Aus der
Medizinischen Universitätsklinik und Poliklinik Tübingen
Abteilung Innere Medizin III
(Schwerpunkt: Kardiologie und Angiologie)

Erkennung fundamentaler Strukturen eines EKG Signals mit einem TensorFlow-basierten rekurrenten neuronalen Netzwerk

Inaugural-Dissertation
zur Erlangung des Doktorgrades
der Medizin

der Medizinischen Fakultät
der Eberhard Karls Universität
zu Tübingen

vorgelegt von

Haefeker, Marc Anthony

2023

Dekan:	Prof. Dr. B. Pichler
1. Berichterstatter:	Prof. Dr. M. Gawaz
2. Berichterstatter:	Prof. Dr. B. Antkowiak
Tag der Disputation:	6. März 2023

Sophie, meiner Familie und meinen Freunden

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Acronyme

Acc Accuracy

ADAM Adaptive moment estimation

ANN artifizielles neuronales Netzwerk

AV-Knoten atrioventrikular Knoten

BD-RNN bidirektionales rekurrentes neuronales Netzwerk

CNN Convolutional Neural Network

CPU Central Processing Unit

CSV comma-separated values

DNN deep neural network

EKG Elektrokardiogramm

GDO Gradient descent optimizer

GRU Gated recurrent Unit

GPU Graphics Processing Unit

LSTM Long short-term Memory

ML Maschinelles Lernen

MSE mean squared error

ms Millisekunde

mV Millivolt

NAG Nesterov accelerated gradient

QTDB QT Database

RAM Random-Access Memory

RNN Rekurrentes neuronales Netzwerk

SCE Sigmoid cross entropy

SSD Solid State Disk

SVM Support Vector Machine

TPU Tensor Processing Unit

1 Einleitung

Die Auswertung und Interpretation der elektrischen Aktivität des Herzens ist ein inzwischen essentieller Teil der Routinediagnostik in der Medizin. Die rechnergestützte Auswertung dieses Elektrokardiogramms (EKG) stellt seit Jahrzehnten einen wichtigen Forschungszweig dar. Durch die in letzter Zeit geschaffenen Fähigkeiten der „künstlichen Intelligenz“ in Form artifizieller neuronaler Netzwerke (ANN), haben sich neue Möglichkeiten im Bereich der rechnergestützten EKG-Diagnostik aufgetan. In der vorliegenden Arbeit wird dieses Potential aufgriffen und die Erkennung der lokalen Maxima der wichtigsten Bestandteile des Elektrokardiogramm (EKG)s mit einem ANN implementiert. Die Ergebnisse werden mit ähnlichen Arbeiten und etablierten Methoden verglichen.

1.1 Das Organsystem Herz

1.1.1 Anatomie

Das Herz ist ein muskuläres Hohlorgan, welches als Pumpe einen zentralen und essentiellen Bestandteil des Blutkreislaufs darstellt. Es ist in eine rechte und eine linke Hälfte unterteilt. Jede Herzhälfte besteht jeweils aus einem Vorhof (Atrium), der das Blut aus Körper oder Lunge zunächst „sammelt“, und einer Kammer (Ventrikel), die das Blut aus dem Vorhof ansaugt und wieder in den Körper- bzw. Lungenkreislauf presst. Hierbei fließt das Blut aus den beiden großen Hohlvenen in die rechte Hälfte und wird in den kleinen Kreislauf durch die Lungen gepumpt. Die linke Herzhälfte erhält das Blut aus den Lungenvenen und befördert es in den großen Körperkreislauf. (Aumüller 2010, Kapitel 3.1.1)

Um den Blutfluss in die richtige Richtung zu lenken, sind Ventile in Form von Herzklappen jeweils am Ausgang der Atrien beziehungsweise der Ventrikel angelegt. Sie verhindern den Rückfluss aus der jeweils nachgeschalteten Struktur. Die nach ihrer Form benannten Klappen zwischen Atrien und Ventrikel werden als Segelklappen (rechts Trikuspidalklappe, links Bikuspidalklappe) bezeichnet, während sich am Ausgang der Ventrikel Taschenklappen (rechts Pulmonalklappe, links Aortenklappe) befinden. Diese vier Klappen liegen in einer räumlichen Ebene, Ventilebene genannt, welche zudem auch als elektrophysiologische Barriere zwischen den Atrien und Ventrikeln dient. (Aumüller

2010, Kapitel 3.1.3)

1.1.2 Physiologie

Das Muskelgewebe des Herzens (Myokard) lässt sich funktionell in zwei Untergruppen gliedern: das Gewebe des Arbeitsmyokards und das Gewebe des Reizleitungssystems aus spezialisierten Muskelzellen. Während das Arbeitsmyokard die Pumpleistung durch das synchrone Zusammenziehen der Muskelzellen gewährleistet, dient das Reizleitungssystem der zeitlich abgestimmten Erregung des Arbeitsmyokards. Um diese Abstimmung zu gewährleisten, sind die Muskelzellen des Reizleitungssystems auf eine schnelle Erregungsfortleitung spezialisiert. Ohne diese Reizleitungsstrukturen würde sich der Reiz langsam von Muskelzelle zu Muskelzelle ausbreiten und das Herz wäre nicht zu einer kraftvollen, synchronen Kontraktion in der Lage. (So 1978, Seite 3)

Ablauf der Herzaktion Der Herzschlag wird in zwei Abschnitte geteilt: die Austreibungsphase (Systole) und die Füllphase (Diastole). Die Systole beginnt mit der Kontraktion der Atrien, welche so das in ihnen enthaltene Blut durch die Segelklappen in die Ventrikel treiben. Nach einer kurzen Pause ziehen sich die Ventrikel zusammen und drücken so das Blut durch die Taschenklappen in die beiden Blutkreisläufe. Hierbei verhindern die Segelklappen einen Rückfluss des Blutes in die Atrien. In der Diastole fließt Blut aus den Blutkreisläufen in die Atrien, während die Taschenklappen einen Rückfluss in die Vetrikel verhindern. (Behrends et al. 2016, Kapitel 4.3)

Aufbau des Reizleitungssystems Das Steuerzentrum des Reizleitungssystems ist der Sinusknoten, welcher sich im rechten Vorhof an der Einmündung der *Vena cava superior* befindet. Beim gesunden Herzen gehen vom Sinusknoten alle Erregungen für die rhythmischen Kontraktionen des Herzens aus. Die Erregungen der Systole erfolgen mit einer Ruhefrequenz von 60 bis 80 Schlägen pro Minute. Hintergrund hierbei ist, dass im Gegensatz zur Skelettmuskulatur, welche durch Nervenfasern erregt wird, der Herzmuskel sich selbst erregt und nur durch modulierende Nervenfasern von außen beeinflusst wird. Diese initiale Erregung wird über atriale Leitungsbahnen zum atrioventrikular Knoten (AV-Knoten) geleitet. Dieser befindet sich ebenfalls im rechten Vorhof, im Bereich

der Einmündung des Sinus coronarius (venöser Rückfluss aus dem Myokards) und der Trikuspidalklappe. Der AV-Knoten stellt beim gesunden Herzen die elektrophysiologisch einzige Möglichkeit des Übertritts einer Erregung von den Vorhöfen zu den Kammern dar und hat somit eine Wächterfunktion. Zudem ist eine wichtige Aufgabe die verzögerte Fortleitung der Vorhofserregung, welche notwendig ist, um den Vorhöfen Zeit zu geben sich vollständig zu kontrahieren, bevor die Kontraktion in den Kammern stattfindet. Von der unteren Hälfte des AV-Knotens aus, auch *His*-Bündel genannt, erfolgt die Reizleitung über die beiden *Tawara*-Schenkel. Der rechte *Tawara*-Schenkel versorgt den rechten Ventrikel, der linke *Tawara*-Schenkel teilt sich und ist für die Erregung des linken Ventrikels zuständig. Alle Äste verzweigen sich schlussendlich in ein Netz aus *Purkinje*-Fasern, welche den subendokardialen Raum durchziehen. (So 1978, Seiten 3-4)

Autonomie des Herzens Wie im vorhergehenden Absatz beschrieben, ist das Herz selbstständig in der Lage Erregungen zu erzeugen und somit eine Kontraktion auszulösen. Fällt der Sinusknoten als wichtigste Struktur für diese Erregungsbildung aus, können der AV-Knoten als sogenanntes Sekundärzentrum oder sogar der Ventrikel selbst die Auslösung von Herzaktionen übernehmen und somit ein Überleben sichern. Die Erregungsbildung des AV-Knotens erfolgt jedoch mit einer verminderten Frequenz von 40-60/min, die der Ventrikel mit einer Frequenz von lediglich 20-40/min. (So 1978, Seiten 3-4)

Modulierung der Herzaktion Trotz der Autonomie des Herzens hat der Körper die Möglichkeit, auf die Herzfunktion Einfluss zu nehmen. Dies erfolgt über das vegetative Nervensystem (Sympathikus und Parasympathikus), welches insbesondere Verbindungen zum Sinusknoten und AV-Knoten besitzt. Die beiden Gegenspieler Sympathikus und Parasympathikus können sowohl die Herzfrequenz als auch die Kontraktilität, Erregbarkeit und Erregungsleitung beeinflussen. (Behrends et al. 2016, Kapitel 4.4)

1.1.3 Elektrophysiologie

Durch das elektrophysiologische Zusammenspiel der unterschiedlichen Zellen des Herzgewebes sowie die elektromechanische Kopplung (siehe unten) in den Zellen des Ar-

beitsmyokards entsteht die koordinierte Kontraktion des Herzens. Diese Vorgänge werden durch die Interaktion verschiedener Ionenkanäle gesteuert und im Folgenden erläutert.

Arbeitsmyokard Im Zellinneren einer Herzmuskelzelle und dem sie umgebenden Äußeren herrschen unterschiedliche Konzentrationen der Ionen Kalium (K^+), Natrium (Na^+) und Kalzium (Ca^2+). Diese Konzentrationsunterschiede entstehen einerseits durch das aktive Auseinanderführen von Kalium- (K^+) und Natriumionen (Na^+) durch die Ionenpumpe Na^+-K^+-ATP-ase. Dieses Protein ist unter Energieaufwand in der Lage, die Kalium- und Natriumionen entgegen ihrer Konzentrationsgradienten zu bewegen. Andererseits steuert eine Reihe von Ionenkanälen deren Rückfluss entlang des Konzentrationsgradienten der Ionen. Dieser elektrische Gradient im unerregten Zustand wird auch als Ruhemembranpotential bezeichnet und beträgt beim Arbeitsmyokard *-80 Millivolt (mV)*. (Behrends et al. 2016)

Erreicht eine elektrische Erregung von ausreichender Stärke das Arbeitsmyokard, kommt es zur Depolarisation, also zur Spannungsumkehr der erregten Zelle. Diese Spannungsumkehr liegt im Bereich der Vorhöfe für eine Dauer von ca. 150 Millisekunden (ms) vor, in dem der Ventrikel dauert sie etwa 200-400 ms. (Behrends et al. 2016)

Diese Spannungsumkehr wird als Aktionspotential bezeichnet und gliedert sich grob in drei Abschnitte, welche durch den Einfluss jeweils eines anderen Ions und seiner regulierenden Kanäle geprägt werden:

- Depolarisation: ausgelöst durch einen Na^+-Einstrom
- verzögerte Repolarisation/Plateauphase: ausgelöst durch einen temporären Ca^{2+}-Einstrom
- schnelle Repolarisation: ausgelöst durch einen K^+-Ausstrom

Der Verlauf des Aktionspotentials lässt sich damit erklären, dass unterschiedliche Spannungen die Eigenschaften der jeweiligen Ionenkanäle verändern. Außerdem wird der dargestellte zeitliche Verlauf vom Konzentrationsgefälle der beteiligten Ionen beeinflusst.

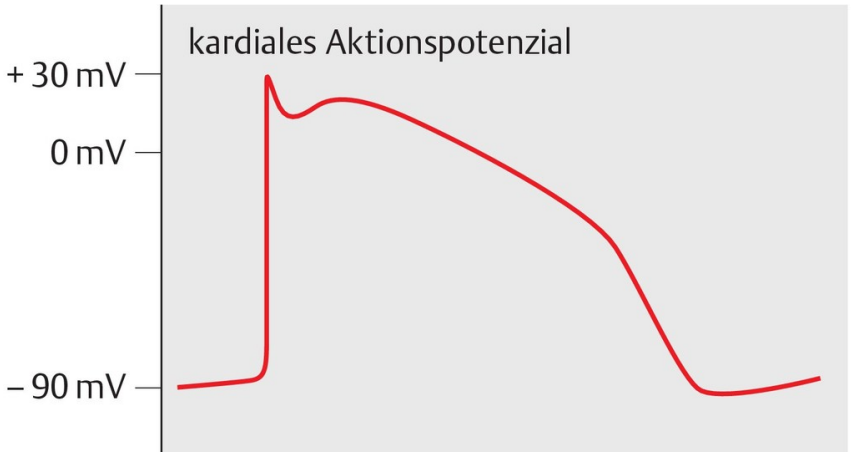


Abbildung 1: Schematischer Zeitverlauf des kardialen Aktionspotentials. Modifiziert nach Behrends et al. (2016)

Elektromechanische Kopplung Bei der Umsetzung der elektrischen Erregung in eine Muskelkontraktion - elektromechanische Kopplung genannt - spielt Kalzium eine große Rolle. Der durch die initiale Depolarisation ausgelöste Ca^{2+}-Einstrom aus dem Extrazellulärraum führt innerhalb der Zelle zu einer zusätzlichen Freisetzung von Ca^{2+} aus dem sarkoplasmatischen Retikulum, einer intrazellulären Speicherstruktur. Die dadurch rasch zunehmende Konzentration von Ca^{2+}-Ionen führt zur Bindung dieser Ionen an das Protein Troponin C. Diese Bindung wiederum löst eine Wechselwirkung der beiden Proteine Aktin und Myosin aus, welche die Herzmuskelzelle kontrahieren lassen. Das eingeströmte Ca^{2+} wird anschließend durch Ionenpumpen unter Energieaufwand aus dem Zellinneren wieder entfernt. (Behrends et al. 2016)

Reizleitungssystem Die Aufgabe des Reizleitungssystems besteht darin, die periodische Erregung des Herzens automatisch zu erzeugen und diese Erregung möglichst schnell und synchron an das Arbeitsmyokard weiterzuleiten.

Um diese beiden Aufgaben erfüllen zu können, unterscheiden sich die Zellen des Reizleitungssystems strukturell deutlich von denen des Arbeitsmyokards. Die Zellen des Reizleitungssystems weisen unter anderem weniger kontraktile Elemente und Mitochondrien (Zellorganellen, welche die Zelle mit chemischer Energie versorgen) auf. Sie sind untereinander durch eine hohe Dichte an „Gap junctions“ verbunden. So werden spezielle tunnelartige Strukturen bezeichnet, die das Zellinnere unterschiedlicher Zellen mitein-

ander verbinden und damit einen raschen Informationsfluss in Form von Ionenströmen ermöglichen.

Als zusätzliche Besonderheit sind vor allem die Zellen des Reizleitungssystems im Sinus- und AV-Knoten zur spontanen Depolarisation in der Lage. Den Grund dafür bildet - im Gegensatz zum Arbeitsmyokard - das Fehlen stabilisierender Ionenströme, ein geringeres Ruhemembranpotential von lediglich -60 mV und das Vorliegen von „funny Kanälen“, die den „funny Strom“ I_f verursachen. Diese Kombination aus Instabilität und niedrigerem Ruhemembranpotential macht das Auftreten spontaner Aktivitäten beziehungsweise Aktionspotentialen möglich.

Die Wahrscheinlichkeit des Auftretens dieser spontanen Depolarisationen wird von der Vor- und Nachlast des Herzens (Frank-Starling-Mechanismus) und dem Einfluss der Neurotransmitter Noradrenalin und Acetylcholin des vegetativen Nervensystems, welches Teile des Herzens erreicht, beeinflusst. (Behrends et al. 2016)

1.2 Das Elektrokardiogramm

Die elektrophysiologischen Vorgänge des Herzens erzeugen auf der Körperoberfläche des Menschen mess- und ableitbare elektrische Felder, die im EKG aufgezeichnet und zu diagnostischen Zwecken herangezogen werden können.

Die erste Anwendung eines EKGs beim Menschen fand durch *Augustus Waller* in den 1880er Jahren statt. Diese Pionierleistungen wurden im frühen 20. Jahrhundert durch *Willem Einthoven* fortgesetzt, als er zum einen die technischen Aspekte des EKGs weiterentwickelte und zum anderen die noch heute gebräuchlichen Platzierungen der Messelektroden und Bezeichungen der Signalabschnitte definierte. Für diese Arbeit erhielt Einthoven 1924 den Nobelpreis für Medizin. Seither wurde das EKG vielfach weiterentwickelt und stellt heute eine Selbstverständlichkeit im klinischen Alltag dar. (Sörnmo und Laguna 2005)

1.2.1 Physikalische Grundlagen

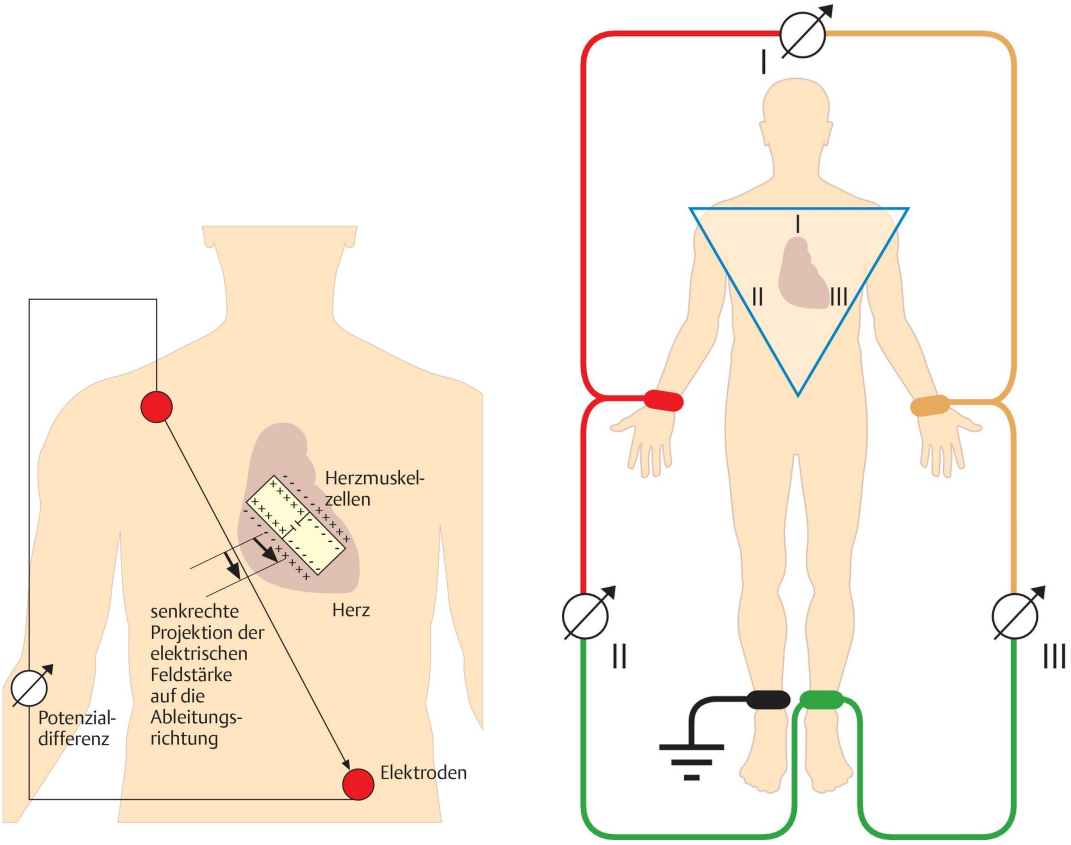
Die Voraussetzung für die in den vorherigen Abschnitten beschriebenen Aktionspotentiale sind Ladungsdifferenzen zwischen dem Zellinneren und seiner Umgebung. Diese Vorgänge haben Auswirkungen, welche auch außerhalb der Zelle an der Körperoberfläche des Patienten messbar sind.

Hierbei spielt das sogenannte Membranpotential eine entscheidende Rolle: es stellt die an der Oberfläche der Zelle vorherrschende Ladung dar. Wechselt nun eine Zelle im Rahmen der Depolarisation zügig ihr Membranpotential, fehlen lokal Na^{+}-Ionen und die Zelle erscheint im Vergleich zu ihrer Umgebung negativ. Es entsteht ein elektrischer Dipol, welcher ein auf der Körperoberfläche messbares elektrisches Feld erzeugt. Wie in Abbildung Abbildung 2a auf Seite 8 zu sehen, ist dieses Feld in Form eines Vektors darstellbar. Dieser zeigt von der negativ geladenen Zelle hin zur ungeladenen/positiven Zelle. Das EKG misst und bildet die Auswirkung dieser elektrischen Felder relativ des Vektors zwischen zweier miteinander verschalteten Elektroden ab. (Behrends et al. 2016)

Bei der Durchführung des EKGs erfolgt die Anordnung der Elektroden meistens gemäß der von Einthoven definierten Konvention. Danach wird jeweils eine Elektrode an beiden Armen und Beinen platziert, was als Extremitätenableitung bezeichnet wird. Je nachdem welche Elektroden miteinander verschaltet werden, bilden sich unterschiedliche Vektoren relativ zur elektrischen Aktivität des Herzens aus. Das ist gleichzusetzen mit einer Betrachtung aus verschiedenen räumlichen Perspektiven. Dabei kommt die Ableitung II der anatomischen Herzachse am nächsten. Über die Jahre haben sich eine Vielzahl weiterer Ableitungspositionen entwickelt, die eine größere Detailschärfe bieten. Hier sind zum Beispiel die sechs Brustwandableitungen *V1-V6* nach *Wilson* zu nennen. (So 1978)

1.2.2 Aussehen und Interpretation eines EKG-Signals

Der Grundzustand des EKG-Signals ist die „isoelektrischen Linie“, auch „Nulllinie“ genannt. Sie stellt den horizontal flach verlaufenden Teil des Signals dar, in dem keine Potentialveränderungen messbar sind. Im weiteren Verlauf spiegelt sich die elektrische Aktivität des Herzens wider als eine charakteristische Folge auf- und absteigender Wellen und



(a) Darstellung der bioelektrischen Grundlagen eines EKGs.

(b) Platzierung der Elektroden und Verschaltung der Ableitungen nach *Einthoven*.

Abbildung 2: Schematische Darstellungen der bioelektrischen Grundlagen eines EKGs und der Ableitungen nach Einthoven. Modifiziert nach Behrends et al. (2016)

Zacken. Diese Bereiche werden als „Intervalle“ bezeichnet, während die isoelektrischen Bereiche dazwischen „Strecken“ genannt werden. Nach *Einthoven* erhalten die charakteristischen Signalanteile alphabetische Bezeichnungen, beginnend mit dem Buchstaben „P“. (Behrends et al. 2016; So 1978)

Charakteristisches Aussehen des EKG-Signals eines gesunden Menschen Das EKG-Signal beginnt mit der Depolarisation der Vorhöfe, welche sich als P-Welle hervorhebt. Im Anschluss an die P-Welle ist das Signal für eine kurze Zeit wieder isoelektrisch - dies spiegelt die Verzögerung im AV-Knoten wider. Gefolgt wird dieser Abschnitt von einer Serie schnell ab- und zunehmender Zacken, wobei die erste negative als Q-Zacke und die erste positive als R-Zacke bezeichnet wird. Der R-Zacke folgt in der Regel eine weitere negative S-Zacke. Diese Gesamtheit wird als QRS-Komplex bezeichnet und

stellt die Depolarisation der Ventrikel dar. Nach einer erneuten kurzen isoelektrischen Phase zeigt sich eine weitere Welle, welche als T-Welle bezeichnet und durch die Repolarisation des Ventrikelmyokards hervorgerufen wird. Ein auf diese Weise konfigurierter und regelmäßig auftretender Herzschlag wird „Sinusrhythmus“ genannt. (Behrends et al. 2016)

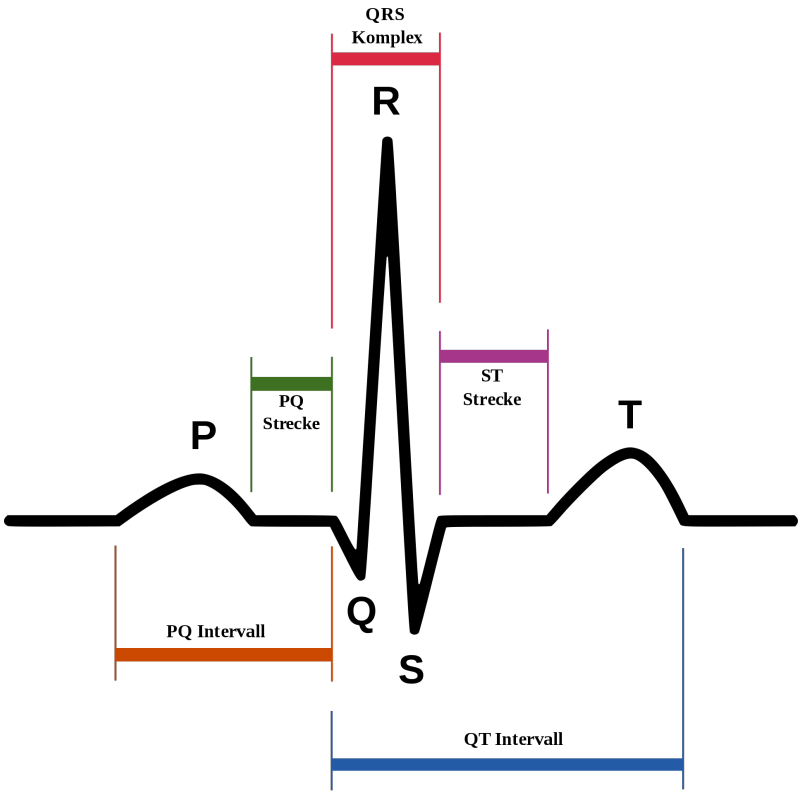


Abbildung 3: Darstellung eines gesunden EKG Signals. Modifiziert nach Atkielski (2007)

Interpretation eines EKGs Aus den gemessenen Kurven lassen sich eine Vielzahl von Parametern bestimmen, die bei der Beurteilung des Zustandes des untersuchten Herzens hilfreich sind.

So lässt sich beispielsweise aus der Kombination der Ableitungen nach *Einthoven* der elektrische Hauptvektor des Herzens berechnen. Hierbei handelt es sich um den „Lagetyp“. Eine starke Abweichung dieses Hauptvektors von der Norm, kann auf eine akute oder chronische Belastungssituation des Herzens hinweisen. Zusätzlich wird auch die Konfiguration der einzelnen Wellen und Zacken untersucht. Eine überhöhte T-Welle gilt als Anzeichen für eine Hyperkaliämie, eine Erhöhung der Strecke zwischen S-Zacke und T-Welle als Anzeichen für einen Herzinfarkt. Darüber hinaus lassen sich auch durch die

Analyse der zeitlichen Abstände zwischen den Zacken und Wellen wichtige Informationen gewinnen. Ein geschädigter AV-Knoten kann auf diese Weise an der Länge des PQ-Intervalls erkannt werden. (So 1978)

Die am häufigsten genutzte Information ist jedoch die Berechnung der Herzfrequenz. Sie ergibt sich aus der Anzahl der QRS-Komplexe pro Minute. Mithilfe der Herzfrequenz kann zum einen eine Aussage zum aktuellen Zustand des Herz-Kreislaufsystems getroffen, aber auch eine gefährliche Rhythmusstörung diagnostiziert werden. Die Betrachtung des Verlaufs der Herzfrequenz über einen längeren Zeitraum gestattet sogar Einblicke in den Zustand des vegetativen Nervensystems, da dessen Einfluss sich in einer gewissen Varianz der Ruheherzfrequenz nachweisen lässt. (Boehm et al. 2018)

1.2.3 Computergestützte EKG-Analyse

Das EKG war eines der ersten Anwendungsgebiete für die Einführung der computergestützten Signalverarbeitung in den medizinischen Sektor. In der Anfangszeit wurden mit IT-basierten Lösungen hauptsächlich Entscheidungsbäume erstellt, mit denen die Interpretation des EKG-Sinals durch einen Kardiologen ersetzt werden sollten. Dabei wurde schnell klar, dass die Signalverarbeitung selbst der Schlüssel zu genaueren Ergebnissen ist. Insbesondere die exakte Erkennung des QRS-Komplexes ist von großer Bedeutung, da viele nachfolgende Analysen davon abhängen. (Sörnmo und Laguna 2005)

Pan und Tompkins Algorithmus Eine häufig verwendete Methode zur Erkennung von QRS-Komplexen wurde 1985 von Pan und Tompkins eingeführt. Sie implementiert einen zweistufigen Ansatz zur Interpretation des EKG-Signals. Zunächst wird durch Vorverarbeitung mit einer Vielzahl von Filtern das Rauschen reduziert, um auf diese Weise niedrigere Schwellenwerte für die QRS-Erkennung zu ermöglichen. Anschließend folgt eine Serie aus Entscheidungsregeln, um die Zuverlässigkeit der QRS-Erkennung zu steigern.

Im Vorverarbeitungsschritt wird zunächst ein Bandpass-Filter eingesetzt, der nur Signalanteile in einem Frequenzband von 5 - 15 Hz zulässt und den übrigen Anteil dämpft. Diese Dämpfung soll störende Einflüsse minimieren. Dazu gehören muskuläre Aktivitäten, die

Netzspannung und weitere Artefakte, die beispielsweise durch die Atmung des Patienten verursacht werden. Anschließend wird mithilfe einer Ableitungsfunktion die Steigung des Signals über fünf Datenpunkte berechnet. Durch eine anschließende Quadrierung erreicht man, dass sämtliche Werte positiv und hochfrequente Signale (welche zu diesem Zeitpunkt am ehesten vom tatsächlichen EKG abstammen) verstärkt werden. Die Einstellungen beider Filter werden variabel implementiert, damit der Algorithmus sich anhand logischer Grenzen dem aktuellen Signal anpassen kann.

Auf diesen Vorverarbeitungsschritt folgt der Entscheidungsblock, der nach Signalspitzen innerhalb variabler Schwellenwerte sucht und eine auf physiologischen Regeln basierende Logik anwendet, um die falsch-positive Erkennungsrate weiter zu reduzieren und gegebenenfalls zunächst übersehene QRS-Komplexe doch noch zu erkennen.

Diese Arbeit erreichte bereits damals eine richtig-positive Erkennungsrate von QRS-Komplexen von 99,325% und war in der Lage, diese Leistung auch in Echtzeit zu liefern. (Pan und Tompkins 1985)

1.3 Maschinelles Lernen

Während Algorithmen wie der von Pan und Tompkins hervorragende Leistungen hervorbringen, bedürfen sie vorher einer langwierigen Prozedur manueller Entwicklung.

Mit dem fortwährenden Fortschritt der Computertechnologie standen immer mehr Ressourcen zur Automatisierung der Entwicklung solcher Algorithmen zur Verfügung. Ziel ist es, sich selbst trainierende Algorithmen zu entwickeln, welche aufgrund von Erfahrung in der Lage sind intelligente Entscheidungen treffen zu können. Dieser Bereich der Informatik wird unter dem Begriff „Maschinelles Lernen (ML)“ zusammengefasst. (Mitchell 1997)

Hier spielen die Art und Weise des Trainings eine entscheidende Rolle. Diese hängt von der zu Grunde liegenden Aufgabe ab:

- **supervised learning** (Englisch für beaufsichtigtes Lernen) eignet sich am besten für Klassifizierende Aufgaben. Es werden also Datensätze zum Trainieren verwen-

det, bei denen die Beispiele bereits bekannten Rollen/Klassen zugeordnet sind und der Algorithmus deren Zuteilung erlernen soll. (IBM Cloud Education 2020b)

- **unsupervised learning** (Englisch für unbeaufsichtigtes Lernen) wird verwendet um Zusammenhänge (z.b. Cluster oder Hierarchien) in großen unsortierten Datensätzen zu finden und benötigt daher keine vordefinierten Klassen. (IBM Cloud Education 2020a)

- **self-supervised learning** (Englisch für selbst-beaufsichtigtes Lernen) ist eine Fortentwicklung des Supervised Learnings. Es adressiert dessen größtes Nachteil: dem manuellen Erstellen der Trainingsdatensätze und der Definition der zu erlernenden Klassen. Dieser Prozess erfordert klassischerweise ebenfalls viele Arbeitsstunden und kann insbesondere bei Text-bezogenen Daten durch verschiedene Herangehensweisen automatisiert werden. (Leszczynski 2020)

Die Erkennung von Wellen in einem EKG kann hierbei als typische Form eines Klassifizierungsproblems betrachtet werden - es geht darum Signalabschnitte vordefinierten Wellen oder Strecken zuzuordnen. Dementsprechend müssen bei der Entwicklung eines ML-basierten Verfahrens zur EKG-Wellen-Erkennung Verfahren mit „supervised learning“ zum Einsatz kommen.

Hierfür kommen eine Vielzahl von Methoden in Frage, wovon einige populäre hier aufgelistet seien (IBM Cloud Education 2020b):

- **Logistische Regression** versucht bei kategorischen Daten einen Zusammenhang zwischen einem Eingabewert und einer Kategoriezugehörigkeit über eine das Verhältnis abbildende Variable zu entscheiden. Dies kann auch bei einer Vielzahl von Eingabeparametern, im Sinne einer multiplen logistischen Regression, erfolgen. Logistische Regression wird bei binären Fragestellungen wie Gut/Böse oder Richtig/Falsch eingesetzt.

- **Random forest** beschreibt eine Klassifizierungsmethode welche aus einer Vielzahl an Entscheidungsbäumen besteht. Diese werden am Ende zu einer gemeinsamen Entscheidungsinstanz gebündelt, um zusammen eine bessere Entscheidung treffen

zu können.

- **Support Vector Machine (SVM)** (zu Deutsch Entscheidungsverktormaschine) sind Klassifizierungsalgorithmen, die versuchen die verschiedenen Klassen durch eine Hyperebene (e.g. den Entscheidungsvektor) optimal voneinander zu trennen. Dieser Vektor stellt somit die Entscheidungsgrenze zwischen den verschiedenen Kategorien dar.

- **artifizieller neuronaler Netzwerke (ANN)** sind Algorithmen, welche das biologische Vorbild eines Nervensystems - mehrere Schichten miteinander vernetzter Zellen - nachzuahmen. Die Zellen haben einen oder mehrere Eingangskanäle, verarbeiten die Informationen und leiten diese gegebenenfalls mittels einer Aktivierungsfunktion an die nächste Neuronenschicht weiter. Die letzte Schicht übernimmt dann die Aufgabe der Klassifikation.

Zum Zeitpunkt der Konzeption der Arbeit waren ANN ein noch spärlich untersuchtes Feld im Bereich der EKG-Signalanalyse, während Methoden die beispielsweise SVM (Salam und Srilakshmi 2015) oder logistischer Regression (Dora und Biswal 2016) einsetzen bereits existierten. Aus diesem Grunde wurde beschlossen, diese Arbeit der Untersuchung der Einsatzfähigkeit von ANN bei der Identifizierung von charakteristischen Wellen im EKG zu untersuchen.

1.4 Artifizielle neuronale Netzwerke

In den vergangenen zwei Jahrzehnten wurde das Feld der Mustererkennung durch den immer vielseitigeren Einsatz artifizieller neuronaler Netzwerke (ANN) revolutioniert. Die Leistungsfähigkeit der dafür genutzten Systeme nimmt stetig zu, so dass mit den ausgeklügelten Algorithmen hochkomplexe Aufgaben gelöst werden können. Dazu zählen beispielsweise die Bilderkennung, Übersetzungen oder autonomes Fahren. Dies alles funktioniert ohne die Notwendigkeit, einen allgemeingültigen Algorithmus dafür zu entwickeln. (Schmidhuber 2015; Lecun, Bengio et al. 2015) Im Kern bilden ANN Mustererkennungsgraphen: Nach der Betrachtung ausgewählter Trainingsdaten, sind diese in der Lage Zusammenhänge in großen Datenmengen mit hoher Effizienz zu erkennen. Bei der Analy-

se von medizinischen Daten, wie zum Beispiel Biosignalen oder bildgebenden Verfahren handelt es sich ebenfalls um Mustererkennungsprobleme. Deshalb kann davon ausgegangen werden, dass ANN vergleichbare Erkennungsraten mit größerer Effizienz erreichen können.

1.4.1 Geschichte

Die Anfänge von ANN finden sich in der Mitte des 20. Jahrhunderts, als die Wissenschaft eine Reihe wegweisender Entdeckungen machte. So wurde von Neurologen die Funktionsweise des Gehirns als Geflecht unzähliger miteinander verbundener Nervenzellen erkannt. Gleichzeitig wurde im Bereich der Informatik durch *Turing*, *Shannon* und *Wiener* die theoretische Möglichkeit bewiesen, Gedankenprozesse durch digitale Computerarchitekturen abzubilden. Dadurch entstand das Konzept eines „elektronischen Gehirns“. (Flasiński 2016; Loiseau 2019) Diese parallele Entwicklung und Nähe zu den biologischen Erkenntnissen und Vorbildern lässt sich auch heute noch in der Fachsprache der ANN wiederfinden.

Trotz einiger temporärer Rückschläge setzte sich das 1958 von *Frank Rosenblatt* vorgestellte „Perzeptron“(aus dem Englischen „perception“- Wahrnehmung) als einer der erfolgreichsten Ansätze im Bereich der ANN durch.

Das Ziel war ein System, welches in der Lage sein sollte, mit Umweltreizen umzugehen ohne die Notwendigkeit vorhergehender menschlicher Informationsaufbereitung. In starker Anlehnung an das biologische Vorbild schlugen Rosenblatt und seine Kollegen ein System vor, in dem eine mathematische Funktion das Verhalten eines Neurons nachahmt. Diese Funktion kombiniert einen Eingabewert mit verstellbaren Gewichtungen und einen definierten Schwellenwert zu einem Ausgabewert. Durch die Kombination mehrerer dieser Neuronen zu einem Netzwerk, auch „Perceptron“ genannt, wurde das Lösen des Problems wahrscheinlicher.

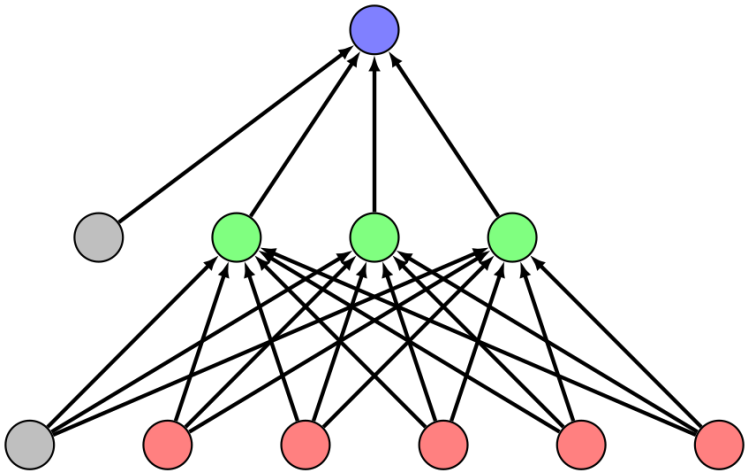


Abbildung 4: Schematische Darstellung eines zweilagigen Perzeptrons. Modifiziert nach Thoma (2013)

Dank der Rückführung des Fehlers im Training zur Modifikation der verschiedenen Gewichtungen und Schwellwerte, konnte in einem iterativen Prozess die Leistung zunehmend verbessert werden. (Rosenblatt 1958)

Der Ansatz mehrerer miteinander verbundener Perzeptrone und das Konzept der Fehlerrückführung wurden in den 1980er Jahren durch *David Rumelhart* und *Geoffrey Hinton* popularisiert. (D. E. Rumelhart und MacClelland 1986; D. Rumelhart et al. 1986) Dank der stetig zunehmenden Rechenkapazität bei konstant bleibenden Kosten gemäß *Moore's Gesetz* (Moore 1965) wurde die nötige Rechenkapazität für ANN erschwinglicher und damit auf breiterer Ebene einsetzbar.

Durch die zunehmende Verfügbarkeit ausreichender Rechenkapazität wurden ab Ende der 1990er Jahre immer komplexere Netzwerke mit deutlich mehr Schichten - auch tiefe Netzwerke genannt - möglich. Dieser Trend wurde zudem durch weitere Fortschritte in der Netzwerkarchitektur beflügelt.

1.4.2 Formen und Funktionsweise

Eine sehr populäre Entwicklung im Feld der ANNs stellt das Convolutional Neural Network (CNN) (aus dem Englischen „convolution" = Faltung) dar.

Das CNN adressiert eines der Kernprobleme eines wie ursprünglich von *Rosenblatt* beschriebenen vielschichtigen Perzeptrons: das „Overfitting“ (= Überanpassung). Da in einem vielschichtigen Perzeptron jedes Neuron mit allen Neuronen der folgenden Schicht verbunden ist, entsteht schnell eine extrem hohe Anzahl trainierbarer Parameter. Ist diese Zahl zu groß, wird das Netzwerk einerseits zu komplex um eine effiziente Lösung des Problems zu finden und stößt andererseits schnell an Limitationen bezüglich der verwendbaren Hardware.

Die Entwickler des CNNs orientierten sich daher auch in diesem Fall an der Natur und versuchten unter anderem die Struktur des visuellen Cortex (= Sehrinde) bei Tieren nachzuahmen. Dieser Anteil des Gehirns funktioniert durch eine hierarchische Aufteilung der Aufgabe auf einen vielschichtigen Verbund, bei dem die Zahl der Neuronen pro Schicht stetig abnimmt. Zusätzlich werden die ersten Schichten räumlich in getrennte rezeptive Felder unterteilt.

Jedes Neuron der ersten Zwischenschicht erhält nur einen kleinen Ausschnitt des Gesamtbildes, während die darauf folgende Schicht erneut nur mit einem Teil aller vorhergehenden Neuronen verbunden ist. Durch diese Aufteilung können die einzelnen Schichten spezifische Aufgaben erlernen. Gleichzeitig können mit zunehmender Schichtdicke komplexere Zusammenhänge und Strukturen erkannt werden.

Um den Einfluss von Fehlern eines einzelnen Abschnittes an der Gesamtleistung zu verringern, werden die initialen Ausschnitte der Daten überlappend angelegt, um eine Streuung des Problems über mehrere Kanäle zu erreichen.

Die Verjüngung zur jeweils nächsten Schicht erfolgt durch das Aktivieren eines oder mehrerer trainierbarer Gewichtsmatrizen (auch Filter genannt), die auf die Eingabewerte angewendet werden und das jeweilige Skalarprodukt der Eingabewerte und des Filters bilden.

Jede Schicht eines CNNs verwendet an jeder Stelle die identischen Filter, wodurch zusätzliche Robustheit gegenüber Fehlern durch Rauschen, Verzerrung oder Fokusverschiebung gewonnen wird. (Lecun, Bottou et al. 1998)

Deshalb benötigen Neuronale Netzwerke dieser Architektur nur sehr wenig Signalvor-

verarbeitung im Vergleich zu herkömmlichen Ansätzen. CNN finden in vielen Feldern Verwendung, beispielsweise in der Bild- (Krizhevsky et al. 2017) und der Erdbebenerkennung (Perol et al. 2018), aber auch in der Medizin bei der Klassifizierung von Herzrhythmen. (Acharya et al. 2018; Swapna et al. 2018; Oh et al. 2018)

Ein großer Nachteil eines solchen Netzwerkes ist sein Mangel an zeitlichem Bewusstsein. Das bedeutet, dass ein CNN nicht in der Lage ist, sich auf vorher oder in der Zukunft getroffene Vorhersagen zu beziehen. Dieser Sachverhalt kann bei Daten, die nur in einer strikten Reihenfolge vorkommen dürfen, von Nachteil sein. (Nielsen 2015, Kapitel 6) Davon betroffen sind beispielsweise die Sprache oder diverse Biosignale.

1.4.3 Rekurrente neuronale Netzwerke

Wie in den vorherigen Abschnitten beschrieben, stellt der Mangel an zeitlichem Bewusstsein einen der wichtigsten Nachteile von CNNs bei Signalen oder Daten mit einer strikten Reihenfolge dar.

Im Gegensatz hierzu sind Rekurrente neuronale Netzwerke (RNN) in der Lage, sich auf andere Zeitzustände zu beziehen, das heißt sich zu „erinnern“. Diese Information kann so in die aktuelle Vorhersage einbezogen werden, was für die im vorhergehenden Absatz erwähnten Anwendungen einen großen Vorteil bietet.

In seiner Urform wurde das RNN 1997 von *Hochreiter* und *Schmidhuber* beschrieben. In dieser Form löste es durch die Möglichkeit der Fehlerrückführung über die Zeit (aus dem Englischen - „backpropagation through time“) eines der Hauptprobleme neuronaler Netzwerke im Kontext von sequentiellen Daten. Dieses Problem tritt beim Trainingsprozess auf, wenn der Optimierungsalgorithmus versucht, die Fehlerrückführung über viele Zeitstufen hinweg anzuwenden. Beispielsweise ist bei der Bilderkennung mittels CNN die Fehlerrückführung problemlos anwendbar, da die einzelnen Trainingsbeispiele nicht miteinander in Verbindung stehen. Dagegen stellt sich bei sequentiellen Daten die Frage, über wie viele Zeitschritte hinweg eine Fehlerrückführung erfolgen muss. Wird die optimale Anzahl hierbei überschritten, wächst die Menge an verstellbaren Parametern schnell ins Unermessliche und damit kommt das Trainieren zum Sistieren oder wird sehr zeitauf-

wendig.

„Hochreiter“ und „Schmidhuber“ lösten das Problem mit der Entwicklung des Long short-term Memory (LSTM)-Neurons. Hierbei handelt es sich um ein Neuron, welches im der Lage ist sich auf seine Zustände in der Vergangenheit und in der Zukunft zu beziehen. Das Ausmaß dieser Verbindungen wird via trainierbarer Gates (englisch für Schranke) regelt. Diese trainierbaren Gates erlauben es dem Netzwerk automatisch zu lernen, über wie viele und zu welchen Zeitzuständen es Verbindungen halten soll und zu welchen nicht.

Damit konnte der Prozess der Fehlerrückführung nun auch erfolgreich auf den Bereich sequentieller Daten ausgeweitet werden. (Hochreiter und Schmidhuber 1997)

Mathematische Grundlagen eines RNN Insgesamt verfügt ein LSTM-Neuron über drei Gates und ein „Zellinneres“. Die drei Gates regeln den Informationsfluss zu den anderen Zuständen sowie das Vergessen nicht mehr benötigter Werte:

- Input Gate: kontrolliert die Menge neuer Eingabewerte, die in das Neuron aufgenommen werden
- Output Gate: kontrolliert das Ausmaß, in dem die ermittelten Werte anderen zeitlichen Zuständen des Neurons zur Verfügung stehen sollen
- Forget Gate: kontrolliert, welche Werte in der Zelle gespeichert und welche vergessen werden.

Das Forget Gate wurde nachträglich durch Gers et al. (1999) etabliert, stellt aber inzwischen den Standard bei LSTM-Neuronen dar.

Jede Schranke enthält drei trainierbare Gewichtsmatrizen *W, U, V* und einen Bias-Vektor *b* (englisch für Tendenz/Neigung). Die Gewichtsmatrize *W* wird mit dem aktuellen Eingabevektor x_t multipliziert, während *U* mit dem Ausgangsvektor $h_{t\text{-}1}$ des vorherigen Zeitzustandes multipliziert wird. Aus *V* und dem Wert des Zellinneren $c_{t\text{-}1}$ wird das Hadamard Produkt berechnet. (Hochreiter und Schmidhuber 1997)

Das Hadamard Produkt stellt eine spezielle Unterform der Matrixmultiplikation dar, bei

der zwei Matrizen identischer Dimensionierung miteinander multipliziert werden. Somit lassen sich die jeweiligen Komponenten exakt miteinander multiplizieren. (Horn und Johnson 2017)

Über die Summe aller vorherigen Berechnungen und dem Bias-Vektor b wird zuletzt eine Sigmoidfunktion gelegt, welche die Werte des Gates nichtlinear auf eine Spanne zwischen 0 und 1 verteilt.

Folglich ergeben sich für die drei Gates folgende Gleichungen:

$$\text{Input Gate: } i_t = \sigma(W_i * x_t + U_i * h_{\text{t-1}} + V_i \circ c_{\text{t-1}} + b_i) \tag{1}$$

$$\text{Output Gate: } o_t = \sigma(W_o * x_t + U_o * h_{\text{t-1}} + V_o \circ c_{\text{t-1}} + b_o) \tag{2}$$

$$\text{Forget Gate: } f_t = \sigma(W_f * x_t + U_f * h_{\text{t-1}} + V_f \circ c_{\text{t-1}} + b_f) \tag{3}$$

Das Zellinnere verfügt über zwei Gewichtsmatrizen W_c, U_c und einen Bias Vektor b_c. Diese werden zusammen mit den aktuellen Trainingsdaten und den Werten der Input- und Forget Gates kombiniert und ergeben sozusagen die neue „Aktivität“ des Neurons. Es gilt hierfür folgende Formel:

$$c_t = f_t \circ c_{\text{t-1}} + i_t \circ \sigma_c(W_c * x_t + U_c * h_{\text{t-1}} + b_c) \tag{4}$$

Die „Aktivierung“ des Zellinneren stellt jedoch nicht die endgültige Ausgabe des Neurons dar. Der Ausgabevektor h_t des Neurons ergibt sich aus dem Hadamard Produkt des Vektors des Output Gates o_t und dem hyperbolen Tangens σ_h des Wertes des Zellinneren c_t. (Hochreiter und Schmidhuber 1997) Die Formel lautet demzufolge:

$$h_t = o_t \circ \sigma_h(c_t) \tag{5}$$

Damit lässt sich festhalten, dass das LSTM-Neuron insgesamt drei Eingabewerte (Wert des Zellinneren zum vorherigen Zeitpunkt $c_{\text{t-1}}$, Ausgabevektor zum vorherigen Zeitpunkt

$h_{\text{t-1}}$ und den aktuellen Eingabevektor x_t) erhält und zwei Ausgabewerte (Ausgabevektor h_t und den aktuellen Wert des Zellinnneren c_t) erzeugt.

Aus den genannten Formeln kann die schematische Abbildung 5 auf Seite 21 eines LSTM-Neurons zur Verdeutlichung des Informationsflusses durch ein Neuron abgeleitet werden.

Gated recurrent Units Eine Weiterentwicklung des LSTM-Neurons stellt das Gated recurrent Unit (GRU) dar. Es vereinfacht die Zahl der verwendeten Gates auf zwei, indem das Input und das Output Gate zu einem Update Gate vereint werden. Für beide Gates gelten analog zum LSTM folgende Berechnungswege:

$$\text{Update Gate: } u_t = \sigma(W_u * x_t + U_u * h_{\text{t-1}} + V_u \circ c_{\text{t-1}} + b_u) \tag{6}$$

$$\text{Forget Gate: } f_t = \sigma(W_f * x_t + U_f * h_{\text{t-1}} + V_f \circ c_{\text{t-1}} + b_f) \tag{7}$$

Das Gated recurrent Unit (GRU)-Neuron liefert nicht zwei, sondern nur einen Ausgabevektor. Das ist ein weiterer Vorteil gegenüber dem LSTM, da somit weniger Rechenschritte notwendig sind. Dies erreicht das GRU-Neuron, indem es über das Update Gate die relevanten Informationen aus den vorherigen Zeitzuständen in den aktuellen Zeitpunkt integriert. Das Innere des Neurons selbst besteht wie das LSTM-Neuron aus zwei Gewichtsmatrizen W_h, U_h und einen Bias Vektor b_h.

Es ergibt sich folgende Formel zur Berechnung des Ausgangsvektors h_t eines GRU-Neurons:

$$h_t = u_t \circ h_{\text{t-1}} + (1 - u_t) \circ \sigma_h(W_h * x_t + U_h * (f_t \circ h_{\text{t-1}}) + b_h) \tag{8}$$

Eine schematische Darstellung eines GRU-Neurons ist in der Abbildung 6 auf Seite 21 abgebildet.

Mit dieser Weiterentwicklung wurden in einigen Beispielen vergleichbarer Ergebnisse bei geringerem Ressourcenaufwand erzielt. (Cho et al. 2014)

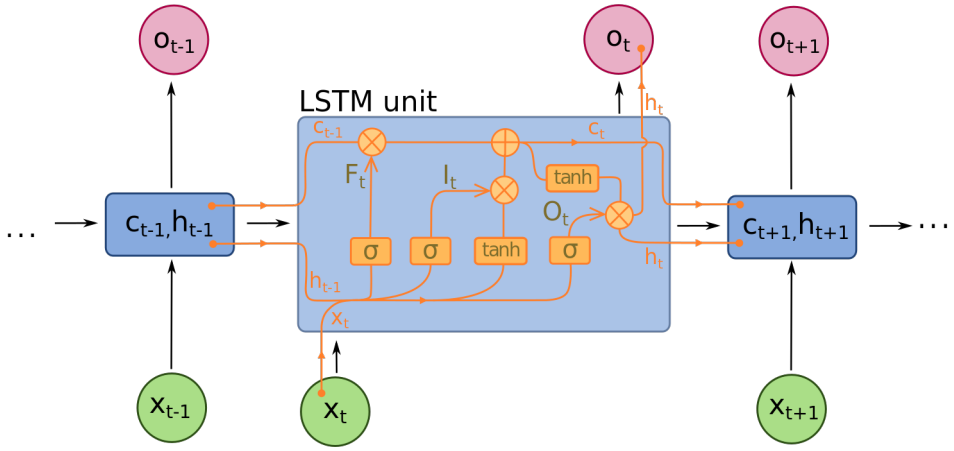


Abbildung 5: Vereinfachte schematische Darstellung eines LSTMs-Neurons. Entnommen aus Ixnay (2017a)

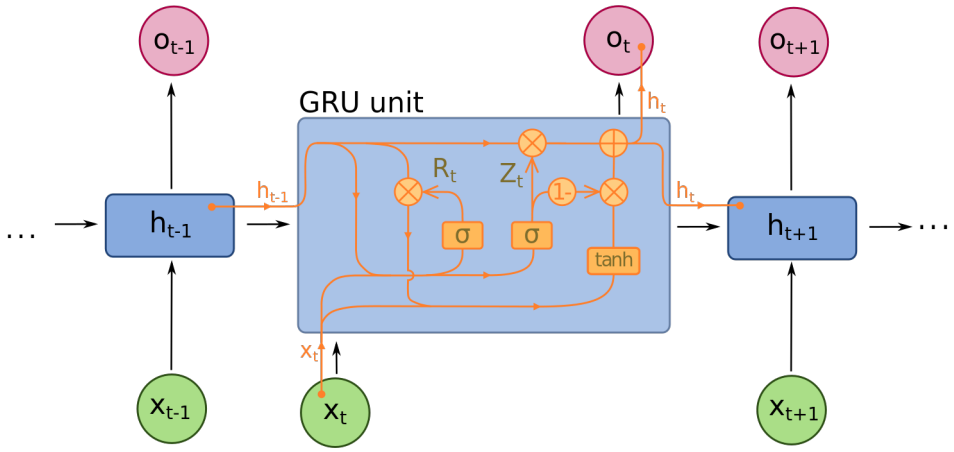


Abbildung 6: Vereinfachte schematische Darstellung eines GRU-Neurons. Entnommen aus Ixnay (2017b)

Mehrschichtige Rekurrentes neuronales Netzwerk RNN in ihrer ursprünglichen Form bestehen aus einer Schicht an Neuronen, welche die Gesamtheit aller anstehenden Aufgaben lösen müssen. Hermans und Schrauwen (2013) zeigten, dass der Ansatz eines tiefen, d.h. mehrschichtigen RNNs möglich ist und deutliche Vorteile gegenüber den klassischen Vertretern bietet.

Die Idee von tiefen neuronalen Netzwerken entstammt den frühen 2000er Jahren, als man dank verbesserter Rechenkapazitäten begann, mehrere Schichten von Neuronen hintereinander zu schalten. Damit können pro Schicht verschiedene Abstraktionsebenen der Daten angesprochen werden. Dadurch ergibt sich die Möglichkeit, eine bessere Entscheidung zu treffen als es für eine Schicht allein möglich wäre. (Szegedy et al. 2015)

An einem Beispiel der Bilderkennung lässt sich dieser Sachverhalt gut erläutern: Soll ein Netzwerk darauf trainiert werden, Autos in einem Bild zu erkennen, ist ein einschichtiges Netzwerk mit hoher Wahrscheinlichkeit überfordert oder liefert nur schlechte Resultate. Führt man jedoch mehrere Schichten ein, kann sich die erste Schicht auf das Differenzieren von Ecken, Rundungen und Farben im Bild fokussieren. Währenddessen ist die nächste Schicht beispielsweise bereits in der Lage, daraus verschiedene Texturen zu erkennen. Eine weitere Schicht wäre schließlich womöglich in der Lage, nun aus den Informationen bezüglich Textur und Formen im Bild eine Entscheidung zu treffen, ob sich ein Auto in dem Bild befindet oder nicht. Dieser Vorgang wird heutzutage in der Regel mit CNN abgebildet.

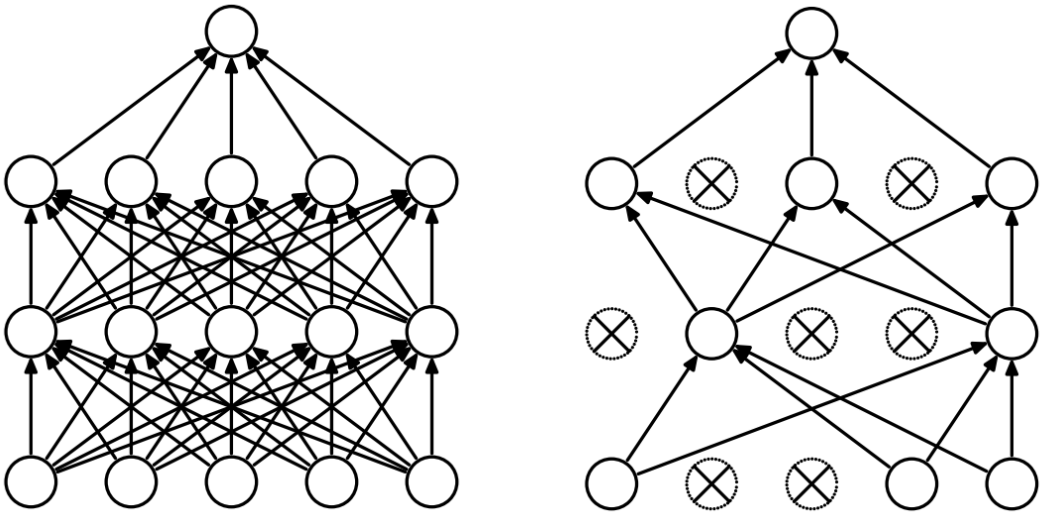
Ähnlich verhält es sich bei tiefen RNNs. Der Ausgangswert der ersten Schicht dient als Eingangswert der nächsten. Auf diese Weise kann sich wieder jede Schicht dem Erlernen einer bestimmten Unteraufgabe widmen und so von der Arbeit der vorhergehenden profitieren. (Hermans und Schrauwen 2013)

1.4.4 Dropout

Ein häufig auffallendes Phänomen beim Trainieren von neuronalen Netzwerken ist das sogenannte „Overfitting“ (englisch für Überanpassung). Damit wird ein beobachtbarer Abfall von einer zuvor erreichten Leistung beim mehrmaligen Verwenden desselben Trai-

ningsdatensatzes beschrieben. Dieses Problem wird verursacht durch ein Überangebot verstellbarer Parameter, die zu einer zu komplexen Lösung des Problems führen. Die Folge ist ein schlechteres Ergebnis als mit einem Ansatz mit weniger Parametern erreicht werden kann. Da eine optimale Anzahl und Konfiguration von Neuronen sich nicht im Voraus berechnen lässt, muss der Trainingsprozess selbst die Lösung dieses Problems übernehmen.

Eine Lösung hierfür stellt die Einführung des sogenannten „Dropouts“ (englisch für herausfallen) dar. Entsprechend einer vordefinierten Wahrscheinlichkeit klammert diese Methode zufällig Neuronen aus dem Netzwerk aus. Dadurch wird mit jeder Iteration eine etwas andere, „ausgedünnte“ Netzwerkkombination mit der Trainingsaufgabe betraut und nur diese Kombination dem Optimierungsprozess unterworfen. Mit diesem Prozess können tatsächlich relevante Verschaltungen mit höherer Wahrscheinlichkeit identifiziert werden. Dies führt zu deutlich besseren Leistungen neuronaler Netzwerke. In der Regel stellt eine Ausfallwahrscheinlichkeit von 50% pro Neuron während des Trainings bei den meisten Netzwerken einen guten Wert dar. Bei der Evaluierung wird die Ausfallwahrscheinlichkeit anschließend auf 0% herabgesetzt, um die Leistung aller Neuronen gemeinsam nutzen zu können. (Srivastava et al. 2014)



(a) Typisches neuronales Netzwerk. (b) Nach der Anwendung von Dropout.

Abbildung 7: Illustration des Dropouts. Rechts ist das ausgedünnte Netzwerk dargestellt nach der Anwendung des Dropouts mit einer Rate von 50% auf das linke Netzwerk. Die gekreuzten Neuronen sind vom Dropout ausgeklammert. Modifiziert nach Srivastava et al. (2014)

1.4.5 Logits

Am Ende eines jeden klassifizierenden Netzwerkes steht ein sogenanntes „Logits layer“. So wird im Bereich der künstlichen Intelligenz eine Schicht von Neuronen bezeichnet, die jeweils mit allen Neuronen der vorhergehenden Schicht verbunden ist und der endgültigen Einteilung eines Trainingsbeispiels in eine Klasse dient. Damit entspricht diese Schicht der ursprünglich von *Rosenblatt* dargelegten Bauweise eines Perzeptrons. Durch die dichte Vernetzung mit der vorhergehenden Schicht, sind die Neuronen in der Lage, ein Maximum an Informationen zur Bearbeitung einer spezifischen Klassifizierungsfrage heranzuziehen. Während die vorherigen Schichten also beliebig groß und unterschiedlich miteinander verschaltet sein können, fokussieren die Logit layers die Ausgabe des Netzwerkes auf die Zahl der erlernbaren Klassen.

Zusammenfassend sind es also die Ausgabewerte dieser Neuronen, die den letztendlichen Zweck des Netzwerkes darstellen und die zur Evaluierung herangezogen werden.

Grundlage eines Logit layers sind eine Gewichtsmatrize W_l und eine Bias Matrize B_l. Durch Multiplikation der Eingabewerte mit der Gewichtsmatrize und anschließender Ad-

dition mit der Bias Matrize entsteht der Ausgabewert des Logit layers:

$$o_l = W_l \cdot x_t + B_l \tag{9}$$

In manchen Fällen wird der Ausgabewert noch durch die Anwendung einer weiteren Funktion ergänzt, die das Zahlenspektrum verändert oder bestimmte Werte nicht zulässt. Bei dieser Methode wird analog zum biologischen Vorbild ein Schwellenwert darstellt. Stark an die biologischen Pendants angelehnt, spricht man hierbei von „Aktivierungsfunktionen“. (Nielsen 2015, Kapitel 1)

Populär sind zum einen klassische sigmoidale Funktionen, aber auch spezielle Funktionen wie beispielsweise eine „lineare Rektifizierung“ (ReLU) (Nair und Hinton 2010) oder „softplus“ (Glorot et al. 2011). Einige gebräuchliche Aktivierungsfunktionen sind in Abbildung Abbildung 8 auf Seite 26 schematisch dargestellt.

1.4.6 Fehlerfunktion

Die Fehlerfunktion ist die mathematische Instanz, welche die Differenz zwischen den Ausgabewerten des Netzwerkes und den annotierten Zielwerten ermittelt. Nur wenn diese Funktion in der Lage ist, diesen Fehler richtig und gleichmäßig verteilt abzubilden, kann der Optimierungsalogrithmus die richtigen Veränderungen herbeiführen.

Demzufolge gibt es für unterschiedliche Aufgaben auch eine Vielzahl an Fehlerfunktionen. Einige davon sind ausschließlich für Szenarien geeignet, in denen nur eine Klasse positiv sein kann, während andere mit mehreren Klassen gleichzeitig umgehen können. (Nielsen 2015, Kapitel 3)

Der mean squared error (MSE) ist eines der bekanntesten Beispiele für eine Fehlerfunktion und wurde durch *Carl Friedrich Gauss* entwickelt. Er berechnet die durchschnittliche quadrierte Differenz aus den Annotationen A_i (also den Sollwerten) und der Ausgabe des Netzwerkes P_i. Er berechnet sich wie folgt:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (A_i - P_i)^2 \tag{10}$$

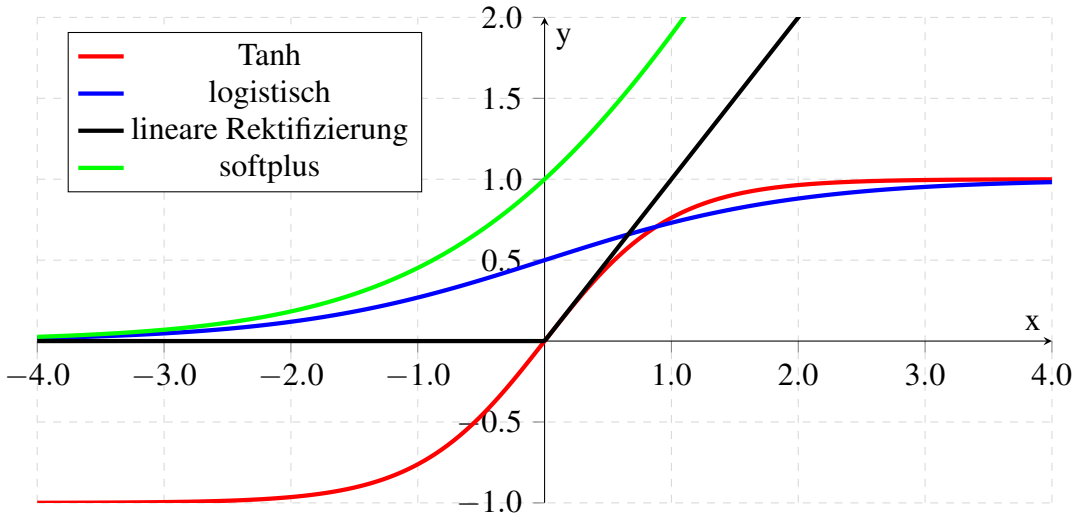


Abbildung 8: Illustration verschiedener Aktivierungsfunktionen: Tanh, logistisch, lineare Rektifizierung und softplus

Der Hauptkritikpunkt am MSE ist die Tatsache, dass er initial zu einem sehr langsamen Trainingsfortschritt führen kann und demnach mehr Trainingsschritte notwendig sein könnten um die gewünschte Leistung zu erreichen. (Nielsen 2015, Kapitel 3)

Sigmoid cross entropy (SCE) behebt das Problem der Verlangsamung des Lernens. Er ist stets positiv und tendiert gegen 0, wenn das Netzwerk bessere Ergebnisse erzielt. Er ist wie folgt definiert:

$$SCE = -P_i * A_i + log(1 + exp(P_i)) \tag{11}$$

1.4.7 Optimierungsfunktion

Der Wert der Fehlerfunktion ist Kernparameter der Optimierungsfunktion. Dieser Algorithmus ist das Herzstück des maschinellen Lernens, welches die Anpassung des Netzwerkes hin zu immer besseren Leistungen automatisiert. Der Prozess erfolgt durch die sogenannte Backpropagation (englisch für „Fehlerrückführung“), welche 1986 durch *David Rumelhart*, *Geoffrey Hinton* und *Ronald Williams* (D. Rumelhart et al. 1986) weite Bekanntheit erlangte, nachdem die ersten Ansätze bereits in den 1970ern entwickelt wurden.

Das elegante Prinzip hinter der Fehlerrückführung liegt in der effizienten Berechnung der Derivate, d.h. der einzelnen Anteile eines jeden Gewichts oder Bias an dem Gesamtfehler (oder Fehlergradient). Kann diese Aufschlüsselung berechnet werden, ist es möglich die Einstellungen dieser entsprechend schrittweise zu ändern und so den Fehlergradienten zu verkleinern. Diese Form der Optimierungsfunktion wird auch Gradient descent optimizer (GDO) (englisch für „Gradienten-Abstiegs-Optimierer“) genannt. (Nielsen 2015, Kapitel 2)

Seit seiner initialen Einführung 1986 wurde der GDO immer weiterentwickelt um noch mehr Details und Probleme beim Trainieren von ANN zu lösen. Eines der ersten Probleme ist die in seiner ursprünglichen Form sehr sprunghafte Reaktion der Fehlerrückführung auf die Unterschiede in den einzelnen Trainingsbeispielen. Dieses hohe Maß an Fluktuation und der daraus überschießenden Anpassung der Parameter verlangsamte den Prozess.

Der erste Verbesserungsansatz liegt in der Bildung sogenannter *Mini-Batches* (englisch für Charge oder Bündel) aus mehreren Trainingsbeispielen, über die gleichzeitig die Fehlerrückführung berechnet wird. Die Hoffnung liegt darin, dass sich über mehrere Trainingsbeispiele hinweg eine gewisse Normalisierung ergibt und somit der allgemeine Trainingsfortschritt schneller in die richtige Richtung gelenkt wird.

Weitere Entwicklungen stellen Optimierungsalgorithmen dar, die das Problem der Fluktuation mittels „Momentum“ adressieren. Diese berechnen den Hauptvektor der einzustellenden Veränderungen und dämpfen somit den Effekt der Oszillationen durch Unterschiede in den Trainingsdaten.

Der Forscher *Yurii Nesterov* erkannte jedoch die Gefahr einer Fehlerrückführungsfunktion, die blind dem Momentum immer weiter folgt. Diese tendiert nämlich dazu, das Optimum zu überschießen und benötigt daher unnötige Zeit um das Überschießen zu korrigieren. Er schlug daher 1983 vor, den Wert des Momentums regelmäßig zu überprüfen, indem er diesen testweise für viele Schritte anwendete und dann den Gradienten überprüfte. Die daraus gewonnene Erkenntnis wurde dann auf die tatsächliche Fehlerrückführung angewandt. (Nesterov 1983) Diesen Ansatz nennt man „Nesterov accelerated gradient (NAG)“.

(a) Optimierung mit GDO ohne Momentum. (b) Optimierung mit GDO mit Momentum.

Abbildung 9: Schematische Darstellung des des GDO. Ohne Momentum oszilliert der Algorithmus in kleinen Schritten dem Optimum entgegen, während Momentum schneller und steiler Richtung Optimum führt.

Ein weiteres, bei NAG noch nicht adressiertes Problem, liegt in der Implementierung der Lernrate. Dieser Wert definiert, mit welchem Grad an Intensität pro Trainingsschritt die Werte der Gewichts- und Bias-Matrizen angepasst werden dürfen. Bei frühen Optimierungsalgorithmen wurde die Lernrate global vom Entwickler fest implementiert. Dies ist jedoch problematisch, da die einzelnen Anteile am Gesamtgradienten teilweise sehr weit auseinander liegen und die optimalen Lernraten für die einzelnen Parameter in der Regel größer oder kleiner einer fest definierten Lernrate sind. Daher implementieren einige neuere Ansätze durch verschiedene Herangehensweisen eine variable Lernrate, die entweder global oder sogar individuell für jeden Parameter angepasst wird. Somit muss die optimale Lernrate nicht mehr empirisch vom Entwickler ermittelt werden. Die gängigsten Vertreter dieser Vorgehensweise sind *Adaptive moment estimation (ADAM) und RMSprop*. (Kingma und Ba 2015; Walia 2017)

1.4.8 Software-Bibliotheken und Hardware-Beschleunigung

Das Trainieren von ANN ist ein komplexes Feld der Informatik. Die Kernaspekte der Komplexität liegen sowohl in der Menge der zu verarbeitenden Datenmengen als auch in der zuverlässigen und effizienten Implementierung der involvierten Rechenschritte. Deshalb war dieses Forschungsgebiet für lange Zeit wenigen Experten vorbehalten und wurde

in anderen Feldern selten eingesetzt.

Um die Möglichkeiten der ANN einem größeren Forscherkreis zugänglich zu machen, wurden über die letzten Jahre hinweg mehrere Projekte aufgesetzt, die sogenannte Software-Bibliotheken für ANN entwickeln.

Mit diesen Software-Bibliotheken soll den Entwicklern ein Gerüst zur Verfügung gestellt werden, was sie in die Lage versetzt mithilfe eines Baukastenprinzips ANN zu entwickeln und trainieren. Der Vorteil hierbei liegt darin, dass die Entwicklung der einzelnen Bausteine Experten überlassen wird, welche sich um die korrekte und effiziente Umsetzung der Logik und Mathematik kümmern, während der Nutzer sich dem Problem von einer rein logischen Perspektive nähern kann. (Abadi, Barham et al. 2016)

Bei der Wahl der Software-Bibliothek sollten zum einen die zu Grunde liegende Programmiersprache, die Menge implementierter ANN-Funktionen und -strukturen, die Zahl der aktiven Nutzer und die Fähigkeit zur Hardware-Beschleunigung eine Rolle spielen.

Viele der heutzutage populären Bibliotheken werden in der Programmiersprache Python angeboten (beispielsweise *Theano* (The Theano Development Team et al. 2016) oder *TensorFlow* (Abadi, Barham et al. 2016)), während beispielsweise *Caffe* (Jia et al. 2014) auf C++ basiert. Die Mathematik-Software *Matlab* bietet zudem ebenfalls Bibliotheken zur Implementierung von ANN. (Kim 2017)

Hardware-Beschleunigung Da das Trainieren von neuronalen Netzwerken ein sehr rechenaufwändiger Prozess ist, gibt es inzwischen Ansätze, diesen Vorgang durch spezialisierte Computerkomponenten zu beschleunigen. Der Fokus liegt hierbei auf der Effizienzsteigerung der zahllosen Matrix-Operationen, welche das mathematische Grundgerüst eines jeden Netzwerkes darstellen. Dieser Prozess begann mit dem Einsatz optimierter Befehlssätze für Grafikkarten (Graphics Processing Units (GPUs)), da diese bereits hochgradig parallelisierte Prozessorarchitekturen besitzen, welche die Berechnung dieser Matrizen effizienter und schneller bewerkstelligen können als gängige Central Processing Units (CPUs). (NVIDIA Developer 2016)

Eine zusätzliche Neuerung stellen sogenannte „Tensor Processing Unit (TPU)“ dar, wel-

che 2015 von Google entwickelt wurden und durch eine weitere Verfeinerung der Prozessorarchitektur und optimierte Integration in die *TensorFlow* Bibliothek eine weitere Beschleunigung um ein Vielfaches im Vergleich zu CPUs und GPUs bieten. (Jouppi et al. 2017)

1.4.9 Derzeitige Verwendung im Bereich der EKG-Interpretation

Der Einsatz von ANN zur Analyse von EKGs ist ein Feld zunehmender Aktivität. Hierbei liegt der Fokus in der Ausführung sehr komplexer Aufgaben wie der Herzschlag-Klassifizierung und der Erkennung von Rhythmusstörungen. (Lyon et al. 2018) Zudem gibt es Arbeiten über die Möglichkeit, Patienten anhand ihres EKGs und ANN im biometrischen Sinne wiederzuerkennen. (Wieclaw et al. 2017)

Während obige Arbeiten sich mit komplexen Aufgaben beschäftigen, gibt es neuerdings auch Ansätze, die Erkennung und Abgrenzung grundlegender EKG-Strukturen durch ANN zu implementieren. Damit stößt diese Methode in einen Bereich vor, der bislang klassischen EKG-Analysemethoden von Methoden ohne ANN-Ansatz dominiert wird. (Camps et al. 2018; Abrishami et al. 2018)

2 Ziel der Arbeit

Ziel dieser Dissertation ist die *TensorFlow*-basierte Implementierung eines ANN zur Abgrenzung der lokalen Maxima der charakteristischen Strukturen eines EKGs mittels *TensorFlow*. Als Netzwerkarchitektur wird ein RNN verwendet und im Rahmen einer Hyperparameter-Studie die optimale Konfiguration identifiziert. Anschließend wird die Leistungsfähigkeit des Netzwerkes mit etablierten Methoden und ähnlichen ANN-Ansätzen verglichen.

3 Material und Methoden

3.1 Hardware

Sämtliche Schritte dieser Arbeit wurden auf einem 13”Apple MacBook Pro (Modelljahr 2016) mit einem 2,9 GHz dual-core Intel Core i5 Rechenkern (CPU), 16 GB Arbeitsspeicher (Random-Access Memory (RAM)) und 512 GB Hauptspeicher in Form einer Solid State Disk (SSD) ausgeführt. Als Betriebssystem diente OS X High Sierra.

3.2 Trainingsdatensatz

Zur besseren Validierung gegenüber anderen EKG-Erkennungsalgorithmen wurde ein öffentlich zugänglicher und oft referenzierter Datensatz als Trainings- und Evaluierungsgrundlage gesucht.

3.2.1 Physionet

Für diesen Zweck ist das online-Repositorium *Physionet* geeignet. Es bietet eine Reihe an Biosignalen, welche mit einer Vielzahl von klinischen Informationen und Annotationen versehen sind und der Allgemeinheit frei zur Verfügung stehen. Zusätzlich liefert es fertige Werkzeuge, um die Daten zu laden und zu bearbeiten. (Goldberger et al. 2000)

3.2.2 QT Database

Von den zahlreichen auf Physionet zur Verfügung stehenden Datenbanken stellte sich die QT Database (QTDB) als die geeignetste heraus. Die QTDB enthält 105 Zweikanal-EKGs von jeweils 15 Minuten Dauer in einer Auflösung von 250 Hz mit detaillierten Annotationen bezüglich der Wellengrenzen und -maxima. (Laguna, Mark et al. 1997)

Das Vorhandensein dieser Annotationen war ein essentielles Kriterium für die Auswahl der QTDB, da diese für die angestrebte Form des überwachten Lernens benötigt werden. Sie werden beim Training und der Evaluation die „Ground Truth“ (Englischer Begriff in der Statistik und ML, welcher für die optimale erwartete Lösung eines Problems verwendet wird. (Kozyrkov 2020)) darstellen, anhand derer das Netzwerk optimiert und dessen

Leistung gemessen wird. Im Folgenden wird jedoch der Begriff der Annotation für die „Ground Truth“ eingesetzt.

Vor der weiteren Vorverarbeitung (Preprocessing) wurden die EKGs hinsichtlich ihrer tatsächlichen Eignung als Trainings- und Evaluationsdaten untersucht. Folgende Kriterien wurden bei der Auswahl verwendet:

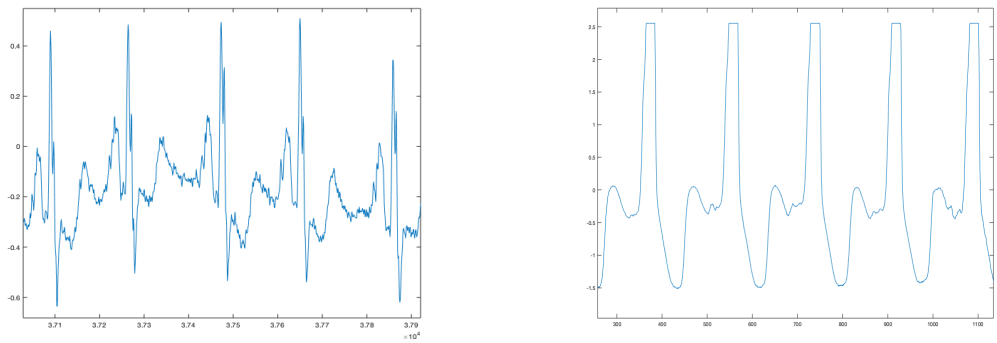
1. EKG enthält vollständige Annotationen für die Maxima von P- und T-Welle und Mittelpunkte der QRS-Komplexe.

2. EKG-Signal ist in keiner Weise abgeschnitten oder verändert.

Insgesamt konnten 10 EKGs das erste Kriterium nicht erfüllen, da Annotationen für entweder die P- oder T-Welle fehlten.

Ein weiteres EKG scheiterte am zweiten Kriterium. Bei diesem kam es wahrscheinlich aufgrund einer Übervoltage zu einer überdurchschnittlich hohen Amplitude des EKG-Signals. Der Grund dafür liegt in der Regel am fehlerhaften Platzieren der EKG-Elektroden und führt zu einem „Abschneiden“ der Spitzen der R-Zacken. Verdeutlicht wird diese Tatsache anhand des Vergleichs in Abbildung 10 auf Seite 32.

Unter Berücksichtigung der Ausschlusskriterien konnten von den ursprünglichen 105 EKGs der QTDB schlussendlich 94 für die Arbeit verwendet werden.



(a) Eingeschlossenes EKG - Signal `sel39` (b) Ausgeschlossenes EKG - Signal `sel42`

Abbildung 10: Darstellung des zweiten Einschlusskriteriums durch Vergleich eines eingeschlossenen EKGs mit einem ausgeschlossenen EKG.

3.2.3 Preprocessing in MATLAB®

Für das Preprocessing wurde die Mathematik-Software *MATLAB* eingesetzt. *Physionet* bietet hierfür ein fertiges Softwarepaket zum Herunterladen und Bearbeiten der Biosignale und ihrer Annotationen. Die geladenen EKGs wurden anschließend alle denselben Vorverarbeitungsschritten unterzogen und in den Trainings- und Evaluationsdatensatz verteilt.

Zunächst bestand die Option, das Rohsignal mittels des *Savitzky-Golay-Filters* (Savitzky und Golay 1964) zu glätten. Der Filter wurde mit den Einstellungen eines Polynomgrades von 0 und einer Fensterbreite von 15 verwendet. Es wurden Datensätze sowohl mit als auch ohne Filter erzeugt und untersucht. Hintergrund dafür war die Absicht zu untersuchen, welchen Effekt vorgefilterte Daten auf die Leistungsfähigkeit des Netzwerks haben.

Daraufhin wurden alle Werte der EKG-Signale auf eine Skala von $1 \cdot e^{-10}$ bis 1 redimensioniert, um dem Netzwerk einheitliche Eingangswerte zu liefern.

Anschließend erfolgte die Vorbereitung der Signale für die Speicherung als comma-separated values (CSV)-Dateien. Dieses Dateiformat bietet den Vorteil einer weiten Kompatibilität über alle Programmiersprachen hinweg, da es sich um Textdateien handelt, in denen die Spalten einer Matrix durch Kommata voneinander getrennt sind. Da der Prozess des Lesens iterativ Zeile für Zeile geschieht, wurde beschlossen, die bis zu 225000 Datenpunkte eines einzelnen EKGs auf Reihen von je 1000 Zeitschritten (entspricht 4 Sekunden Signallänge) herunterzubrechen. Zudem verfügte jede Zeile über eine Spalte, in der die Identität des jeweiligen Signals abgespeichert wurde und zusätzlich 3000 Spalten für die korrespondierenden 1000 Annotationen pro Welle/QRS-Komplex. Somit ergab sich eine Datenstruktur mit 4001 Spalten und einer der Signaldauer entsprechenden Zahl von Reihen pro EKG.

Um während des Trainingsprozesses nicht zu lange auf Daten des gleichen EKG-Beispiels zu trainieren, wurde beschlossen, sämtliche Zeilen der Trainings-EKGs untereinander zu mischen, um so einen jederzeit möglichst homogenen Trainingsdatensatz zu erhalten. Dieser Datensatz und der Evaluierungsdatensatz wurden anschließend jeweils in ei-

ner CSV-Datei gespeichert. Insgesamt enthielten der Trainingsdatensatz 14817 und der Evaluierungsdatensatz 6287 Zeilen an EKG- und Annotationsdaten. Eine Schematische Darstellung des Preprocessing in *MATLAB* findet sich in der Abbildung 11 auf Seite 35.

3.2.4 Erzeugte Trainingsdatensätze

Es wurden zunächst zehn Trainingsdatensätze erzeugt, um die wichtigsten Fragestellungen nach der optimalen Annotationsbreite und dem Einsatz des Savitzky-Golay-Filters zu untersuchen. Es wurden Annotationsbreiten von 1, 3, 5, 7 und 11 Datenpunkten untersucht und diese jeweils im gefilterten und ungefilterten Zustand. Eine genaue Auflistung der generierten Datensätze ist in Tabelle 1 auf Seite 36 zu finden.

3.3 TensorFlow - Software-Bibliothek für maschinelles Lernen

Wie im Abschnitt 1.4.8 bereits erläutert, stehen Forschern im Bereich der künstlichen Intelligenz diverse Software-Bibliotheken zur Verfügung, um Netzwerke zu programmieren und zu trainieren.

Die Wahl fiel auf die Open-Source-Bibliothek *TensorFlow*, welche vom Google Brain Team seit 2011 entwickelt wird. (Abadi, Agarwal et al. 2015) Ausschlaggebend für diesen Entschluss waren die Vielzahl implementierbarer Netzwerkkomponenten, der automatische Lastausgleich entsprechend der verwendeten Hardware und das Basieren auf der modernen Programmiersprache Python.

In den folgenden Abschnitten soll dem Leser ein grundlegendes Verständnis für den Aufbau und die Funktionsweise der *TensorFlow*-Bibliothek vermittelt werden. Die konkret in der Arbeit verwendeten Komponenten und Schritte werden im Abschnitt 3.4 auf Seite 39 genauer erläutert.

3.3.1 Grundlagen

Der Name *TensorFlow* stammt vom mathematischen Begriff des „Tensors“ („tendere“, lateinisch für spannen und „flow“ englisch für fließen) ab. Er dient als Oberbegriff für Matrizen, Skalaren und Vektoren. In der Tat kann man das Konstrukt eines neuronalen

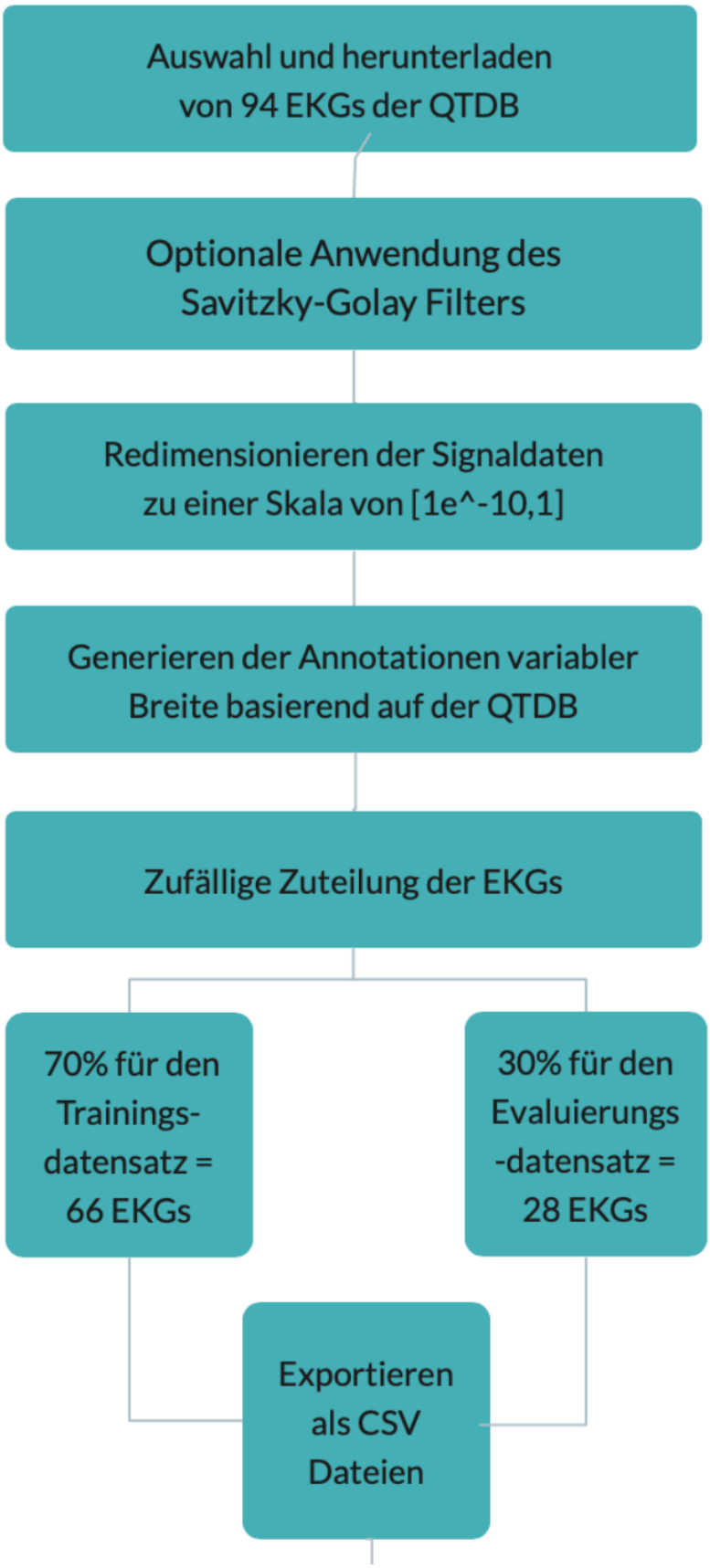


Abbildung 11: Schematische Darstellung des Preprocessings der EKG Daten in *MATLAB*.

Tabelle 1: Auflistung der erzeugten Datensätze.

Datensatz	**Annotationsbreite**	**Filter**
PRT1-NF	1	nein
PRT3-NF	3	nein
PRT5-NF	5	nein
PRT7-NF	7	nein
PRT11-NF	11	nein
PRT1-F	1	aktiv
PRT3-F	3	aktiv
PRT5-F	5	aktiv
PRT7-F	7	aktiv
PRT11-F	11	aktiv

Netzwerkes betrachten als das Fließen von Daten durch eine Aneinanderreihung von Matrizen, die durch Operationen miteinander verknüpft sind.

TensorFlow basiert auf der Repräsentation aller Rechenschritte als Knotenpunkte eines Graphen. Jeder Knotenpunkt kann Eingangs- und Ausgangsparameter besitzen, die den Fluss von Daten oder Einstellungen repräsentieren. Durch die Verknüpfung der Ein- und Ausgänge miteinander entsteht die Logik, welche letztendlich das ANN und die benötigten Operationen zum Trainieren, Evaluieren und Abspeichern darstellt.

Ein- und Ausgabedaten und variable Einstellungen (auch Hyperparameter genannt) werden im Graphen durch sogenannte Platzhalter-Knotenpunkte dargestellt. Sie definieren die Formatierung dieser Werte und erlauben so eine automatische Überprüfung der Logik des kompletten Graphen noch bevor echte Daten durch den Graphen fließen.

Knotenpunkte, welche Operationen ausführen können, werden abstrakt anhand ihrer Operation (zum Beispiel `add` für eine Addition) benannt. Im Hintergrund beinhalten Knotenpunkte verschiedene, für Berechnungen auf CPUs oder GPUs optimierte „Kernels“ (aus dem Englischen für Systemkerne), d.h. die jeweiligen Befehlsketten, um die Operation auf dem Rechner durchzuführen. Der reine Fokus auf die operativen Eigenschaften des

Knotenpunktes erlaubt es dem Programmierer, sich ausschließlich auf die Logik des Graphen zu konzentrieren. *TensorFlow* kann währenddessen flexibel auf die unterschiedlichen Gegebenheiten der Hardware des verwendeten Computersystems eingehen, indem es individuell den passenden Kernel auswählt.

3.3.2 Aufbauen eines Netzwerk-Modells

Wie im Abschnitt 1.4.2 bereits erwähnt, bestehen neuronale Netzwerke aus Schichten verknüpfter Neuronen, welche wiederum in sich Verkettungen von mathematischen Operationen darstellen.

TensorFlow hält sich ebenfalls bei der Programmierung eines Netzwerkes an diese Sichtweise und bietet für jede Betrachtungsstufe entsprechende Knotenpunkte. Schicht-Knotenpunkte bieten eine Vielzahl an einstellbaren Parametern und erzeugen anhand dieser automatisch die erforderlichen Neuronen beziehungsweise mathematische Operationsknoten im Graphen.

Das Verknüpfen der einzelnen Schichten oder Neuronen untereinander erfolgt, indem die Ausgangswerte der vorhergehenden Schicht als Eingangswerte für die darauf folgende Schicht definiert werden. Für die erste Schicht von Neuronen wird dabei der Platzhalter-Knotenpunkt für eingehende Daten verwendet.

3.3.3 Einlesen von Daten

Auch das Einlesen der Daten erfolgt als Teil des Graphen und wird in der abstrakten Betrachtungsweise definiert. Es werden zu Beginn die Dateiart und Formatierung der Daten festgelegt, damit die logische Überprüfung des gesamten Graphen ohne echte Daten erfolgen kann.

Obwohl *TensorFlow* mehrere Datenformate unterstützt, soll im Folgenden jedoch nur auf die in dieser Arbeit verwendete Dateneingabe eingegangen werden, da sich die Vorgänge abhängig des Dateiformates stark unterscheiden.

Wie im Abschnitt 3.2.3 auf Seite 33 beschrieben, lagen die vorverarbeiteten Daten in Form von CSV-Dateien vor, welche in ihrer grundlegenden Struktur einfache Textdatei-

en darstellen. Für dieses Datenformat bietet *TensorFlow* einen `text_line_reader` (frei übersetzt „zeilenweiser Texteinleser“), welcher in der Lage ist, Textdateien automatisch zu öffnen und zeilenweise einzulesen. Anschließend werden mit der Operation `decode_csv` die Werte der eingelesenen Zeile automatisch anhand der Kommata in eine Matrix überführt. Dieser Operationsknotenpunkt benötigt eine Definition der Struktur der einzulesenden Daten (d.h. wie viele Spalten/Werte sind pro Textzeile zu erwarten). Die erzeugten Ausgabewerte können nun an dieser Stelle vom Programmierer genutzt werden, um weitere Vorverarbeitungsschritte an den Daten zu unternehmen (siehe hierzu Abschnitt 3.4.2 auf Seite 40). Am Ende dieses Schrittes hat der Programmierer einen für den spezifischen Datensatz angepassten Einlesemechanismus/Knotenpunkt im Graphen definiert.

TensorFlow betrachtet die Datenaufnahme jedoch auf einer noch höheren Metaebene, der sogenannten „Input pipeline“. Hier kann der Entwickler Operationsknoten nutzen, die beispielsweise in der Lage sind, die Daten als Pakete mehrerer Zeilen zu bündeln (aus dem Englischen „batching“) oder zu durchmischen. Erst diese Stufe der Datenaufnahme wird anschließend als Eingabewert für die neuronalen Komponenten des Graphen genutzt. Alle Kernels der Komponenten dieser Input pipeline und des Einlesemechanismus führen im Hintergrund einen Lastausgleich (aus dem Englischen „load balancing“) durch. Das bedeutet, dass die *TensorFlow*-Komponenten immer nur anhand der aktuell vorhandenen Ressourcen des Rechners arbeiten und es infolgedessen zu keinen Fehlern durch Überlastung kommen kann.

3.3.4 Ausführen von Prozessen

Um nun echte Daten in das Netzwerk einlesen und Berechnungen ausführen zu können, wird der Graph in einer Session (aus dem Englischen für Sitzung) ausgeführt. Zu Beginn einer Sitzung wird der Graph automatisch hinsichtlich seiner mathematischen Logik überprüft und anschließend alle erforderlichen Knotenpunkte im RAM angelegt. Die nun initialisierte Sitzung kann über den Befehl `run` Operationen des Graphen ausführen und die Ergebnisse ausgeben. Neben der Angabe der auszuführenden Operation(en) erwartet der `run`-Befehl ein sogenanntes „Einspeisungslexion“ (frei übersetzt aus dem Englischen von „feed dictionary“), welches die realen Werte für die Platzhalterknotenpunkte enthält.

Mit jeder Iteration wird dieses Einspeisungslexion erneuert, vor allem die Werte für die Dateneingabe der zu trainierenden Werte. Die ausgegebenen Werte können nun entweder für weitere `run`-Befehle genutzt, dem Nutzer angezeigt oder wieder auf dem Rechner abgespeichert werden.

3.3.5 Performancemonitoring mittels TensorBoard

Die Vielzahl an Parametern und Rechenoperationen, die ein modernes neuronales Netzwerk beinhaltet, sind von einem Menschen nicht zu überblicken. Aus diesem Grund sind diese visuellen Schnittstellen essentiell, um Veränderungen im Trainingsverhalten und potentielle Fehler zu verstehen. *TensorFlow* bietet zur Überwachung des Trainingsfortschritts, aber auch der Struktur des Graphen für Visualisierungszwecke oder Fehlersuche, eine fertige Web-Umgebung namens *TensorBoard*. Dieses Werkzeug erlaubt die Anzeige von Leistungsparametern, welche zuvor als *TensorBoard*-Knotenpunkte im Graphen angelegt werden. Es ist zudem in der Lage, ein interaktives Schema des gesamten Graphen zu generieren.

3.4 Implementierung des RNNs in TensorFlow

Da es sich bei den in dieser Arbeit verwendeten Trainingsdaten um sequenzielle Daten handelt, also Daten, die in einer strikten zeitlichen Reihenfolge zueinander stehen, bietet sich eine RNN-Architektur für das Netzwerk an. Es wurde beschlossen, die Zahl der Neuronen pro Schicht und die Anzahl dieser variabel zu implementieren, um so während der Trainingsläufe die optimale Netzwerkkonfiguration zu identifizieren. Deshalb wurden sämtliche variablen Parameter zunächst als Platzhalter-Knotenpunkte im Graph definiert, die dann während des Trainings- und Evaluierungsprozesses über das Einspeisungslexion mit den Einstellungswerten versehen werden.

3.4.1 Erstellen des Netzwerkes

Die Erstellung des RNN erfolgt zunächst mit der Definition des zugrundeliegenden Neuronentyps - für LSTM `tf.contrib.rnn.LSTMCell` und für GRU `tf.contrib.rnn.GRUCell`. Beide Funktionen nehmen als Parameter die gewünschte Zahl von Neuronen an.

Man erhält somit eine Schicht mit der gewünschten Anzahl an Neuronen eines Typs. Um RNN mit mehreren Schichten zu erzeugen, existiert in TensorFlow die Hüllfunktion `tf.nn.rnn_cell.MultiRNNCell`, welche mehrere Instanzen von RNN-Schichten zu einem gesamten RNN-Objekt bündeln kann. Dieses gebündelte RNN-Objekt ist wiederum für die folgenden Schichten verwendbar.

Da sich sowohl GRU- als auch LSTM- Netzwerke als leistungsfähig herausgestellt haben (Chung et al. 2014), wurde beschlossen beide Typen in dieser Arbeit zu untersuchen. So kann die optimale Architektur für die gestellte Aufgabe gefunden werden.

Als nächstes wurde der Dropout implementiert, der in diesem Fall als Hülle um das RNN betrachtet werden kann. Hierzu wird die Hüllfunktion `tf.nn.rnn_cell.DropoutWrapper` verwendet. Sie erhält als Eingabeparameter das mehrschichtige RNN-Objekt und die gewünschte Dropout-Rate. Diese Funktion übernimmt automatisch die zufällige Deaktivierung der einzelnen Zellen im RNN während des Trainings anhand der angegebenen Dropout-Rate.

Dieses komplette Konstrukt wird nun an die Logits-Schicht als Eingabe weitergegeben, welche die letztendliche Klassifizierung des Eingabesignals übernimmt. Die fertige Schicht lässt sich mit der `tf.layers.dense`-Methode erzeugen und ebenfalls durch verschiedene Eingabeparameter in ihrer Größe, Aktivierungsfunktion und anderen Details anpassen. In diesem Falle wurde eine aus drei Neuronen (korrespondierend mit den drei Zielannotationen) bestehende Logits-Schicht ohne Aktivierungsfunktion gewählt.

Wie in Abbildung 12 auf Seite 41 zu sehen ist und in den Abschnitten 3.5 und 3.6 erläutert wird, erfolgt anschließend der Vergleich der Ausgabewerte mit den Annotationen. Daraus können einerseits Messgrößen zur Netzwerkleistung berechnet und andererseits der anstehende Trainingsschritt abgeleitet werden. Zunächst soll jedoch der Prozess des Einlesens der Daten und die temporale Iteration über diese genauer beschrieben werden.

3.4.2 Iteratives Zeitfenster

Wie im Abschnitt 3.2.3 auf Seite 33 geschildert wurde, liegen die Daten nach dem Preprocessing in MATLAB als CSV-Dateien vor. Diese werden im Rahmen der in Abschnitt

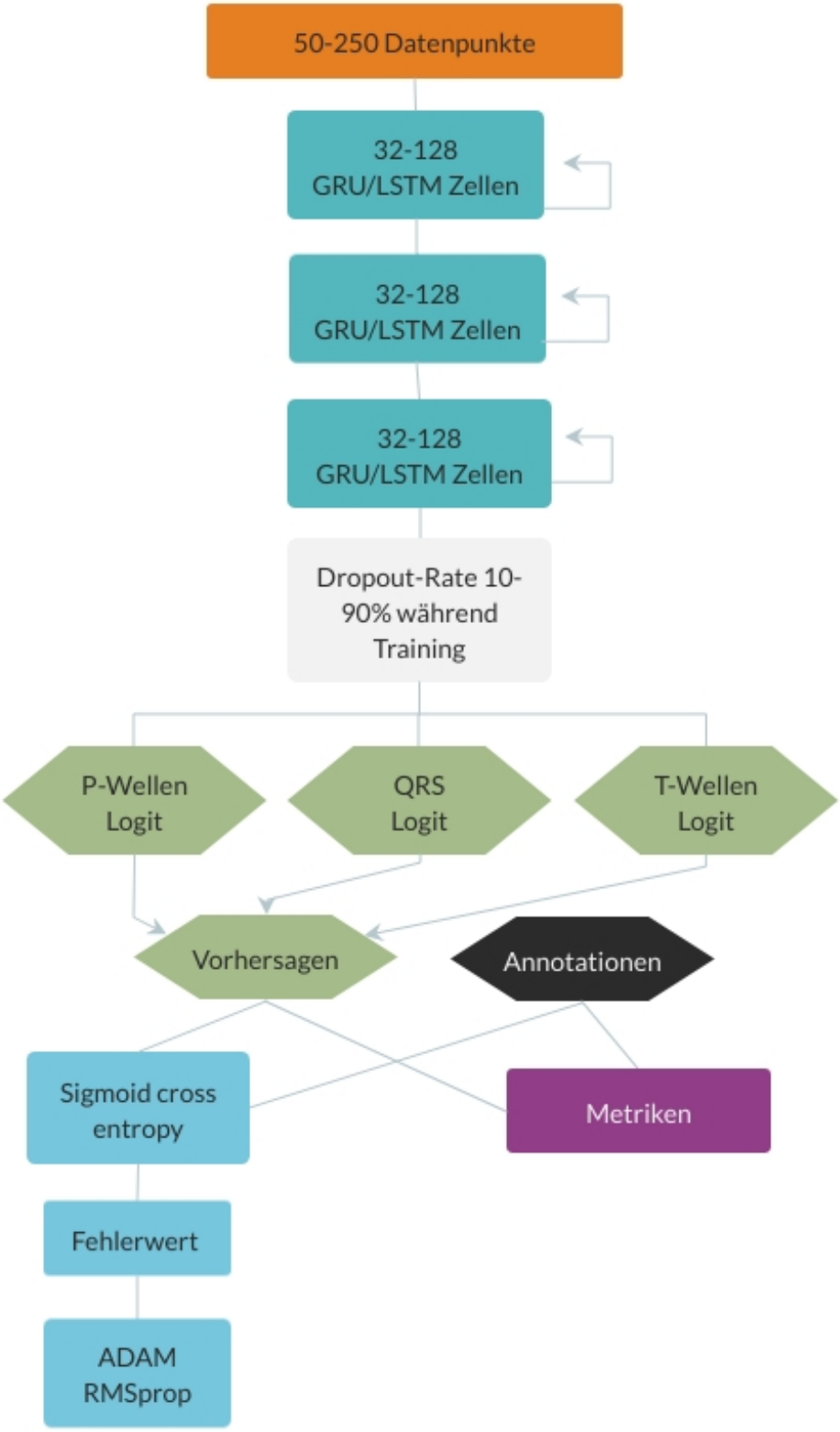


Abbildung 12: Schematische Darstellung des eingesetzten Netzwerkes.

3.3.3 auf Seite 37 vorgestellten Input pipeline nun einem weiteren Preprocessing-Schritt unterzogen. Dieser ermöglicht die variable Einstellung des Einlesefensters, die Assoziation dieses Fensters zu den entsprechenden Annotationen und die Generierung einer zeitlichen Iteration über die Daten.

Es wurde ein „iteratives Zeitfenster“ variabler Breite implementiert, zu dem automatisch die Annotationen für die drei Klassen des zentralen Datenpunktes mit dem aktuellen Ausschnitt assoziiert werden. Dieses Fenster wird iterativ bis ans Ende der verfügbaren Daten geschoben und alle Werte in temporären Matrizen gespeichert. Bei einer Fensterbreite von beispielsweise 100 Datenpunkten, werden also in der ersten Iteration auch die ersten 100 EKG-Werte herangezogen und mit den drei Annotationen des 50. Datenpunktes assoziiert. Da pro CSV-Zeile 1000 Datenpunkte vorhanden sind, kann bei dieser Einstellung die Iteration insgesamt 900 mal stattfinden bis die nächste Zeile geladen werden muss. Da die EKG- und die Annotationsdaten in zwei getrennten Matrizen gespeichert werden, ergeben sich für die Matrizen in diesem Beispiel folgende Dimensionen:

- EKG-Daten-Matrize: (Anzahl Iterationen, Zeitfensterbreite) $\rightarrow$ (900,100)
- Annotationsdaten-Matrize: (Anzahl Iterationen, Zeitfensterbreite, Anzahl von Annotationen) $\rightarrow$ (900,100,3)

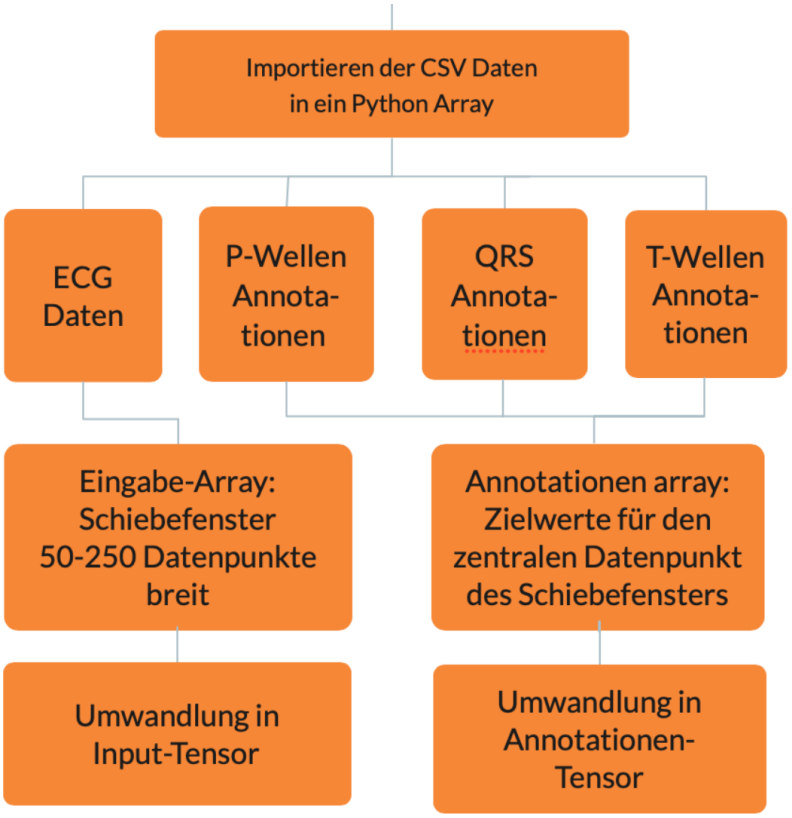


Abbildung 13: Schematische Darstellung des Preprocessings der EKG-Daten in *TensorFlow*.

3.5 Trainieren des Netzwerkes

3.5.1 Fehlerfunktion

In dieser Arbeit wurden sowohl der SCE als auch der MSE untersucht und implementiert. Der MSE ist über die Methode `tf.keras.losses.MeanSquaredError` aufrufbar, während der SCE sich hinter der Methode `tf.nn.sigmoid_cross_entropy_with_logits` verbirgt.

3.5.2 Optimierungsfunktion

Das Ergebnis der Fehlerfunktion wird anschließend an die Optimierungsfunktion als Parameter weitergegeben. Die Optimierungsfunktion liefert ein ausführbares Objekt, welches später in der laufenden Sitzung aufgerufen wird und somit die Logik der Funktion anwendet. Abhängig von der Optimierungsfunktion kann der Entwickler weitere Parameter, wie

zum Beispiel die gewünschte Lernrate, als weitere Variable definieren.

In dieser Arbeit wurden folgende Fehlerfunktionen untersucht:

- ADAM: `tf.train.AdamOptimizer`
- RMSprop: `tf.train.RMSPropOptimizer`

3.6 Evaluierung der Netzwerkleistung

Die Evaluierung der Netzwerkleistung erfolgte anhand der 28 EKGs des Evaluierungsdatensatzes. Diese wurden nicht zum Trainieren des Netzwerkes verwendet und dienten somit der Simulation der Leistungsfähigkeit des Netzwerkes bei unbekannten Testpersonen. In diesem Modus wurde das Dropout deaktiviert, um auf diese Weise das volle Potential aller trainierten Neurone zu nutzen. Die Daten wurden in 6287 Evaluationsschritten (entsprechend der Zahl der CSV-Zeilen) abgearbeitet.

Neben der Bestimmung der statistischen Leistungsfähigkeit im Sinne der Rate der korrekten Abgrenzung der Wellengipfel beziehungsweise der QRS-Komplexe wurde zusätzlich die benötigte Zeit für die Evaluierung bestimmt, um einen Anhaltspunkt für die Fähigkeit zum Echtzeiteinsatz zu erhalten.

Es wurde weder bei dem Training noch bei der Evaluation eine zusätzliche Hardwarebeschleunigung eingesetzt.

3.6.1 Messgrößen

Das beschriebene Netzwerk stellt einen typischen Klassifikator dar, welcher gegebene Daten drei verschiedenen Klassen zuordnet. Die statistische Bestimmung der Leistungsfähigkeit liegt also in der Quantifizierung der Zahl der Abweichungen von den zuvor definierten Annotationen, also der Wahrheit. Generell spricht man bei diesem statistischen Feld von der „Informationswiedergewinnung“ (aus dem englischen Information retrieval).

Beim Vergleich der Ausgabewerte mit den „wahren“ Klassen können insgesamt vier Fälle auftreten, die sich in der sogenannten „Wahrheitsmatrix“ abbilden lassen und in Tabelle 2

auf Seite 45 dargestellt sind.

Stellt man die vier möglichen Fälle zueinander in Bezug, lassen sich verschiedene Messwerte ableiten. (Powers 2011) Folgende wurden in dieser Arbeit verwendet:

Als *Sensitivität* wird die Richtig-Positiv-Rate des Klassifikators bezeichnet, also der Anteil korrekt positiv erkannter Beispiele an der Gesamtzahl positiver Beispiele in Prozent.

$$\text{Sensitivität} = \frac{r_p}{r_p + f_n} \tag{12}$$

Die *Spezifität* oder Richtig-Negativ-Rate wiederum misst den Anteil der korrekt als negativ klassifizierten Beispiele an der Gesamtzahl der negativen Beispiele in Prozent.

$$\text{Spezifität} = \frac{r_n}{r_n + f_p} \tag{13}$$

Die *Präzision* beschreibt die Wahrscheinlichkeit, mit der eine positive Markierung des Netzwerkes auch tatsächlich einem positiven Beispiel entspricht.

$$\text{Präzision} = \frac{r_p}{r_p + f_p} \tag{14}$$

Als *Korrektklassifikationsrate* (englisch Accuracy (Acc)) wird die Korrektheit des Klas-

Tabelle 2: Wahrheitsmatrix

	Annotation ist positiv	**Annotation ist negativ**
Vorhersage ist positiv	richtig positiv	falsch positiv
$(r_p + f_p)$	r_p	f_p
Vorhersage ist negativ	falsch negativ	richtig negativ
$(r_n + f_n)$	f_n	r_n

sifikators bezeichnet, sowohl für negative als auch positive Beispiele in Prozent.

$$\text{Acc} = \frac{r_n + r_p}{\text{alle Fälle}} \tag{15}$$

Die zuvor beschriebenen Maße beeinflussen sich gegenseitig, weshalb zudem Messwerte definiert wurden, die diese Maße miteinander kombinieren. Der wichtigste hierbei ist der sogenannte F1-Wert. Er kombiniert die Präzision und die Sensitivität miteinander durch ein gewichtetes harmonisches Mittel.

$$F_1 = 2 \cdot \frac{\text{Präzision} \cdot \text{Sensitivität}}{\text{Präzision} + \text{Sensitivität}} \tag{16}$$

Die Bestimmung der Messwerte in TensorFlow erfolgt durch bereits vorhandene Funktionen für die Korrektklassifikationsrate (`tf.metrics.accuracy`), Präzision (`tf.metrics.precision`) und Sensitivität (`tf.metrics.recall`). Hingegen müssen die Spezifität und der F1-Wert selbst implementiert werden. Die bereits implementierten Messwerte erhalten als Eingabewerte die Ausgabewerte des Netzwerkes und die korrespondierenden Annotationen.

Zur Berechnung weiterer statistischer Größen bietet TensorFlow Funktionen, welche die Korrekt-Negativ-Rate (tf.metrics.true_negatives) und die Falsch-Positiv-Rate (tf.metrics.false_positives) liefern. Aus diesen Raten lassen sich mittels einfacher Multiplikations-, Divisions- und Additionsbefehle die entsprechenden Ergebnisse für die Spezifität und F1-Werte berechnen.

All diese Werte werden für jeden einzelnen Evaluationsschritt bestimmt und zu den vorherigen Ergebnissen addiert. Am Ende der Evaluierung wird dann der jeweilige Durchschnitt der Einzelwerte berechnet und ausgegeben.

4 Ergebnisse

Im folgenden Abschnitt werden die Ergebnisse dieser Arbeit dargelegt. Neben der Kernaufgabe, dem Finden einer optimalen Netzwerkkonfiguration, wurden zusätzlich Untersuchungen hinsichtlich der Evaluierungsgeschwindigkeit und der Abhängigkeit der Netzwerkleistung von der zum Training zur Verfügung stehenden Zahl an EKG-Beispielen unternommen.

4.1 Vergleich verschiedener Hyperparameter

Im Rahmen einer sogenannten Hyperparameterstudie wurde nach den optimalen Einstellungen verschiedener Netzwerk- und Trainingsparameter gesucht. Dieser, in der Regel rein empirische Prozess, benötigt viel Rechenzeit und ist deshalb häufig der limitierende Faktor in der Entwicklung eines ANN.

Da es sich hierbei um ein Problem mit vielen Variablen handelt, wurde in den einzelnen Disziplinen jeweils nur die Tendenz und nicht die Werte an sich untersucht.

4.1.1 Breite des Lesefensters

Es wurde zunächst begonnen, die Breite des iterativen Lesefensters zu untersuchen. Hierbei spielten physiologische Überlegungen eine Rolle: so dauert ein gesamter Erregungszyklus des normalen Herzens etwa 600 ms. Bei einer Abtastrate von 250 Hz betrug demzufolge die zeitliche Auflösung der verwendeten EKG-Signale 4 ms. Ein durchschnittlich langer Erregungszyklus müsste also etwa 150 Datenpunkte im EKG-Signal ausmachen. Es wurden deshalb Fensterbreiten von 50 und 100 Schritten über und unter diesem errechneten Wert untersucht.

Wie im Abbildung 14 auf Seite 48 zu sehen ist, stellte sich heraus, dass die Einstellungen mit einem 50 und 100 Datenpunkte breiten Lesefenster beim QRS-Komplex nahezu ebenbürtig sind. Jedoch lieferte die Einstellung mit einem 50 Datenpunkte breiten Lesefenster in der Gesamtschau bessere Ergebnisse. Es stellte sich generell eine negative Leistungstendenz bei zunehmender Fensterbreite dar.

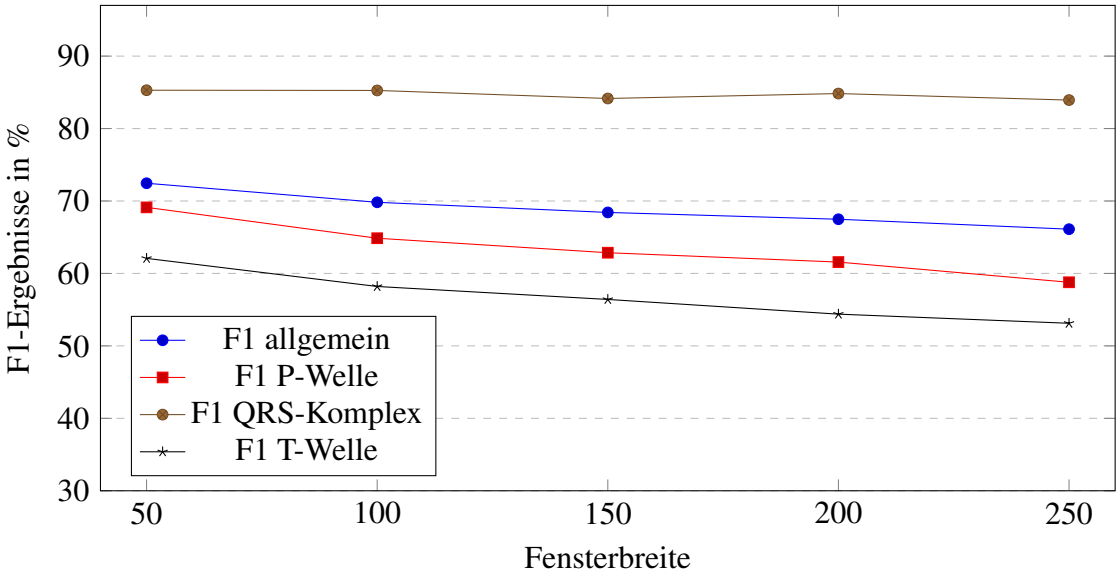


Abbildung 14: Diagramm zum Vergleich der F1-Ergebnisse in Abhängigkeit von der Fensterbreite. Das Netzwerk bestand bei allen Untersuchungsläufen aus 3 Schichten zu je 128 LSTM-Neuronen und wurde mit einem Trainingsdropout von 50%, PRT1-NF als Datensatz und einer RMSprop-Optimierungsfunktion für 7 Epochen trainiert.

4.1.2 Zahl und Art der verwendeten Neuronen und deren Schichtung

Als nächstes wurde das Layout des Netzwerkes untersucht. Hierbei ging es zum einen um die Betrachtung der Anzahl der Neuronen pro Schicht, als auch der Zahl der verwendeten Schichten betrachtet. Beide Parameter wurden schrittweise erhöht, bis ein Abfall der Netzwerkleistung zu verzeichnen war. So wurden bis zu 4 Schichten mit 32 bis zu 128 Neuronen pro Schicht untersucht und die Leistungsfähigkeit im Form des allgemeinen F1-Werts analysiert. Aufgrund der begrenzten Rechenkapazität wurden Netzwerkmodelle mit mehr als 128 Neuronen je Schicht nicht untersucht.

Wie in Abbildung 15 auf Seite 49 dargestellt, nahm die Leistungsfähigkeit bei einer Vergrößerung der Zahl der verwendeten Schichten kontinuierlich bis 3 zu, flacht dann jedoch bei einem Hinzufügen einer vierten Schicht ab. Die Ergebnisse stiegen mit der Zahl der verwendeten Neurone pro Schicht im untersuchten Spektrum kontinuierlich an. Ein lokales Optimum stellte die Kombination aus drei Schichten zu je 128 LSTM-Neuronen dar.

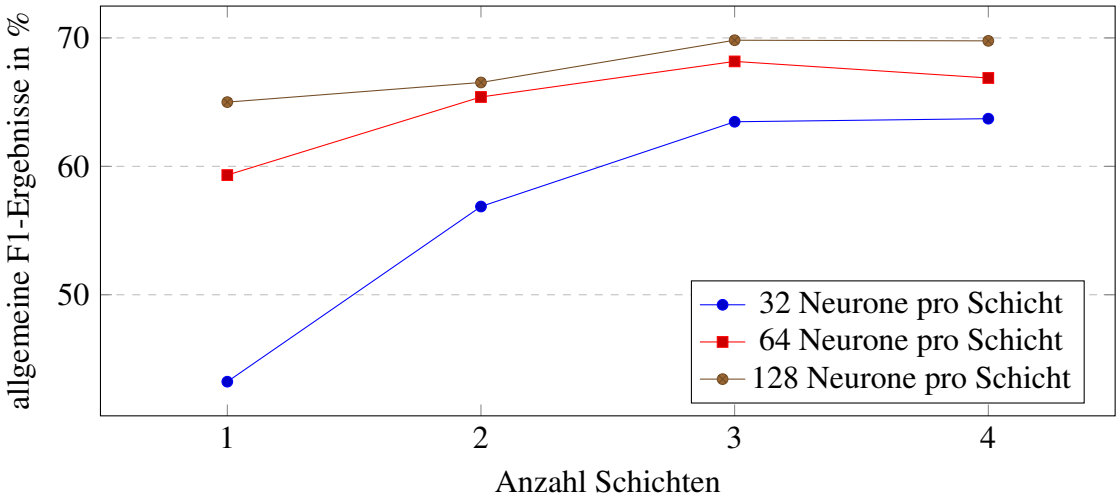


Abbildung 15: Diagramm zum Vergleich der F1-Ergebnisse in Abhängigkeit der Anzahl der Schichten und Zahl von LSTM Neuronen im RNN. Alle Untersuchungsläufe wurden mit einem Trainingsdropout von 50%, einer Lesefensterbreite von 100, PRT1-NF als Datensatz und einer RMSprop- Optimierungsfunktion für 7 Epochen trainiert.

Zudem wurde ein Vergleich zwischen GRU- und LSTM-Neuronen gestellt. Hierzu wurden Netzwerke, bestehend aus drei Schichten zu jeweils 128 Neuronen, untersucht und deren Leitungsfähigkeit in Form des allgemeinen F1-Werts verglichen. Wie in Tabelle 3 dargestellt, zeigt sich eine allgemeine Überlegenheit des GRU-Neurons gegenüber Netzwerken mit LSTM heraus.

Tabelle 3: Vergleich der Leistungsfähigkeit von Netzwerken mit GRU oder LSTM als Neuronentyp. Es wurden Netzwerke, bestehend aus drei Schichten zu je 128 Neuronen, untersucht. Das Training erfolgte mit einem Dropout von 50%, einer Lesefensterbreite von 100 und einer RMSprop-Optimierungsfunktion für 7 Epochen trainiert.

		F1 Allgemein	**F1 P-Welle**	**F1 QRS-Komplex**	**F1 T-Welle**
PRT1-NF	GRU	72,99%	68,12%	87,64%	62,20%
	LSTM	69,82%	64,86%	85,26%	58,20%
PRT3-NF	GRU	84,39%	81,86%	93,14%	78,29%
	LSTM	82,93%	80,16%	92,44%	76,20%

4.1.3 Optimierungsfunktion

Anschließend erfolgte ein Vergleich zwischen den Optimierungsalgorithmen ADAM und RMSprop. Wie in Abschnitt 3.5.2 auf Seite 43 geschildert, sind beide Algorithmen in der Lage, ihre inneren Parameter, insbesondere die Lernrate, laufend anzupassen. Dadurch sinkt die Zahl der zu betrachtenden Hyperparameter deutlich.

Ein Untersuchungsaspekt beschäftigte sich damit, welcher Algorithmus schneller beginnt, das Netzwerk in die richtige Richtung zu optimieren. Im weiteren Verlauf war von Interesse, welcher letztendlich zur besseren Netzwerkleistung führt. Wie in Abbildung 16 dargestellt, begann RMSprop früher mit dem Erzielen besserer Leistungen als ADAM. Nach wenigen Trainingsschritten jedoch erzielten beide gleichwertige Ergebnisse, die im weiteren Verlauf zugunsten von ADAM stark auseinander drifteten. Ein detaillierter Vergleich der erzielten Werte in Tabelle 4 auf Seite 51 dargestellt.

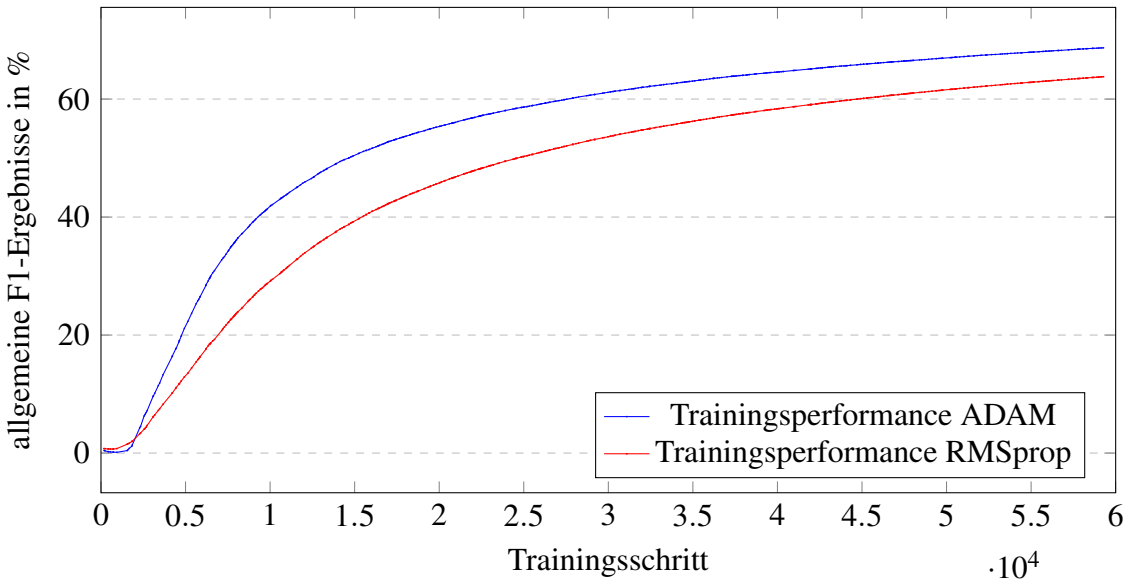


Abbildung 16: Diagramm zum Vergleich der F1-Ergebnisse bei Einsatz von ADAM und RMSprop als Optimierungsalgorithmen. Alle Untersuchungsläufe wurden mit einem Trainingsdropout von 50%, einer Lesefensterbreite von 100, PRT1-NF als Datensatz und einem Netzwerk, bestehend aus 3 Schichten zu je 128 LSTM-Neuronen, für 7 Epochen trainiert.

Tabelle 4: Vergleich der F1-Ergebnisse abhängig von ADAM oder RMSprop als Optimierungsalgorithmus. Bei beiden Untersuchungen wurden Netzwerke, bestehend aus drei Schichten zu je 128 Neuronen, mit einer Lesefensterbreite von 100, einer Dropoutrate von 50% und PRT1-NF als Datensatz für 7 Epochen trainiert.

	F1 Allgemein	**F1 P-Welle**	**F1 QRS-Komplex**	**F1 T-Welle**
ADAM	73,46%	68,67%	87,36%	63,15%
RMSprop	69,82%	64,86%	85,26%	58,20%

4.1.4 Dropout

Die Untersuchung der Dropout-Rate während des Trainings war ein weiterer Schwerpunkt. Ausgehend von einer Wahrscheinlichkeit von 50%, wurden zusätzlich die Ergebnisse bei 10%, 25%, 75% und 90% analysiert. Wie in Abbildung 17 dargestellt, steigt die Netzwerkleistung mit einer sinkenden Wahrscheinlichkeit, dass ein Neuron vom Dropout ausgeklammert wurde. Das äußert sich in einer steigenden Dropout-Zahl.

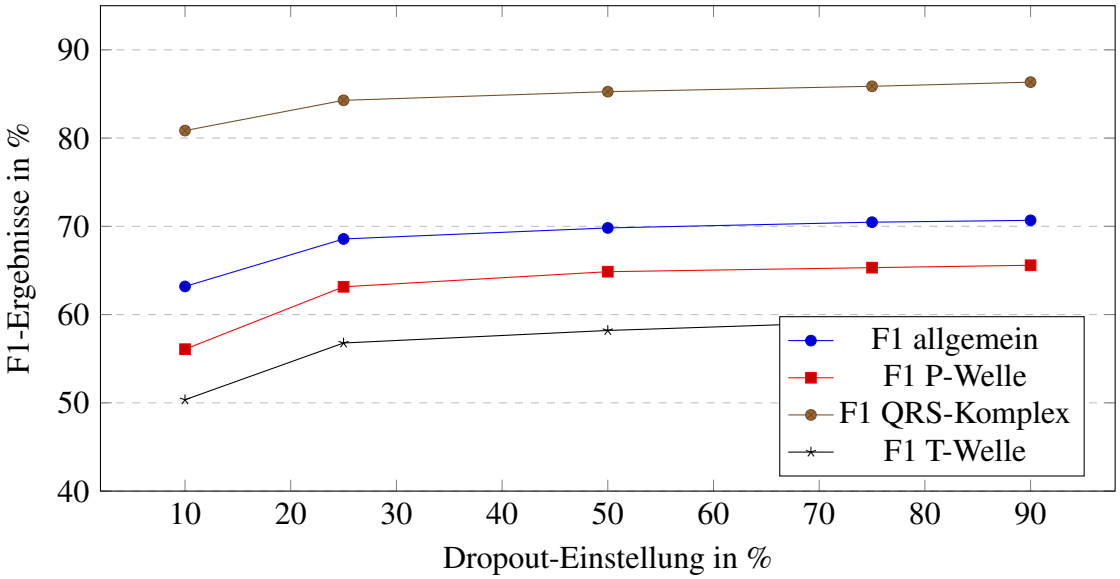


Abbildung 17: Diagramm zum Vergleich der F1-Ergebnisse in Abhängigkeit vom Dropout. Alle Untersuchungsläufe wurden mit einer Lesefensterbreite von 100 Datenpunkten, RMSprop, PRT1-NF als Datensatz und einem Netzwerk bestehend aus 3 Schichten zu je 128 LSTM-Neuronen für 7 Epochen trainiert.

4.1.5 Einsatz des Savitzky-Golay Filters und Definition des Zielfensters

Außerdem erfolgte die Untersuchung des Einsatzes eines Savitzky-Golay-Filters im Preprocessing (Einstellungen wie in Abschnitt 3.2.3 auf Seite 33 beschrieben) und die Definition der Breite des Zielfensters.

Wie in Abbildung 18 dargestellt, zeigt sich ein deutlicher Vorteil für den Einsatz ungefilterter Daten und einer breiteren Definition des Zielbereichs der Annotationen.

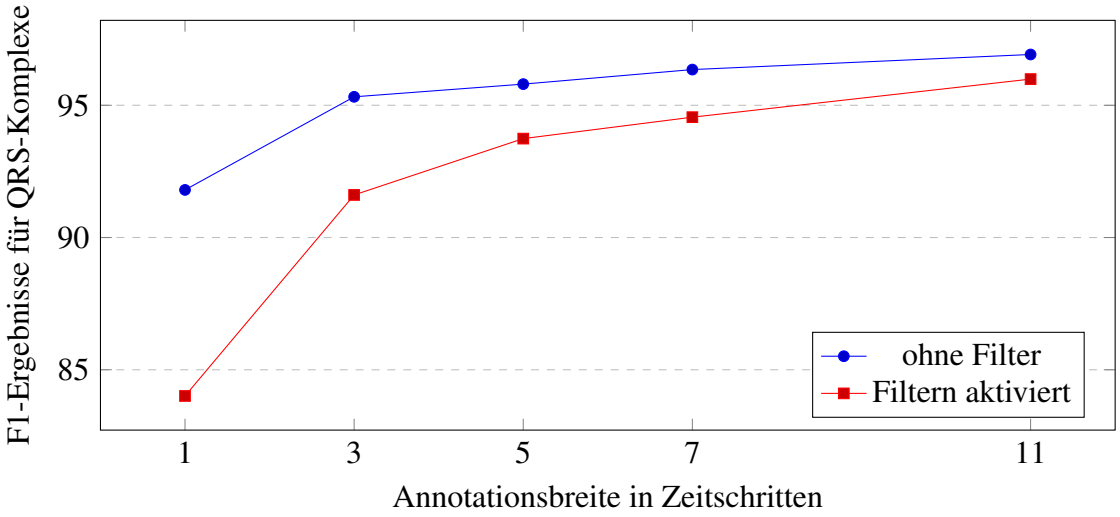


Abbildung 18: Diagramm zum Vergleich der F1-Ergebnisse bei QRS-Komplexen in Abhängigkeit von der Annotationsbreite und den Filtereinstellungen. Alle Untersuchungsläufe erfolgten mit einer Lesefensterbreite von 100 Datenpunkten, RMSprop als Optimierungsalgorithmus, einer Dropout-Rate von 50% und einem Netzwerk, bestehend aus 3 Schichten zu je 128 LSTM-Neuronen, für 10 Epochen trainiert.

4.2 Evaluierung des Zusammenspiels aller Hyperparameter

Zusammenfassend lässt sich aus den oben genannten Untersuchungen eine theoretisch optimale Gesamtkombination aus folgenden Hyperparametern ableiten:

- Neuronen pro Schicht: 128
- Schichten: 3
- Neuronentyp: GRU

- Lesefensterbreite: 50
- Dropout-Rate während des Trainings: 90% (i.e. Ausfallwahrscheinlichkeit eines Neurons = 10%)
- Optimierungsalgorithmus: ADAM
- Filterung der Daten: nein

Netzwerke mit diesen Einstellungen, welche mit ungefilterten Daten mit Annotationsbreiten von 1 und 3 trainiert wurden, erzielten die in Tabelle 5 festgehaltenen Leistungsparameter. Hier erfolgte das Training der Netzwerke für insgesamt 10 Epochen.

Tabelle 5: Vergleich der Netzwerkleistung bei Trainingsläufen mit den zuvor ermittelten optimalen Trainingsparametern.

		Allgemein	$P_{\textbf{Spitze}}$	$QRS_{\textbf{Mitte}}$	$T_{\textbf{Spitze}}$
Präzision	PRT1-NF	87,94%	85,16%	93,81%	84,23%
	PRT3-NF	92,21%	90,25%	96,50%	89,80%
Sensitivität	PRT1-NF	79,81%	76,66%	92,65%	70,98%
	PRT3-NF	87,56%	85,93%	95,42%	82,00%
F1-Leistung	PRT1-NF	83,68%	80,69%	93,22%	77,04%
	PRT3-NF	89,83%	88,04%	95,96%	85,72%

4.3 Vergleich Signallänge zu benötigter Evaluierungszeit

Um zu beurteilen, ob die Auswertungsgeschwindigkeit für Echtzeit-Patientenüberwachungsszenarien ausreichend ist, wurde die Signaldauer mit der verstrichenen Auswertungszeit verglichen. Wie aus Tabelle 6 auf Seite 54 hervorgeht, liegt das durchschnittliche Auswerteverhältnis bei etwa 10,87. Dies zeigt, dass die Auswertung fast elf mal schneller ist als die tatsächliche Signaldauer.

Tabelle 6: Bewertung der Evaluierungsgeschwindigkeit bei allen sechs Trainingseinstellungen. PRT1 beschreibt die ursprünglichen Annotationseinstellungen der QT-Datenbank, während PRT3 eine zusätzliche positive Annotation in beiden Richtungen darstellt. Die Zeit ist als `hh:mm:ss` formatiert.

	Signaldauer	**benötigte Evaluierungszeit**	**Verhältnis**
PRT1-NF	06:59:08	00:38:32	10,88
PRT3-NF	06:59:08	00:38:35	10,86
		Durchschnitt:	10,87

4.4 Vergleich Menge verwendeter EKGs

Zu guter Letzt wurde analysiert, wie das Netzwerk in Abhängigkeit von der Menge der verwendeten Trainings-EKGs funktioniert. Dazu wurde das Netzwerk mit Trainingsdatensätzen trainiert, die 2 bis 64 EKGs in einer Sequenz von zwei mit der Potenz von 1 bis 6 enthielten. Entsprechend beinhalteten die Evaluierungsdatensätze die restlichen verfügbaren EKGs aus der QT-Datenbank. Zur Anwendung kamen die originalen Annotationen der QT-Datenbank (entspricht PRT1-NF in den vorherigen Untersuchungen).

Es ergab sich die einer Sigmoidalfunktion ähnliche Steigerung der Leistungsergebnisse des Netzwerks bei Zunahme der zur Verfügung stehenden EKGs im Trainingsdatensatz.

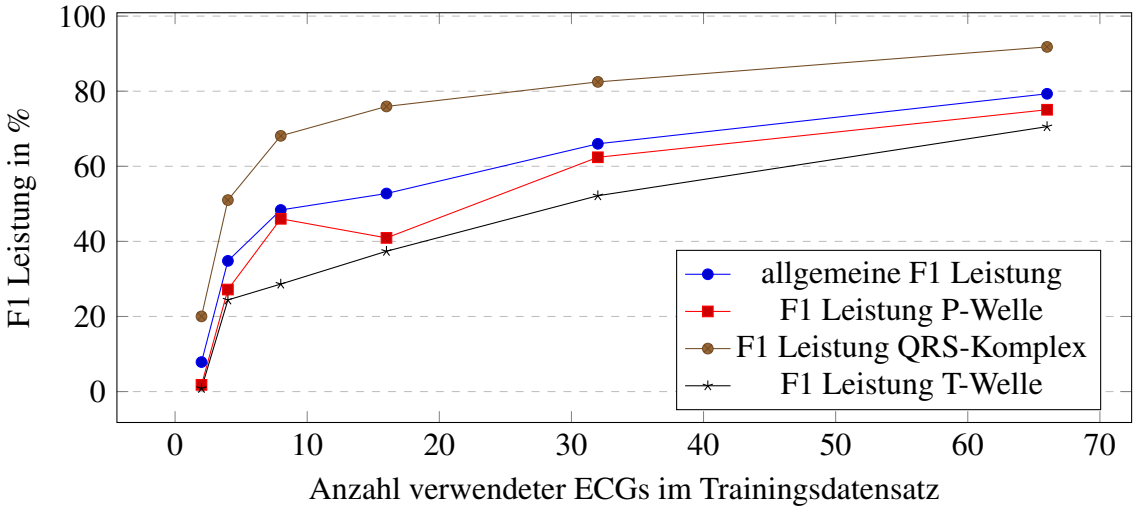


Abbildung 19: Vergleich der Netzwerkleistung bei unterschiedlicher Anzahl der für das Training verwendeten EKGs. Als Referenzparameter diente die F1-Leistung für allgemeine und wellen-weise Vorhersagen.

4.5 Zusammenfassung der Methodik und Ergebnisse

Die besten Ergebnisse wurden mit einem dreischichtigen Netzwerk aus Schichten zu je 128 GRU Neuronen mit einer anschließenden Logits-Schicht bestehend aus 3 Neuronen erzielt. Die Ausgabe der RNN-Schichten wurde mit einem Dropout 10% versehen. Während des Trainings wurde der Vorhersagefehler mittels „sigmoid cross entropy“ berechnet und mittels „ADAM“ iterativ reduziert. Es stellte sich heraus, dass ungefilterte Daten zu besseren Ergebnissen geführt haben als bei Trainingsläufen mit gefilterten Signalen.

70% aus der Gesamtzahl an EKGs der QT-Datenbank wurden zufällig für den Trainingsdatensatz ausgewählt, welches 66 EKG-Beispielen von je 15 Minuten Länge entspricht. Es erfolgte anschließend eine Zerstückelung der Signale in Ausschnitte zu je 100 Datenpunkten, welche im Sinne eines sich um je ein Datenpunkt vorschiebenden Rasters berechnet und dann dem Netzwerk zum trainieren präsentiert wurden. Als Trainingsziel wurden die Annotationen der Wellenspitzen, beziehungsweise der Mitte der QRS-Komplexe, verwendet und in variabler Breite implementiert. Insgesamt wurde im Sinne von Epochen 10 Mal über den Trainingsdatensatz iteriert bevor die Evaluierung der Netzwerkleistung mit den anderen 30% der Daten erfolgte.

Wie in Tabelle 5 auf Seite 53 genau aufgeschlüsselt zu sehen, erzielt das Netzwerk insbesondere bei den QRS-Komplexen hohe F1-Werte über 90%. Bei der hoch-präzisen Annotationseinstellung PRT1-NF ohne Toleranz wird ein F1-Wert von 93,22% erzielt. Lässt man eine Toleranz von $\pm 4ms$ wie im Datensatz PRT3-NF zu, so erzielt das Netzwerk bei den QRS-Komplexen einen F1-Wert von 95,96%.

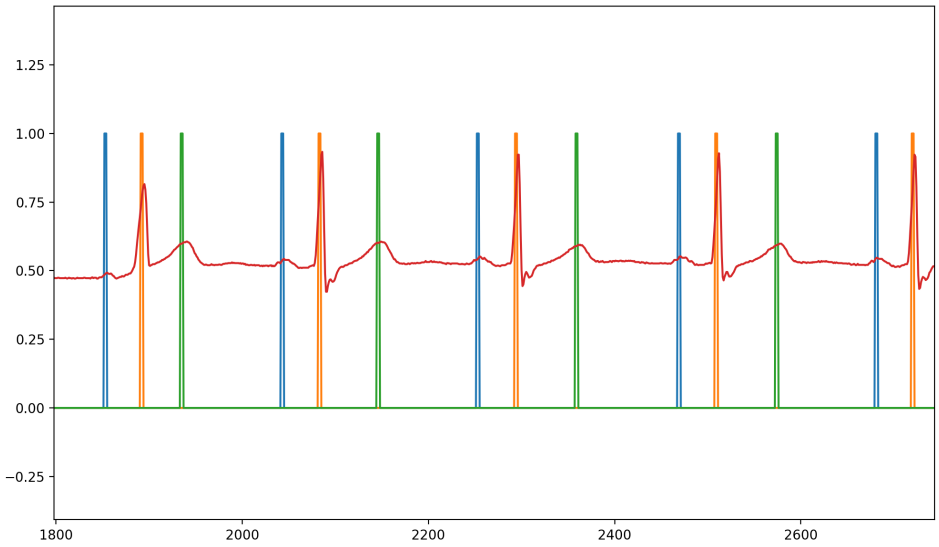


Abbildung 20: Visualisierung der Annotationsleistung des Netzwerkes. Die rote Kurve stellt das zu bewertende EKG-Signal dar. Die blaue Kurve repräsentiert die Aktivität des Logit Neurons, welches die P-Welle markieren soll. Korrespondierend verhalten sich die orange Kurve für den QRS-Komplex und die grüne Kurve für die T-Welle. Die Y-Achse stellt die berechnete Wahrscheinlichkeit des Netzwerkes in % dar, während die X-Achse der Zeit (je Zeitschritt 4ms) entspricht.

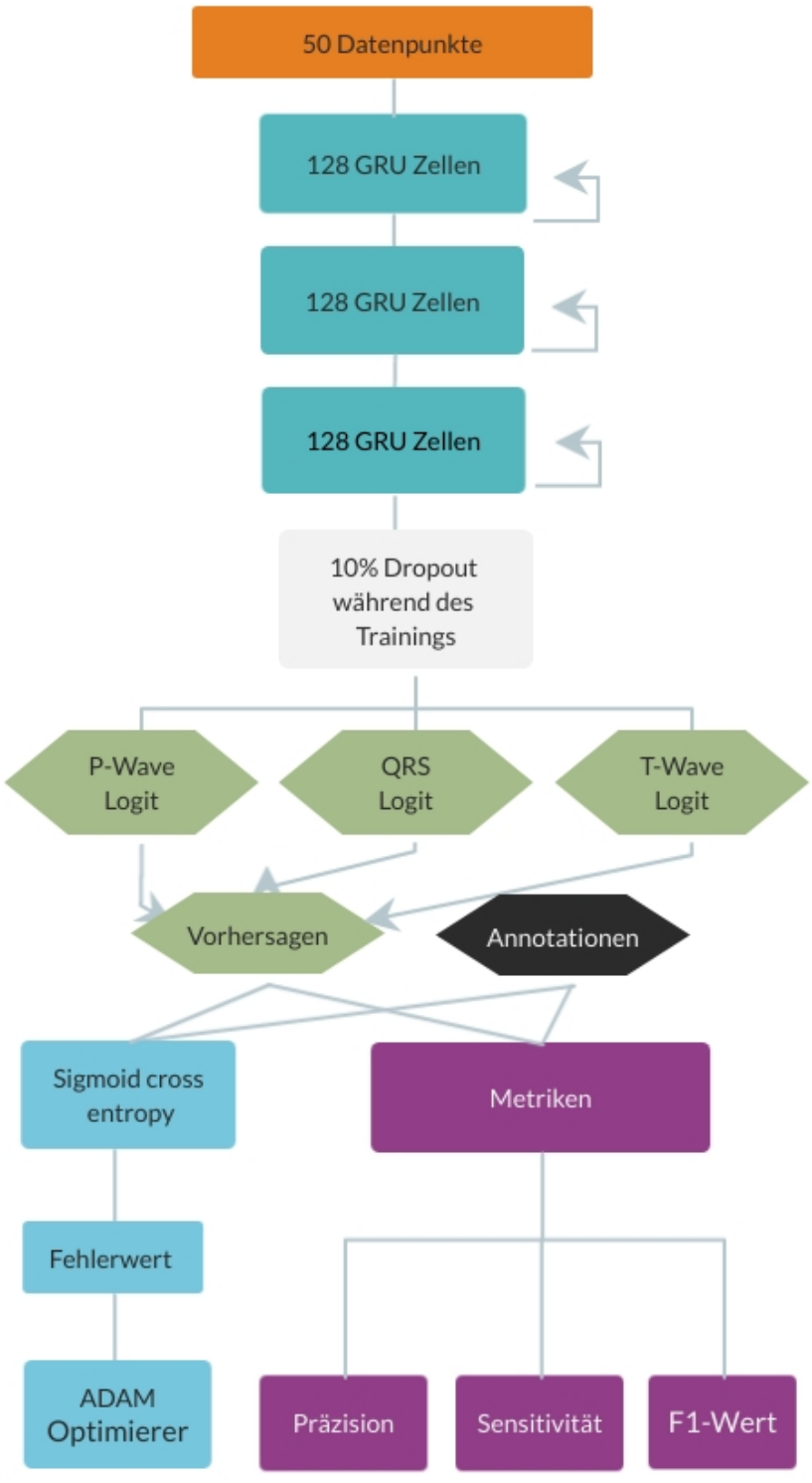


Abbildung 21: Schematische Darstellung der erfolgreichsten Netzwerkkonfiguration. Die hellblauen Punkte entsprechen den Berechnungen, welche während des Trainings stattfinden während die lilafarbenen denen der Evaluierung entsprechen.

5 Diskussion

5.1 Leistungsvergleich mit klassischen EKG-Analyse-Algorithmen

Obwohl das Netzwerk bereits eine insgesamt sehr zufriedenstellende Leistung erzielt, übertrifft es nicht die konventioneller Methoden ohne den Einsatz von ANNs. Deren positive Vorhersagewerte (Präzision) für die Erkennung von QRS-Wellen erreichen bis zu 99,46% [13, 14] und 99,88% [14] in der QT-Datenbank, und auch bei der P- und T-Wellenspitzenerkennung übertreffen sie die Leistung des entwickelten Netzwerks. Insbesondere im Bereich der QRS-Komlexe und der T-Wellenspitzenerkennung liegen in den untersuchten Arbeiten noch deutliche Unterschiede von ca. 3-3,5% bei den QRS-Komplexen und ca. 8% bei der T-Welle vor. Die Leistung bei den P-Wellen weist hingegen lediglich einen Unterschied von ca. 1% auf.

Tabelle 7: Vergleich der Präzision des vorgeschlagenen Netzwerks bei der Spitzenerkennung mit dreier Arbeiten ohne ANN-Ansatz. (Das Netzwerk wurde entsprechend der in Abschnitt 4.2 auf Seite 52 beschriebenen Einstellungen für 10 Trainingsepochen trainiert). Leistungswerte der anderen Ansätze extrahiert aus Martínez et al. 2004.

	$P_{\textbf{Spitze}}$	$QRS_{\textbf{Mitte}}$	$T_{\textbf{Spitze}}$
PRT1-NF	85,16%	93,81%	84,23%
PRT3-NF	90,25%	96,50%	89,80%
Martínez et al. (2004)	91,03%	99,88%	97,79%
Laguna, Jané et al. (1994)	91,17%	N/A	97,71%
Moody und Mark (1983)	N/A	99,46%	N/A

Eine Erklärung für die Überlegenheit der klassischen Methoden lässt sich auf zweierlei Weise darlegen:

ANN beinhalten eine Vielzahl an trainierbaren Parametern, welche zusammengenommen der Lösung eines Problems dienen. Ist die Zahl der Parameter zu groß oder zu klein, so wird das Ergebnis des Netzwerkes schlechter ausfallen. Diese Phänomene werden als Über- oder Unteranpassung (aus dem englischen „overfitting“ und „underfitting“) (Reed

und Marks ll 1997, Seite 241) bezeichnet und können beispielsweise durch Dropout umgangen werden. (Srivastava et al. 2014) Eine ausführliche Erklärung über Ansätze zur Verminderung der Über- und Unteranpassung ist in Abschnitt 5.4 auf Seite 65 zu finden.

Insbesondere die Überanpassung lässt sich bei eher simplen mathematischen Problemen beobachten, bei denen klassische mathematische Formeln in der Regel deutlich besser als falsch dimensionierte ANN funktionieren. Wird die Aufgabe jedoch komplexer, nimmt die Wahrscheinlichkeit zum Finden einer optimalen mathematischen Funktion ab und ANN beginnen, bessere Leistungen zu erzielen.

Wie in Abschnitt 1.2.3 auf Seite 10 bereits dargelegt, stellen viele klassische EKG-Erkennungsalgorithmen eine geschickte Aneinanderreihung relativ einfacher und effektiver mathematischer Operationen in Kombination mit leicht programmierbaren physiologischen Regeln dar. Die Wahrscheinlichkeit, durch diese „klassischen“ Methoden relativ nah an das Optimum zu gelangen, ist bereits sehr hoch.

Hinzu kommt der zweite Nachteil bei der Verwendung eines ANN im Vergleich zu solch bewährten Konkurrenten: die langwierige Hyperparameterstudie. Dieser empirische und zeitaufwändige Prozess stellt einen der limitierenden Faktoren für die Entwicklung der ANN dar und begrenzt auch die Möglichkeiten, mittels eines ANN auf eine optimale Konfiguration zu stoßen.

Auf die Nuancen und potentielle Verbesserungsmöglichkeiten der hier stattgefundenen Hyperparameterstudie wird in Abschnitt 5.5 auf Seite 67 eingegangen.

Zusammenfassend lässt der Vergleich mit anderen klassischen EKG-Erkennungsmethoden folgende zwei Schlüsse zu: Die Aufgabe der Wellenspitzenerkennung in einem EKG könnte durch klassische mathematische Ansätze leichter zu lösen sein. Oder eine ebenbürtige Lösung durch den Einsatz von ANN ist prinzipiell möglich, jedoch ist die richtige Kombination im Rahmen der hier durchgeführten Hyperparameterstudie nicht aufgetreten, oder es wären mehr Trainingsepochen nötig gewesen.

5.2 Leistungsvergleich mit ähnlichen Arbeiten mit ANN-Ansatz

Neben dem Vergleich mit etablierten Methoden, sollte auch eine Gegenüberstellung zu ähnlichen Arbeiten mit ANN-Ansatz in der Literatur erfolgen. Allerdings haben sich bisher keine der in der Literatur zu findenden Arbeiten auf die alleinige Erfassung der lokalen Wellenspitzen beziehungsweise QRS-Komplexe unter Verwendung von ANN konzentriert. Diese Tatsache macht einen diesbezüglichen Vergleich unmöglich.

5.2.1 CampsNet

Die erste untersuchte Arbeit von Camps et al. (2018) nutzte eine tiefe CNN-Struktur (im folgenden als *CampsNet* bezeichnet), um QRS-Komplexe in EKG-Signalen zu erkennen. Die Autorinnen geben an, ähnliche Ergebnisse wie andere klassische Methoden der QRS-Erkennung zu erzielen. Leider sind die in dieser Quelle angegebenen Leistungsparameter nicht mit denen in der vorliegenden Arbeit erhobenen vergleichbar. Die Parameter werden als zeitliche Abweichung von den Annotationen in Millisekunden (ms) dargelegt. Auf die technischen Unterschiede zwischen der vorliegenden Arbeit und *CampsNet* wird in Abschnitt 5.3 auf Seite 61 eingegangen.

5.2.2 SegNet

„ECG-SegNet“ von Abrishami et al. (2018) (im folgenden als *SegNet* bezeichnet) kommt dem Ziel dieser Arbeit am nächsten, obwohl der Schwerpunkt der Autoren auf der Abgrenzung der gesamten Wellen und nicht nur ihrer Spitzen/Zentren lag. Hierzu wurde ebenfalls eine Form des RNN zum Einsatz gebracht und mit der QT-Datenbank trainiert. Ein Vergleich der Ergebnisse des vorgeschlagenen Netzwerkes mit *SegNet* ist in Tabelle 8 auf Seite 62 zu finden. Jedoch ist anzumerken, dass der Vergleich der Leistungen zweier unterschiedlicher Ziele keine Aussagekraft birgt.

Es ist zu erkennen, dass das hier präsentierte Netzwerk trotz deutlich schmalerer Zieldefinitionen ähnliche Ergebnisse wie *SegNet* erzielt. So übertrifft das entwickelte Netzwerk bei der QRS-Komplex-Erkennung sowohl in der Präzision als auch im F1-Wert *SegNet* um ca. 2-2,5%. Bei den P- und T-Wellen bietet *SegNet* jedoch einen Vorteil und übersteigt das hier entwickelte Netzwerk beim F1-Wert der P-Welle um ca. 3% und der T-Welle um

ca. 5%.

5.3 Vergleich von Modellarchitektur und Trainingsmethoden zwischen CampsNet und SegNet

Trotz der schlechten Vergleichbarkeit der Metriken von *CampsNet* und *SegNet* sind die Unterschiede zwischen der dort angewandten Trainingsmethodik und den Netzwerkstrukturen und denen in der vorliegenden Arbeit verwendeten Methoden interessant.

5.3.1 Modellarchitektur

Während *CampsNet* zwei tiefe CNN-Netze (auch dCNN genannt - englisch für „deep CNN“) als Netzwerkarchitektur verwendete, wurden in dieser Arbeit und in *SegNet* mehrschichtige RNN eingesetzt.

Das dCNN bezeichnet ein Netzwerk, in dem eine Vielzahl von CNN-Schichten hintereinander geschaltet sind und durch den Einsatz von zusätzlichen Zwischenschichten stabilisiert werden. Diese Vorgehensweise wurde unter anderem von Szegedy et al. (2015) vorgestellt. Sie ist erst durch die seit wenigen Jahren verfügbare Menge an erschwinglicher Rechenkapazität praktisch umsetzbar. Durch die Vielzahl an Schichten ist es möglich, beeindruckende Leistungen im Bereich der Bildanalyse zu erzielen. Voraussetzung dafür ist das Vorhandensein einer ausreichenden Menge an Trainingsbeispielen und der korrekte Umgang mit dem Problem der Überanpassung.

Deshalb ist die Signalverarbeitung eines EKGs mithilfe von dCNN in *CampsNet* durchaus nicht abwegig. Allerdings erscheint der Verzicht auf das zeitliche Bewusstsein eines RNNs bei einer solchen Datenquelle zunächst als untypisch. Es ist zu vermuten, dass das aus insgesamt 24 CNN- und 10 vollständig verbundenen Schichten bestehende dCNN in *CampsNet* durch die Vielzahl an Neuronen in der Lage ist, auch ohne zeitliches Bewusstsein gute Vorhersagen zu treffen. Der größte Vorteil eines dCNNs liegt hier mit hoher Wahrscheinlichkeit an der durch die gefaltete Struktur des Netzwerkes relativ hohen Resistenz gegenüber Störquellen und Rauschen.

Die in *SegNet* eingesetzte Form des RNN variiert leicht zu der hier eingesetzten Form.

Tabelle 8: Vergleich der Leistung des Netzwerks nach 10 Trainingsepochen mit den in Abschnitt 4.2 auf Seite 52 beschriebenen Einstellungen mit den Ergebnissen in *SegNet*. Die *SegNet*-Werte wurden direkt aus der ursprünglichen Quelle (Abrishami et al. 2018) extrahiert.

		P_{Spitze}/ P-Welle	QRS_{Mitte}/ QRS-Komplex	T_{Spitze}/ T-Welle
Präzision	PRT1-NF	85,16%	93,81%	84,23%
	PRT3-NF	90,25%	96,50%	89,80%
	SegNet	92%	94%	92%
Sensitivität	PRT1-NF	76,66%	92,65%	70,98%
	PRT3-NF	85,93%	95,42%	82,00%
	SegNet	90%	95%	92%
F1-Leistung	PRT1-NF	80,69%	93,22%	77,04%
	PRT3-NF	88,04%	95,96%	85,72%
	SegNet	91%	94%	91%

Es handelt sich bei *SegNet* um ein zweischichtiges bidirektionales rekurrentes neuronales Netzwerk (BD-RNN) mit je 250 bidirektionalen LSTM-Neuronen pro Richtung in der ersten, respektive 125 in der zweiten Schicht. BD-RNNs, ursprünglich von Schuster und Paliwal (1997) beschrieben, haben den Vorteil, nicht nur Kontakt zu zurückliegenden Netzwerkzuständen halten zu können, sondern auch zu zukünftigen. Dieser Vorteil könnte jedoch bei der Verwendung in Echtzeitszenarien von Nachteil sein. Das fällt vor allem dann ins Gewicht, wenn das Netzwerk einen zu lange dauernden Pufferspeicher benötigt, um seine Vorhersageleistung zu erzielen. Dennoch gibt es Beispiele in der Literatur, die den Einsatz von BD-RNNs in Echtzeitszenarien erfolgreich implementieren. (Farsad und Goldsmith 2018)

Während die Abgrenzung von Strukturen in EKG-Signalen eher einen Nebenschauplatz der ANN-basierten EKG-Interpretation darstellt, gibt es eine Vielzahl von Beispielen erfolgreicher Anwendung von ANN im Bereich der Arrhythmieerkennung.

Einige dieser Arbeiten wenden ebenfalls bereits mehrschichtige RNN an, beispielsweise Yildirim (2018). Damit leistet der in der vorliegenden Arbeit und *SegNet* beschriebene

Ansatz einen weiteren Beitrag zu dieser Entwicklung.

Einen interessanten Anstoß für weitere Forschung und Verbesserung dieser Netzwerkstrukturen stellt die Arbeit von Andersen et al. (2019) dar. Er kombiniert CNN- und RNN-Strukturen miteinander, um eine typische Rhythmusstörung des Herzens (Vorhofflimmern) zu erkennen. Die Zielsetzung hierbei ist es, die Vorteile beider Netzwerktypen zu kombinieren und damit zu einer noch besser funktionierenden Symbiose zu gelangen. Die Autoren beschreiben sehr gute Erkennungsraten und den Vorteil ohne aufwendige Vorverarbeitungsschritte (wie beispielsweise in *SegNet* beschrieben) auszukommen.

Zusammenfassend ist festzustellen, dass das in dieser Arbeit beschriebene Netzwerk bezüglich der Architektur und der verwendeten Neuronentypen auf etablierte Methoden zurückgreift. Während auf die Wahl der weiteren Hyperparameter in Abschnitt 5.5 auf Seite 67 eingegangen wird, kann insbesondere der Ansatz der Kombination von CNN- und RNN-Strukturen als eine beachtenswerte zukünftige Erweiterung der derzeitigen Architektur angesehen werden.

5.3.2 Trainingsmethoden und Aufarbeitung der Daten

CampsNet *CampsNet* verwendet analog zu dieser Arbeit einen Schiebefenster-Ansatz, um über die EKG-Daten zu iterieren. Die normalisierten Signaldaten werden dabei über ein sich bewegendes 512 Datenpunkte breites Fenster eingelesen. Pro Trainingsschritt wird dieses Fenster um 1% weiterbewegt, das entspricht ca. 5 Datenpunkten. Die Normalisierung in *CampsNet* erfolgt durch „die Subtraktion aller Werte um den allgemeinen Durchschnittswert des Fensters und der Division der Werte durch den Betrag der Standardabweichung“. (Camps et al. 2018)

Die Aufgabe der Erkennung des QRS-Komplexes und der Markierung seiner genauen Abgrenzungen erfolgt in einem zweistufigen Ansatz: zunächst wird ein als „Segmentierer“ bezeichneter Teil des Netzwerkes darauf trainiert, die Stellen im EKG grob zu markieren, in denen ein QRS-Komplex vorkommt. Anschließend wird dessen Ausgabe zusammen mit dem dazugehörigen EKG-Signal von einem als „Beschreiber“ bezeichneten zweiten Teil des Netzwerkes verwendet, um die letztendliche Abgrenzung des QRS-Komplexes zu

vollziehen. Einige der vollständig verbundenen Schichten wurden während des Trainings mit einem Dropout von 50% belegt.

Die Ausgabe erfolgt durch eine Logits-Schicht, welche zu jedem Datenpunkt eine Vorhersage zum Beginn und zum Ende des QRS-Komplexes ausgab. Bei einem 512 Datenpunkte breiten Lesefenster und einer trainierbaren Klasse, enthält diese Schicht also 512 Neuronen, die alle vollständig mit jedem Neuron der vorhergehenden Schicht verbunden sind. Das ist mit der in dieser Arbeit präsentierten Logits-Schicht vergleichbar. Dabei musste die Logits-Schicht hier nur eine Vorhersage zum mittleren Datenpunkt des Lesefensters treffen und beinhaltete deshalb nur einen Bruchteil der Menge an trainierbaren Parametern der in *CampsNet* eingesetzten Logits-Schicht.

Die in *CampsNet* beschriebene Aufteilung der Aufgabe in zwei Teile bringt gewiss Vorteile mit sich. So kann sich der Segmentierer auf das Aussortieren der irrelevanten Signalanteile spezialisieren, während der Beschreiber die detaillierte Abgrenzung des QRS-Komplexes lernen kann. Auf diese Arbeit übertragen, wäre eine Aufteilung der Erkennung der einzelnen Wellenspitzen auf einzelne Unternetzwerke eine interessante Überlegung. Ein Netzwerk nämlich, das gleichzeitig lernen muss, alle Wellenspitzen zu erkennen, kann seine Ressourcen zwangsläufig weniger fokussieren als ein Konstrukt aus mehreren Netzwerken, von denen jedes für die jeweilige Disziplin spezialisiert trainiert werden kann.

SegNet wählt einen anderen Ansatz: zunächst werden neben dem Rohsignal noch drei auf unterschiedliche Weise aufgearbeitete Signale erzeugt. Dazu wird der lokale Durchschnittswert eines Datenpunktes und seiner umgebenden 10 Datenpunkte und zudem die erste und zweite Ableitung des Signals berechnet.

Die Iteration über die Daten erfolgt nicht im Sinne eines sich immer weiter schiebenden Lesefensters wie in dieser Arbeit und *CampsNet*, sondern abschnittsweise. Das bedeutet die Unterteilung des Signals in Abschnitte von jeweils 500 Datenpunkten, über die dann im Rahmen eines Trainingsschrittes iteriert wurde. Pro Iteration standen dem Netzwerk der aktuelle Datenpunkt x_t und der zukünftige Datenpunkt x_{t+1} zur Verfügung.

Darüber hinaus wurden mithilfe der Anwendung eines Filters kurzzeitige Markierungs-

aussetzer des Netzwerkes kaschiert. Die Filteranwendung erfolgte im Rahmen eines „post-processings“ auf die Ausgabewerte der letzten Logits-Schicht.

Zusätzlich zu den drei Wellenklassen führten die Autoren eine vierte „neutrale“ Klasse ein, die als Gegenpol zu den Wellenannotationen stets positiv war, wenn keine der Wellen im Signalabschnitt annotiert war.

Einige Ansätze in *SegNet* sind auch für diese Arbeit interessant. So wurde hier zwar der Einsatz gefilterter Daten ebenfalls untersucht, allerdings nicht die Kombination des Rohsignals mit gefilterten Daten. Der Effekt eines mehrkanaligen Inputs könnte Inhalt zukünftiger Forschung sein. Das in der vorliegenden Arbeit vorgestellte Netzwerk konnte lediglich mit dem Rohsignal bereits vergleichbare und teilweise bessere Ergebnisse als *SegNet* erzielen.

Dass *SegNet* trotz seiner beachtlichen Zahl eingesetzter Neuronen komplett ohne Dropout auskommt, korreliert mit den in dieser Arbeit festgestellten Ergebnissen. Für eine reine Kombination mehrerer RNN-Schichten, gefolgt von einer Logits-Schicht, ist es anscheinend weder notwendig noch praktikabel, Dropout einzusetzen.

Die Einführung einer neutralen Klasse stellt zusätzlich einen interessanten Ansatz dar. Denn dadurch muss zu jedem Zeitpunkt im Signal zu einer Aktivierung eines Neurons in der Logits-Schicht kommen. Dadurch könnte es möglich werden, das Gesamtnetzwerk zu einer besseren und ausgeglichneren Differenzierung zu bewegen. Damit ergibt sich für die Zukunft auch ein Verbesserungspotential für das in der vorliegenden Arbeit präsentierte Netzwerk.

5.4 Umgang mit dem Problem der Überanpassung

Neben dem bereits erwähnten und untersuchten Ansatz des Dropouts, gibt es noch weitere Möglichkeiten, eine Überanpassung eines Netzwerkes zu umgehen. Darüber wird im Goodfellow et al. (2016, Kapitel 7) berichtet.

- Gewichtsregulierung („Weight constraint“)
- Eingangsrauschen („Noise“)

- Frühes Aufhören („Early stopping“)

Die Gewichtsregulierung setzt den diversen Gewichtsmatrizen innerhalb eines ANN gewisse Grenzwerte, die nicht über- oder unterschritten werden dürfen. Damit sollen Extremfälle vermieden werden, die bei der Überanpassung auftreten und das Netzwerk empfindlich gegenüber Daten die nicht im Trainingsdatensatz vorkommen machen. Dies war in der hier verwendeten Version von TensorFlow ein sehr aufwendiges und fehleranfälliges Unterfangen, weshalb von einer solchen Gewichtsregulierung abgesehen wurde. In der aktuellen Version von TensorFlow (v.2.1.0) existieren jedoch im Paket `tf.keras.constraints` übersichtliche Funktionen, mit denen die Einstellung dieser Parameter möglich wird. Der Einsatz solcher Funktionen wäre demzufolge für zukünftige Untersuchungen eine lohnende und zudem leicht umzusetzende Maßnahme.

Mit der Erzeugung eines Eingangsrauschens soll bei kleinen Datensätzen der Effekt durch die unvermeidbare Wiederholung von Trainingsbeispielen abgemildert werden. Trotz wiederholter Verwendung desselben Beispiels erhält das Netzwerk jedes Mal eine etwas andere Form und kann daher besser lernen zu generalisieren. Da bei dem hier vorgestellten Netzwerk noch keine klaren Anzeichen einer Überanpassung zu bemerken waren, wurde dieser Schritt nicht implementiert. Sollte das Netzwerk jedoch für deutlich mehr als die bislang explorierten 10 Epochen trainiert werden, sollte die Erzeugung eines Eingangsrauschens in Erwägung gezogen werden..

Das frühe Aufhören ist die einfachste Vorgehensweise, um eine Überanpassung zu verhindern. In diesem Fall wird das Training beendet, sobald die Leistungsparameter beginnen, wieder signifikant zu fallen. Die im Rahmen dieser Arbeit durchgeführten 10 Trainingsepochen entsprachen mit der verwendeten Hardware einer Rechenzeit von etwa 36 Stunden. In dieser Zeit war keine Überanpassung zu beobachten, deshalb wurde auf den Einsatz der oben beschriebenen Routine verzichtet.

Interessanterweise wurde lediglich in zwei Arbeiten (diese und *CampsNet*) Dropout eingesetzt. „Early stopping“ fand lediglich in *SegNet* Anwendung, als das Netzwerk nach 68 Epochen die Abnahme der Leistungsparameter erkannte. Das Eingangsrauschen wurde in keiner der drei Arbeiten eingesetzt.

Das führt zu der Annahme, dass die Menge an EKG-Beispielen und deren Daten in genügend hoher Vielfalt vorlagen und demzufolge nicht zu einer Überanpassung führten. Insbesondere tiefe CNN-Architekturen scheinen jedoch Dropout zu benötigen, was bei der oben erwähnten zukünftigen Kombination aus CNN- und RNN-Architekturen bedacht werden sollte.

5.5 Wahl der Hyperparameter

5.5.1 Annotationsbreite

Die Definition der Annotationsbreite stellte eine der Kernfragen während des Trainings dar. Um einen belastbaren Vergleich mit den gängigsten Arbeiten treffen zu können, sollte nur die Netzwerkleistung mit den Trainingsdatensätzen PRT1-F und PRT1-NF in Betracht gezogen werden, da hier der direkte Vergleich mit den Annotationen in der QT-Datenbank vorgenommen wird. Auf der anderen Seite lässt sich jedoch annehmen, dass die Annotationseinstellungen in PRT3-F und PRT3-NF, welche also ein 8ms breites Zielfenster definieren, im klinischen Alltag keine signifikanten Probleme aufweisen würden. So geben die Autorinnen von *CampsNet* an, ebenfalls „hochklassige“ Ergebnisse zu erzielen, obwohl der durchschnittliche Fehler bei der Erkennung des Beginns und Endes des QRS-Komplexes bei 12.1 ± 0.5ms beziehungsweise 18.5 ± 1.1ms liegt.

Die Annotationseinstellungen PRT(5-11)-F und PRT(5-11)-NF liefern interessante Vergleichsmöglichkeiten mit *SegNet*, sollten aber dann korrekterweise nicht mehr als die Erkennung von Wellenspitzen bezeichnet werden, sondern eher der Markierung von Bereichen dienen, in denen die Welle vorkommt.

5.5.2 Optimierungsalgorithmus

Wie in der vorliegenden Arbeit wurden sowohl bei *CampsNet* als auch bei *SegNet* ADAM als Optimierungsalgorithmus verwendet. Die Wahl von ADAM als Optimierungsalgorithmus wird also, neben den empirischen Untersuchungen in dieser Arbeit, auch durch die Fachwelt bestätigt.

5.5.3 Modellstruktur, Dropout, etc.

Die Wahl der Hyperparameter stellt einen in der Praxis häufig rein empirischen Prozess dar. (Bengio 2012) Die erreichten Ergebnisse führen zu der Schlussfolgerung, dass eine Vielzahl an Trainingsdurchläufen notwendig ist, um eine weitestgehend richtige Konfiguration des Netzwerkes zu identifizieren. Aufgrund der begrenzten technischen und zeitlichen Ressourcen muss aber mit der Möglichkeit gerechnet werden, dass die limitierte Zahl an Trainingsläufen nicht das tatsächliche Optimum eines ANNs dieser Bauart hervorbringt.

Wie im Ergebnisteil geschildert, wurde die Zahl der verwendeten Neuronen pro Schicht nicht vollständig bis zu den oberen Grenzen untersucht. Außerdem wurden keine Kombinationen mit unterschiedlichen Anzahlen an Neuronen pro Schicht (wie in *SegNet* geschehen) getestet. Darauf musste aufgrund der limitierten technischen Kapazitäten verzichtet werden. Diese erweiterten Untersuchungen könnten Gegenstand zukünftiger Forschung sein.

Es gibt jedoch seit wenigen Jahren Ansätze, die das Thema der zeitraubenden Hyperparameterstudie revolutionieren könnten. In Karpathy (2018) wird beschrieben, wie die Bestimmung der Hyperparameter selbst zu einem Teil der Lern- und Trainingsverfahren werden kann. Der Autor dieser Studie ist Chef der Abteilung für künstliche Intelligenz bei Tesla Motors Inc. Ihm zufolge liegt die zukünftige Rolle eines menschlichen Programmierers lediglich in der klaren und wohlüberlegten Anfertigung von Trainingsdaten und Zieldefinitionen für sich selbst programmierende Systeme (neuronale Netzwerke). Er zeigt Möglichkeiten auf, die Wahl der Hyperparameter komplett zu automatisieren. Dies übersteigt aktuell die Erfahrung und Kompetenz der meisten in diesem Bereich aktiven Forscher und benötigt die Erweiterung des gegenwärtigen Ressourcenpools um das beschriebene Training zu ermöglichen. Allerdings stellen eine vermutlich vergleichbare bis kürzere Trainingsdauer (bei entsprechend großem Ressourcenpool) und zugleich bessere Netzwerkkombination wichtige Anreize dieses Ansatzes dar. Es ist daher zu vermuten, dass in der nahen bis mittelfristigen Zukunft auch diese Form der Erstellung neuronaler Netzwerke dem breiteren Forschungsfeld zugänglich werden wird und somit mehr Wissenschaftler in der Lage sein werden, noch besser funktionierende ANNs zu erstellen.

Die übergreifende Verwendung der QT-Datenbank als Trainingsdatenquelle in Arbeiten dieser Art, scheinen diese Datenbank als die geeignetste dieser Art für den ausgeführten Zweck zu markieren. Allerdings zeigen die Ergebnisse in Abschnitt 4.4 auf Seite 54, dass eine Ausweitung der zur Verfügung stehenden Trainingsdaten einen positiven Effekt auf die Leistung des Netzwerkes hat. Das daraus folgende Leistungsverhalten führt zu der Schlussfolgerung, dass eine Datenbank mit der doppelten oder gar dreifachen Menge an EKGs einen immer noch messbaren Effekt zeigen könnte. Dieses Verhalten lässt sich ganz einfach am Lernprinzip eines ANNs erklären: Während klassische EKG-Erkennungsalgorithmen von der Erfahrung ihrer Entwickler profitieren und daher an relativ kleinen Datenbanken suffizient getestet werden können, benötigen ANN eine idealerweise um ein vielfaches größere Menge an Beispielen, um das zugrunde liegende Problem richtig erfassen zu können, ohne in den Bereich der Überanpassung zu geraten. Wie zuvor in Abschnitt 5.4 auf Seite 65 beschrieben, gibt es viele Ansätze um dem Problem der Überanpassung zu begegnen. Der beste und einfachste Ansatz allerdings ist die Vergrößerung des Trainingsdatensatzes.

Die Erstellung und Veröffentlichung einer solchen Datenbank wäre daher für die Forschung an EKG-Signalen mit ANN sehr sinnvoll. Diese sollte neben vollständiger Annotationen zu Wellengrenzen und pathologischen EKG-Mustern auch Informationen zu klinischen Daten wie Diagnosen und Laborparameter enthalten, da in der Korrelation dieser das größte Potential für ANN zu vermuten ist.

5.6 Einsatz in Echtzeitszenarios

Zum Schluss soll die Einsatzfähigkeit in Echtzeitszenarien diskutiert werden. Wie im Abschnitt 4.3 auf Seite 53 gezeigt, erfolgt die Evaluierung der Signale fast elfmal schneller als das Signal selbst dauert. Es ist also davon auszugehen, dass prinzipiell genügend Zeitreserven für weitere Berechnungen und die Pufferung der EKG-Daten vorhanden sind. TensorFlow ist für den Echtzeiteinsatz konzipiert und wird von der Entwicklerfirma Google (Alphabet Inc.) auch in solchen Szenarien eingesetzt.(Sullivan 2016)

Daher ist die technische Machbarkeit generell gegeben. Es müsste allerdings die Datenaufnahme des Netzwerkes auf die neue Quelle umgebaut und zusätzlich untersucht wer-

den, wie groß der Puffer und die daraus resultierende Fensterbreite des Lesefensters sein kann. Je kleiner dieser ist, desto zeitnaher zur tatsächlichen Herzaktion ist dann auch die Ausgabe des Netzwerkes.

Aktuell auf den Markt gebrachte Computertechnik bietet immer häufiger fertig optimierte Hardware-Komponenten zur effizienten Ausführung von ANNs. Demnach wäre der Einsatz eines ANNs zur Echtzeiterkennung von EKG-Anteilen (beispielsweise im Rahmen der Pulsfrequenzmessung oder der Kardioversion) zu einem solchen Zweck sogar auf mobilen medizinischen Endanwenderprodukten durchaus möglich.

5.7 Ausblick: Hybrid-AI

Die in dieser Arbeit vorgestellten und aktuell gängigen Techniken der künstlichen Intelligenz, also mehrschichtige ANN, stoßen inzwischen an ihre Grenzen. In einer weit zitierten und diskutierten Arbeit „The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence“ von Gary Marcus (Marcus 2020), argumentiert der Autor, dass die Entwicklung der künstlichen Intelligenz der letzten Jahre hauptsächlich auf dem Schaffen größerer Datensätze und Netzwerke basiere, jedoch wenige Netzwerke hervorgebracht habe, die gut in der Umwelt generalisieren. Hiermit sei gemeint, dass diese Netzwerke sehr schlecht mit unerwarteten Daten/Situationen - welche also ihren Trainingshorizont überschreiten - umgehen, und hier schnell zu Fehlern neigen.

Die Ursache hierfür sieht er darin, dass die Netze zwar sehr gut darin sind spezifische Muster und Gemeinsamkeiten in großen Datensätzen zu erkennen, es ihnen aber an der Fähigkeit mangle die zu Grunde liegenden Regeln dieser Gegebenheiten zu erkennen und zu nutzen.

Es kann hier der Kontrast zum Lernen beim Menschen gezogen werden: der Mensch sei deshalb in der Lage sich in neuen Situationen so gut zurecht zu finden, weil sich dieser aufgrund seiner Erfahrungen durch die Interaktion mit der Umwelt ein abstraktes Netzwerk aus Zusammenhängen und Gegebenheiten bilde. Dieses könne dann bei der Bewertung neuer Situationen und der Entscheidungsfindung angewendet werden. Ein Mensch muss also nicht zwangsläufig jede Situation zuvor erlebt haben um diese zu meistern,

sondern kann sich basierend auf den bisherigen Erfahrungen die erfolgversprechendste Lösung der Situation errechnen.

Marcus schlägt daher vor, die aktuelle Technik von deep neural network (DNN) (tiefen/mehrschichtigen neuronalen Netzwerken) mit sogenannten „Symbolischen Systemen“ (auch „Symbolic-AI“ genannt) zu verbinden. Hierunter versteht er eine Technik, welche in den 80er Jahren des 20. Jahrhunderts (vor dem Siegeszug der DNN) als erfolgverheißender Ansatz zur Lösung komplexer Probleme mittels künstlicher Intelligenz gehandelt wurde. Die Idee besteht darin, einen aus Regeln und Zusammenhängen bestehenden Graphen zu schaffen, welcher einem Netzwerk als Entscheidungsgrundlage dient. Wenn der Graph nur komplex und abstrakt genug ist, so könnte ein Netzwerk auch mit unerwarteten und schwierigen Aufgaben eine intelligente Lösung errechnen.

Ein System welches neben den Mustererkennungsfähigkeiten von DNN auch noch Zugriff auf einen abstrakten Wissensgraphen hätte, könnte die eingehenden Informationen besser einordnen und somit besseres Verhalten zeigen.

Dieses Konzept wurde beispielsweise mit dem „Neuro-symbolic Concept Learner“-Netzwerk (NSCL) (Mao et al. 2019) erfolgreich demonstriert. Das NSCL kombiniert neuronale Netze zur Lösung von Problemen der visuellen Fragenbeantwortung - einer Aufgabe, bei der eine textbasierte Frage mittels der Interpretation eines Bildes beantwortet werden muss. Dies ist ein mit klassischen neuronalen Netzwerken nur schwer zu lösendes Problem. Hierbei lernt ein „Wahrnehmungsmodul“ visuelle Konzepte auf der Grundlage der sprachlichen Beschreibung des Objekts, auf das verwiesen wird und ähnelt somit stark dem menschlichen Vorbild. Zusätzlich können die erlernten Konzepte zur Bildung neuer Sätze verwendet werden und erleichtern das Erlernen neuer visuellen Zusammenhänge.

Auf das in dieser Arbeit behandelte Problem der Erkennung typischer Strukturen in EKG Signalen wäre Hybrid-AI auch ein denkbarer Verbesserungsansatz.

So könnte man einem Netzwerk, neben den Anteilen der Mustererkennung im Eingangssignal, eine zusätzliche Wissens-Komponente hinzufügen, welche beispielsweise die physiologische Reihenfolge der verschiedenen Wellen und Zacken erklärt. Es wäre sogar

denkbar, ähnlich der Methodik des NSCL-Netzwerkes, diesen Graphen im Rahmen des Trainings automatisch generieren zu lassen.

Den größten Effekt hätte diese Entwicklung jedoch vermutlich bei der Bewältigung noch komplexerer Probleme der EKG-Interpretation, indem man beispielsweise zusätzliche Diagnosen oder (Labor-)Parameter des Patienten über einen Wissensgraphen mit in die Vorhersagen des Netzwerkes einbezieht.

6 Zusammenfassung

Die fortschreitende Entwicklung der Leistungsfähigkeit moderner Computersysteme hat in den letzten beiden Jahrzehnten zu einem sprunghaften Wachstum im Bereich der künstlichen Intelligenz in Form von artifizieller neuronaler Netzwerke (ANN) geführt. Diese selbst-optimierenden statistischen Algorithmen sind in der Lage Muster in Datensätzen zu erkennen und dieses Wissen auf neue Daten anzuwenden. Da es sich bei der Erkennung und Interpretation von EKG-Signalen ebenfalls um ein Mustererkennungsproblem handelt, ist das EKG erneut von zentraler Bedeutung im Bereich der Forschung und Entwicklung von ANN im medizinischen Sektor.

Die hier vorliegende Arbeit hatte das Ziel, ein ANN zu entwickeln, welches in der Lage ist die Wellenspitzen der P- und T-Wellen und die Mitte der QRS-Komplexe in einem ihm vorgelegten EKG zu markieren. Die Leistung dieses Netzwerkes sollte anschließend mit Vertretern sowohl klassischer Methoden, als auch ähnlichen Ansätzen mit ANN-Komponenten, verglichen werden.

Als Datensatz zum Trainieren und Evaluieren des Netzwerkes wurden 96 EKGs der QT-Datenbank eingesetzt. Das ANN wurde mit Hilfe der TensorFlow Softwarebibliothek in Python implementiert. Als ANN-Form wurde ein Rekurrentes neuronales Netzwerk (RNN) gewählt und sowohl Long short-term Memory (LSTM) als auch Gated recurrent Units (GRUs) als Neuronentypen untersucht. Diese RNN-Schicht wurde von einer Dropout-Schicht umgeben und von einer vollständig-verbundenen Logits-Schicht gefolgt. Letztere hatte die Aufgabe die Klassifizierung des vorliegenden Einlesefensters in eine der drei Annotationsklassen (entsprechend der drei Wellenspitzen) vorzunehmen. Zur Analyse der Netzwerkleistung wurden Metriken in Form der Sensitivität, Spezifität, Präzision, Korrektklassifikationsrate und dem F_1-Wert berechnet.

Im Rahmen einer ausführlichen Hyperparameterstudie wurden zunächst die optimalen Einstellungen der jeweiligen Parameter untersucht. Die jeweils relativ am besten abschneidende Parametereinstellung wurde dann anschließend als Einstellung eines „optimalen“ Netzwerkes verwendet. Dieses wurde dann mehrfach mit den unterschiedlichen Annotationseinstellungen der EKG-Daten jeweils für 10 Epochen trainiert und evaluiert.

Die Ergebnisse aus diesen Testläufen wurden anschließend sowohl mit den Leistungen klassischer Methoden (ohne ANN-Einsatz) als auch zweier anderer Arbeiten, die ANN für ähnliche Zwecke zum Einsatz bringen.

Während das Netzwerk, angesichts des für das Training eines ANNs sehr kleinen Datensatzes, bereits gute Leistungen erzielt, kann das entwickelte Netzwerk die Leistungen klassischer Methoden aktuell noch nicht überbieten. So liegt es bei der Erkennung des QRS-Komplexes im Durchschnitt ca. 3%, und bei der P- und T-Welle 1% beziehungsweise 8% zurück. Es wird jedoch gezeigt, dass eine deutliche Leistungssteigerung des Netzwerkes mit größeren Trainingsdatensätzen zu erwarten ist.

Der Vergleich mit den beiden anderen ähnlichen Arbeiten fällt schwierig aus, da eine Arbeit (*CampsNet*) andere Metriken benutzt und die zweite (*SegNet*) eine etwas andere Trainingsaufgabe zum Ziel hatte. Bei vorsichtiger Betrachtung, kann postuliert werden, dass das in dieser Arbeit erstellte Netzwerk *SegNet* mindestens in der Disziplin der QRS-Komplex-Erkennung überbietet und im Bereich der P- und T-Wellen ähnliche Ergebnisse erzielt.

Im Rahmen der Diskussion werden die unterschiedlichen Ansätze der ANN-verwendenden Arbeiten und dieser miteinander verglichen und die jeweiligen Stärken und Schwächen erörtert. Ferner wird auf die Wahl der Netzwerkarchitektur, die Hyperparameterstudie und mögliche zukünftige Verbesserungen beider eingegangen. Zusätzlich werden die möglichen Vorteile einer größeren allgemein zugänglichen EKG-Datenbank und die Einsatzfähigkeit des Netzwerkes in Echtzeitszenarien diskutiert.

Zusammenfassend kann festgehalten werden, dass das in dieser Arbeit entwickelte Netzwerk in der Lage ist, sich mit ähnlichen ANN in der Literatur zu messen, diese aber allesamt in ihrer Leistungsfähigkeit von konventionellen Methoden aktuell noch überschattet werden. Verbesserungen, sowohl in Form einer veränderten Netzwerkstruktur als auch eines, noch zu schaffenden, vergrößerten Trainingsdatensatzes, könnten jedoch vielversprechende Leistungssteigerungen ermöglichen und für zukünftige medizinische Forschung einen wichtigen Beitrag leisten.

Literaturverzeichnis

Abadi, Martín, Ashish Agarwal et al. (2015). „TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems“. In: *White Paper*. URL: http://download.tensorflow.org/paper/whitepaper2015.pdf.

Abadi, Martín, Paul Barham et al. (2016). „TensorFlow: A System for Large-Scale Machine Learning TensorFlow: A system for large-scale machine learning“. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. ISBN: 978-1-931971-33-1. DOI: 10.1038/nn.3331.

Abrishami, Hedayat et al. (2018). „Supervised ECG Interval Segmentation Using LSTM Neural Network“. In: *BIOCOMP'18*, S. 71–77.

Acharya, U. Rajendra et al. (2018). „Automated identification of shockable and non-shockable life-threatening ventricular arrhythmias using convolutional neural network“. In: *Future Generation Computer Systems* 79. ISSN: 0167739X. DOI: 10.1016/j.future.2017.08.039.

Andersen, Rasmus S., Abdolrahman Peimankar und Sadasivan Puthusserypady (2019). „A deep learning approach for real-time detection of atrial fibrillation“. In: *Expert Systems with Applications* 115. ISSN: 09574174. DOI: 10.1016/j.eswa.2018.08.011.

Atkielski, Anthony (Wikimedia) (2007). *Schematic diagram of normal sinus rhythm for a human heart as seen on ECG (with English labels)*. URL: https://en.wikipedia.org/wiki/Electrocardiography#/media/File:SinusRhythmLabels.svg.

Aumüller, Gerhard et al. (2010). „Duale Reihe Anatomie“. In: *Duale Reihe Anatomie*. 2. Auflage. Stuttgart: Thieme. ISBN: 978-3-13-136042-7. DOI: 10.1055/b-0034-100799.

Behrends, Jan C. et al. (2016). „Duale Reihe Physiologie“. In: *Duale Reihe Physiologie*. Stuttgart: Thieme. DOI: 10.1055/b-0034-24754.

Bengio, Yoshua (2012). „Practical recommendations for gradient-based training of deep architectures“. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7700 LECTU, S. 437–478. ISSN: 03029743. DOI: 10.1007/978-3-642-35289-8-26.

Boehm, K. et al. (2018). „Heart rate variability for rapid risk stratification of emergency patients with malignant disease“. In: *Supportive Care in Cancer*. ISSN: 14337339. DOI: 10.1007/s00520-018-4144-y.

Camps, Julia, Blanca Rodriguez und Ana Minchole (2018). „Deep Learning Based QRS Multilead Delineator in Electrocardiogram Signals“. In: *Computing in Cardiology* 2018-Septe, S. 1–4. ISSN: 2325887X. DOI: 10.22489/CinC.2018.292.

Cho, Kyunghyun et al. (2014). „Learning phrase representations using RNN encoder-decoder for statistical machine translation“. In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, S. 1724–1734.

Chung, Junyoung et al. (Dez. 2014). „Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling“. In: URL: http://arxiv.org/abs/1412.3555.

Dora, Chinmayee und Pradyut Kumar Biswal (2016). „Robust ECG artifact removal from EEG using continuous wavelet transformation and linear regression“. In: *2016 Inter-*

national Conference on Signal Processing and Communications, SPCOM 2016. DOI: 10.1109/SPCOM.2016.7746620.

Farsad, Nariman und Andrea Goldsmith (2018). „Sliding Bidirectional Recurrent Neural Networks For Sequence Detection In Communication Systems“. In: *Stanford University*, S. 2331–2335.

Flasiński, Mariusz (2016). „History of Artificial Intelligence“. In: *Introduction to Artificial Intelligence*, S. 3–13. DOI: 10.1007/978-3-319-40022-8{_}1.

Gers, Felix A., Jurgen Schmidhuber und Fred Cummins (1999). „Learning to forget: Continual prediction with LSTM“. In: *IEE Conference Publication* 2.470, S. 850–855. ISSN: 05379989. DOI: 10.1049/cp:19991218.

Glorot, Xavier, Antoine Bordes und Yoshua Bengio (2011). „Deep sparse rectifier neural networks“. In: *Journal of Machine Learning Research*.

Goldberger, A. L. et al. (2000). „PhysioBank, PhysioToolkit, and PhysioNet : Components of a New Research Resource for Complex Physiologic Signals“. In: *Circulation*. ISSN: 0009-7322. DOI: 10.1161/01.CIR.101.23.e215.

Goodfellow, Ian, Yoshua Bengio und Aaron Courville (2016). *Deep Learning*. Adaptive C. The MIT Press. ISBN: 978-0262035613.

Hermans, Michiel und Benjamin Schrauwen (2013). „Training and analyzing deep recurrent neural networks“. In: *Advances in Neural Information Processing Systems*.

Hochreiter, Sepp und Jürgen Schmidhuber (Nov. 1997). „Long Short-Term Memory“. In: *Neural Computation* 9.8, S. 1735–1780. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

Horn, Roger A und Charles R Johnson (2017). *Chapter 5 The Hadamard product*, S. 298–381. ISBN: 9780511840371.

IBM Cloud Education (Aug. 2020a). *What is supervised learning?* URL: https://www.ibm.com/cloud/learn/supervised-learning.

— (Sep. 2020b). *What is Unsupervised Learning?* URL: https://www.ibm.com/cloud/learn/unsupervised-learning.

Ixnay (2017a). *A diagram for a one-unit Gated Recurrent Unit (GRU)*. URL: https://commons.wikimedia.org/wiki/User:Ixnay#/media/File:Gated_Recurrent_Unit.svg.

— (2017b). *A diagram for a one-unit Long Short-Term Memory (LSTM)*. URL: https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg.

Jia, Yangqing et al. (2014). „Caffe“. In: DOI: 10.1145/2647868.2654889.

Jouppi, Norman P. et al. (2017). „In-datacenter performance analysis of a tensor processing unit“. In: *Proceedings - International Symposium on Computer Architecture*. ISBN: 9781450348928. DOI: 10.1145/3079856.3080246.

Karpathy, Andrej (Director of AI at Tesla Motors Inc.) (2018). „Building the Software 2.0 Stack“. In: *Spark+AI Summit*. URL: https://databricks.com/session/keynote-from-tesla.

Kim, Phil (2017). *MATLAB Deep Learning*. DOI: 10.1007/978-1-4842-2845-6.

Kingma, Diederik P. und Jimmy Lei Ba (2015). „Adam: A method for stochastic optimization“. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, S. 1–15.

Kozyrkov, Cassie (Feb. 2020). *What is "Ground Truth" in AI? (A warning.)* URL: https://towardsdatascience.com/in-ai-the-objective-is-subjective-4614795d179b.

Krizhevsky, Alex, Ilya Sutskever und Geoffrey E. Hinton (2017). „ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM*. ISSN: 15577317. DOI: 10.1145/3065386.

Laguna, Pablo, Raimon Jané und Pere Caminal (1994). „Automatic detection of wave boundaries in multilead ECG signals: Validation with the CSE database". In: *Computers and Biomedical Research* 27.1, S. 45–60. ISSN: 00104809. DOI: 10.1006/cbmr.1994.1006.

Laguna, Pablo, Roger G. Mark et al. (1997). „A Database for Evaluation of Algorithms for Measurement of QT and Other Waveform Intervals in the ECG". In: *Computers in Cardiology* 24, S. 673–676.

Lecun, Yann, Yoshua Bengio und Geoffrey Hinton (2015). „Deep learning". In: *Nature* 521.7553, S. 436–444. ISSN: 14764687. DOI: 10.1038/nature14539.

Lecun, Yann, Le'on Bottou et al. (1998). „Gradient-Based Learning Applied to Document Recognition". In: *proc. OF THE IEEE*. URL: http://ieeexplore.ieee.org/document/726791/#full-text-section.

Leszczynski, Megan (2020). *Self-Supervised Learning*.

Loiseau, Jean-Christophe B. (2019). *Rosenblatt's perceptron, the first modern neural network*.

Lyon, Aurore et al. (Jan. 2018). *Computational techniques for ECG analysis and interpretation in light of their contribution to medical advances*. DOI: 10.1098/rsif.2017.0821.

Mao, Jiayuan et al. (Apr. 2019). „The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision". In: URL: http://arxiv.org/abs/1904.12584.

Marcus, Gary (Feb. 2020). „The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence". In: URL: http://arxiv.org/abs/2002.06177.

Martínez, Juan Pablo et al. (2004). „A Wavelet-Based ECG Delineator Evaluation on Standard Databases". In: *IEEE Transactions on Biomedical Engineering* 51.4, S. 570–581. ISSN: 00189294. DOI: 10.1109/TBME.2003.821031.

Mitchell, Tom M. (Okt. 1997). *Machine Learning (McGraw-Hill International Editions Computer Science Series)*. 1st. McGraw-Hill. ISBN: 978-0071154673.

Moody, George B. und Roger G. Mark (1983). *Development and Evaluation of a 2-Lead Ecg Analysis Program.*

Moore, G. E. (1965). „Cramming more components onto integrated circuits. In: Electronics". In: *Electronics*.

Nair, Vinod und Geoffrey E. Hinton (2010). „Rectified linear units improve Restricted Boltzmann machines". In: *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*. ISBN: 9781605589077.

Nesterov, Yurii (1983). „A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)". In: *Doklady ANSSR*.

Nielsen, Michael A. (2015). *Neural Networks and Deep Learning*. Determination Press.

NVIDIA Developer (2016). *NVIDIA cuDNN — NVIDIA Developer*.

Oh, Shu Lih et al. (2018). „Automated diagnosis of arrhythmia using combination of CNN and LSTM techniques with variable length heart beats“. In: *Computers in Biology and Medicine* 102. ISSN: 18790534. DOI: 10.1016/j.compbiomed.2018.06.002.

Pan, Jiapu und Willis J. Tompkins (1985). „A Real-Time QRS Detection Algorithm“. In: *IEEE Transactions on Biomedical Engineering*. ISSN: 0018-9294. DOI: 10.1109/TBME.1985.325532.

Perol, Thibaut, Michaël Gharbi und Marine Denolle (2018). „Convolutional neural network for earthquake detection and location“. In: *Science Advances*. ISSN: 23752548. DOI: 10.1126/sciadv.1700578.

Powers, David Martin Ward (2011). „ABC Unconscious Computer Interface View project Autonomous Robotics View project EVALUATION: FROM PRECISION, RECALL AND F-MEASURE TO ROC, INFORMEDNESS, MARKEDNESS & CORRELATION“. In: 2.1, S. 37–63. ISSN: 2229-3981. DOI: 10.9735/2229-3981. URL: http://www.bioinfo.in/contents.php?id=51.

Reed, Russel und Robert J Marks ll (1997). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks (A Bradford Book)*. A Bradford Book. ISBN: 978-0262527019.

Rosenblatt, F. (1958). „The perceptron: A probabilistic model for information storage and organization in the brain“. In: *Psychological Review*. ISSN: 0033295X. DOI: 10.1037/h0042519.

Rumelhart, David, Geoffrey Hinton und Ronald Williams (1986). „Learning representations by back-propagating errors“. In: *Nature* 323.6088, S. 533–536. ISSN: 00280836. DOI: 10.1038/323533a0.

Rumelhart, David E. und James L. MacClelland (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Foundations. 1, Volume 1*. Bd. 1. June, S. 318–362. ISBN: 0262132184. URL: http://books.google.co.uk/books/about/Parallel_Distributed_Processing.html?id=SCHaQwAACAAJ&pgis=1.

Salam, K. Amtul und G. Srilakshmi (2015). „An algorithm for ECG analysis of arrhythmia detection“. In: *Proceedings of 2015 IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT 2015*. DOI: 10.1109/ICECCT.2015.7226130.

Savitzky, Abraham und Marcel J.E. Golay (1964). „Smoothing and Differentiation of Data by Simplified Least Squares Procedures“. In: *Analytical Chemistry*. ISSN: 15206882. DOI: 10.1021/ac60214a047.

Schmidhuber, Jürgen (2015). *Deep Learning in neural networks: An overview*. DOI: 10.1016/j.neunet.2014.09.003.

Schuster, Mike und Kuldip K. Paliwal (1997). „Bidirectional recurrent neural networks“. In: *IEEE Transactions on Signal Processing* 45.11, S. 2673–2681. ISSN: 1053587X. DOI: 10.1109/78.650093.

So, Cook-Sup (1. Medizinische Klinik und Poliklinik rechts der Isar der Technischen Universität München) (1978). *Praktische Elektrokardiographie*. 2. Aufl. München: Selecta-Verlag, Dr. Ildar Idris, Planegg vor München, S. 255. ISBN: 3-921500-12-5.

Sörnmo, Leif und Pablo Laguna (2005). „The Electrocardiogram—A Brief Background“. In: *Bioelectrical Signal Processing in Cardiac and Neurological Applications*. Elsevier, S. 411–452. DOI: 10.1016/b978-012437552-9/50006-4.

Srivastava, Nitish et al. (2014). „Dropout: A simple way to prevent neural networks from overfitting“. In: *Journal of Machine Learning Research* 15, S. 1929–1958. ISSN: 15337928.

Sullivan, Danny (Search Engine Land) (2016). *FAQ: All about the Google RankBrain algorithm*. URL: https://searchengineland.com/faq-all-about-the-new-google-rankbrain-algorithm-234440.

Swapna, G., K. P. Soman und R. Vinayakumar (2018). „Automated detection of cardiac arrhythmia using deep learning techniques“. In: *Procedia Computer Science*.

Szegedy, Christian et al. (2015). „Going deeper with convolutions“. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594.

The Theano Development Team et al. (2016). „Theano: A Python framework for fast computation of mathematical expressions“. In: S. 1–19. DOI: 1605.02688. URL: http://arxiv.org/abs/1605.02688.

Thoma, Martin (2013). *Schema eines Feed Forward Perzeptrons*.

Walia, Anish Singh (2017). *Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent*. URL: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f.

Wieclaw, Lukasz et al. (Nov. 2017). „Biometrie identification from raw ECG signal using deep learning techniques“. In: *Proceedings of the 2017 IEEE 9th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2017*. Bd. 1. Institute of Electrical und Electronics Engineers Inc., S. 129–133. ISBN: 9781538606971. DOI: 10.1109/IDAACS.2017.8095063.

Yildirim, Özal (2018). „A novel wavelet sequences based on deep bidirectional LSTM network model for ECG signal classification“. In: *Computers in Biology and Medicine* 96. ISSN: 18790534. DOI: 10.1016/j.compbiomed.2018.03.016.

Erklärung zum Eigenanteil der Dissertationsschrift

Die Arbeit wurde in der Medizinische Klinik 3, Abteilung für Kardiologie und Angiologie an der Universitätsklinik Tübingen unter Betreuung von Prof. Dr. med. Meinrad Gawaz durchgeführt.

Die Konzeption der Arbeit erfolgte gemeinsam mit Dr. med. Christian Eick.

Die Programmierung, das Trainieren und das Auswerten des Netzwerkes und dessen Leistung erfolgte eigenständig und in Absprache mit Dr. Eick.

Ich versichere, das Manuskript selbstständig verfasst zu haben und keine weiteren als die von mir angegebenen Quellen verwendet zu haben.

Tübingen, den 14.10.2022