieg die Größe und Komplexität von Softwareprojekten drastisch an. Softwareentwicklung wurde immer mehr kommerziell genutzt wodurch sich neue Arten von Kunden bildeten. Darunter vor allem das Militär. Durch die Vergrößerung der Branche war dies immer weniger effektiv ohne vorherige Planung zu entwickeln. Als Lösung für das Problem entstand das Software Projekt Management(SPM), welches auch heutzutage noch eine zentrale Rolle in der Softwareentwicklung spielt.[14] Einer der Punkte mit dem sich SPM beschäftigt ist das project cost management(PCM). Nach Bajita[14] unterteilt sich PCM in die Kategorien software cost estimation(SCE) und software effort estimation(SEE). Aktuelle Forschungen zeigen, dass genaue SCE und SEE die Chance auf ein qualitativ hochwertiges Produkt erhöht.[15][16] Jeffery et. al. beschreiben, dass SCE eines der wichtigen Werkzeuge ist, um einen kompetitiven Vorteil im Bereich des IT-Service zu erhalten.[17] Aus dieser Notwendigkeit entstanden viele SCE und SEE Tools, welche Experten bei der Projektplanung unterstützen sollen.

Im Zuge dieses Kapitels werden als Erstes die bestehenden SCE und SEE Techniken vorgestellt und kategorisiert. Als nächstes wird das **klassische constructive cost estimation model**(COCOMO) sowie neuere Soft Computing Modelle vorgestellt und deren Vorteile und Probleme aufgelistet. In den weiteren Absätzen wird erklärt, wie die Ergebnisse der Schätzungen gemessen und in Zahlen gefasst werden können, um danach auf einige Beispiele von direkten Vergleichen der mathematischen und **Soft Computing** einzugehen. Zum Schluss wird anhand der vorherigen Abschnitte beantwortet, ob NNs in der **Softwaretechnik(SWT)** im Bereich der SCE und SEE eingesetzt werden können und wo deren derzeit bekannte Grenzen liegen.

3.1 Stand der Technik

In der Planungsphase eines Softwareprojekts ist es essentiell, die Kosten und die Dauer des Projekts **zu kennen**. Kunden geben Aufträge und müssen wissen, wie viel welche Funktion kosten würde und wann das Projekt voraussichtlich abgeschlossen sein wird. Grobe Schätzungen können allein durch Erfahrung der Entwickler abgegeben werden, wie zum Beispiel in SCRUM, bei dem User Stories je nach Komplexität Story Points gegeben werden. Bei ansteigender Komplexität **Projekten** kann die manuelle Schätzung nicht ausreichen. Das zeigt sich im großen Zweig des SPM, welcher sich allein mit der Planung und Schätzung von Softwareprojekten auseinandersetzt. Eine wichtige Metrik in der SPM ist die Größe der Software. Dabei kann man zwei Arten von Methoden der Größenbestimmung unterscheiden. **Zum Einen die gezählten Lines of Code (LOC)** und die daraus abgeleiteten **1000 Lines of Code (KLOC)**. **Zum anderen die function points (FPs)**. Anhand der Arten der Modellkonstruktion lassen sich SCE und SEE Modelle in vier Kategorien unterteilen. Erstens Expertensysteme, die sich auf die Erfahrung der Personen verlassen[18]. Zweitens die **Linearen Systeme**, die auf der Größenmessung mit FP basieren[15]. Drittens die **nonlinearen Mathematischen Modelle**. Viertens die neueren Soft Computing Ansätze, die entweder bisherige Methoden erweitern oder komplett neu aufbauen[19].[20]. Eines der beliebtesten Tools ist COCOMO.[21] COCOMO lässt sich den nichtlinearen Systemen zuordnen.

3.2 COCOMO

COCOMO wurde 1981 von Boehm entworfen, um Fehler im SPM zu verringern. COCOMO ist ein algorithmisches Modell mit vielen **Eingabeparametern(zahl)**. In COCOMO gibt es drei Typen von Projekten, gemessen an der Komplexität. Organic Mode für einfache kleine Projekte, embedded mode

für innovative komplexe Projekte mit hohen Anforderungen sowie den semi-detached mode für Projekte zwischen dem organic und embedded type. Der Projekttyp wirkt sich neben den LOC signifikant auf die Endschätzung aus. Zudem nutzt COCOMO einige **cost drivers**, welche je nach Art variieren. In Boehms ersten COCOMO gab es 15 cost drivers. Cost drivers werden mit Werten zwischen *sehr niedrig* und *sehr hoch* bewertet. Die Werte stellen dabei Gewichtungen dar, die sich ebenfalls auf den Endwert auswirken. Das Ergebnis stellt die Schätzung der Kosten in einer gewünschten Währung dar, sowie die benötigten Personenmonate. Die klassischen Modelle bringen Probleme mit sich, von denen hier einige beschrieben werden.

- Algorithmische Tools sind generisch und somit nicht an das spezifische Projekt angepasst, daraus ergibt sich ein Mangel an bestehenden Daten für die eigene Nische. Daten von anderen Firmen sind in der Regel nicht verfügbar.
 - Erfahrung in großen Projekten ist schwierig, da diese eine lange Lebensdauer haben, bis sie abgeschlossen werden und die Schätzung mit dem Ergebnis verglichen werden kann.
 - Oft werden die Schätzungen nebenbei gemacht und nicht als eigenständiger Punkt in der Planung beachtet, was die Wahl der Eingabeparameter negativ beeinflusst. Die Eingabeparameter beeinflussen die Schätzung jedoch maßgeblich
 - Mangel an Flexibilität bei sich ändernden Anforderungen. **Besonders im Agilen Bereich.**
 - Schwierigkeit und Aufwand, Projekt spezifische Daten zu sammeln, zu beurteilen und in konkreten Zahlen auszudrücken. [22][18][23]
- Einige der genannten Probleme versuchen verschiedene Forscher mit Soft Computing Ansätzen zu lösen[16]

3.3 Soft Computing Ansätze

Unter Soft Computing versteht man vor allem Fuzzy Logic, Evolutionäre Algorithmen und Künstliche Neuronale Netze (NNs). Mit ihnen kann das Problem gelöst werden, **nur an ein Projekt angepasst zu sein**, da sie sich leicht generalisieren lassen. Zudem sind sie sehr flexibel und anpassungsfähig, was sie bei sich ändernden Anforderungen wertvoll macht.[24] Damit sie zufriedenstellende Ergebnisse liefern, müssen die Trainingsdaten passend gewählt sein. NNs mit zufällig gewählten Trainingsdaten liefern keine besseren Schätzungen als Regressionsmodelle. [25]

3.4 Leistungsbeurteilung

Um die Modelle untereinander vergleichen zu können, gibt es verschiedene Methoden die Qualität zu messen. Nach Finnie et. al. ist der mean absolute relative error (MARE) die bevorzugte Fehlerbestimmungsmethode von Software-Messforschern. Der MARE wird wie folgt berechnet:

$$\left| \frac{estimate - actual}{estimate} \right| \quad (1)$$

estimate ist dabei der vom Modell geschätzte Wert und *actual* der nach der Durchführung tatsächlich gemessene Wert. Um herauszufinden, ob das Modell über- oder unterschätzt, wird der mean relative error (MRE) wie folgt berechnet:

$$\left(\frac{estimate - actual}{estimate} \right) * 100 \quad (2)$$

Ein Wert von Null ist dabei ein perfektes Ergebnis. Je höher oder niedriger die Zahl, desto größer wurde über- oder unterschätzt.[26] Ein Ergebnis zwischen -25 und 25 gilt als gutes Ergebnis in SEE.[23]. Viele der vorgestellten Modelle werden an laufenden Industrieprojekten getestet. Das am weitesten akzeptierte Evaluationskriterium ist der MRE aus einem Satz von 63 NASA Projekten.[5]



3.5 Vergleich

Abrahamsson et. al. vergleichen in ihrem Artikel inkrementelle Ansätze mit globalen Ansätzen im Kontext der agilen Entwicklung. Die verwendeten Modelle sind einerseits die Schrittweise lineare Regression und andererseits zwei verschiedene Arten von NNs. Eines ihrer Ergebnisse war, dass NNs nicht generell besser sind als einfache Regression. Das Multilayer Perceptron (MLP) aus dem Test liefert zum Beispiel schlechtere Ergebnisse als die meisten Regressionsmodelle. Das Radial Basis Function (RBF) NN lieferte ähnliche Ergebnisse wie das Regressionsmodell.[23] TODO - Besser als COCOMO[5] TODO - Auch besser als COCOMO in NASA Testprojekten in über 90 Prozent[27]

3.6 Fazit

Dieses Kapitel hat verschiedene Möglichkeiten aufgezeigt, wie NNs in den wichtigen Bereichen der SCE und SEE innerhalb der SWT eingesetzt werden. Wenn für gewisse Projekte passende NNs ausgewählt und korrekt konfiguriert

werden, liefern NNs bessere Ergebnisse als reine mathematische Modelle. Vorallem in der agilen Entwicklung lösen NNs viele Probleme anderer Ansätze, auch in Verbindung und als Erweiterung der anderen Modelle. Dennoch sind sie nicht in ausschließlich allen Fällen die beste Lösung. Es existiert noch kein NN, welches eine Universallösung für alle Softwareschätzungen darstellt. Für manche Projekte bietet sich die Lösung mit NNs mehr an als für andere, vorallem da die optimale Lösung noch nicht während der Planung bekannt ist. Vorallem in Umgebungen mit wenigen Daten oder wenig Zeit sind NNs noch nicht praktikabel, da ein passendes NN zuerst entwickelt und trainiert werden muss.

4 Software quality (Konstantin Rosenberg)

”Softwarequalität ist die Gesamtheit der Merkmale und Merkmalswerte eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen” [28, p. 20]. F. Klaus zitiert die Norm ISO/IEC 25000, dabei unterscheidet er zwischen funktionalen und nicht-funktionalen Merkmalen von Softwarequalität (SQ). Funktionale Merkmale beziehen sich auf die reine Funktionalität einer Software. Die nicht-funktionalen Anforderungen hingegen beschreiben die Zuverlässigkeit, Benutzbarkeit, Effizienz, Anpassungsfähigkeit und Übertragbarkeit eines Softwareprodukts. Um SQ zu erreichen gibt es verschiedene Ansätze. In Kontrast stehen dabei analytische (AQS) und konstruktive Qualitätssicherungsmaßnahmen (KQS). AQS beziehen sich auf das Vermeiden von Fehlern im Vorfeld. KQS hingegen sollen nach Abschluss der Arbeit Fehler aufdecken. [28, p. 29] Um SQ konstruktiv zu sichern, schlägt F. Klaus das Einführen von Testmethoden, Testwerkzeugen und die Schulung der Mitarbeiter als Maßnahmen vor. Für eine analytische Sicherung der Qualität nennt er das Überprüfen von Tests und Dokumenten. Im Folgenden Abschnitt zeige ich Verfahren, um die genannten Maßnahmen mit NN’s umzusetzen. Abschließend wird auf die Frage eingegangen, ob NN’s zur Qualitätssicherung in Software verwendet werden können.

4.1 Automatisiertes Testen durch neural networks

Softwaretests haben die Aufgabe zu erkennen, ob Software den qualitativen Ansprüchen genügt und um Fehler zu finden und zu beheben. [29] L. Wu beschreibt in [29] ein Verfahren, um mit BPNN’s automatisiert funktionale Tests zu erstellen. Bei diesem Vorgehen gibt es eine Reihe an Akteuren. Ein bestehendes System oder Programm. Außerdem die Spezifikation der

Funktionen, einen Testgenerator, eine Testdatenbank und das BPNN. Der Testgenerator erzeugt zufällige Input-Variablen, passend zu den festgelegten Systemspezifikationen. Die Variablen werden in der Testdatenbank abgespeichert. Das Softwaresystem erzeugt Erwartungswerte. Als Input für das BPNN dienen die erzeugten Testwerte. Das NN verarbeitet die Eingabewerte und generiert Ausgaben, die gegen den Erwartungswert geprüft werden. Mit diesem Training nähert sich das NN dem entsprechenden Erwartungswert an. Sobald die Ausgabewerte einen entsprechenden Toleranzwert erreichen, kann der Input der einzelnen Knoten des NN verwendet werden, um Tests abzuleiten. Der Vorgang wird wiederholt bis die Testabdeckung vollständig ist. Anhand eines Versuchs werden in [29] Tests für ein Softwaresystem erzeugt. Dabei wird eine vollständige Testabdeckung für ein Beispielsystem erzeugt.

In [30] beschreiben M. Negar und S. Morteza ebenfalls einen Versuch, der die Effektivität von NN's deutlich macht. In dem Versuch werden zwei NN's verwendet. Eines davon dient dem white box testing, das andere dem black box testing. Nach dem Training der NN's, wird mit dem ersten NN ein Programm mit 394 Zeilen Code getestet. Dabei wird eine vollständige Testabdeckung erzielt. Das zweite NN wird mit absichtlich gesetzten Fehlern getestet. Dabei erkannte das System in 92% der Fälle, dass ein Fehler vorhanden ist. Mit diesem Vorgehen lassen sich nach [30] haltbare Softwaretests produzieren, sowie Zeit und Aufwand einsparen.

4.2 Evaluation von Softwarequalität durch neural networks

Es bietet Vorteile den qualitativen Stand von Software zu kennen, um Vorbereitungen zu treffen oder früh Mängel aufzudecken und zu beheben [31], deshalb stellen O. Pomorova und T. Hovorushchenko in [31] eine Methode vor, um mit NN's die Softwarequalität in einer Projektphase zu evaluieren. Als Input für das NN dienen SM's. Davon enthält eine SM geschätzte, die andere die tatsächlichen Werte. Durch die Ausgabe des NN's wird die evaluierte Softwarequalität beschrieben oder eine Vorhersage über die zukünftige Komplexität des Projekts gemacht. In [31] wird ein Versuch beschrieben in dem vier verschiedene Softwareprojekte mit dieser Methode eingeschätzt wurden. Dabei erzielte das System genaue Ergebnisse. Es stellte sich jedoch auch heraus, dass eine unzulängliche Anzahl von Bewertungskriterien zu ungenauen Ergebnissen führen kann. Wenn die Faktoren gegeben sind, bietet dieses

Verfahren aber eine genaue Möglichkeit um SQ erkennbar zu machen und dadurch Rückschlüsse auf die Entwicklungsphase zu ziehen.

4.3 Vorhersage von SQ durch neural networks

Durch eine Vorhersage der SQ können Defizite frühzeitig erkannt und verbessert werden. Mitarbeiter können sich besser auf bevorstehende Probleme vorbereiten und man erhält die Möglichkeit, Zusammenhänge zwischen getroffenen Entscheidungen und SQ zu erkennen [32]. In [32] wird deshalb ein Vorgehen beschrieben, um eine Vorhersage zu SQ in objektorientierten Systemen zu treffen. Dabei werden SQ's und NN's verwendet, um Vorhersagen zu machen. Bei den verwendeten Metriken handelt es sich um objekt orientierte Metriken wie die **depth of the inheritance tree (DIT9)**-Metrik. Die Metriken werden über ein sog. fuzzy neural network verarbeitet. fuzzy neural networks können auch mit wenig Input genaue Aussagen treffen. Vorteile dieser Variante zu herkömmlichen Ansätzen zum Vorhersagen von SQ werden in einem Beispiel in [32] deutlich. Dabei zeigt sich, dass die Zusammenhänge zwischen SQ und beeinflussenden Faktoren deutlicher zu erkennen sind, als auch dass das Vorgehen mit einer Vielzahl an Dateiformaten auskommt.

4.4 Fazit

Betrachtet man die genannten Beispiele für Einsatzmöglichkeiten von NN's, stellt man fest, dass es Anwendungsfälle gibt in denen sich NN's als sehr geeignet und effizient erweisen. In diesem Abschnitt sehen wir wie Softwaretests automatisch generiert werden können, um Zeit und Arbeitsaufwand zu sparen. Darüber hinaus erkennen wir die Vorteile von NN's zur Vorhersage und Bestimmung von SQ. Gerade das Vorhersagen von SQ, um Rückschlüsse auf Ursachen ziehen zu können, bietet eine große Chance, um die Arbeit an Softwareprojekten nachhaltig zu prägen und zu verbessern. Darum kann bestätigt werden, dass NN's im Bereich der Softwareentwicklung eingesetzt werden können, um die Qualität zu verbessern.

5 Fazit und Ausblick (Alle)


Vergleicht man die aufgeführten Beispiele und dessen Ergebnisse miteinander, so kommt man zu dem Schluss, dass neuronale Netze durchaus Anwendung im Bereich pattern recognition finden. Auch wurde einerseits gezeigt, dass sowohl CNNs als auch Resnets für Speech recognition eingesetzt werden. Andererseits wurde aufgezeigt, dass sich CNNs auch im Bereich emo-

tion recognition etabliert haben und wie in [11] bewiesen, sogar am Besten für facial emotion expression recognition in unkontrollierten Umgebungen geeignet sind. Ebenfalls wurden verschiedene Ansätze für den Einsatz von CNNs gegeben, was zeigt, dass CNNs auf unterschiedliche Weise fungieren können, um diverse Aufgaben zu erledigen. Auch wird bezogen auf die aufgeführten Beispiele deutlich, dass die genannten CNNs mitunter zu den momentan besten Lösungen für die Aufgaben **Speech-** sowie emotion recognition gehören. Erwähnenswert ist außerdem, dass unter aktuelle Grenzen u.a. die Fehlerrate für das Erkennen von Sprachsignalen und Emotionen gehört, welche momentan noch keine einhundert prozentige Zuordnung erreicht, Nebeneffekte wie die Überanpassung von Neuronen aufgrund zu tiefer Spezialisierung und das Zusammenführen von Wut und Neutralität in der Disziplin emotion recognition aufgrund der nicht zu identifizierbaren Unterschiede. Desweiteren wurde gezeigt, dass NNs bei korrekter Konfiguration, bessere Ergebnisse im Bereich SCE und SEE aufweisen als herkömmliche mathematische Modelle. Der Einsatz von NNS bei SCE und SEE ist noch nicht vollständig sinnvoll, da die Anwendung situations- und projektabhängig ist. Auch sind NNs dann nicht sinnvoll, wenn noch nicht genügend qualitative Trainingsdaten vorliegen, oder die Zeit für das Training der NN nicht gegeben ist. **U.a.** hat sich auch ergeben, dass NNs gute Ergebnisse auf dem Gebiet der Softwarequalität erzielen. Dabei wurden mit NNs automatisierte Tests erstellt, welche sowohl Zeit als auch Arbeitsaufwand einsparen konnten. Zusätzlich bieten NNs im Bereich SQ eindeutig bessere Vorhersagen von SQ, was die Arbeit an Softwareprojekten nachhaltig prägt. Aus den genannten Ergebnissen können wir bestätigen, dass neuronale Netze in der Softwaretechnik eingesetzt werden.

References

- [1] Meng, J., Zhang, J., Zhao, H.: Overview of the speech recognition technology. Proceedings - 4th International Conference on Computational and Information Sciences, ICCIS 2012, pp. 199–202 (2012). <https://doi.org/10.1109/ICCIS.2012.202>
- [2] Guiming, D., Xia, W., Guangyan, W., Yan, Z., Dan, L.: Speech recognition based on convolutional neural networks. 2016 IEEE International Conference on Signal and Image Processing, ICSIP 2016 pp. 708–711 (2017). <https://doi.org/10.1109/SIPROCESS.2016.7888355>
- [3] Vydana, H.K., Vuppala, A.K.: Residual neural networks for speech recognition. 25th European Signal Processing

- Conference, EUSIPCO 2017 2017-Janua, 543–547 (2017).
<https://doi.org/10.23919/EUSIPCO.2017.8081266>
- [4] Tang, X.: Hybrid hidden markov model and artificial neural network for automatic speech recognition. Proceedings of the 2009 Pacific-Asia Conference on Circuits, Communications and System, PACCS 2009 pp. 682–685 (2009). <https://doi.org/10.1109/PACCS.2009.138>
- [5] Khalifelu, Z.A., Gharehchopogh, F.S.: Comparison and evaluation of data mining techniques with algorithmic models in software cost estimation. Procedia Technology **1**, 65–71 (2012). <https://doi.org/10.1016/j.protcy.2012.02.013>
- [6] Padmanabhan, J., Premkumar, M.J.J.: Machine learning in automatic speech recognition: A survey. IETE Technical Review (Institution of Electronics and Telecommunication Engineers, India) **32**(4), 240–251 (2015). <https://doi.org/10.1080/02564602.2015.1010611>, <http://dx.doi.org/10.1080/02564602.2015.1010611>
- [7] A.~K.~Jain, R.~P.~W.~Duin, J.~Mao: {S}tatistical {P}attern {R}ecognition: {A} {R}eview. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 22, NO. 1, JANUARY 2000 **22**(1), 4–37 (2000)
- [8] Li Deng, D.Y.: Automatic speech recognition, vol. 9 (2015). <https://doi.org/10.1007/BF02747521>
- [9] Santos, R.M., Matos, L.N., Macedo, H.T., Montalvao, J.: Speech recognition in noisy environments with convolutional neural networks. Proceedings - 2015 Brazilian Conference on Intelligent Systems, BRACIS 2015 pp. 175–179 (2016). <https://doi.org/10.1109/BRACIS.2015.44>
- [10] Aracena, C., Basterrech, S., Snasel, V., Velasquez, J.: Neural Networks for Emotion Recognition Based on Eye Tracking Data. Proceedings - 2015 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2015 pp. 2632–2637 (2016). <https://doi.org/10.1109/SMC.2015.460>
- [11] Surace, L., Patacchiola, M., Sönmez, E.B., Spataro, W., Cangelosi, A.: Emotion Recognition in the Wild using Deep Neural Networks and Bayesian Classifiers. Proceeding ICMI 2017 Proceedings of the 19th ACM International Conference on Multimodal Interaction Pages 593–597 pp. 593–597 (2017). <https://doi.org/10.1145/3136755.3143015>, <http://arxiv.org/abs/1709.03820>

- [12] Huang, C.: Combining convolutional neural networks for emotion recognition. 2017 IEEE MIT Undergraduate Research Technology Conference, URTC 2017 **2018-Janua**, 1–4 (2018). <https://doi.org/10.1109/URTC.2017.8284175>
- [13] Awasthi, A.: Facial Emotion Recognition Using Deep Learning. IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI) Dec. 22, 2017 **1**(September), 9–12 (2013). <https://doi.org/10.1145/2818346.2830593>
- [14] Bajta, M.E., Idri, A., Ros, J.N., Fernandez-Aleman, J.L., Gea, J.M.C.D., Garcia, F., Toval, A.: Software project management approaches for global software development: a systematic mapping study. Tsinghua Science and Technology **23**(6), 690–714 (2018). <https://doi.org/10.26599/TST.2018.9010029>
- [15] Matson, J.E., Barrett, B.E., Mellichamp, J.M.: Software development cost estimation using function points. IEEE Transactions on Software Engineering **20**(4), 275–287 (1994). <https://doi.org/10.1109/32.277575>
- [16] Bilgaiyan, S., Mishra, S., Das, M.: A Review of Software Cost Estimation in Agile Software Development Using Soft Computing Techniques. In: 2016 2nd International Conference on Computational Intelligence and Networks (CINE), Computational Intelligence and Networks (CINE), 2016 2nd International Conference on, cine. p. 112. IEEE (2016). <https://doi.org/10.1109/CINE.2016.27>
- [17] Jeffery, D.R., Low, G.: Calibrating estimation tools for software development. Software Engineering Journal **5**(4), 215–221 (1990). <https://doi.org/10.1049/sej.1990.0024>
- [18]  Heemstra, F.J.: Software Cost Estimation. Handbook of Software Engineering, Hong Kong Polytechnic University **34**(10) (1992). https://doi.org/10.1142/9789812389701_014
- [19] Huang, X., Ho, D., Ren, J., Capretz, L.F.: Improving the COCOMO model using a neuro-fuzzy approach. Applied Soft Computing **7**(1), 29–40 (**jan** 2007). <https://doi.org/10.1016/J.ASOC.2005.06.007>
- [20] Huang, S.J., Lin, C.Y., Chiu, N.H.: Fuzzy decision tree approach for embedding risk assessment information into software cost estimation model. Journal of Information Science and Engineering **22**(2), 297–313 (2006)

- [21] Jain, R., Sharma, V.K., Hiranwal, S.: Reduce mean magnitude relative error in software cost estimation by HOD-COCOMO algorithm. In: 2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). pp. 708–712 (2016). <https://doi.org/10.1109/ICCICCT.2016.7988044>
- [22] Chen, Z., Menzies, T., Port, D., Boehm, B.: Feature subset selection can improve software cost estimation accuracy. *ACM SIGSOFT Software Engineering Notes* **30**(4), 1 (2005). <https://doi.org/10.1145/1082983.1083171>
- [23] Abrahamsson, P., Moser, R., Pedrycz, W., Sillitti, A., Succi, G.: Effort Prediction in Iterative Software Development Processes – Incremental Versus Global Prediction Models. In: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007). pp. 344–353 (2007). <https://doi.org/10.1109/ESEM.2007.16>
- [24] Boetticher, G.D.: Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains. *Model Based Requirements Workshop* pp. 17–24 (2001). <https://doi.org/10.1.1.19.111>
- [25] Setyawati, B.R., Sahirman, S., Creese, R.C.: Neural Networks for Cost Estimation. *AACE International Transactions* p. 13.1 (jun 2002)
- [26] Finnie, G.R., Wittig, G.E.: AI tools for software development effort estimation. *Software Engineering: Education and Practice*, 1996. Proceedings. International Conference pp. 346–353 (1996). <https://doi.org/10.1109/SEEP.1996.534020>
- [27] Gharehchopogh, F.S.: Neural networks application in software cost estimation: A case study. In: 2011 International Symposium on Innovations in Intelligent Systems and Applications. pp. 69–73 (2011). <https://doi.org/10.1109/INISTA.2011.5946160>
- [28] Franz, K.: Handbuch zum Testen von Web- und Mobile-Apps (2015). <https://doi.org/10.1007/978-3-662-44028-5>, <https://link.springer.com/book/10.1007%2F978-3-662-44028-5>
- [29] Wu, L., Liu, B., Jin, Y., Xie, X.: Using back-propagation neural networks for functional software testing. In: 2nd International Conference on Anti-counterfeiting, Security and Identification, ASID 2008 (2008). <https://doi.org/10.1109/IWASID.2008.4688385>, <https://ieeexplore.ieee.org/document/4688385>



- [30] Majma, N., Babamir, S.M.: Software test case generation & test oracle design using neural network. 22nd Iranian Conference on Electrical Engineering, ICEE 2014 (Icee), 1168–1173 (2014). <https://doi.org/10.1109/IranianCEE.2014.6999712>
- [31] Pomorova, O., Hovorushchenko, T.: Artificial neural network for software quality evaluation based on the metric analysis. Proceedings of IEEE East-West Design and Test Symposium, EWDTS 2013 pp. 0–3 (2013). <https://doi.org/10.1109/EWDTS.2013.6673193>
- [32] Peng, W., Yao, L., Miao, Q.: An approach of software quality prediction based on relationship analysis and prediction model. In: Proceedings of 2009 8th International Conference on Reliability, Maintainability and Safety, ICRMS 2009 (2009). <https://doi.org/10.1109/ICRMS.2009.5270097>, <https://ieeexplore.ieee.org/document/5270097>