



SAT
WS2018

Prof. Dr. Christian Kücherer

Zwischenversion
Neuronale Netze in der Softwaretechnik und
ihre 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 (KNNs) haben in den letzten zwanzig Jahren ein exorbitantes Interesse 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 werden die Disziplinen Mustererkennung, Kosten- und Aufwandsschätzung sowie Qualitätsmanagement näher betrachtet und Beispiele aufgezeigt, um den Einsatz von KNNs sowie die aktuell bekannten Grenzen zu erläutern. Dafür werden relevante Begriffe und Methoden beschrieben. Die verschiedenen Arten von neuronalen Netzen und deren Vor- sowie Nachteile werden in dieser Arbeit aufgezeigt. Das Ergebnis der 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.

Inhaltsverzeichnis

1	Einleitung (Alle)	3
2	Neuronale Netzwerke	3
3	Mustererkennung (Ravell Heerdegen)	6
3.1	Convolutional neural networks	6
3.2	Spracherkennung	7
3.2.1	Spracherkennung und convolutional neural networks . .	7
3.2.2	Spracherkennung und residual neural networks	8
3.3	Gefühlserkennung	9
4	Kosten- und Aufwandsschätzung (Robin Schramm)	11
4.1	Stand der Technik	11
4.2	COCOMO	12
4.3	Soft-Computing Ansätze	13
4.4	Leistungsbeurteilung	13
4.5	Vergleich	14
5	Softwarequalität (Konstantin Rosenberg)	14
5.1	Testabdeckung	15
5.2	Software Metriken	15
5.3	Automatisiertes Testen durch neural networks	15
5.4	Evaluation von Softwarequalität	16
5.5	Vorhersage von Softwarequalität	16
6	Fazit und Ausblick (Alle)	17

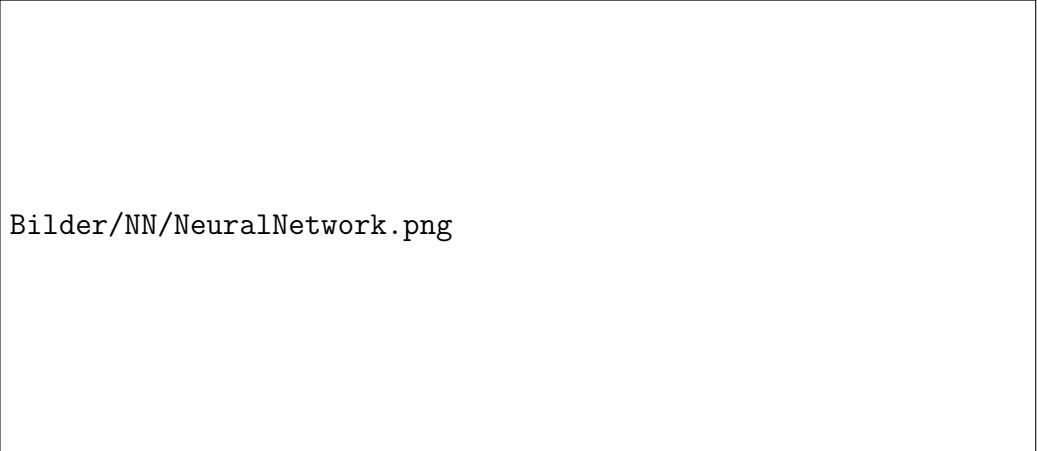
1 Einleitung (Alle)

Diese Arbeit beantwortet, ob neuronale Netze (NNs) in der Softwaretechnik (SWT) eingesetzt werden können und wo die Grenzen der Anwendungen liegen. Die Arbeit ist keine Umfrage, ob NNs in der Realität eingesetzt werden. Vielmehr werden drei Themengebiete vorgestellt, in denen Forscher NNs testeten und einsetzten. Die Auswahl wurde aufgrund von persönlichen Interessen und vorherrschender Brisanz getroffen. Unser Ziel war 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. KNNs basieren auf der Funktion des biologischen Nervensystems. Informationen werden, wie bei einem menschlichen Gehirn auch, durch eine Vielzahl miteinander verknüpfter parallel laufender Neuronen 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), Residual neural networks (Resnets) und multilayer perception (MLP)-NNs. In dieser Arbeit wird näher auf CNNs und Resnets eingegangen. CNNs werden im Kapitel Spracherkennung näher erläutert, da die aufgeführten Beispiele auf CNNs bevorzugt eingehen. Resnets gehören zur Kategorie der DNNs. Jeder Layer des Resnets stellt ein sogenanntes Res dar und besitzt mehrere trainierbare Neuronen. Neuronen sind in Bezug auf KNNs Leiter und Überträger von Daten und Werten. Das Besondere ist der Dropout, welcher durch zufälliges Ausschalten von Neuronen Überanpassungen vermeidet [2]. Mehrschichtige neuronale Netze (Multilayer perceptrons) 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 [3].

2 Neuronale Netzwerke

NNs werden verwendet, um unübersichtliche und komplizierte Probleme lösen. NNs bestehen aus Knotenelementen, den sog. Neuronen. Mehrere Neuronen bilden einen Layer. Bei herkömmlichen NNs wird zwischen drei Layer-Arten unterschieden, den Inputlayer, Hiddenlayer und Outputlayer. Inputlayer und Outputlayer sind in einem NN genau ein Mal vorhanden und dienen der Ein- und Ausgabe des NN. Zwischen diesen Schichten befinden sich

beliebig viele Hiddenlayer. Jedes Neuron besitzt eine Verbindung zu allen Neuron der Folgenden Schicht. Von der jeweils oberen Schicht werden über diese Verbindungen Signale an die darunterliegende Schicht weitergeleitet. Eine Aktivierungsfunktion entscheidet in jedem Neuron, ob die eingetroffenen Signale für eine Weiterleitung ausreichen. Auf den Verbindungen liegen unterschiedliche Gewichtungen, die die Intensität eines Signals beeinflussen. Beim Training des NN werden die Gewichtungen angepasst. Es gibt zwei grundsätzliche Trainingsansätze. Das supervised learning setzt einen Datensatz mit erwarteten Ergebnissen voraus. Nach Verarbeitung der Daten werden die Gewichte des NN anhand der Erwartungswerte angepasst. Ansätze für unsupervised learning kommen auch ohne gekennzeichnete Daten aus, sind aber derzeit noch nicht einsatzfähig. NN können durch ihre trainierbarkeit komplexe Sachverhalte verarbeiten und bieten einen Lösungsansatz für zuvor unlösbare Probleme [?]



Bilder/NN/NeuralNetwork.png

Abbildung 1: CNN-Architektur vgl. [?]

Das Hidden Markov Model (HMM) ist bekannt für seinen Nutzen im Bereich der speech recognition [2]. 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 [5]. Ein MLP-NN weist während des Trainings jedem Neuron eine Gewichtung zu, um den Einfluss zwischen Input und Output zu maximieren [6]. Radial Basis Function-NNs (RBF) werden in den Gebieten der pattern recognition und Funktionsapproximation eingesetzt. In Kapitel 3 wird die Disziplin Mustererkennung und dazugehörige Verfahren und Möglichkeiten näher beschrieben. Anschließend wird in Kapitel 4 der Einsatz von NNs im Gebiet Kosten- und Aufwandsschätzung erläutert und damit verbundene Methoden und Praktiken aufgezeigt. Kapitel 5 befasst sich mit NNs im Bereich Qualitätsmanagement 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.

Flächen. Eine Fläche besitzt mehrere unabhängige Neuronen. Eingegebene Spracheingaben werden in eine bestimmte Menge an Frequenzen unterteilt. Jede einzelne Frequenz wird mit sogenannten Gewichten verglichen, welche mit einer lokalen Variable im Layer multipliziert werden. Die Gewichte werden auf den gesamten Eingaberaum verteilt. Die Verteilung aktiviert die convolutional layers und teilt diese in verschiedene Bereiche auf. Jeder convolutional layer besitzt diverse Filter. Durch die Filter entstehen sogenannte hidden units. Hidden units sind beschreibbare Plätze auf einem Layer. Im Anschluss werden die hidden units in einen max-pooling layer gegeben. Im max-pooling layer werden Varianzen, also Abweichungen der einzelnen hidden units entfernt. Die Varianzen können u.a. aus Dialekten, undeutlicher Sprache oder Störungen in der Übertragung kommen. Die nun bereinigten hidden units werden in den convolutional layers als Trainingsparameter eingespeichert. Somit erkennt jeder convolutional layer verschiedene Merkmale, aufgrund der ihm zur Verfügung stehenden Merkmalsammlung [3].

3.2 Spracherkennung

Speech recognition (SR) 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].

3.2.1 Spracherkennung und convolutional neural networks

In [3] beschreibt Du Guiming et al. einen Versuch, bei dem SR 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) [10] eingesetzt, welcher das Frequenzspektrum kompakt darstellen kann. 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 [3]

Rafael M. Santos et al. zeigen in [4] 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 werden. Die Aufnahmen werden in einer unkontrollierten Umgebung mit einem Mobiltelefon aufgezeichnet.

Bei dem Versuch hat sich ergeben, dass die Kombination von CNN und HMM, sowohl Merkmale als auch Signale erfolgreich filtern und erkennen kann. Die Studie zeigt, dass sich die Kombination CNN-HMM am wenigsten von Lautstärke und Geräuschen beeinflussen lässt, im Vergleich zu anderen getesteten Kombinationen wie GMM [7] oder SVM [7] in Verbindung mit HMM.

3.2.2 Spracherkennung und residual neural networks

Im Versuch von H. Vydana und A. Vuppala [2] wird ein Resnet mit mehreren Res in Kombination mit SR 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, nimmt die Performanz ab. Das bedeutet, dass es eine Grenze gibt, welche sowohl die Performanz, aber auch die Anwendbarkeit eines Resnet eingrenzt. Verglichen wird das sogenannte 8Res dabei mit einem 8-DNN. Dabei hat sich herausgestellt, dass das 8Res bessere Ergebnisse erzielt als das 8-DNN. Schließlich wurde festgestellt, dass HMM-Resnets mit steigender Anzahl an Res und Layern eine

um 0,4% bessere Performanz als ein HMM-DNN [2] wie z.B. CNN mit HMM in Bezug auf die WER erreichten. Somit können Resnets DNNs in Bezug auf SR vorgezogen werden, wenn die Anzahl der Res und DNN-Layer beachtet wird.

3.3 Gefühlserkennung

Emotion recognition (ER) ist ein interdisziplinäres Forschungsgebiet, welches u.a. die Bereiche Informatik und Psychologie vereint. Das Erforschen von emotionalen Zuständen ist hilfreich, um menschliche Aktionen zu verstehen, oder um menschliche Faktoren in künstliche Systeme zu integrieren [11]. Generell werden Gesichtsausdrücke 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 ein System dem Gesicht eine Emotion zuordnen mithilfe von action units analysis [12]. Beim Arbeiten im Bereich ER 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 [13]. ER findet aber auch Anwendung in Verbindung mit Sprache, bzw. Körpersprache.

Mit ER soll die natürliche Kommunikation zwischen Mensch und Maschine hergestellt und verbessert werden. Jedoch auch Zustände eines Menschen erkannt werden, um anschließend passende Werbung oder zugeschnittene Angebote zu erstellen.

Methodisch wird zwischen statischen und dynamischen Verfahren unterschieden. Das bedeutet, einerseits mit statischen Bildern zu arbeiten, andererseits mit Sequenzen von Bildern, u.a. auch Videos [9].

Gefühlserkennung und convolutional neural networks

In [9] 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 Techniken wie z.B. Rotations- oder Farbkorrekturen 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 von L. Surace et al. beschriebene Experiment [12] 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 bottom-up Methodik eignet sich zum Isolieren von Gesichtern aus Bildern, um diese anschließend in das bereits trainierte CNN als Input einzugeben. Die top-down Methode bezeichnet sich als labeling algorithm. Beim labeling werden Bezeichnungen einer Szene zugeordnet, um den Kontext der Szene zu erfassen. Die Bezeichner werden daraufhin in ein bayesian network (BN) [12] gegeben, welches die Bezeichner verarbeitet, um Beziehungen und Abhängigkeiten zwischen den Begriffen herzustellen. Die verarbeiteten Bezeichner 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. Das bedeutet, dass eine Szene z.B. einen traurigen Kontext haben kann, wodurch eine Vorkategorisierung einer Szene durchgeführt werden kann. Dadurch kann Zeit eingespart werden, da z.B. beim Einordnen einer Szene als Traurig, eine gewisse Anzahl an Emotionen, welche positiven, fröhlichen Bezeichnern zugeordnet sind, von vornherein wegfallen. Zum Einsatz kommt außerdem die GAF Datenbank, welche eine Abbildung von 6470 Bildern von Personen in unterschiedlichen Szenen darstellt. Diese Menge wurde aufgeteilt auf die Trainings-, Validierungs- und auf die Testdaten. Zusätzlich wurde ein Dropout eingesetzt, um die Überanpassung der Neuronen zu vermeiden. Der Dropout sorgt dafür, dass teilweise shared weights übergangen bzw. gewollt verloren gehen. Letztendlich 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 des Ergebnisses

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.

4 Kosten- und Aufwandsschätzung (Robin Schramm)

In den sechziger und siebziger Jahren stieg die Größe und Komplexität von Softwareprojekten drastisch an[14]. 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. Im Zuge dieses Kapitels werden als Erstes die bestehenden SCE und SEE Techniken vorgestellt und kategorisiert. Als nächstes wird das 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 Lösungen einzugehen. Zum Schluss wird anhand der vorherigen Abschnitte beantwortet, ob NNs in der SWT im Bereich der SCE und SEE eingesetzt werden können und wo deren derzeit bekannte Grenzen liegen.

4.1 Stand der Technik

In der Planungsphase eines Softwareprojekts ist es essentiell, die Kosten und die Dauer des Projekts geschätzt zu haben. 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 wer-

den. Bei ansteigender Komplexität der Projekte 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 geschätzten nötigen Lines of Code (LOC) und zum anderen die geschätzte Anzahl von 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 lineare Systeme, die auf der Größenmessung mit FP basieren[15]. Drittens die nichtlinearen 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.

4.2 COCOMO

COCOMO wurde 1981 von Boehm entworfen, um Fehler im SPM zu verringern. COCOMO ist ein algorithmisches Modell mit einer Anzahl von Eingabeparametern abhängig von der Implementation. 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 Kostentreiber (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. Dies wirkt sich besonders im Agilen Bereich aus. Schwierigkeit und Aufwand, Projekt spezifische Daten zu sammeln, zu beurteilen und in konkreten Zahlen auszudrücken. Einige der genannten Probleme versuchen verschiedene Forscher mit Soft-Computing Ansätzen zu lösen[16][22][18][23].

4.3 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, für jedes Projekt ein neues Schätzungsmodell erstellen zu müssen, da sie sich leicht generalisieren lassen. Zudem sind 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]

4.4 Leistungsbeurteilung

Um die Modelle untereinander vergleichen zu können, gibt es verschiedene Methoden die Qualität zu messen. Nach Finnie et al[26]. 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.[6]

4.5 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]. - Besser als COCOMO[6] - Auch besser als COCOMO in NASA Testprojekten in über 90 Prozent[27]

5 Softwarequalität (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 in [28, p. 20] 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 Merkmale 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. Dieser Abschnitt stellt Verfahren vor, um die genannten Maßnahmen mit NNs umzusetzen und beantwortet die Frage nach Einsatzmöglichkeiten von NNs in der Qualitätssicherung von Softwareprodukten.

5.1 Testabdeckung

Die Testabdeckung bezeichnet in der Softwaretechnik die Relation, der durch Tests abgedeckter Ereignisse und der Menge der insgesamt möglichen Fälle. Eine komplette Testabdeckung ist praktisch nicht möglich. Darum werden zur Feststellung einer ausreichenden Testabdeckung Kriterien, z.B die Menge der abgedeckten Befehle oder die Zahl der gesicherten Methoden, angelegt [29].

5.2 Software Metriken

Softwaremetriken (SM) dienen in der Softwareentwicklung als Ansatzpunkt zur Bestimmung von Qualität. In einer SM werden Softwareeigenschaften Zahlen zugeordnet. Anhand der Zahlen lassen sich Rückschlüsse auf die Qualität der Software ziehen. Aspekte einer Softwaremetrik können z.B. die Zahl der Verzweigungen im Quellcode oder die Komplexität der Vererbungshierarchie sein [30, p. 3].

5.3 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 [31]. L. Wu et al. beschreibt in [31] ein Verfahren, um mit BPNNs automatisiert funktionale Tests zu erstellen. Bei diesem Vorgehen gibt es eine Reihe an Akteuren. Ein bestehendes System oder Programm, 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 kann wiederholt werden, bis die Testabdeckung zufriedenstellend ist. Die Methode bringt Aufwand mit sich, weshalb sie sich erst ab einer gewissen Systemgröße lohnt. Anhand eines Versuchs werden in [31] Tests für ein Softwaresystem erzeugt. Dies zeigt die Benutzbarkeit und Effizienz der Methode. In [32] beschreiben M. Negar und S. Morteza ebenfalls einen Ansatz, der die Effektivität von NNs deutlich macht. In der Methode werden zwei NNs verwendet. Eines davon dient dem white-box-testing, das andere

dem black-box-testing. Das NN für das white-box-testing kann Testfälle für ihm bekannten Code erzeugen. Das zweite NN ermöglicht das Finden von Fehlern, ohne dass es den Code kennt. Beide Varianten ergänzen sich für eine bessere Testabdeckung. Mit diesem Vorgehen lassen sich nach [32] haltbare Softwaretests produzieren, sowie Zeit und Aufwand einsparen.

5.4 Evaluation von Softwarequalität

Es bietet Vorteile den Stand von Software in Bezug auf Qualität zu kennen. Dadurch können Vorbereitungen getroffen und früh Mängel aufgedeckt und behoben werden [33]. Deshalb stellen O. Pomorova et al. in [33] eine Methode vor, um mit NNs die Softwarequalität in einer Projektphase zu evaluieren. Als Input für das NN dienen SMs. Eine SM enthält geschätzte, die andere die tatsächlichen Werte. Durch die Ausgabe des NNs wird die evaluierte Softwarequalität beschrieben oder eine Vorhersage über die zukünftige Komplexität des Projekts gemacht. In [33] wird ein Versuch beschrieben in dem vier verschiedene Softwareprojekte mit dieser Methode eingeschätzt wurden. Dabei erzielte das, auf NN basierende System, genaue Ergebnisse im Vergleich zu manuellen Testverfahren.

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.

5.5 Vorhersage von Softwarequalität

Durch eine Vorhersage der SQ können Defizite frühzeitig erkannt und verbessert werden. Mitarbeiter können sich besser auf bevorstehende Probleme vorbereiten und erhalten die Möglichkeit, Zusammenhänge zwischen getroffenen Entscheidungen und SQ zu erkennen [34]. In [34] wird deshalb ein Vorgehen beschrieben, um eine Vorhersage zu SQ in objektorientierten Systemen zu treffen. Dabei werden SQs und NNs verwendet um Vorhersagen zu machen. Die Metriken werden über ein sog. fuzzy neural network verarbeitet. Fuzzy neural networks können auch mit wenig Input genaue Ergebnisse generieren. Vorteile dieser Variante zu herkömmlichen Ansätzen zum Vorhersagen von SQ werden in einem Beispiel in [34] deutlich. Dabei zeigt sich, dass die Zusammenhänge zwischen SQ und beeinflussenden Faktoren zu erkennen sind. Außerdem kommt das Verfahren mit einer Vielzahl an Dateiformaten aus.

6 Fazit und Ausblick (Alle)

Die aufgeführten Versuche im Bereich Mustererkennung haben gezeigt, dass neuronale Netze vielerlei Anwendung in der Sprach- und Gefühlserkennung finden. Vor allem CNNs und Resnets werden in Verbindung mit anerkannten Methoden und Vorgehensweisen eingesetzt. Dabei haben CNNs im Falle der facial emotion expression recognition gegenüber Resnets bessere Ergebnisse erzielt. Aktuelle Grenzen im Bereich der Mustererkennung in Bezug auf NNs liegen momentan in der Skalierbarkeit der Schichten, in der Qualität der Trainings- und Testdaten sowie in der hohen Fehlerrate für das Erkennen von Sprachsignalen und Emotionen. Ebenfalls wurden 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. Vor allem 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, vor allem da die optimale Lösung noch nicht während der Planung bekannt ist. In Umgebungen mit wenigen Daten oder wenig Zeit sind NNs noch nicht praktikabel, da ein passendes NN zuerst entwickelt und trainiert werden muss. Die genannten Beispiele für Einsatzmöglichkeiten von NNs im Bereich Softwarequalität zeigen, dass es Anwendungsfälle gibt, in denen sich NNs als geeignet und effizient erweisen. Aufgezeigt wurde auch, wie Softwaretests automatisch generiert werden können, um Zeit und Arbeitsaufwand zu sparen. Darüber hinaus werden die Vorteile von NNs zur Vorhersage und Bestimmung von SQ offen gelegt. 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. Aus den genannten Ergebnissen kann bestätigt werden, dass NNs im Bereich der Softwareentwicklung erfolgreich eingesetzt werden können, um verschiedene Aufgaben zu lösen und Prozesse zu verbessern. Zukünftig wären neue Kombinationen oder sogar neue Arten von neuronalen Netzen möglich, um noch bessere und vor allem schnellere Ergebnisse zu erzielen, ohne immenses vorheriges Training oder Konfigurieren.

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] Vydana, H.K., Vuppala, A.K.: Residual neural networks for speech recognition. 25th European Signal Processing Conference, EUSIPCO 2017 pp. 543–547. <https://doi.org/10.23919/EUSIPCO.2017.8081266>
- [3] 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>
- [4] 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>
- [5] 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>
- [6] 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>
- [7] Jayashree, P., Melvin Jose, J., Premkumar: 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>
- [8] A. K. Jain, Robert P. W. Duin, J. Mao: Statistical Pattern Recognition: A Review. IEEE Transactions on Pattern Analysis And Machine Intelligence, Vol. 22, No. 1, January 2000 **22**(1), 4–37 (2000)
- [9] 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>

- [10] Deng, L., Yu, D.: Automatic speech recognition, Springer Verlag, vol. 9 (2015). <https://doi.org/10.1007/BF02747521>
- [11] 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>
- [12] 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>
- [13] Huang, C.: Combining convolutional neural networks for emotion recognition. 2017 IEEE MIT Undergraduate Research Technology Conference, URTC 2017 pp. 1–4 (2018). <https://doi.org/10.1109/URTC.2017.8284175>
- [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/10.1142/97898123897010014>
- [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 (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, Springer-Verlag Berlin Heidelberg (2015). <https://doi.org/10.1007/978-3-662-44028-5>, <https://link.springer.com/book/10.1007%2F978-3-662-44028-5>
- [29] Antinyan, V., Derehag, J., Sandberg, A., Staron, M.: Mythical unit test coverage. IEEE Software (3), 73–79 (2018). <https://doi.org/10.1109/MS.2017.3281318>
- [30] Committee, S.&.S.E.S., Others: IEEE Std 1061-1998-IEEE Standard for a Software Quality Metrics Methodology. IEEE Computer Society, Tech. Rep (1998)
- [31] 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>
- [32] Majma, N., Babamir, S.M.: Software test case generation & test oracle design using neural network. 22nd Iranian Conference on Electrical Engineering, ICEE 2014 pp. 1168–1173 (2014). <https://doi.org/10.1109/IranianCEE.2014.6999712>
- [33] 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>
- [34] 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>