Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELORTHESIS

Prototyp für eine virtuelle Tastatur basierend auf IMUs und maschinellem Lernen - Anwendung

vorgelegt von

Carolin Konietzny

MIN-Fakultät

Fachbereich Informatik

Arbeitsbereich Technische Aspekte Multimodaler Systeme

Studiengang: Informatik

Matrikelnummer: 6523939

Erstgutachter: Dr. Norman Hendrich

Zweitgutachter: Florens Wasserfall

Kurzbeschreibung

Das Projekt „Prototyp für eine virtuelle Tastatur basierend auf IMUs und maschinellem Lernen" zielt darauf ab, ein System zu entwickeln, das als Alternative zu einer traditionellen Computertastatur Texteingaben aus den beim Tippen gemachten Bewegungen ermittelt. In dieser Arbeit wird ein Verfahren erläutert, aus den Bewegungsdaten von 6 IMUs, welche an den Fingern und der Rückseite einer Hand angebracht sind, die beim Tippen durchgeführten Tastenanschläge zu bestimmen. Hierzu werden die benötigten Vorverarbeitungsschritte vorgestellt und gezeigt, dass die ein *Convolutional Neural Network* für diese Aufgabe sehr geeignet ist. Bei langsamem Tippen von 10 verschiedenen Tasten konnte eine Genauigkeit von 85% erreicht werden.

Abstract

In the project "Prototype for a virtual keyboard based on IMUs and machine learning" a system was developed to deduce keyboard input from the movements made while typing, to be used as an alternative to a traditional computer keyboard. In this thesis an approach is presented to detect keystrokes from the movement data recorded by 6 IMUs attached to the fingers and the back of the hand. The required preprocessing of this data is discussed. It is shown that *convolutional neural networks* are suitable for the task at hand. During slow typing of 10 different keys an accuracy of 85% was accomplished. und maschinellem Lernen"

Inhaltsverzeichnis

Abkürzungsverzeichnis

3D	drei-dimensional
CNN	*convolutional neural network*, Faltungsnetzwerk
EEPROM	*electrically erasable programmable read-only memory* (Speicherbaustein)
HAR	*human activity research* (Erkennung menschlicher Bewegungsabläufe und Aktivitäten durch Sensoren)
HMM	*Hidden Markov Model* („a tool for representing probability distributions over sequences of observations."[1])
IMU	*inertial measurement unit*, inertiale Messeinheit
KNN	*k nearest neighbours*, k-nächste-Nachbarn (Algorithmus) *oder* künstliches neuronales Netz (vgl. NN)
ML	*machine learning*, maschinelles Lernen
MSE	*mean squared error*, mittlere quadratische Abweichung
NN	neuronales Netz (manchmal auch *nearest neighbor*, vgl. KNN)
RNN	*recurrent neural network*, rekurrentes neuronales Netz
ROS	*Robot Operating System*
UDP	*User Datagram Protocol* (ein Netzwerkprotokoll)

[1] (Ghahramani, 2001), eigene Übersetzung: „ein Werkzeug zur Darstellung von Wahrscheinlichkeitsverteilungen über Sequenzen von Beobachtungen"

1 Einleitung

In dieser Arbeit wird der Prototyp eines Datenhandschuhs vorgestellt, der mit Sensoren die Finger- und Handbewegungen beim Schreiben auf einer Tastatur aufzeichnet und mittels geeigneter Methoden des maschinellen Lernens (ML) diese Bewegungen lernt, um später Tastendrücke aus den Bewegungen erkennen zu können. Dieses System soll die Tastatur als Eingabegerät ersetzen können.

Im folgenden Kapitel wird die Motivation zu diesem Projekt und der Aufbau der Arbeit erläutert. Außerdem wird der Projektumfang abgegrenzt.

1.1 Motivation

In der heutigen Zeit sind Computer aus dem Alltag nicht mehr wegzudenken. In den meisten Fällen wird für die Eingabe von Befehlen eine Tastatur verwendet.

Für den Einsatz einer Tastatur sprechen viele Kriterien. Zum einen ist die Nutzung relativ schnell zu erlernen. Anfangs ist die Eingabegeschwindigkeit zwar oft nicht sonderlich hoch, jedoch wird diese mit einem gewissen Grad an Erfahrung und dem richtigen Schreibsystem rasch verbessert.

Hinzu kommt, dass die Tastatur sehr präzise in ihren Eingaben ist. Wenn eine Taste gedrückt wird, ist eine Fehlinterpretation des Computers nicht möglich. Sofern also das Tippen beherrscht wird, ist eine schnelle und stabile Kommunikation mit dem Computer gewährleistet.

Ein weiterer Grund für die Nutzung einer Tastatur ist, dass sie in der Produktion ausgesprochen günstig ist. Kaum eine andere Eingabemethode kommt an dieses Preis-Leistungs-Verhältnis heran.

Es gibt jedoch auch viele Gründe, die gegen die Verwendung einer Tastatur sprechen. Eine Tastatur ist beispielsweise relativ schlecht an motorische Fähigkeiten anpassbar. Mit weniger als 10 Fingern wird das Abdecken jeder Taste erschwert, wodurch die Effektivität negativ beeinflusst werden kann.

Außerdem ist eine Tastatur kaum an die Aufgabendomäne anpassbar. Die vorgegebe-

nen Tasten und deren Anordnungen können nach der Herstellung nicht mehr verändert werden und das Ändern von Tastenbelegungen ist nur in bedingtem Maße nützlich. Sogenannte „Short-Cuts", welche es ermöglichen bestimmte Funktionalitäten auf eine Kombination zwei oder mehrerer Tasten zu legen, reduzieren die Einschränkungen lediglich in geringem Umfang.

Das wahrscheinlich größte Problem mit konventionellen Tastaturen ist wohl der ergonomische Aspekt. Das Schreiben an einer Tastatur bietet einen sehr kleinen Spielraum für die Position der Hände. Der Mensch muss sich mit seiner gesamten Haltung an diese Position anpassen. Dies fördert diverse gesundheitliche Probleme, wie zum Beispiel das Repetitive-Strain-Injury-Syndrom (Gerr, Marcus und Monteilh, 2004).

In Anbetracht dieser Nachteile ist es schwer nachvollziehbar, dass an der Tastatur in den letzten Jahrzehnten kaum Veränderungen vollzogen wurden, und das, obwohl die Endgeräte, für welche die Tastatur benötigt wird, sich drastisch verändert haben.

Die Tastatur ist noch immer eng an das Layout einer Schreibmaschine angelehnt. Es wurden zwar verschiedene Anpassungen gemacht, wie zum Beispiel das Aufteilen der Tasten in zwei Hälften bei manchen Tastaturen, um die Handposition beim Schreiben zu verbessern, die Anordnung der Tasten im Ganzen sind jedoch kaum verändert.

Da die Geräte jedoch immer kleiner werden, ist die Tastatur nun oft der einschränkende Faktor. Bei Smartphones wird die virtuelle Tastatur oft nur noch mit zwei Finger bedient, der benötigte Platz umfasst jedoch trotzdem knapp den halben Bildschirm und ist somit nicht unerheblich.

1.2 Vision

Mit diesem Projekt wollen wir[1] einen Datenhandschuh entwickeln, welcher die herkömmliche Tastatur ersetzen kann. Mithilfe des Handschuhs soll es möglich sein, in jeder Position flüssig zu schreiben, ohne an hardwarebedingte Vorgaben gebunden zu sein. Dies soll ein deutlich ergonomischeres Schreiben ermöglichen, bei dem die Haltung der Person, welche den Datenhandschuh nutzt, nicht mehr an die Tastatur angepasst werden muss. Somit ist ein regelmäßiges Ändern der Schreibposition möglich, ebenso wie das Schreiben an verschiedenen Orten, an denen die Verwendung einer Tastatur eher unpraktisch oder kaum möglich wäre. Beispielsweise soll es möglich sein, den Handschuh im Stehen (ohne einen Tisch vor sich zu haben), im Liegen oder unterwegs zu benutzen.

Zu der Positionsunabhängigkeit kommt hinzu, dass auch die Bewegungen für die verschiedenen Eingaben unabhängig von externen Vorgaben sind. Jede Person kann ein eigenes Schreibsystem entwickeln und für sie wichtige Funktionalitäten, die mit einer Tastatur nur über Short-Cuts zu ermöglichen wären, mit eigenen Bewegungen belegen. Dadurch

[1] In dieser Arbeit verwende ich die erste Person Singular, um mich auf meine eigene Arbeit zu beziehen, und die erste Person Plural, um auf Entscheidungen innerhalb des Projekts einzugehen.

Abbildung 1: Der fertiggestellte Datenhandschuh während des Aufzeichnens der Lerndaten. Zu sehen sind die mit Gummiband befestigten Finger-IMUs, die Hand-IMU seitlich auf dem Handrücken sowie der Mikroprozessor mit Platine, welche auf einem Armband befestigt sind.

entsteht ein Eingabegerät, das in sehr vielen verschiedenen Kontexten verwendet werden kann.

Anforderungen an den Handschuh sind unter anderem, dass er robust und möglichst universal gebaut sein sollte, so dass er verschiedenen Handtypen passt. Die Kosten für die Herstellung sollten möglichst gering sein, um eine eventuelle Vermarktung zu ermöglichen und das Produkt für viele Nutzer attraktiv zu machen.

Die Daten, mit deren Hilfe die Bewegungsabläufe gelernt werden sollen, werden mithilfe einer Tastatur aufgezeichnet. Dies ermöglicht, dass Nutzer des Datenhandschuhs die bereits im Muskelgedächtnis gespeicherten Bewegungen für verschiedene Tasten nutzen können und sich nicht für jede Tastatureingabe eine Bewegung ausdenken müssen.

Nach dem Lernen soll auf die Tastatur verzichtet werden können. Anhand der Sensor- und Tastaturdaten soll ein maschineller Lernalgorithmus in der Lage sein, die Fingerbewegungen den dazugehörigen Tasten zuordnen zu können.

1.3 Umfang und Abgrenzung der Arbeit

Das Projekt ist im Rahmen zweier Bachelorarbeiten entstanden. Da diese auf jeweils 5 Monate begrenzt sind, war es nicht möglich, die Vision vollständig umzusetzen. Im Folgenden wird darauf eingegangen, auf welche Aspekte der Fokus gelegt wurde und welche Teile eventuell nur stark vereinfacht umgesetzt werden konnten.

Eine Vereinfachung ist, dass wir nur einen Handschuh gebaut haben, um sowohl Resourcen als auch Zeit zu sparen. Desweiteren haben wir den Handschuh nicht auf mehrere, sondern lediglich auf eine Person ausgelegt und nicht für verschiedene Handtypen designed.

Die Voraussetzung für das Nutzen des Muskelgedächtnisses zum Schreiben auf einer Tastatur ist, dass die Person ein geübter Schreiber ist. Wir konzentrierten uns auf diesen Fall, damit die ohne Tastatur durchgeführten Bewegungen denen der Lernphase ähneln. Wir beachteten zunächst nicht den Fall, dass sich die Fingerbewegungen zu einer Taste von Fall zu Fall stark unterscheiden (wenn die Person die Taste etwa erst suchen muss).

Wir haben in diesem Projekt zudem nicht viele verschiedene ML-Methoden evaluiert, sondern lediglich 2 verschiedene neuronale Netze ausprobiert und uns für das Netz entschieden, welches sich für unser Projekt besser eignet. Je komplexer die Aufgabe wird, desto wahrscheinlicher ist es, dass weitere Methoden evaluiert werden müssen, das Projekt bietet also Raum für weitere Forschung.

Unser Projektziel kann in drei große Bereiche eingeteilt werden:

1. Entwurf eines Systems zur Aufzeichnung der charakteristischen Handbewegungen beim Tippen sowie der dazugehörigen Tastatureingaben.

2. Definition eines Ansatzes zum Rückschließen auf Tastatureingaben aus den aufgezeichneten Bewegungen unter der Verwendung von Verfahren des maschinellen Lernens.
3. Bewertung der Qualität dieser Rückschlüsse und der Nutzbarkeit eines solchen Verfahrens als Alternative zur klassischen Tastatur.

Der erste Teil, der Entwurf des Systems, ist Hauptbestandteil der Bachelorarbeit von Paul Bienkowski (2017) und dort nachzulesen. Ich beschränke mich auf diese Aspekte lediglich, soweit sie die Verständlichkeit meiner Arbeit unterstützen.

Die vorliegende Arbeit behandelt die Nutzung von maschinellem Lernen zur Ermittlung von Tastendrücken anhand von aufgezeichneten Bewegungen. In beiden Arbeiten wird auf die Bewertung und die Analyse des Projektes eingegangen.

1.4 Aufbau der Arbeit

Diese Arbeit ist in mehrere Kapitel gegliedert. In Kapitel 1 wurde die Motivation für den erstellten Prototyp erläutert und ein Überblick über die Problematik der jetzigen Eingabemethode gegeben.

In Kapitel 2 wird eine Grundlage über den für das Projekt relevanten Bereich des maschinellen Lernens geschaffen. Unter anderem wird der Unterschied von überwachtem und unüberwachtem Lernen erläutert, die in diesem Projekt verwendeten Typen eines neuronalen Netzes erklärt und auf die Problematik von unbalancierten Datensätzen eingegangen.

Im Abschnitt „State of the Art', Kapitel 3, werden verwandte Projekte und Forschungsarbeiten aufgezeigt und diskutiert.

Es folgt ein Kapitel über das angewandte Verfahren, welches die Bewegungsdaten extrahiert und mit ihrer Hilfe Tastendrücke erkennt. In diesem Kapitel wird der Datenhandschuh und die verwendeteten Bibliotheken beschrieben. Zudem wird auf das Vorverarbeiten der Daten eingegangen und die zu Grunde liegende Netzwerkarchitektur mitsamt aller Konfigurationsmöglichkeiten erläutert.

In Kapitel 5 werden die Versuche beschrieben, welche für die Bewertung des Datenhandschuhs und des vorgestellten Verfahrens verwendet wurden, um dann auf deren Ausgang und die Schlüsse, die wir aus den Versuchen ziehen konnten, einzugehen.

Schließlich werden in Kapitel 6 die Ergebnisse dieser Arbeit zusammengefasst und Verbesserungsmöglichkeiten und weitere mögliche Schritte aufgezeigt.

2 Wissenschaftliche Grundlagen

In diesem Kapitel werden die für das maschinelle Lernen im Rahmen unseres Projektes relevanten Grundlagen erläutert.

2.1 Maschinelles Lernen

Lernen ist, nach Poole und Mackworth (2010, Kapitel 7), die Fähigkeit eines Agenten anhand von Erfahrungen das eigene Verhalten zu verbessern. Hierbei wird sowohl die Genauigkeit als auch die Geschwindigkeit bei der Bearbeitung von Aufgaben verbessert und die Bandbreite der gezeigten Verhaltensmuster erweitert.

Es existieren zahlreiche Methoden des maschinellen Lernens. Diese können in verschiedene Kategorien eingeteilt werden, ich werde hier auf überwachtes und unüberwachtes Lernen [1] eingehen.

Beim überwachten Lernen werden dem Lernalgorithmus sowohl die Eingabedaten, als auch deren Zielwerte zur Verfügung gestellt. Der Algorithmus soll nun die Abbildungsfunktion

$$f : \text{Eingabedaten} \mapsto \text{Zielwerte}$$

erlernen, er versucht anhand der Eingabedaten die Zielwerte zu ermitteln. Zunächst macht der Algorithmus dabei noch Fehler. Anhand der bereitgestellten Zielwerte kann der Fehler ermittelt und das Verhalten angepasst werden, um in der Zukunft bessere Vorhersagen machen zu können. Die Fehler werden in Kosten angegeben, welche durch eine geeignete Kostenfunktion errechnet werden. Diese Kosten gilt es zu minimieren.

Um das Gelernte zu überprüfen, folgt auf eine Trainingsphase mit den zugehörigen Zielwerten eine Testphase ohne Zielwerte. Der Algorithmus muss das vorher Gelernte in der Testphase auf unbekannte Daten anwenden. Die Verwendung eines anderen Datensatzes ist sehr wichtig, da die Gefahr besteht, dass sich der Algorithmus zu sehr auf die Lerndaten einstellt und dadurch die Generalisierungsfähigkeit verliert.

Dieses Phänomen wird auch *Overfitting* genannt (Hawkins, 2004). In Abbildung 2 sind beispielhafte Kosten der Trainings- und der Testphasen im Verlauf der Lernepochen zu sehen. Die Kostenfunktion der Trainingsdaten wird optimiert, sodass bestenfalls die Kosten

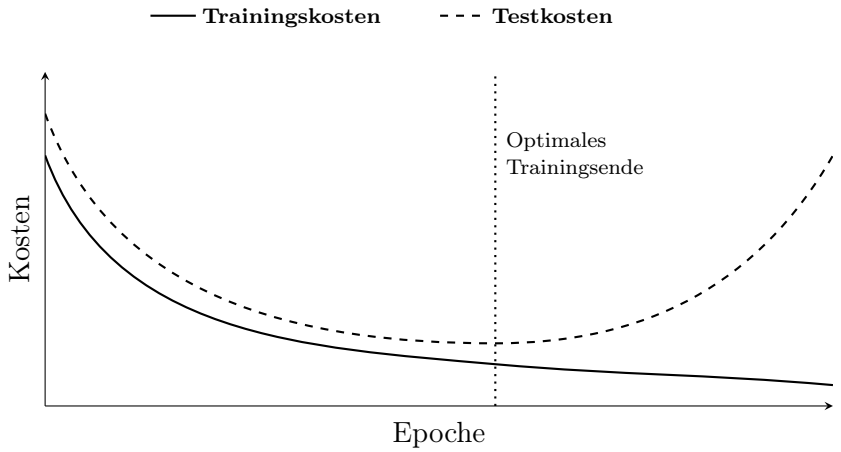


Abbildung 2: Overfitting. Die Trainings- und Testkosten nehmen zunächst beide ab, dann steigen die Testkosten wieder, während die Trainingskosten weiterhin optimiert werden. Dies ist der Zeitpunkt, ab dem der Algorithmus seine Generalisierungsfähigkeit verliert.

bei den Testdaten ebenfalls sinken. Zunächst ist dies auch der Fall, doch ab einem gewissen Punkt sinken die Kosten bei den Trainingsdaten, die Kosten für den Testdatensatz steigen jedoch wieder an. Dies ist der Moment, in dem das Netz die Generalisierungsfähigkeit verliert und ein Overfitting beginnt. Es ist meist sinnvoll, an diesem Punkt den Lernvorgang zu beenden.

Überwachtes Lernen findet vor allem Anwendung in Klassifizierungsproblemen und in sogenannnten Regressionsproblemen, bei denen die Zielwerte numerisch, etwa Geldbeträge oder bestimmte Gewichtsmaße sind.

Im Vergleich zum überwachten Lernen werden beim unüberwachten Lernen keine Zielwerte angegeben, der Algorithmus erhält also ausschließlich die Eingabedaten. Er lernt keine Abbildungsfunktion, sondern vielmehr ein Modell der Struktur beziehungsweise die Verteilung der Eingabedaten [1].

Häufig werden unüberwachte Lernalgorithmen verwendet, um Daten nach bestimmten Eigenschaften zu gruppieren. Es werden dabei Gruppen gebildet, deren Elemente sich möglichst ähnlich sind. Dieser Vorgang wird auch *Clustering* genannt (Poole und Mackworth, 2010, Kapitel 11.1) .

Für unser Projekt haben wir uns für überwachtes Lernen entschieden, da wir einen gut definierten Hypothesenraum haben und zudem Trainingsdaten inklusive Zielwert recht einfach generieren können, da wir beim Tippen die Daten des Handschuhs mit den vom Computer verzeichneten Tastenanschlägen verbinden können.

Es gibt viele verschieden ML-Methoden, die überwachtes Lernen verwenden. Wir haben uns für künstliche neuronale Netze (NN) entschieden. Diese bieten eine breitgefächerte

Variation an Anwendungsmöglichkeiten und sind in der Lage, komplexe Zusammenhänge in den Daten zu erkennen und für die gemachten Ausgaben zu verwenden. Wenn ein neuronales Netz gelernt ist, kann es in der Anwendung die Ausgaben sehr schnell berechnen.

Nachteile eines neuronalen Netzes sind unter anderem, dass sie eine große Menge an Lerndaten benötigen und oft sehr viel Training erfordern, um zufriedenstellende Ergebnisse zu erzielen. Zudem ist es sehr schwer, die vom Netz gelernten Zusammenhänge einzusehen. Außerdem muss darauf geachtet werden, dass kein Overfitting auftritt, neuronale Netze sind hierfür anfällig.

2.2 Künstliche Neuronale Netze

Künstliche neuronale Netze werden im Bereich des maschinellen Lernes häufig verwendet. Durch die verschiedenen Strukturen von neuronalen Netzen finden sie Anwendung in sehr unterschiedlichen Bereichen des Lernens, unter anderem in Muster- und Spracherkennung, aber auch in der Bildverarbeitung oder beim Lernen von Bewegungsabläufen.

Neuronale Netze sind an Gehirnstrukturen angelehnt, gleichen diesen jedoch nicht vollständig [2]. Grundbausteine sind einzelne Neuronen (Einheiten), welche über gewichtete Kanten miteinander verbunden sind. Diese Kanten werden mit zufälligen Werten initialisiert und nach und nach so angepasst, dass sie für die ihnen gestellte Aufgabe möglichst gut geeignet sind.

Die Struktur eines Netzes ist von großer Bedeutung und kann zwischen verschiedenen Netztypen stark variieren. Die einfachste Struktur ist wohl die des *Feedforward-Netz* (Davidian, 1995). Hierbei sind verschiedene Schichten, welche unterschiedlich viele Einheiten enthalten können, hintereinandergeschaltet und mit Kanten verbunden. Das Netz ist nach vorne gerichtet, die Kanten gehen also von einer Schicht ausschließlich in die nächste, ohne Schleifen oder rückwärtsgerichtete Kanten.

Das Netz rechnet, indem die Eingabeschicht (*input layer*) die Eingabedaten erhält. Jede Einheit der Schicht ermittelt mit einfachen mathematischen Mitteln einen eigenen Ausgabewert. Dieser wird von den gewichteten Kanten in die nächste Schicht geleitet, bis die Ausgabeschicht (*output layer*) erreicht wird. Die von der Ausgabeschicht ermittelten Werte werden dann als Ergebnis interpretiert.

Es gibt viele verschieden mathematische Mittel, welche verwendet werden können. Diese sind an sich jedoch primitiv im Vergleich zur Ausdrucksstärke des Netzes. Durch die Verwendung vieler Einheiten steigt die Komplexität der Berechnungen, welche mit dem Netz durchgeführt werden können.

Die *Backpropagation-Methode* wird beim überwachten Lernen eingesetzt. Dabei wird zunächst der Ausgabewert mit dem tatsächlichen Wert verglichen. Der ermittelte Fehler wird auf die Kanten verteilt, welche den Fehler verursacht haben und die Kanten wer-

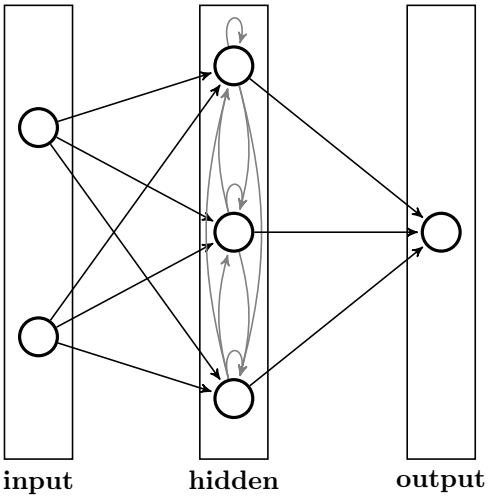


Abbildung 3: Struktur eines RNN, bestehend aus einer Eingabeschicht, einer versteckten Schicht und einer Ausgabeschicht.

den dementsprechend angepasst. Dadurch ist das Netz in der Lage, Ausgabewerte zu generieren, die möglichst oft dem Zielwert entsprechen.

Im Folgenden werde ich auf zwei verschiedene Strukturen neuronaler Netze eingehen, welche wir in diesem Projekt nutzen.

2.2.1 Recurrent Neural Network

Zunächst haben wir mit einem einfachen *Recurrent Neural Network* (RNN; Elman, 1991) gearbeitet. Ein RNN unterscheidet sich vom einfacheren Feedforward-Netz vor allem dadurch, dass es die aufeinanderfolgenden Eingabewerte nicht als unabhängig behandelt, sondern eine Relation zwischen diesen herstellen kann. Das ermöglicht ein Lernen von Sequenzen.

Die Struktur eines RNN ist in Abbildung 3 vereinfacht dargestellt. Nach der Eingabeschicht, in welcher die zu lernenden Daten eingespeist werden, folgt eine versteckte Schicht (*hidden layer*). Jede Einheit dieser Schicht ist mit jeder anderen verbunden, zuzüglich verfügen die Einheiten über Schleifen, also Kanten zu sich selbst. Dies führt dazu, dass die Informationen aus der Eingabeschicht in die versteckte übergehen, und hier beliebig lange verweilen, bevor sie zur Ausgabeschicht (oder gegebenenfalls zu weiteren Zwischenschichten) weitergereicht werden. Das Netz kann sich dadurch Informationen aus den vorherigen Eingaben merken und für die Berechnung nachfolgender Ausgaben verwenden.

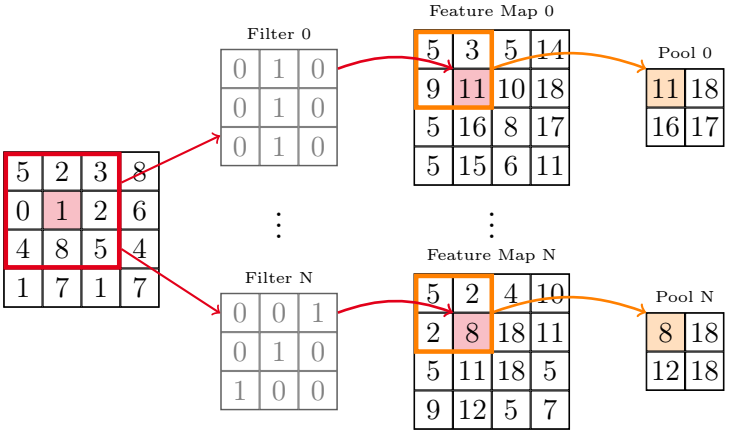


Abbildung 4: Extrahieren von Merkmalen durch ein CNN: Anwendung der Filter und Ausführung eines Max-Poolings. Die Filter betrachten jeden Wert der Eingabedaten und ermitteln anhand der Nachbarschaftswerte und ihrer eigenen Gewichtung neue Werte, welche bestimmte Merkmale hervorheben. Das Max-Pooling wählt aus disjunkten Vierecken den maximalen Wert, um die Datenmenge zu reduzieren und besondere Merkmale hervorzuheben.

2.2.2 Convolutional Network

Eine weitere Art des neuronalen Netzes ist das sogenannte *Convolutional Neural Network* (CNN, manchmal übersetzt als „Faltungsnetz"; Lecun u. a., 1990). Mithilfe von Merkmalserkennung werden prägnante Teile eines Datensatzes ermittelt, um diesen dann zu klassifizieren.

Durch das Extrahieren von Merkmalen sind CNNs außerordentlich gut darin, Muster zu erkennen. Sie finden dementsprechend oft Anwendung in der Textverarbeitung sowie in der Sprach-, Bild- und Handschriftserkennung (Kim, 2014; Abdel-Hamid u. a., 2014).

Abbildung 4 zeigt, wie ein CNN Merkmale aus Eingabedaten extrahiert. Hierbei werden verschiedene Filter angewendet, um Merkmale der Daten hervorzuheben. Zunächst werden die Daten in die Eingabeschicht gegeben. Bei einer Bilderkennung wären dies zweidimensionale Werte, welche die Pixel des Bildes repräsentieren. Es wird nun jeder Wert (in der Grafik rot unterlegt) inklusive seiner Nachbarn (rotes Viereck) mit jedem der n Filter multipliziert. Jeder Filter ist zu Beginn mit zufälligen Werten belegt und wird über die Zeit trainiert, bis er ein bestimmtes Merkmal besonders gut hervorheben kann.

Bei der schrittweisen Multiplikation der Eingangsschicht mit den Filtern entstehen n sogenannte *Feature Maps* (deutsch: Merkmalsabbildungen). In jeder Feature Map ist ein bestimmtes Merkmal der Daten hervorgehoben, abhängig vom zugehörigen Filter. Nach der Extraktion der Merkmale wird im nächsten Schritt das sogenannte *Pooling* (deutsch:

Bündelung) vorgenommen. Hierbei wird die Eingabematrix in disjunkte Vierecke geteilt, aus denen jeweils ein Wert entnommen wird, mit welchem weitergerechnet wird. Es gibt unterschiedliche Methoden des Poolings. In unserem Projekt haben wir uns für das Max-Pooling entschieden, bei dem der höchste Wert aus der 2×2-Matrix verwendet wird.

Das Anwenden der Filter (*Convolution*-Schritt) und das Pooling werden wiederholt. Nach der letzten Iteration werden die aktuellen Feature Maps in eine oder mehrere vollständig verbundene versteckte Schichten geführt, welche die nun bereits stark angepassten und reduzierten Daten schließlich klassifizieren. Durch die vorherige Aufbereitung und Reduzierung der Daten wurde die Aufgabe des Klassifizierens stark vereinfacht, dies ist wesentlich einfacher als eine direkte Klassifikation der Eingabedaten.

Die von dem CNN ermittelte Klasse der eingegebenen Daten wird mit dem eigentlichen Zielwert verglichen. Es findet anschließend der Backpropagation-Schritt statt, in welchem ermittelt wird, welche Kanten des Netzes wie stark für den Fehler verantwortlich sind. Die Kanten (und somit auch die Filter) werden dementsprechend angepasst und sind dadurch bei der nächsten Eingabe besser für die Merkmalserkennung und die korrekte Klassifizierung geeignet.

2.3 Unbalancierte Datensätze

In einem Klassifizierungsproblem gehören die Daten verschiedenen Klassen an und sollen in diese eingeteilt werden. Wenn die Klassen starke Unterschiede in der Anzahl der zugehörigen Beispiele im Lerndatensatz aufweisen, erschwert dies das Klassifizieren, da unterrepräsentierte Klassen oft von ML-Algorithmen übergangen werden.

In der realen Welt müssen viele Klassifizierungsprobleme mit unbalancierte Daten umgehen können. Ein Beispiel ist die Brustkrebsvorsorgeuntersuchung (Rayat u. a., 2016) und dementsprechend das Klassifizieren einer Brustkrebserkrankung bei Frauen. Bei den Untersuchungen in den Jahren 2014 und 2015 wurden in England 99.14% der teilnehmenden Frauen als gesund (im Hinblick auf Brustkrebs) klassifiziert. Bei der Krebsuntersuchung sind die interessanten Fälle jedoch die, bei denen die Frauen zum Zeitpunkt der Untersuchung an Brustkrebs erkrankt sind. Diese Klasse wird oft als „positive Klasse" referenziert.

Ein neuronales Netz, welches mit herkömmlichen Metriken und Kostenfunktionen gelernt wird, tendiert leicht dazu, die Datensätze *ausschließlich* der überproportional häufig vertretenen Klasse zuzuordnen, weil es dadurch bereits sehr viele Daten richtig klassifiziert. Da aber gerade die übergangene Klasse besonders wichtig ist, ist ein so gelerntes Netz nicht nützlich. Wichtig ist also, dass die Leistung eines Klassifikators nicht ausschließlich anhand der Anzahl richtig zugeordneter Daten gemessen wird. Geeignete Metriken für unbalancierte Daten werden ich in Abschnitt 5.2 vorstellen.

Vorerst erläutere ich nur den Umgang mit unbalancierten Datensätzen. Es gibt hier-

für verschiedene Ansätze, wobei zwischen internen und externen Ansätzen unterschieden wird. Die internen Ansätze passen den klassifizierenden Algorithmus an, die externen nutzen die unangepassten Algorithmen, präsentieren diesem jedoch angepasste Datensätze (Estabrooks, Jo und Japkowicz, 2010). Im Folgenden werde ich auf einige Balancierungsansätze eingehen.

Resampling (Estabrooks, Jo und Japkowicz, 2010) Beim *Resampling* wird versucht, das Ungleichgewicht der Klassen manuell anzupassen. Hierbei wird zwischen zwei Arten unterschieden, dem *Under-* und *Oversampling*. Beim Undersampling werden Datensätze der überrepräsentierten, negativen Klasse aus dem Datensatz entfernt, beim Oversampling werden mehr Datensätze der unterrepräsentierten, positiven Klasse eingefügt. Welche der beiden Modelle genutzt werden sollten, oder ob sogar eine Kombination sinnvoll ist, muss von den Daten abhängig gemacht werden. Die optimale Intensität der Anpassung, also wie sehr das Vorkommen der verschiedenen Klassen aneinander angenähert wird, ist ebenso fallabhängig und muss optimiert werden.

Bestrafung Um einen Klassifizierungsalgorithmus davon abzuhalten, sich ausschließlich auf die große Klasse zu konzentrieren, ist es möglich, eine Form der Bestrafung anzuwenden. Dafür wird die Kostenfunktion angepasst, sodass falsche Vorhersagen unterschiedlich bewertet werden. Falsch klassifizierte Datensätze der kleineren Klasse fallen dann deutlich stärker ins Gewicht als Fehler bei der größeren Klasse.

Künstliche Datensätze Wenn künstliche Daten ohne große Schwierigkeiten herzustellen sind, ist dies eine gute Möglichkeit, um dem Ungleichgewicht bei dem echten Problem entgegen zu wirken. Dies ist vor allem in mathematischen Problemstellungen oft möglich.

Dies sind einige Möglichkeiten, um mit unbalancierten Daten umgehen zu können. Die Problemstellung spielt eine große Rolle in der Auswahl einer geeigneten Methode. Eine gründliche Evaluation ist dementsprechend erforderlich.

3 State of the Art

In diesem Kapitel wird auf verwandte Projekte eingegangen. Vorgestellt werden sowohl Projekte, die eine ähnliche Hardware besitzen, als auch welche, bei denen ähnliche Lernmethoden verwendet werden.

3.1 Handschuh und maschinelles Lernen

In „Hand data glove: a wearable real-time device for human-computer interaction“ (Kumar, Verma und Prasad, 2012) wurde ein Datenhandschuh benutzt, um bestimmte Gesten zu lernen und mit deren Hilfe mit einem Endgerät zu kommunizieren. Hierbei wurden unter anderem Gesten gelernt, welche Mausklicks simulieren, oder Rotationsbewegungen ausführen. Es wurde herausgefunden, dass das Nutzen dieses Datenhandschuhs genauer und für den Menschen natürlicher ist, als das Schreiben an einer Tastatur oder das Interagieren mithilfe einer Maus. Zudem ist die räumliche Einschränkung einer Tastatur mit dem Datenhandschuh nicht gegeben.

In dem Projekt „Gestures as input: neuroelectric joysticks and keyboards“ (Wheeler und Jorgensen, 2003) wurden zunächst 4 Gesten aufgezeichnet, welche Bewegungen in 4 Richtungen an einem Steuerknüppel darstellten. Hierfür wurden zunächst 4 trockene Elektroden verwendet, welche anhand von elektrischen Signalen die Muskelaktivitäten messen. Die Elektroden wurden in einen Stoffschlauch eingenäht, welcher um den Unterarm herum gelegt wurde. Diese Manschette war nur schwer an verschiedene Armtypen anzupassen, wodurch die aufgenommenen Daten groß qualitative Unterschiede aufwiesen.

Die Daten wurden nach dem Aufzeichnen gefiltert und in einzelne Abschnitte geteilt, um einzelne Gesten zu separieren. Durch manuelle Selektion wurden unvollständige Gesten entfernt. Für die Gestenerkennung wurde für jede Geste ein eigenes *Hidden Markov Model* (HMM; Ghahramani, 2001) verwendet. Die Ergebnisse waren stark tagesformabhängig und wurden unter Anderem von der Konzentration des Probanden und der richtigen Position der Elektroden beeinflusst.

Nach dem Steuerknüppel wurden auch die Bewegungen beim Tippen auf einem Ziffernblock aufgezeichnet. Hierfür wurden Nass-Elektroden verwendet, unter Anderem, um besser die elektrischen Signale messen zu können und das Rauschen in den Daten zu

verringern. Über mehrere Tage war es jedoch kaum möglich die Sensoren an den gleichen Stellen zu platzieren, was das Aufzeichnen und Anwenden an verschiedenen Tagen sehr erschwerte. Die verschiedenen Ziffern konnten mit einer Genauigkeit zwischen 70% und 100% erkannt, wobei die Ziffern, welche sich in der mittleren Zeile des Ziffernlocks befinden, tendenziell schlechter abschnitten.

Gest [3] ist ein mit jeweils 5 IMUs[1] ausgestattetes, tragbares Gerät, das ähnlich wie unser Datenhandschuh aufgebaut ist. Mithilfe der IMUs soll es möglich sein, nicht nur eine Tastatur zu ersetzen, sondern auch durch verschiedene Gesten Programme wie zum Beispiel Musikprogramme zu bedienen oder Drohnen zu steuern. Da es sich bei Gest um ein kommerzielles Produkt handelt, ist nicht bekannt, wie das Lernen der Bewegungen umgesetzt werden sollte. Das Projekt wurde leider vor der Fertigstellung eingestellt.

Im Gegensatz dazu ist die InerTouchHand (Lobo und Trindade, 2013) nicht für eine Tastatureingabe, sondern für Gestenerkennung ausgelegt. Es handelt sich hierbei um einen Sensorhandschuh, welcher Accelerometerdaten und vibro-taktile Stimulatoren verwendet, um *Human Machine Interaction*, also die Interaktion zwischen Menschen und Computern zu ermöglichen. Mithilfe des Handschuhs soll beispielsweise eine Roboterhand bewegt werden können.

In ihrer vorherigen Arbeit (Lobo und Trindade, 2011) erläutern die Autoren eine Vorgehensweise, mit welcher es möglich ist, Gesten ausschließlich anhand von den Daten eines Accelerometers zu erkennen. Hierbei sind Accelerometer an den Fingern und in der Handinnenfläche platziert, wobei die Daten der Finger relativ zu den Daten des Sensors in der Handinnenfläche berechnet werden. Es war damit zwar möglich bestimmte Gesten zu erkennen, Drehungen um die vertikale Achse konnten jedoch nicht erkannt werden.

3.2 Convolutional Neural Networks (CNNs)

Obwohl CNNs bereits in den 90er-Jahren vorgestellt wurden (Le Cun u. a., 1990), hat sich das Interesse an ihnen erst in den letzten Jahren auch für komplexe Klassifizierungsprobleme entwickelt. Zeiler und Fergus (2014) erklären dies durch die heutige Verfügbarkeit von deutlich größeren Traingsdatensätzen, die Möglichkeit der Nutzung von GPUs für das Lernen ebensolcher großen Datensätze und der verbesserten Regulierungsmethoden für die gelernten Modelle. Sie nennen hier das Beispiel der Dropout-Methode (Hinton u. a., 2012). Hierbei werden zufällige Daten entfernt, um Overfitting zu vermeiden.

[1] *inertial measurement unit*; ein mehrteiliger Sensor, der mithilfe eines Accelerometers, eines Gyroskops und eines Magnetometers die lineare Beschleunigung, Rotationsgeschwindigkeit und Orientierung eines Objektes misst

3.3 Zeitreihen mit CNNs

Dass Convolutional Neural Networks für Bilderkennung sehr geeignet sind, ist relativ bekannt. Doch auch für andere Daten sind diese Netze geeignet.

Es gibt bereits Projekte, in denen CNNs für das Klassifizieren von Zeitreihen verwendet werden. Besonders ähnlich zu unserer Problemstellung ist die *Human Activity Recognition* (HAR), etwa aus „Efficient Dense Labeling of Human Activity Sequences from Wearables using Fully Convolutional Networks“ (Yao u. a., 2017). Hier werden drei verschiedene Datensätze verwendet, in welchen Testpersonen unterschiedliche Aktivitäten des Alltags ausüben. Mithilfe diverser Sensoren[2] wurden dabei komplexe Datensätze erzeugt, die nun klassifiziert werden sollen. Zu den aufgezeichneten Aktivitäten zählen unter anderem Stehen und Sitzen oder das Zubereiten von Kaffee und Sandwiches. Das Ziel des Projektes ist es, mithilfe von CNNs die verschiedenen Aktivitäten zu erkennen. Im Gegensatz zu früheren Methoden ist das CNN dabei in der Lage, mehrere der komplexen Muster innerhalb eines Zeitabschnittes zu erkennen, und ohne sogenannte „sliding windows” auszukommen. Dies ermöglicht eine feinere Körnung der Klassifikation.

[2] Im Opportunity-Datensatz (Roggen u. a., 2010) werden beispielsweise unter anderem Accelerometer, Gyroskope, Magnetometer, Mikrofone, Drucksensoren, Kameras, Thermometer auf der Haut und Belastungssensoren an der Kleidung der Testpersonen verwendet.

4 Verfahren

In den folgenden Abschnitten wird das Verfahren erläutert, welches das Lernen von Bewegungsdaten eines Datenhandschuhs ermöglicht. Zunächst wird der Systemaufbau erklärt und der verwendete Datenhandschuh vorgestellt. Dabei wird jedoch auf eine detaillierte Beschreibung der Entwicklung des Handschuhs verzichtet. Die genauen Details sind in der Arbeit von Paul Bienkowski (2017) beschrieben.

Desweiteren werden die verwendeten Python-Bibliotheken vorgestellt und die Datenvorverarbeitung sowie das Einteilen der Daten in geeignete Abschnitte erläutert.

Schließlich werden die benutzten Methoden des maschinellen Lernens vorgestellt.

4.1 Systemaufbau

Der Datenhandschuh misst mit den an den Fingern positionierten Sensoren die Beschleunigung (Accelerometer), die Rotationsgeschwindigkeit (Gyroskop) und das Erdmagnetfeld (Magnometer). Diese Daten werden zu sogenannten *Quaternionen*[1] fusioniert, welche wir für das Lernen verwenden.

In Abbildung 5 sind die Komponenten unseres Systems abgebildet. Die Daten der sechs IMUs werden mithilfe von drei I^2C-Bussen zum Mikroprozessor übertragen. Über WLAN oder USB-Kabel werden die Daten an den Computer geleitet. Es gibt zwei mögliche Modi, den Lernmodus (gestrichelte Linien) und den Anwendungsmodus (durchgezogene Linien).

Im Lernmodus werden die Sensordaten vom *Serial Parser* und die Tastendrücke vom *Keyboard Listener* in ein *ROS-Bag* gespeichert. Das *Robot Operating System* (ROS) [4] ist eine Sammlung von Bibliotheken und Werkzeugen für die Entwicklung von Robotersystemen.

Für die Vorverarbeitung (*Preprocessing*) werden die Daten dann aus dem ROS-Bag gelesen. Die Vorverarbeitung dient unter anderem der Reduzierung der Datendimensiona-

[1] Quaternionen sind ein mathematisches Konstrukt, nützlich um die Orientierung von Objekten im Raum oder Drehungen zu beschreiben.

lität. Zudem werden die benötigten Daten zu regelmäßigen Zeitpunkten ermittelt, die durch das zeitversetzte Senden der IMUs eventuell nicht direkt vorliegen. Nach dem Vorverarbeiten werden die Daten dann in geeignete Samples geteilt. Jedes Sample besteht hierbei aus einer festen Anzahl an Zeitschritten und enthält jeweils einen Tastendruck. Außerdem werden Samples für die Klasse $\emptyset$, also „keine Taste gedrückt", angelegt. Das *Sampling* dient dazu, einheitliche, zeitdiskrete Daten an das neuronale Netz zu geben.

Beim *Learning* werden die Samples von dem Netz gelernt, die gelernten Netzparameter bilden das Modell und werden gespeichert, um sie später anwenden zu können.

Im Anwendungsmodus werden die Lernschritte ausgelassen. Die Daten werden zwar immer noch in Samples geteilt, diese werden jedoch nicht mehr gezielt ausgesucht. Stattdessen wird jeweils eine feste Anzahl der zuletzt eingetroffenen Zeitschritte als ein Sample zusammengefasst und ausgewertet. Dadurch entsteht ein gleitendes Zeitfenster, einzelne Datenpunkte sind in mehreren Samples enthalten. Für das Anwenden werden die im Vorfeld gelernten Netzparameter geladen. Die vom Netz getätigten Ausgaben werden an das Betriebssystem weitergegeben und als Tastatureingaben interpretiert.

4.1.1 Der Datenhandschuh

Der fertige Datenhandschuh ist in Abbildung 6 zu sehen. Er besteht aus

- sechs Inertial Measurement Units (IMUs) des Typs BNO055, befestigt an den Fingern und dem Handrücken,
- einem fingerlosen dünnen Baumwollhandschuh zur Befestigung der Hand-IMU,
- 5 Ringen aus elastischem Textilband zur Befestigung der IMUs am zweiten Segment der Finger,
- dem Mikroprozessor Adafruit Feather M0 WiFi („Featherboard"),
- einer Entwicklungsplatine, welche auf den Mikroprozessor gesteckt wird und sowohl die Verkabelung vereinfacht, als auch Platz für ein EEPROM und die benötigten Pull-Up-Widerstände bietet,
- einem breiten Band ums Handgelenk, auf welchem der Mikroprozessor mit der Entwicklungsplatine befestigt ist, sowie
- einer Steckverbindung und diversen Kabeln zur Verbindung der Komponenten.

Die IMUs enthalten jeweils ein Accelerometer, ein Gyroskop und einen 3D-Kompass (Magnetometer). Außerdem ist ein Mikroprozessor mit einem vorinstallierten Fusionsalgorithmus enthalten, mit dessen Hilfe die IMU die einzelnen Sensordaten zu einer Quaternion fusioniert. Diese Quaternion gibt die absolute Orientierung des Sensors im Raum an.

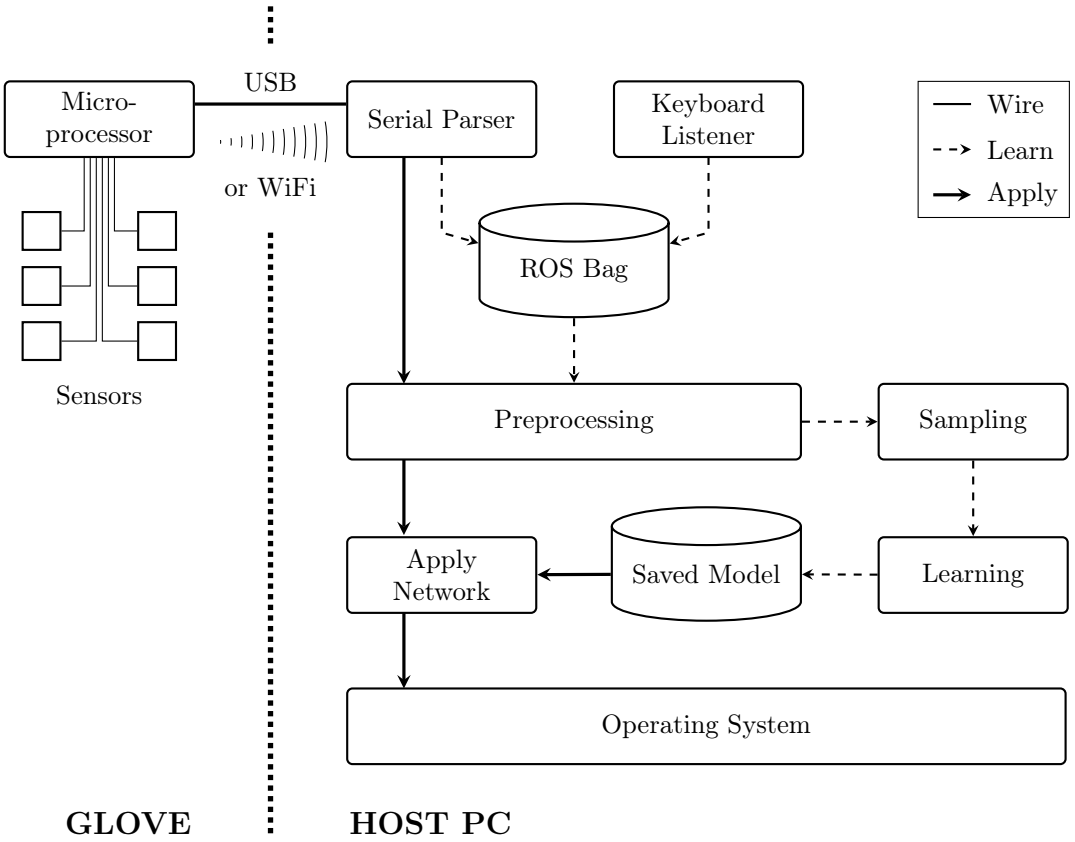
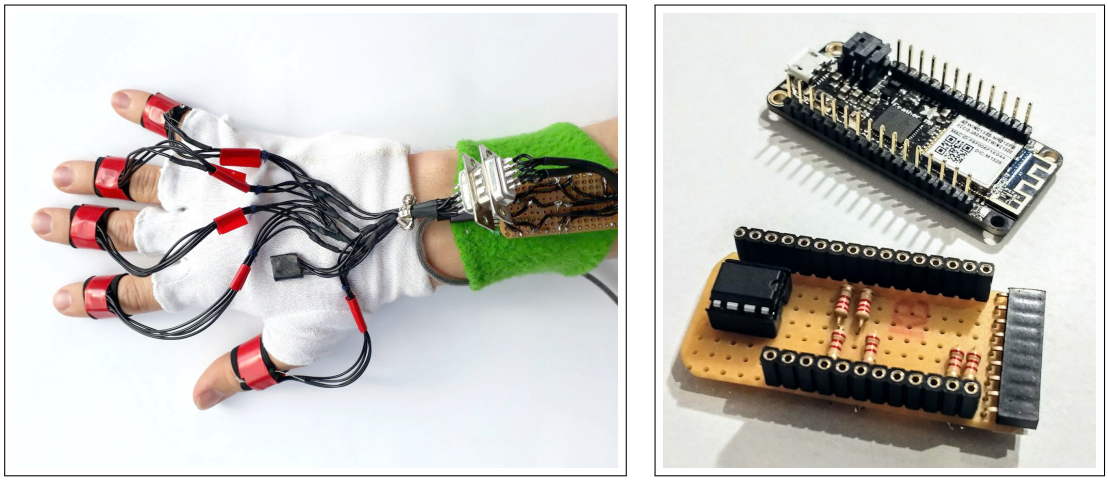


Abbildung 5: Aufbau des Systems. Die Sensoren zeichnen Daten auf, welche vom Mikroprozessor entweder über USB oder über WLAN an den *Serial Parser* übermittelt werden. Im Lernmodus werden die Daten mitsamt der dazugehörigen Tastendrücke aus dem *Keyboard Listener* in ein *ROS-Bag* gespeichert. Nach der Vorverarbeitung und der Einteilung in geeignete Samples wird ein neuronales Netz anhand der Daten trainiert und das gelernte Modell gespeichert. Im Anwendungsmodus werden die Lernschritte übersprungen und das bereits gelernte Netz geladen, um nach dem Vorverarbeiten der Daten diese zu klassifizieren.



(a) Fertiggestellter Handschuh mit 6 IMUs auf den Fingern und dem Handrücken. Der Mikroprozessor wird durch eine Steckverbindung mit den IMUs verbunden.

(b) Featherboard mit Entwicklungsplatine; die Platine wird auf den Mikroprozessor aufgesteckt. Diese enthält dem Mikroprozessor fehlende Widerstände und ein EEPROM. Durch die Platine wird zudem die Verkabelung vereinfacht.

Abbildung 6: Hier ist der Prototyp des Datenhandschuhs mit den einzelen Komponenten abgebildet.

4.1.2 Datenübertragung

Die IMUs erfassen die Bewegungen der Finger und der Hand. Diese Daten werden über einen I^2C-Bus vom Featherboard ausgelesen. Da die IMUs über jeweils 2 verschiedene Adressen ansprechbar sind, reichen die insgesamt 3 freien I^2C-Busse des Featherboards aus. Vom Featherboard werden die IMU-Daten in einzelnen Paketen an den Host-PC übermittelt. Jedes Paket enthält die Accelerometer- und Gyroskopdaten sowie die fusionierte Quaternion. Dies ergibt pro Paket 10 Werte mit insgesamt 20 Bytes. Die Pakete können seriell per USB oder über WLAN und UDP übermittelt werden.

Mithilfe eines in Python geschriebenen Parsers werden die Pakete dann in ROS-Nachrichten verwandelt. Dies ermöglicht ein leichtes Aufzeichnen und späteres Verwenden der Daten, um mit diesen zeitlich unabhängig lernen zu können. Ein Live-Betrieb zur Anwendung des gelernten Modells ist ebenfalls möglich.

4.2 Bibliotheken für maschinelles Lernen

Für das Arbeiten mit neuronalen Netzen gibt es einige Frameworks und Bibliotheken, um die Anforderungen an die Komplexität der Netze leichter erfüllen zu können. Wir haben uns für die Kombination aus Theano (Al-Rfou u. a., 2016) und Lasagne [5] entschieden.

Theano ist eine open-source Python-Bibliothek, die auf komplexe und große wissenschaftliche Berechnungen ausgelegt ist und auf NumPy (Walt, Colbert und Varoquaux, 2011) basiert. Wir nutzen NumPy zudem für einfache Datenverarbeitungsschritte, wie beispielsweise beim Vorverarbeiten. Theano optimiert Berechnungen mit multi-dimensionalen Arrays mit einem Fokus auf Effizienz und Schnelligkeit. Desweiteren ermöglicht Theano ohne großen Aufwand eine Ausführung der Berechnung auf einer oder mehreren CPUs oder sogar auf GPUs. Hierfür muss lediglich eine entsprechende Umgebungsvariable gesetzt werden:

```
1 # CPU (4 Threads)
2 export THEANO_FLAGS=openmp=True
3 export OMP_NUM_THREADS=4
4 python script.py
5
6 # GPU
7 export THEANO_FLAGS=device=cuda,floatX=float32
8 python script.py
```

Codebeispiel 1: Theano Umgebungsvariablen zur Steuerung des Ausführungsziels. Durch das Setzen dieser kann entschieden werden, ob auf einer GPU oder einer oder mehreren CPUs gelernt werden soll.

Lasagne ist eine auf Theano basierende Bibliothek, die auf neuronale Netze spezialisiert ist. Mit Hilfe von Lasagne ist es leicht möglich, verschiedene Arten von Schichten eines neuronalen Netzes als Theano-Formeln zu konfigurieren. Zudem kann das Gesamtnetz ohne große Komplikationen initialisiert, trainiert und schließlich auch angewendet werden. Außerdem ist es ist mithilfe von Lasagne, Theano und NumPy auch möglich, die gelernten Netzparameter zu serialisieren, sodass gelernte Netze leicht gespeichert und später für weitere Nutzung geladen werden können. Lasagne formulierte alle Netzwerkoperationen so, dass Stapelverarbeitung mehrerer Samples leicht möglich ist.

Das Verwenden von Lasagne soll hier anhand eines sehr einfachen CNN vorgestellt werden:

Zunächst wird eine Eingabeschicht erzeugt. Das `shape`-Argument beschreibt die Dimensionalität der Eingabedaten. Hierbei bleibt die erste Dimension `None`, dadurch wird die Stapelgröße anhand der Eingabedaten ermittelt. Ein Stapel besteht in der Regel aus mehreren Samples, um die Berechnung zu beschleunigen.

```
l_current = l_in = lasagne.layers.InputLayer(
        (None, SEQUENCE_LENGTH, N_INPUTS))

for x in range(CONVOLUTION_LAYER_PAIRS):
    l_current = lasagne.layers.Conv2DLayer(
            l_current, FILTER_COUNT, FILTER_SIZE)

    l_current = lasagne.layers.MaxPool2DLayer(
            l_current, POOLING_SIZE)

l_out = lasagne.layers.DenseLayer(l_current, N_OUTPUTS)
```

Codebeispiel 2: Modellierung eines einfachen CNN in Lasagne mit einer Eingabeschicht, einigen Convolution/Pooling-Schicht-Paaren und schließlich einer Ausgabeschicht.

In der Schleife werden die Convolution- und Max-Pooling-Schichten angelegt. Die Größen `FILTER_SIZE` und `FILTER_COUNT` geben die Dimension und die Anzahl der genutzten Filter an, die `POOLING_SIZE` bestimmt die Größe der Region, aus welcher im Pooling-Schritt der größte Wert übernommen wird.

Um die Reihenfolge der Schichten zu definieren, wird die aktuelle Schicht in einer Variable gespeichert und in der nächsten Schicht als Vorgänger angegeben.

Schließlich wird die vollständig verknüpfte Ausgabeschicht (*dense layer*) ans Ende des Netzes gestellt.

4.3 Vorverarbeitung

Bevor die Daten an das neuronale Netz gegeben werden, werden sie vorverarbeitet. Ziel der Vorverarbeitung ist es, die wichtigen Informationen besser zugänglich zu machen und die zu evaluierende Datenmenge zu reduzieren.

Wir gehen zunächst davon aus, dass die Quaternionen bereits einen guten Überblick über die Bewegungsabläufe der verschiedenen Tastendrücke geben, weshalb wir zu Beginn ausschließlich diese verwenden und auf die rohen Sensorwerte des Accelerometers und des Gyroskops verzichten. Dies spart uns Komplexität im Modell, und wir müssen uns zudem nicht um eine geeignete Kalibrierung während des Tippens kümmern.

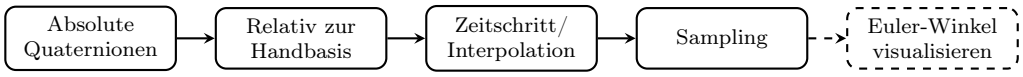
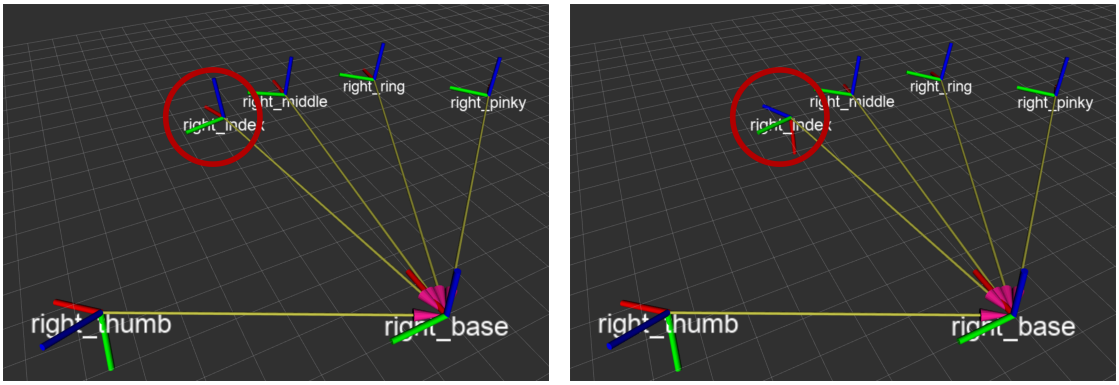


Abbildung 7: Schritte des Preprocessings



(a) Alle Finger sind gerade gestreckt.

(b) Der Zeigefinger wird gebeugt, die restlichen Finger bleiben gestreckt.

Abbildung 8: Darstellung der Hand- und Fingerorientierungen mithilfe des ROS-Visualisierungs-Werkzeugs „RViz". Für die Darstellung wurden vordefinierte Sensorpositionen verwendet, welche jedoch nicht für den Lernvorgang benutzt werden.

Wie in Abbildung 7 zu sehen, beginnen wir bei der Vorverarbeitung damit, dass wir ausschließlich die Quaternionen nutzen. Durch das Ignorieren der Beschleunigung und der Rotationsgeschwindigkeit gehen zwar nützliche Informationen verloren, es reduziert jedoch auch drastisch die Datenmenge. Da es sich in diesem Projekt um einen Prototyp handelt ist dies von großem Nutzen, denn Ergebnisse sind so schneller zu erzielen. In der Zukunft wird es sich höchstwahrscheinlich als sinnvoll erweisen, die bisher ungenutzten Daten wieder mit einzubeziehen, vor allem wenn mit dem Handschuh immer größere Bewegungen ermöglicht werden sollen, um weit entfernte Tasten drücken zu können, oder auch um verschiedene Gesten durchführen zu können.

Um Rückschlüsse auf die Fingerbewegungen machen zu können, berechnen wir die Quaternionen der Finger relativ zu der Quaternion der Hand-IMU. Relative Quaternionen erhält man durch die Multiplikation mit der Inversen der Bezugsorientierung:

$$Q_{\text{rel}}^{\text{finger}} = Q_{\text{abs}}^{\text{finger}} \cdot inverse(Q_{\text{abs}}^{\text{basis}})$$

Abbildung 8 zeigt einen Screenshot von dem ROS-Visualisierungs-Werkzeug RViz, bei dem der Zeigefinger bewegt wird. In Abbildung 8a sind sowohl der Zeigefinger als auch die anderen Finger gestreckt. Beim Abknicken des Zeigefingers ist in Abbildung 8b der relative Winkel der IMU des Zeigefingers zur IMU auf dem Handrücken zu erkennen.

Die Hand-IMU wird relativ zu ihrem gleitenden Mittelwert errechnet. Der gleitende Mittelwert wird verwendet, um einen lokalen Durchschnittswert zu ermitteln. Die zuletzt erfassten Werte werden darin am stärksten gewichtet. Änderungen des Mittelwertes über einen größeren Zeitraum werden also vom gleitenden Mittelwert erfasst, schnelle Ände-

rungen jedoch nicht.

Der gleitende Mittelwert wird wie folgt berechnet:

$$X^i_{avg} = lerp\left(X^{i-1}_{avg}, X^t, \left(t^i - t^{i-1}\right) \cdot r\right)$$

X^i ist der Wert und X^i_{avg} der gleitende Mittelwert im Zeitschritt i, t^i ist die tatsächliche Zeit während des Zeitschrittes i, die Anpassungsrate wird als r bezeichnet. Die Funktion $lerp(a, b, f)$ beschreibt die lineare Interpolation zwischen den Werten a und b im Verhältnis f:

$$lerp(a, b, f) := a + (b - a) \cdot f$$

Für Quaternionen verwenden wir statt *lerp* die sphärische lineare Interpolation *slerp*. Diese berücksichtigt die Interpretation der Quaternion als Angabe für eine Rotation im Raum und interpoliert entsprechend der kürzesten Strecke zwischen zwei Punkten auf der Oberfläche einer Kugel. Das Ergebnis ist wieder eine Einheitsquaternion, die die Teilrotation angibt (Shoemake, 1985).

Das Berechnen der relativen Quaternionen hat mehrere Vorteile:

1. Da das Magnometer das Erdmagnetfeld misst, liefert jede IMU absolut ausgerichtete Werte. Die zur Basis-IMU relativen Quaternionen der Finger-IMUs sind hingegen unabhängig davon, in welche Richtung die Hand zeigt.
2. Die Quaternionen stehen nicht mehr für jeden Finger einzeln, sondern sind in einen Zusammenhang gebracht. Die Fingerbewegung ist unabhängig von der Handbewegung erkennbar. Wird nur die komplette Hand bewegt, aber nicht die einzelnen Finger, ist keine Veränderung an den Daten der Finger erkennbar.
3. Um auch die Basis-IMU von der absoluten Orientierung unabhängig zu erhalten, haben wir den gleitenden Mittelwert als Bezug gewählt. Dieser passt sich langsam der durchschnittlichen Orientierung der Hand an. Dadurch ist es möglich, zwischen Handbewegungen zu unterscheiden, welche für das Drücken entfernter Tasten nötig sind und solchen, die einer Positionsveränderung beim Schreiben dienen.

4.3.1 Fester Zeitschritt und Interpolation

Wir haben uns bei der Datenerhebung für die einfache Variante eines festen Zeitschrittes entschieden. Zeitschritte, welche je nach Datenaufkommen kürzer oder länger (also dynamisch) sind, können zwar besser mit bewegungsintensiven beziehungsweise ruhigen Phasen umgehen, benötigen jedoch eine aufwändigere Implementation.

Da die Datenübertragung aller IMUs nicht synchron stattfindet, kommt es vor, dass zu einem festen Zeitschritt die aktuellen Daten der IMUs fehlen. Um trotzdem eine

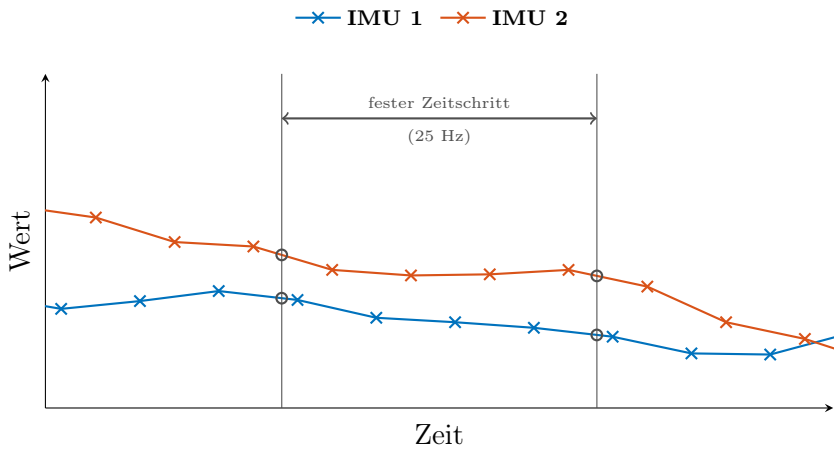


Abbildung 9: Interpolation der IMU-Daten. Die IMUs senden mit 100 Hz, die festen Zeitschritte zu denen neue Daten benötigt werden haben jedoch eine Frequenz von 25 Hz. Durch Interpolation werden die benötigten Daten ermittelt. Dies erfordert das Abwarten der nachfolgenden Datenpunkte.

präzise Berechnung zu ermöglichen, versuchen wir die Daten der IMUs möglichst genau anzunähern. Das Vorgehen, welches wir hierfür benutzen, nennt sich Interpolation.

In Abbildung 9 sind beispielhaft zwei Zeitschritte sowie zwei Wertekurven zu sehen. Um zum Zeitpunkt des Zeitschrittes die aktuellen Werte zu bekommen nutzen wir lineare Interpolation zwischen dem vorherigen und dem nachfolgenden Wert. Für Quaternionen verwenden wir erneut die sphärische lineare Interpolation.

Um den auf den Zeitschritt folgenden Wert für jede IMU zu erhalten und korrekt interpolieren zu können, müssen wir zunächst darauf warten, dass alle IMUs neue Werte gesendet haben. Da die Zeitschritte mit einer Frequenz von 25 Hz stattfinden, die IMUs jedoch mit einer Rate von knapp 100Hz senden, ist diese Wartezeit relativ gering, sie beträgt höchstens 10 ms.

4.3.2 Sampling (CNN)

Für den Ansatz mit dem CNN teilen wir die Datenströme in Samples auf. Jedes Sample enthält eine feste Anzahl an Zeitschritten vor und nach einem Tastendruck. Für die Samples der Klasse $\emptyset$ werden die selbe Anzahl aufeinanderfolgender Zeitschritte verwendet, jedoch aus einem Zeitabschnitt, in dem kein Tastendruck vorhanden ist.

Bisher befanden sich die Tastendrücke in der Mitte der Samples, es ist jedoch denkbar, die Anzahl der Zeitschritte nach dem Tastendruck zu reduzieren. Dadurch würde die Verzögerung veringert, die bei der Anwendung des gelernten Netzes durch das Warten auf die restlichen Zeitschritte entsteht.

4.4 Visualisierung der Daten

Um genau zu sehen, mit welchen Daten wir es zu tun haben, beziehungsweise auf welchen Daten das CNN lernen soll, haben wir die Winkel *Yaw* und *Pitch* aus den Quaternionen extrahiert. *Roll* ist in unserem Fall ausschließlich für die Hand-IMU, nicht aber für die Finger-IMUs von Interesse, da die Handkinematik ein Rollen der Finger relativ zur Hand nicht ermöglicht.

Zum Visualisieren haben wir die Daten genau wie für das Lernen in Samples geteilt und diese mithilfe der Bibliothek Matplotlib (Hunter, 2007) übereinander gezeichnet, sodass jeweils ein Tastenanschlag am Zeitpunkt 0 angeordnet war. Das Ergebnis ist in Abbildung 10 zu sehen.

Wir haben diese Graphen für verschiedene Finger- und Tastenkombinationen erstellt. Die entstehenden Muster sind für die verschiedenen Tasten gut unterscheidbar, daher konnten wir davon ausgehen, dass auch ein ML-Algorithmus dazu in der Lage sein würde, sie zu unterscheiden.

4.5 Netzarchitektur

Dieser Abschnitt behandelt die Konfiguration der verwendeten neuronalen Netze. Es wird zunächst die Konfiguration des RNN beschrieben, anschließend wird auf die Konfiguration des CNN eingegangen.

4.5.1 Recurrent Neural Network

Unser erster Ansatz für eine Netzarchitektur basierte auf einem RNN. Hiermit versprachen wir uns, die Bewegungsabläufe effizient lernen zu können, da RNNs für sequezielle Daten geeignet sind. Wir stießen recht schnell auf diverse Probleme. Ein Problem, welches man oft bei der Nutzung eines RNN lösen muss, ist das *vanishing gradient problem* (Hochreiter, 2004). Es bezeichnet das Phänomen, dass die vorderen Kanten eines Netzes kaum angepasst werden, da der ermittelte Anteil am gemachten Fehler vor allem auf die der Ausgabeschicht nahen Kanten verteilt werden. Da ein RNN meist ein sehr tiefes Netz ist, verschwindet der Anteil immer weiter, die ersten Kanten akkurat zu trainieren dauert dementsprechend lange.

Ein weiteres Problem entsteht durch die Struktur der Daten, mit welchen wir das Netz trainieren. Unsere Aufgabenstellung ist ein Klassifikationsproblem mit mehreren Klassen. Die positiven Klassen beinhalten ausschließlich die Zeitpunkte, zu denen gerade eine Taste hiuntergedrückt wird. Die große, negative Klasse beinhaltet alle Zeitpunkte, in denen eine gedrückte Taste im Vorfeld bereits gedrückt war, oder aber in denen gar keine Taste gedrückt ist. Es handelt sich um stark unbalancierte Daten, wie sie in Kapitel 2

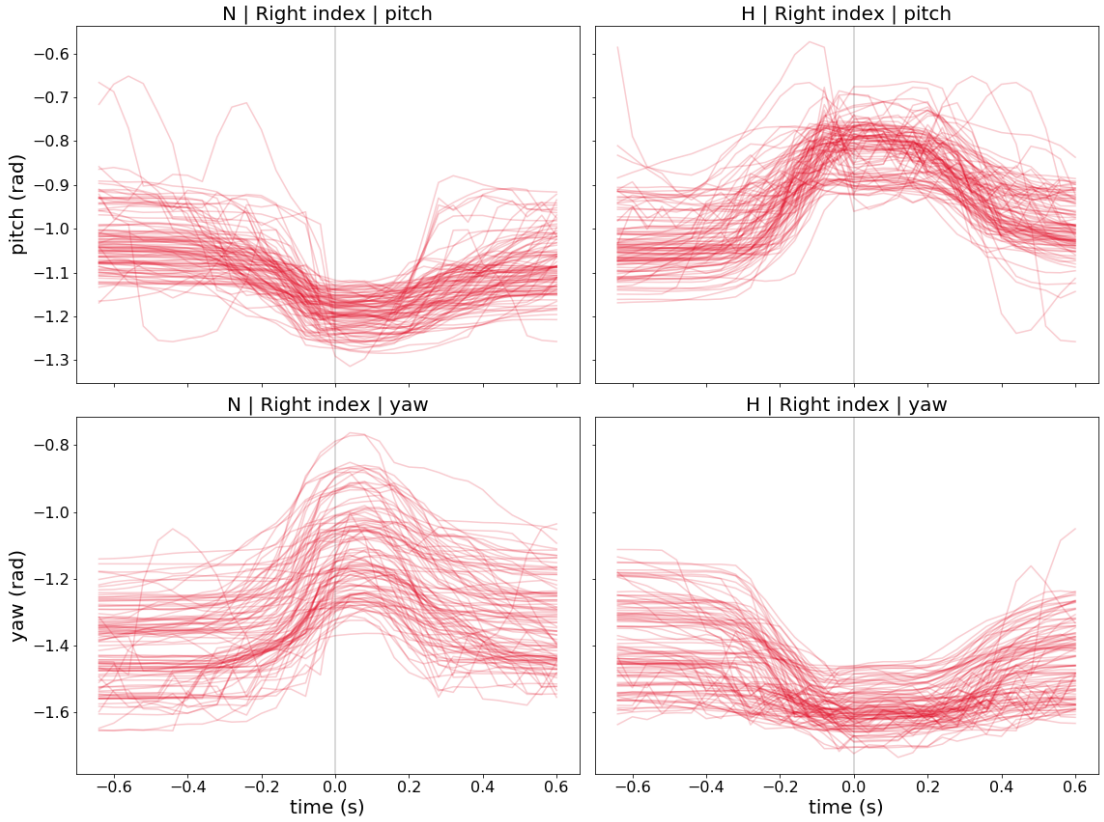


Abbildung 10: Mehrfache Wiederholungen des Drückens der Taste N (links) und H (rechts), im Moment des Tastendrucks übereinander gelegt. Zu sehen sind jeweils der extrahierte Pitch- (oben) und Yaw-Winkel (unten) des rechten Zeigefingers.

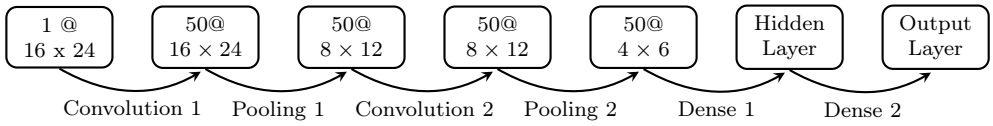


Abbildung 11: Die Architektur des verwendeten CNN. Es ist die Veränderung in der Anzahl und der Größe der Feature Maps zu sehen, welche durch die abwechselnden Convolution- und Poolingschritten entstehen.

bereits beschrieben werden.

Dort wurden auch verschiedene Methoden vorgestellt, um mit unbalancierten Daten umzugehen.

Leider erwiesen sich diese als nur bedingt geeignet für unseren Anwendungsfall. Unsere Daten stellen Bewegungsabläufe dar, welche von dem RNN auch als solche gelernt werden sollen. Es ist nicht möglich, die Daten in einzelne Sequenzen zu teilen, ohne genau zu wissen, an welcher Stelle man die Daten teilen kann, ohne einen zusammengehörenden Bewegungsablauf zu durchtrennen. Resampling war demnach nicht möglich.

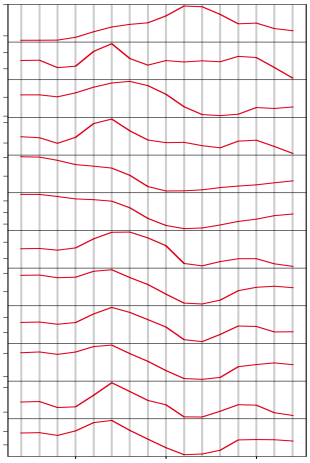
Auch bei der Verwendung diverser bestrafender Kostenfunktionen erzielten wir kein gutes Ergebnis. Lasagne bietet einige bereits implementierte Kostenfunktionen, welche falsches Klassifizieren unterschiedlich stark gewichten. Leider konnten wir jedoch keine Kostenfunktion finden, welche eine signifikante Verbesserung des Lernens erbrachte.

Nach der ersten Versuchs-Phase (Abschnitt **??**) entschieden wir uns dazu, ein anderes neuronales Netz zu verwenden und zusätzlich die Vorverarbeitung der Daten zu verbessern.

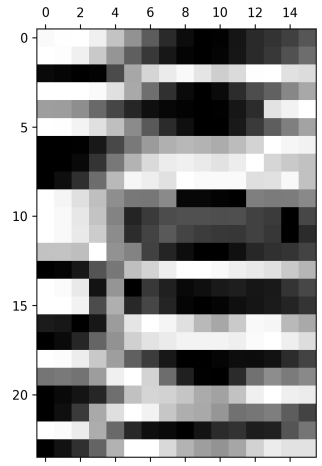
4.5.2 Convolutional Neural Network

Wir änderten die Netzwerkarchitektur und verwendeten nun ein CNN. Ein großer Vorteil ist hierbei, dass die Erkennung der Muster auf festen Zeitabschnitten möglich ist, es ist also nicht mehr nötig, zusammenhängende Sequenzen zu lernen. Somit konnten wir die Daten recht leicht in die verschiedenen Samples teilen, wodurch auch das Resampling erleichtert wurde. Für jede zu lernende Taste wurden ähnlich viele Tastendrücke aufgezeichnet. Dadurch waren die Klassen der Tastendrücke gleichmäßig verteilt. Um auch die Klasse $\emptyset$ lernen zu können, fügten wir zu den Tasten-Samples entsprechend viele weitere Samples hinzu, in denen kein Tastendruck enthalten war. Hierbei achteten wir darauf, einen Minimalabstand zu den Tastendrücken einzuhalten, damit die Klasse $\emptyset$ auch keine Tastendrücke am Rand der Samples enthält.

Wir arbeiten mit der in Abbildung 11 gezeigten Netzstruktur. Die Eingabedaten sind die Quaternionen aller 6 IMUs zu den festen Zeitschritten. Diese ergeben eine 16×24 große Eingabematrix, bei einer Samplegröße von 16 Zeitschritten. Der Tastendruck befindet



(a) Einzelwerte, hier Pitch- und Yaw-Winkel, aller IMUs (vertikal) über die 16 Zeitschritte des Samples (horizontal)



(b) Alternative Darstellung der Eingabewerte, hier die kompletten Quaternionen als Datenmatrix (zeilenweise normalisiert). Jede Zeile in der Matrix entspricht einem Einzelwert, jede Spalte einem der 16 Zeitschritte.

Abbildung 12: Eingabedaten für das CNN auf verschiedene Arten visualisiert.

sich in der Mitte der Zeitschritte.

Mithilfe der 50 Filter werden 50 Feature Maps der gleichen Dimension erzeugt. Beim Pooling werden diese Feature Maps dann auf die Größe 8×12 reduziert. Der nächste Convolution-Schritt führt wiederum zu 50 Feature Maps. Dass sich die Anzahl der Feature Maps nicht erhöht, liegt daran, dass jeder Filter alle vorherigen Feature Maps verwendet. Die Filter haben also die Form $3 \times 3 \times 50$.

Nach dem nächsten Pooling-Schritt werden die Feature Maps in die erste versteckte Schicht des einfachen Klassifikators gereicht und von dort aus wird in der Ausgabeschicht die Klasse ermittelt.

In Abbildung 12 sind die Eingabedaten des CNN zu sehen. In Abbildung 12a wurden aus den Quaternionen die Yaw- und Pitch-Winkel extrahiert, da diese visuell leichter zu deuten sind als ganze Quaternionen, welche eigentlich für das Lernen genutzt werden. Ein Sample besteht aus 16 Zeitschritten und beinhaltet die kompletten Quaternionen, also pro IMU 4 Werte. In sind die Eingabedaten, in diesem Fall tatsächliche Quaternionen, als Graustufenmatrix gezeigt. Jedes Sample ist eine solche Matrix, die Eingabeschicht des CNN hat eine entsprechende Form.

4.6 Konfiguration

Der Aufbau des neuronalen Netzes bietet viele Variablen, die sein Verhalten beeinflussen. Diese Variablen müssen angepasst werden, um für den Anwendungsfall optimale Eigenschaften zu erhalten.

Um Experimente nachvollziehbar und wiederholbar und die verschiedenen Konfigurationen leicht veränderbar zu gestalten, wurde eine Konfigurationsdatei angelegt (siehe Codebeispiel 3), deren Variablen hier vorgestellt werden.

```yaml
# General / sampling
imu_ids: [0, 1, 2, 3, 4, 5]
key_codes: [20, 34, 48, 21, 35, 49, 22, 36, 50, 57]

# Sampling
sequence_length: 16
sequence_ratio: 0.5
sampling_mode: quat
sampling_rate: 25
base_imu_relative_average_rate: 0.2

# Learning / network
epochs: 0
batch_size: 0
training_ratio: 0.8
cost_function: mse
network_type: cnn2d
learning_rate: 0.002

# RNN
n_hidden: 20

# CNN
convolution_filter_count: 50
convolution_filter_size: [3, 3]
convolution_iterations: 2
convolution_dense_layer_units: [10]
convolution_deep_filters: true
```

Codebeispiel 3: Konfigurationsdatei für den Lernprozess. Die Konfigurationsdatei wird geladen und anhand ihrer Einstellungen wird das Verhalten der Software angepasst. Unter anderem können das Lernverhalten, die Netzwerkarchitektur sowie die verwendeten Daten und deren Zerlegung in Samples eingestellt werden.

`imu_ids` Hier wird definiert, welche IMUs des Handschuhs verwendet werden sollen. Zu

Beginn des Projektes wurden oft nur eine oder zwei IMUs benötigt, um die Daten, welche die IMUs liefern zu analysieren.

`key_codes` Die Key Codes der zu lernenden Tasten stammen vom Linux Input Event System[2]. Diese Variable beeinflusst, welche Tasten das neuronalen Netz zu erkennen lernt. Anhand ihrer wird die Größe der Ausgabeschicht ermittelt. Sie ist außerdem in der Anwendung des gelernten Modells nötig, um die Ausgaben in Tastendrücke zurück zu übersetzen.

`sequence_length` Mit der Länge der Sequenz wird die Anzahl der verwendeten Zeitschritte pro Sample festgelegt.

`sequence_ratio` Durch diese Variable wird die Verteilung der Zeitschritte vor und nach dem Tastendruck bestimmt. Bisher haben wir eine gleichmäßige Aufteilung gewählt, es ist jedoch denkbar, die Anzahl der Zeitschritte nach einem Tastendruck zu verringern, um entstehende Verzögerungen zu reduzieren.

`sampling_mode` Hier wird festgehalten, welche Daten zum Lernen verwendet werden. Mögliche Ausprägungen sind `quat`, `angles` und `full`. Bisher lernen wir ausschließlich auf den Quaternionen, im weiteren Verlauf könnte es aber sinnvoll sein, die rohen Daten des Accelerometers und des Gyroskops ebenfalls zu verwenden.

`sampling_rate` Die Frequenz der Zeitschritte wird in Hertz angegeben und bestimmt die Abstände der Zeitpunkte, in denen die IMU-Daten gebündelt und aufgezeichnet werden. Wenn zu den Quaternionen weitere Werte zum Lernen verwendet werden, kann es sinnvoll sein, eine höhere Frequenz zu verwenden.

`base_imu_relative_average_rate` Dieser Wert bezeichnet die Anpassungsrate des gleitenden Mittelwertes der Hand-IMU. Hierfür haben wir experimentell den Wert $r = 0.2$ als geeignet ermittelt.

`epochs` Mit dieser Einstellung lässt sich bestimmen, wie viele Epochen ein Netz lernen soll. Der Wert 0 besagt hier, dass solange immer weitere Epochen gelernt werden sollen, bis der Vorgang abgebrochen wird.

`batch_size` Die Stapelgröße, mit der gelernt wird, ist über diese Variable anpassbar. Ein Stapel ist ein Zusammenschluss von in unserem Fall 100 Samples, jede Epoche trainiert mit einem Stapel dieser Größe.

`training_ratio` Das Lernen eines neuronalen Netzes findet mit zwei disjunkten Teilen eines Datensatzes statt. Der größere Teil ist der Trainingsdatensatz. Dieser wird benutzt, um das Netz zu trainieren. Der andere Teil des Datensatzes ist für die Evaluation vorgesehen. Hierbei wird getestet, ob das Netz das Gelernte auch auf ihm unbekannte Daten anwenden kann, oder ob es die Generalisierungsfähigkeit durch zu starkes Lernen verloren hat. In unserem Fall ist die Aufteilung 0.8, das heißt 80% der Daten stehen als Trainingsdaten zur Verfügung, der Rest wird zum

[2] in einem Linux Betriebsystem unter `/usr/include/linux/input-event-codes.h` einzusehen

Testen verwendet.

cost_function Die Kostenfunktion errechnet den Fehler der Ausgabe eines neuronalen Netzes. Wir verwenden den *Mean Squared Error* (MSE). Diese einfache Kostenfunktion ist für das CNN ausreichend, da auf einem balancierten Datensatz gelernt wird. Beim RNN testeten wir viele verschiedene Kostenfunktionen aus, um trotz unbalanciertem Datensatz eine gute Genauigkeit zu erreichen, ohne die unterrepräsentierten Klassen zu übergehen.

network_type Hier kann zwischen den zwei verschiedenen Netztypen gewählt werden, einem RNN oder einem CNN.

learning_rate Die Lernrate eines neuronalen Netzes bezeichnet den Grad der Anpassung anhand der Fehlerrate. Eine zu geringe Lernrate bewirkt, dass das Netz Anpassungen nur sehr langsam vollzieht, das Lernen dauert dementsprechend lange. Ist die Lernrate jedoch zu hoch, kann es passieren, dass nie ein optimal gelerntes Netz erhalten wird, da die Kantengewichtsanpassungen dazu führen, dass über das Ziel hinausgeschossen wird und die Vorhersagen des Netzes zu oszillieren beginnen. Wir beginnen mit einer Lernrate von 0.002 und nutzen den AdaGrad-Algorithmus von Duchi, Hazan und Singer (2011), um die Lernrate mit vergehender Zeit zu verringern. Dies sorgt dafür, dass spätere Anpassungen nur noch geringe Veränderungen bewirken.

n_hidden Bei dieser Variable handelt es sich um die Anzahl der Einheiten innerhalb der versteckten Schicht des RNN.

convolution_filter_count Die Anzahl der Filter pro Convolution-Schritt.

convolution_filter_size Die Filter im CNN haben eine feste Größe. Diese bestimmt den Einfluss der Nachbarschaft auf den vom Filter ermittelten Wert. Wir haben uns für Filter der Größe 3×3 entschieden.

convolution_iterations Ein CNN vollzieht mehrere Iterationen der Convolution- und Poolingschritte. Wir verwendeten 2 Iterationen, ob diese Anzahl genügt, oder ob mehr Iterationen gemacht werden sollten, muss in weiteren Versuchen herausgefunden werden.

convolution_dense_layer_units Die Anzahl der Einheiten der vollständig verbundenen Schicht zur Klassifizierung am Ende des CNN haben wir zu Beginn auf 10 gesetzt. Da die Ausgabeschicht im weiteren Verlauf der Experimente mehr als 10 Einheiten enthielt, wurde diese Zahl nach oben korrigiert. Diese Variable verwendet eine Liste von Werten, sodass eine beliebige Anzahl vollständig verbundener Schichten unterschiedlicher Größe wählbar ist.

convolution_deep_filters Hier wird ausgewählt, ob tiefe Filters verwendet werden sollen. Tiefe Filter werden auf allen vorherigen Feature Maps angewandt, flache Filter nur auf jeweils einer. Dadurch führen flache Filter zu einer exponentiell wachsenden Anzahl an Feature Maps. Wir verwenden tiefe Filter, im Grunde haben

diese ab der zweiten Convolution-Schritt also die Größe $3 \times 3 \times 50$. Dies reduziert die Komplexität des Netzes.

5 Evaluation

In diesem Kapitel geht es um die Bewertung der Vorverarbeitung und die durchgeführten Experimente. Die zur Evaluation der Vorverarbeitung, des genutzten ML-Algorithmus und der Experimente benötigten Metriken werden erläutert. Die jeweiligen Ziele und Ergebnisse dieser Experimente werden in diesem Kapitel vorgestellt.

Die einzelnen Experimente lassen sich in 3 Phasen einteilen, die sich sowohl in der Anzahl der genutzten Finger als auch der zu lernenden Tasten unterscheiden. Das Ziel der verschiedenen Versuche ist es, die Einstellungen des Netzes anzupassen und zu verbessern, aber auch die Einschränkungen und Grenzen des Datenhandschuhs und der verwendeten ML-Methoden zu ermitteln.

5.1 Vorverarbeitung

Die Vorverarbeitung der Daten erwies sich als sehr sinnvoll und gut geeignet, um den Lernprozess zu vereinfachen. In Abbildung 13 sind die von uns erhofften Muster für verschiedene Tastendrücke in den vorverarbeiteten Bewegungsdaten deutlich zu erkennen.

In den einzelnen Graphen sind die durchschnittlichen Werte sowohl der Quaternionen als auch der Accelerometerdaten einiger Tasten[1] aus Phase 2 zu sehen. Die X-Achse beschreibt die vergangene Zeit vor und nach einem Tastendruck. Dieser ist mit einer vertikalen grauen Linie gekennzeichnet. Auf der Y-Achse stehen die gesamten Werte der Quarternions (von oben nach unten: w, x, y, z) und des Accelerometers (ax, ay, az) nach Durchlaufen der Vorverarbeitung. Zusätzlich zum Vorverarbeiten wurden die durchschnittlichen Werte aller aufgenommenen Tastendrücke der jeweiligen Taste errechnet und im Graphen als jeweils eine Linie abgebildet.

Aus den Graphen wird ersichtlich, dass die verschiedenen Tasten gut zu unterscheiden sind. Bei einigen Tasten sind diese Unterschiede deutlich zu erkennen, wie zum Beispiel bei den Tasten T und J . Diese Tasten sind auf der Tastatur relativ weit voneinander entfernt und benötigen für das Tippen eindeutig unterscheidbare Fingerbewegungen.

[1] In Phase 2 werden insgesamt 10 Tasten für das Lernen verwendet. Die Abbildung zeigt jedoch lediglich sechs dieser Tasten, um die Übersichtlichkeit zu gewährleisten.

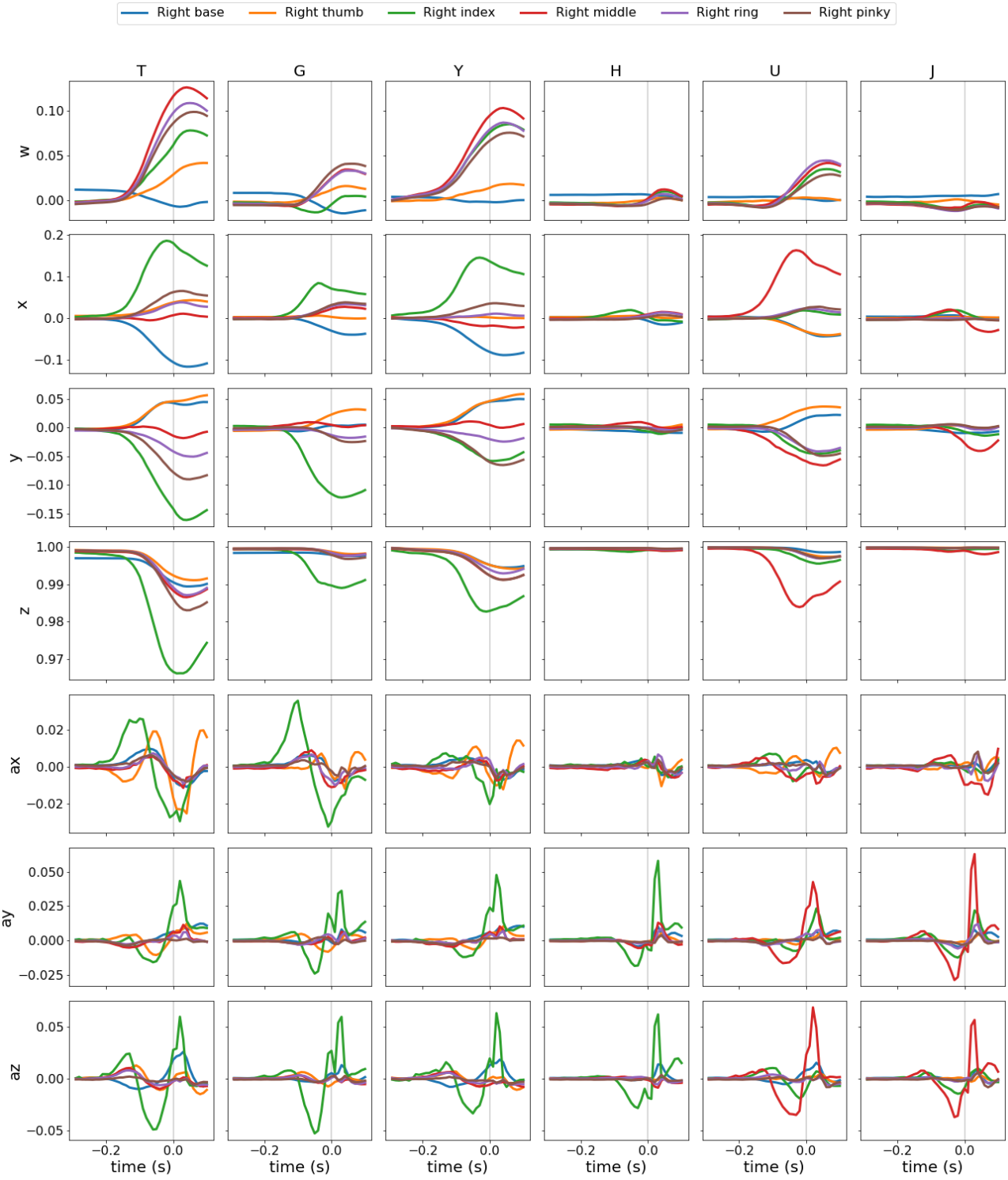


Abbildung 13: Durchschnittswerte der Orientierungs- und Beschleunigungsdaten über je ca. 150 Tastendrücke der in Phase 2 aufgezeichneten Tasten. Die X-Achse zeigt die vergangene Zeit vor und nach dem Tastenanschlag, die Y-Achse zeigt die jeweiligen Werte der Quaternion und des Accelerometers.

Doch auch bei den auf den ersten Blick sehr ähnlich aussehenden Mustern einiger Tasten, können bei genauer Betrachtung Unterschiede festgestellt werden.

Als Beispiel sollen hier die Tasten H und J dienen. Da sich beide Tasten in der mittleren Tastenreihe befinden, sind die Fingerbewegungen zum Drücken der Tasten sehr gering. Dennoch werden diese Bewegungen mit unerschiedlichen Finger getätigt. Die Unterschiede in den Quaterniondaten sind eher gering. Auf der y- und auch der z-Achse der Accelerationsdaten schlagen jedoch bei H die Werte des Zeigefinger und bei J die Werte des Mittelfingers deutlich aus. Da dies der Fall ist, scheint die zusätzliche Nutzung der Accelerometerdaten sinnvoll zu sein.

5.2 Metriken zur Evaluation des Modells

Geeignete Metriken zu finden, um das gelernte Netz bewerten zu können, ist für überwachtes Lernen unabdingbar und sowohl für den Lernprozess, als auch die spätere Analyse sehr wichtig.

Zunächst werde ich auf die Kostenfunktion eingehen. Diese wird beim Lernen verwendet, um zu ermitteln wie gut die gemachten Prädiktionen sind. Hierfür werden die vom Netz ermittelten Werte mit den tatsächlichen Zielwerten verglichen, um dann die Kanten des neuronalen Netzes entsprechend der gemachten Fehler (Kosten) anzupassen.

Wie in Abschnitt 4.6 „Konfiguration" erwähnt, haben wir für das CNN den Mean Squared Error als Kostenfunktion verwendet. Dieser errechnet sich aus den Vorhersagen *pred* und den Zielwerten *act*:

$$\mathrm{MSE}(pred, act) := \frac{1}{n} \cdot \sum_{i=0}^{n} (pred_i - act_i)^2$$

Es wird hierbei für jedes Sample die Differenz zwischen den jeweils n ermittelten und tatsächlichen Zielwerten quadriert. Diese Differenzen werden aufsummiert und durch die Anzahl der Werte geteilt.

Die erhaltenden Werte werden als Kosten angesehen. Im Backpropagation-Schritt werden die Kanten angepasst, um die Kosten zu reduzieren. Diese Anpassung fällt je nach eingestellter Lernrate mehr oder weniger groß aus und ist abhängig von der jeweiligen Kante.

Zur Evaluation eines Klassifizerungs-Algorithmus wird oft die sogenannte Confusion Matrix (siehe Tabelle 1a) zur Hilfe genommen. In dieser Matrix wird festgehalten, wie oft die verschiedenen Klassen vorkamen und als welche Klasse sie erkannt wurden. Links stehen die tatsächlichen, oben die vom Klassifikator ermittelten Klassen.

Meist wird die Confusion Matrix für lediglich zwei Klassen verwendet. Hierbei kann in 4

Kategorien eingeteilt werden, jeweils zwei für richtige und zwei für falsche Klassifikationen.

		Predicted	
		Class 1 positive	Class 2 negative
Actual	Class 1 positive	TP	FN
	Class 2 negative	FP	TN

(a) Confusion Matrix; (TP) korrekt als positiv, (FN) falsch als negativ, (FP) falsch als positiv und (TN) korrekt als negativ erkannte Klassen.

Metrik	**Formel**	**Fragestellung**
Accuracy	$\frac{\mathrm{TP+TN}}{n}$	Anteil korrekter Vorhersagen
Recall	$\frac{\mathrm{TP}}{TP+FN}$	Anteil korrekter Vorhersagen in der positiven Klasse
Precision	$\frac{\mathrm{TP}}{TP+FP}$	Anteil der als positiv vorhergesehenen Samples, der korrekt war
F-Score	$\frac{2\ \mathrm{PR}}{P+R}$	harmonisches Mittel aus Recall (R) und Precision (P), Sasaki u. a. (2007)

(b) Übliche Metriken, welche aus einer Confusion Matrix ablesbar sind; n ist hier die Größe der Testmenge

Tabelle 1: Confusion Matrix mit zwei Klassen und daraus ablesbare Metriken

Wir verwenden die aus der Confusion-Matrix ableitbaren Werte der Accuracy, der Precision und des Recalls, sowie den F-Score (auch F1-Score genannt), um den Lernfortschritt und -erfolg des Experiments zu überwachen. In Tabelle 1b sind diese Begriffe mit ihrer Formel und einer kurzen Beschreibung aufgelistet.

Bei der Accuracy handelt es sich um die Genauigkeit der Prädiktionen. Hierbei wird der Anteil der korrekt klassifizierten Samples berechnet. Die alleinige Verwendung der Accuracy ist in den meisten Fällen nicht zu empfehlen, da sie oft ein falschen Eindruck des Lernerfolgs vermittelt. Wie bereits erwähnt ist die Accuracy bei unbalancierten Datensätzen oft sehr schnell am optimalen Wert, beispielsweise wenn der Klassifikator jedes Sample der negativen Klasse zuordnet. Da diese Klasse einen sehr großen Teil der Daten ausmacht, sind fast alle Samples korrekt klassifiziert, das Lernen war jedoch trotzdem

kein Erfolg.

Der Recall beschreibt den Anteil der korrekt erkannten Samples der positiven Klasse. Dieser Wert ist in Verbindung mit der Precision besonders aussagekräftig, da diese sicherstellt, dass nicht übermäßig oft Samples der positive Klasse zugeordnet werden. Sie berechnet den Anteil der als positiv klassifizierten Samples, welche auch tatsächlich zur positiven Klasse gehören.

Der F-Score verbindet Recall und Precision zu einem Wert und ist dadurch gut geeignet, um Rückschlüsse auf den Lernerfolg ziehen zu können, auch wenn mit unbalancierten Daten gelernt wurde.

In mehrklassigen Problemen lassen sich Recall, Precision und F-Score nur in Bezug auf jeweils eine Klasse angeben. Daher errechnen wir für jede unserer zu erkennenden Tasten getrennt diese Metriken. Wir überwachen diese über den Lernprozess und zeichnen daraus Graphen, um zu erkennen, ob sich der Klassifikator dem Optimum von jeweils 100% annähert.

5.3 Phase 1: Erkennen einer Taste

In der ersten Phase erstellten wir einen Datensatz, in dem die Testperson mit dem Zeigefinger der rechten Hand, an welcher der Datenhandschuh befestigt war, in zufälliger Reihenfolge die zwei Tasten N und K drückte (Abbildung 14). Hierbei ist die eher ungewöhnliche Ruheposition der Hand der Testperson zu erwähnen. Der Zeigefinger der Testperson liegt auf der Taste H und nicht, wie üblich, auf der Taste J. Zudem wurde ein US-Tastaturlayout verwendet.

Die Fingerbewegungen wurden sehr langsam ausgeführt, mit kleinen Pausen zwischen den einzelnen Bewegungen. Nach jeder Bewegung wurden die Finger wieder in die Ausgangsposition bewegt.

Abbildung 14: Verwendete Tasten und Finger in Phase 1.

In dieser Phase wollten wir ermitteln, ob die einzelnen Tastenanschläge in den Daten überhaupt erkennbar sind und ob ein neuronales Netz in der Lage ist, das Rauschen in den Daten zu ignorieren und die Fingerbewegungen für das Drücken einer Taste zu registrieren. Wir wollten herauszufinden, ob selbst direkt benachbarte Tasten unterscheidbar sind.

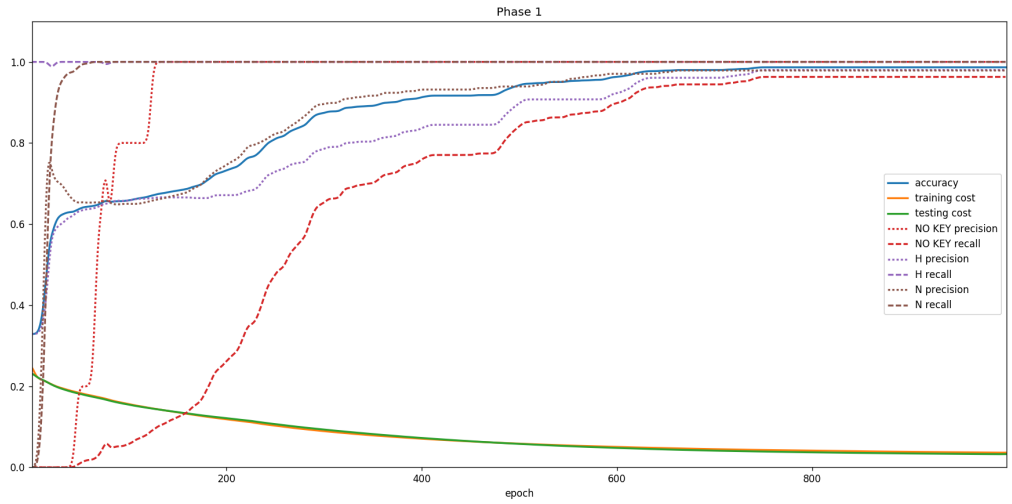


Abbildung 15: Testergebnisse der Phase 1 in den ersten 1000 Epochen. Die Accuracy liegt bei knapp 97%, die Klasse ∅ wird langsamer gelernt als die anderen.

In Abbildung 15 ist die Entwicklungskurve der Accuracy und der Trainings- und Testkosten beim Lernen des Netzes mit dem oben beschriebenem Datensatz zu sehen. Nach 1000 Epochen ist bereits eine Accuracy von ungefähr 97% erreicht, die Kosten wurden sowohl beim Trainings- als auch beim Testdatensatz stark reduziert.

Außerdem sind Precision und Recall aller Klassen eingezeichnet. Hierbei ist zu erkennen, dass der Recall der Klasse ∅ relativ spät optimiert wird, stattdessen werden die dieser Klasse zugehörigen Samples zuvor den anderen Klassen zugeordnet.

Wir waren mit dem so trainierten Modell in der Lage, Tippbewegungen auf einem Tisch durchzuführen, welche dem Tippen auf diesen Tasten ähnelten. Das neuronale Netz konnte diese Tippbewegungen zuverlässig erkennen. Je nach Beugungsgrad des Zeigefingers wurde [N] oder [H] erkannt.

5.4 Phase 2: Differenzieren verschiedener Tasten

Für die zweite Phase wurden drei Finger und ingesamt 10 verschiedene Tasten verwendet.

In dieser Phase legen wir unter anderem Wert darauf, die Vorverarbeitung der Daten zu verbessern und dadurch das Lernen zu erleichtern.

Um zum Beispiel die Taste [T] zu drücken, muss im Gegensatz zur vorherigen Phase die Handposition verändert werden. Die Daten der Hand-IMU sollten helfen, diese Handbewegungen zu erkennen und genauere Rückschlüsse auf die getippte Taste ziehen zu

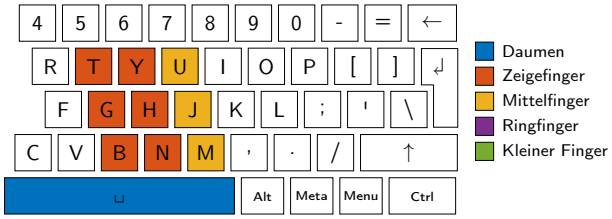


Abbildung 16: Verwendete Tasten und Finger in Phase 2. Mit dem Daumen, dem Zeige- sowie dem Ringfinger werden insgesamt 10 verschiedene Tasten gedrückt.

können.

Ein besonderes Augenmerk liegt auf den Tasten der *mittleren Tastenreihe*. Als mittlere Tastenreihe bezeichne ich die Tasten, welche von den Fingern bedeckt werden, wenn die Hand in der Ruheposition auf der Tastatur liegt. Zum Tippen dieser Tasten ist lediglich eine minimale Bewegung der Finger erforderlich. Dies macht die Erkennung dieser Tasten anhand der Bewegungsdaten schwieriger als die der anderen Tasten.

Die Testperson hat den Zeigefinger in der Ruheposition auf der Taste H liegen. Dadurch sind die Tasten H, J, K, L und ␣ in der mittleren Tastenreihe enthalten.

Wie in Abbildung 16 zu sehen, sind in dieser Phase drei Tasten der mittleren Tastenreihe zu lernen. Dadurch können wir bestehende Schwierigkeiten durch den geringen Bewegungsaufwand erkennen und Lösungswege evaluieren.

In dieser Phase haben wir mehrere Experimente durchgeführt. Im ersten Experiment erzielten wir eine Genauigkeit von knapp 76%, was eine deutliche Verschlechterung im Vergleich zu der ersten Phase darstellte, obwohl mit 160 000 Epochen deutlich mehr Epochen gelernt wurden. Die Confusion Matrix in Abbildung 17 half uns die für die schlechte Accuracy ausschlaggebenden Tasten zu identifizieren.

Die Tasten Y, U, G, B, N und M wurden mit nur wenigen falschen Prädiktionen sehr gut erkannt. Die Tasten H, ␣ und auch das Fehlen eines Tastendrucks, also ∅, wurden fast nie richtig erkannt. Stattdessen wurde die meisten Samples aus diesen Klassen dem Druck der Taste J zugeordnet. Wir kamen zu dem Schluss, dass die Tastatur und die dadurch resultierenden Fingerbewegungen Grund hierfür sein können.

Bei den erwähnten Tasten handelt es sich ausschließlich um die Tasten, die sich in der mittleren Tastenreihe befinden. Um diese zu drücken, müssen sich die Finger also nicht von der Ruheposition wegbewegen, sondern lediglich die unter dem Finger befindliche Taste hinunterdrücken. Ähnliche Beobachtungen konnten, wie bereits erwähnt, in der Confusion Matrix von Wheeler und Jorgensen (2003) gemacht werden.

Da zum Aufzeichnen eine sehr flache Tastatur verwendet wurde, die bereits mit sehr geringem Druck auslöst, war die Fingerbewegung für das Tippen einer dieser Tasten sehr

		Predicted									
		∅	Y	U	G	H	J	B	N	M	␣
Actual	∅	0	13	5	9	5	121	2	6	3	0
	Y	0	175	0	0	0	0	0	0	0	0
	U	0	1	167	0	0	0	0	0	0	0
	G	0	0	0	181	0	0	0	0	0	0
	H	0	1	5	0	1	169	0	0	0	0
	J	0	0	1	0	1	215	0	3	0	0
	B	0	0	0	0	0	0	167	0	0	0
	N	0	0	0	0	0	0	0	180	0	0
	M	0	0	0	0	0	0	0	0	190	0
	␣	0	0	0	0	4	178	1	0	0	0

Abbildung 17: Confusion Matrix des ersten Experiments der Phase 2. [H], [␣] und ∅ werden nicht gut erkannt, in den meisten Fällen wird stattdessen [J] erkannt[2].

gering und das CNN nicht in der Lage diese zu differenzieren.

Eine Wiederholung des Experiments erzielte jedoch ein besseres Ergebnis als der erste Durchlauf. Der erste Lernvorgang wurde demnach zu früh abgebrochen, obwohl er deutlich mehr Epochen benötigte, als der zweite. Dies liegt vermutlich daran, dass das Netz zu Beginn zufällig initialisiert wird und dadurch mit unterschiedlichen Voraussetzungen startet.

Die Confusion Matrix des zweiten Durchlaufs ist in Abbildung 18 abgebildet. Die gewünschte Diagonale durch die Tabellenwerte ist hier deutlicher zu erkennen, als in der Confusion Matrix des ersten Experiments und ein Großteil der Samples der nicht auf der mittleren Tastenreihe befindlichen Tasten, werden mit wenigen Ausnahmen korrekt erkannt. Die Samples der Klasse ∅ werden allen anderen Klassen zugeordnet. Bei den Samples anderer Klassen sind höchstens fünf verschiedene Klassen in den Prädiktionen enthalten. Das Erkennen der ∅-Klasse erfordert also eine gesonderte Optimierung.

In Abbildung 19 ist der Ergebnisgraph des zweiten Experiments abgebildet. Die grauen Linien stehen für die Tasten, welche sich schon im ersten Experiment als leicht erlernbar erwiesen. Die farbigen Linien gehören zu den Tasten der mittleren Tastenreihe. Ein direkter Vergleich der beiden Tastengruppen zeigt, dass die Tasten der mittleren Tastenreihe erst deutlich später gelernt werden als die anderen.

Am Beispiel der Tasten [H] und [N] wird das deutlich. Die Taste [N] wird von Beginn an erkannt und erzielt schon bei ungefähr 20 000 Epochen den F-Score von 95%. Die Taste [H] wird bis ungefähr zur 50 000 Epoche ignoriert. Erst dann wird sie langsam erkannt

[2] Diese Matrix enthält aufgrund eines Konfigurationsfehlers im ersten Experiment nicht alle Tasten wie geplant. Dies haben wir im nachfolgenden Experiment korrigiert.

		Predicted										
		∅	T	G	B	Y	H	N	U	J	M	␣
Actual	∅	39	3	5	4	4	31	6	6	55	5	22
	T	0	166	0	0	0	0	0	0	0	0	0
	G	0	1	180	0	0	0	0	0	0	0	0
	B	0	0	0	167	0	0	0	0	0	0	0
	Y	0	1	0	0	174	0	0	0	0	0	0
	H	2	0	0	0	1	147	0	4	22	0	0
	N	2	0	0	0	0	0	177	1	0	0	0
	U	0	0	0	0	1	0	0	167	0	0	0
	J	8	0	0	0	0	26	3	0	179	0	4
	M	0	0	0	0	0	0	0	0	0	190	0
	␣	4	0	0	0	0	0	0	0	0	0	179

Abbildung 18: Confusion Matrix des zweiten Experiments der Phase 2. H, J, ␣ und ∅ werden wesentlich besser als zuvor erkannt, jedoch immer noch deutlich schlechter als die anderen Klassen.

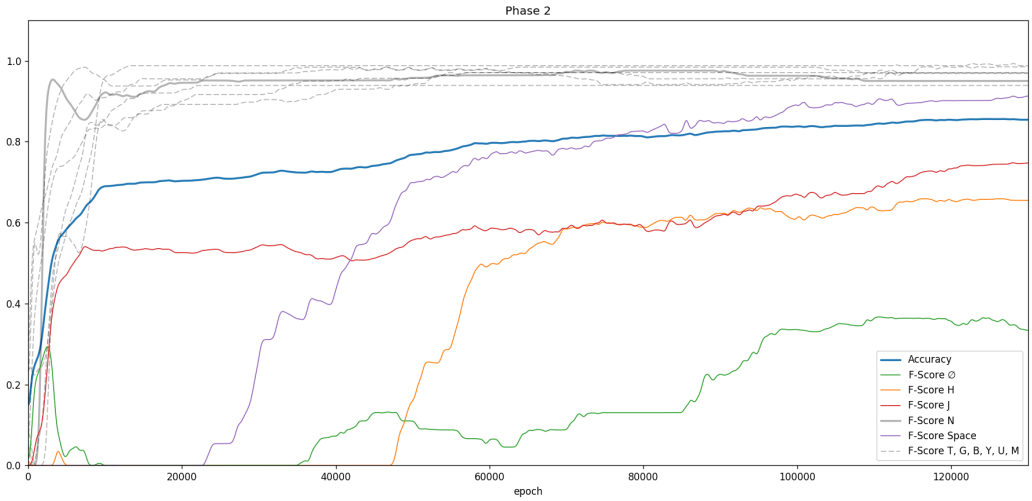


Abbildung 19: Performance-Metriken des zweiten Experiments der Phase 2 (130 000 Epochen). Die F-Scores der verwendeten Tasten sind in zwei Gruppen eingeteilt, Tasten der mittleren Tastenreihe sind farbig, die restlichen Tasten sind grau dargestellt. Hervorgehoben ist der F-Score der Taste N.

und erreicht schließlich einen F-Score von 65.9%, immer noch wesentlich schlechter als der von der Taste N . Insgesamt konnte eine Accuracy von 85% erreicht werden, eine deutliche Verbesserung zum vorherigen Experiment mit 76% Genauigkeit.

Da trotz des besseren zweiten Experiments die Tasten der mittleren Reihe deutlich schlechter abschneiden als die anderen, ist es sinnvoll, ein weiteres Experiment durchzuführen, bei der eine Tastatur verwendet wird, die ein deutlicheres Drücken erfordert.

Eine weiteres Experiment könnte die Daten des Accelerometers zum Lernen hinzunehmen. In Abbildung 13 ist zu sehen, dass diese Daten eine bessere Unterscheidung zwischen den Tasten der mittleren Tastenreihe ermöglichen.

Es sind auch noch weitere Verbesserungsmöglichkeiten aufgefallen, wie zum Beispiel die Verzögerung bei der Anwendung des gelernten Netzes. Diese resultiert vor allem aus der Art der Sampling-Methode. Da wir die Tastendrücke in die Mitte der für ein Sample verwendeten Zeitschritte legen, muss die Hälfte der Zeitschritte nach einem Tastendruck abgewartet werden, um ein Sample zu erhalten. Es wäre möglich, die Anzahl der Zeitschritte nach dem Tastendruck zu verringern. Dies würde zu einer Reduktion der entstehenden Verzögerung führen. Ob hierbei die Erkennung der Tastendrücke erschwert wird, muss evaluiert werden.

5.5 Phase 3: Flüssiges Schreiben

In der dritten Phase sollen alle Finger der rechten Hand dazu verwendet werden, die Tasten, welche für die rechte Hand vorgesehen sind, zu tippen.

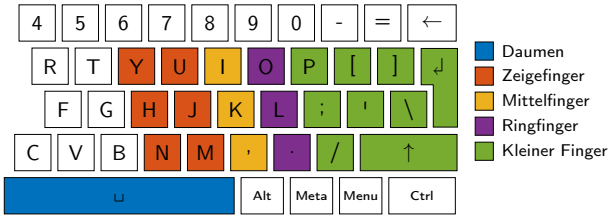


Abbildung 20: Geplante Verwendung von Tasten und Finger in Phase 3 nach korrektem 10-Finger-System. Es sollen alle Tasten der rechten Hand abgedeckt und ein flüssiges Schreiben erlernt werden.

Dies verlangt einen hohen Grad der Differenzierungskapazität zwischen den verschiedenen Bewegungen.

Diese Phase konnten wir jedoch bisher nicht beginnen, da die Genauigkeit der zweiten Phase noch nicht zufriedenstellend genug war, um mit einer so komplexen Aufgabe anzufangen. Die zu erwartenden Probleme in der dritten Phase werde ich jedoch an dieser Stelle bereits erläutern.

Bisher wurden die Daten unter „Laborbedingungen" aufgezeichnet. Jede Taste wurde einzeln gedrückt, das heißt nach einem Tastendruck wurde zuerst die Ruheposition eingenommen, bevor die nächste Taste gedrückt wurde. Dies bedeutet, dass sich die Bewegungen für eine Taste nicht stark unterscheiden, da diese immer von der selben Startposition ausgehen. Beim flüssigen Schreiben würden sich dagegen nicht nur die zeitlichen Abstände drastisch verringern, sondern auch die Bewegungsvielfalt für eine einzelne Taste stark zunehmen.

Zudem werden deutlich mehr Tasten zu lernen sein. Dies erfordert eine präzisere Erkennung der Fingerbewegungen und zusätzlich eine größere Vielfalt der von der Hand-IMU aufgezeichneten Positionsveränderungen.

Das flüssige Schreiben erfordert also eine deutlich höhere Komplexität des Modells. Daraus folgt, dass der Lernvorgang deutlich länger brauchen wird und bedeutend mehr Lerndaten nötig sind. Es könnte sinnvoll sein, zu versuchen, diese Komplexität zu reduzieren. Dies könnte unter anderem dadurch erreicht werden, dass zunächst erkannt wird, zu welchen Zeitpunkten eine Taste gedrückt wird und erst danach, um welche Taste es sich dabei handelte.

In dieser Phase wäre es zudem notwendig, einen zweiten Datenhandschuh zu bauen und zu nutzen, damit tatsächlich alle Tasten der Tastatur abgedeckt werden können.

6 Fazit

In den folgenden Absätzen wird das verwendete Verfahren für das Lernen von Bewegungsdaten zur Prädiktion von Tastendrücken zusammengefasst. Zudem wird ein Ausblick auf weitere Vorgehensmöglichkeiten gegeben. Hierbei werden einige Verbesserungsmaßnahmen erläutert, die innerhalb des Projektes bereits nützlich erschienen, jedoch bisher nicht umgesetzt werden konnten.

6.1 Zusammenfassung

An dieser Stelle möchte ich auf die zu Beginn der Arbeit in Abschnitt 1.3 festgehaltenen Zielsetzungen zurückkommen:

> Definition eines Ansatzes zum Rückschließen auf Tastatureingaben aus den aufgezeichneten Bewegungen unter der Verwendung von Verfahren des maschinellen Lernens.

Der mit 6 IMUs ausgestattete Datenhandschuh liefert Bewegungsdaten der Hand und der Finger mit einer Datenrate von ungefähr 100 Hz. Zur Verfügung stehen unter anderem die Accelerometerdaten, die Daten des Gyroskops und die bereits von den IMUs fusionierten Quaternionen. Für das Lernen verwenden wir die Quaternionen, welche wir zu der IMU am Handrücken beziehungsweise zum gleitenden Mittelwert relativieren. Da die IMUs zu unterschiedlichen Zeitpunkten neue Daten senden, interpolieren wir diese, um zu festen Zeitschritten die aktuellen Werte zu erhalten.

Mithilfe von maschinellem Lernen waren wir in der Lage, Bewegungen mit bestimmten Tastatureingaben zu verbinden. Das von uns beschriebene CNN erwies sich hierbei geeigneter als die Nutzung eines RNN. Die Vorverarbeitung der Daten ist eine große Hilfe, um das Lernen zu erleichtern. Dennoch ist das Rückschließen auf die Tastatureingabe bisher nur bedingt möglich, der Ansatz scheint jedoch weiter optimierbar zu sein. Weitere Experimente und Anpassungen stehen noch an, um die Bandbreite der erkennbaren Tastendrücke zu erhöhen.

> Bewertung der Qualität dieser Rückschlüsse und der Nutzbarkeit eines solchen Verfahrens als Alternative zur klassischen Tastatur.

Innerhalb verschiedener Experimente haben wir evaluiert, in wie weit das Tippen unter Verwendung des Datenhandschuhs möglich ist. Hierbei spielte vor allem die Genauigkeit der Erkennung der Tastendrücke eine Rolle.

In dieser Arbeit waren wir in der Lage 2 Tasten mit einer Genauigkeit von 97% zu unterscheiden. Das Tippen von 3 Fingern und 10 Tasten erreichte eine Genauigkeit von 85%.

Wir stellten fest, dass es sehr schwierig ist, die Tasten zu unterscheiden, auf denen die Finger ruhen. Die Verwendung einer Tastatur mit größerem Tastenanschlagsweg könnte den erhofften Vorteil erbringen. Außerdem könnte die Verwendung der Beschleunigungsdaten vom Accelerometer das Differenzieren dieser Tasten erleichtern.

Obwohl die Genauigkeit bisher nicht optimal ist und es zudem bisher nicht möglich ist, fließend zu schreiben, ist dieses Projekt in meinen Augen erfolgreich. Wir konnten zeigen, dass der Datenhandschuh brauchbare Daten liefert, welche das verwendete CNN in die Lage versetzt, die Fingerbewegungen zu lernen. Die Quaternionen reichen aus, um einige Tasten zu differenzieren. Mit entsprechendem Aufwand halte ich es mithilfe unseres Ansatzes für möglich, eine vergleichbare Eingabemethode zu Tastaturen zu schaffen.

6.2 Ausblick

Da der Datenhandschuh nur ein Prototyp ist, sind Verbesserungen nicht nur möglich, sondern auch nötig.

In unserem Projekt nutzten wir zwei verschiedene Arten von neuronalen Netzen. Es wäre jedoch auch denkbar, einen anderen ML-Algorithmus zu verwenden. Ein Beispiel wäre KNN (*k-nearest-neighbours*). In Abbildung 13 wird deutlich, dass die Graphen für die verschiedenen Tasten stets unterschiedlich sind. Es wäre also möglich, Daten demjenigen Graphen zuzuordnen, welcher am ähnlichsten ist. Denkbar wäre, hierfür ein zweistufiges Modell zu nutzen. Im ersten Schritt könnten Tastendrücke und die dabei beteiligten Finger erkannt werden. Im folgenden Schritt würde dann ermittelt werden, um welche Taste es sich handelt. Für den zweiten Schritt könnte man alternativ zum KNN auch ein Hidden Markov Model verwenden (wie in Wheeler und Jorgensen, 2003).

Neben dem Austauschen des kompletten ML-Algorithmus wäre es auch möglich, Verbesserungen an der bestehenden Konfiguration vorzunehmen. Eine naheliegende Verbesserung wäre, die Verzögerung durch die Sampling-Methode zu verringern. Die Tastendrücke liegen zur Zeit in der Mitte der Samples, dadurch ist es nötig, relativ viele Zeitschritte nach einem Tastendruck abzuwarten, um ein vollständiges Sample zu erhalten.

Zudem könnten auch mehr Daten von dem Datenhandschuh genutzt werden. Bisher haben wir lediglich die Quaternionen verwendet, mit dem Hinzunehmen der Accelerometer- und Gyroskopdaten ist es eventuell möglich, eine genauere Prädiktion der Tasten zu erzielen. Hierfür sollte die Frequenz der festen Zeitschritte von 25 Hz auf 100 Hz erhöht

werden, entsprechend der Frequenz in der die IMUs die Daten senden. Dies ist nötig, da sich die Accelerometerdaten sehr schnell verändern.

Ein Problem mit den Sensordaten ist bisher, dass die IMU im Fusionsmodus die Messung einer Rotationsgeschwindigkeit über $\pm 500\,^{\circ}/\mathrm{s}$ nicht ermöglicht. Um eine schnellere Rotationsgeschwindigkeiten messen zu können, wäre es nötig, das Fusionieren der Quaternionen eigenständig vorzunehmen. Dies erwies sich als umständlich, da dafür eine manuelle Kalibrierung notwendig wäre. Man könnte auch versuchen, die durch die eingeschränkte Messung enstehende Abweichung der Ausrichtung in der Vorverarbeitung zu erkennen und zu korrigieren.

Eventuell wird es beim Lernen von mehr Tasten als bisher auch nötig sein, die Beschleunigung der Handbasis-IMU zu integrieren, um die Position der Hände über der Tastatur in den Algorithmus einfließen zu lassen. Um nicht zu viele Daten verarbeiten zu müssen, wird es sinnvoll sein, ein kinematisches Modell der Hand zu nutzen, um die Dimensionen der Daten reduzieren zu können.

Ein weiterer Verbesserungs- beziehungsweise Implementationsbedarf besteht beim Online-Learning, welches wir bisher leider außer Acht lassen mussten. Online-Learning bedeutet, dass der Algorithmus während der Anwendung weiterhin lernt, um mit Veränderungen der Muster umgehen zu können. In unserem Fall bedeutet dies zum Beispiel das langsame Wegbewegen der Hände von der Tastatur. Für das Online-Learning müssen geeignete Methoden gefunden werden, das überwachte Lernen fortzusetzen. Es ist dann jedoch nötig, eine Methode zu finden, welche die Daten labelt. Das Labeln der Daten war dank der Tastatureingabe bisher kein großes Problem, da durch die Tastatur ein korrekter Zielwert einfach erhalten werden konnte. Ohne diese Rückmeldung muss das Netz gut genug sein, um leichte Abweichungen in der Bewegung trotzdem der richtigen Taste zuordnen zu können. Diese Abweichungen müssen dann nach und nach gelernt und gefestigt werden um von dort aus weitere Veränderungen der Bewegungen lernen zu können.

Nicht nur für das Online-Learning wäre es sinnvoll eine neue Label-Methode zu nutzen. Um den Datenhandschuh auch für Gestenerkennung nutzen zu können, wäre es nötig, die aufgenommenen Daten den unterschiedlichen Gesten zuordnen zu können.

Bisher haben wir mit dem Handschuh lediglich Tastendrücke von der mittleren Tastenreihe aus durchgeführt. Für ein flüssiges Schreiben ist es unabdingbar, eine Taste unmittelbar nach dem vorangegangen Tastendruck zu tippen. Da es sehr viele Buchstabenkombinationen gibt, entstehen viele verschiedene Bewegungsabläufe für dieselbe Taste, wodurch das Lernen komplexer wird.

Für das Schreiben langer Texte mithilfe des Datenhandschuhs wäre es von Vorteil, wenn es eine Wortprädiktion und -korrektur gäbe. Dies ist bereits von Smartphone-Tastaturen bekannt. Eine solche Funktion würde das Tippen stark vereinfachen, da unsauber ausgeführte Fingerbewegungen, aber auch Ungenauigkeiten bei der Tastenerkennung des Klassifikators, durch eine Wortvorhersage automatisch verbessert werden könnten. Diese Funktion sollte jedoch ein- und ausschaltbar sein, um in Situationen, in welchen eine

solche Vorhersage störend sein könnte, ohne Verbesserungsvorschläge tippen zu können. Eine solche Situation ist zum Beispiel das Programmieren, da hierbei viele Sonderzeichen und Wörter ohne die grammatikalischen Regeln einer natürlichen Sprache hintereinandergereiht werden.

Auch beim Datenhandschuh lassen sich einige Verbesserungen vornehmen. Beispielsweise ist die Verkabelung durch die freien Lötstellen recht anfällig, und bei den erforderlichen Bewegungen der Hand gehen diese Verbindungen teilweise kaputt. Außerdem ist die Mehrteiligkeit des Handschuhs recht unpraktisch, das Anziehen dauert lange und die einzelnen Komponenten sind kaum für unterschiedliche Handtypen geeignet, da zum Beispiel die IMUs auf Ringen aus elastischem Band befestigt sind, welche bei schmalen Fingern leicht verrutschen. Außerdem könnte die Datenqualität noch deutlich verbessert werden. Auf diese und weitere mögliche Verbesserungen des Datenhandschuhs wird in der Arbeit von Paul Bienkowski (2017) detaillierter eingegangen.

Mit den aufgelisteten Verbesserungen könnte der Datenhandschuh eine gute alternative Eingabemethode zur Tastatur sein. Weitere Anwendungsgebiete wären ebenfalls denkbar, wie zum Beispiel das Trainieren einer Roboterhand. Ein Proband könnte den Handschuh anziehen und bestimmte Handbewegungen, wie zum Beispiel das Greifen nach einem Gegenstand, aufzeichnen. Ein ML-Algorithmus könnte diese Bewegungen lernen und für die Roboterhand adaptieren. Die vom Menschen schon im Kleinkindalter gelernten Bewegungsabläufe sind sehr schwer in Formeln zu fassen und durch die Vorführung mithilfe des Datenhandschuhs könnte das Muskelgedächtnis des Menschen die Aufgabe des Lehrens übernehmen.

Anhang

1 Quellenverzeichnis

Abdel-Hamid, Ossama u. a. (2014). „Convolutional Neural Networks for Speech Recognition“. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.10, S. 1533–1545.

Al-Rfou, Rami u. a. (2016). „Theano: A Python framework for fast computation of mathematical expressions“. In: *CoRR* abs/1605.02688.

Bienkowski, Paul (2017). *Prototyp für eine virtuelle Tastatur basierend auf IMUs und maschinellem Lernen – Systementwurf.*

Davidian, D. (1995). *Feed-forward neural network.* US Patent 5,438,646.

Duchi, John, Elad Hazan und Yoram Singer (2011). „Adaptive subgradient methods for online learning and stochastic optimization“. In: *Journal of Machine Learning Research* 12.Jul, S. 2121–2159.

Elman, Jeffrey L (1991). „Distributed representations, simple recurrent networks, and grammatical structure“. In: *Machine learning* 7.2-3, S. 195–225.

Estabrooks, Andrew, Taeho Jo und Nathalie Japkowicz (2010). „A Multiple Resampling Method for Learning from Imbalanced Data Sets.“ In: *Computational Intelligence* 20.1, S. 18–36.

Gerr, Fred, Michele Marcus und Carolyn Monteilh (2004). „Epidemiology of musculoskeletal disorders among computer users: lesson learned from the role of posture and keyboard use“. In: *Journal of Electromyography and Kinesiology* 14.1, S. 25–31.

Ghahramani, Zoubin (2001). „An introduction to hidden Markov models and Bayesian networks“. In: *International journal of pattern recognition and artificial intelligence* 15.01, S. 9–42.

Hawkins, Douglas M (2004). „The problem of overfitting“. In: *Journal of chemical information and computer sciences* 44.1, S. 1–12.

Hinton, Geoffrey E. u. a. (2012). „Improving neural networks by preventing co-adaptation of feature detectors“. In: *CoRR* abs/1207.0580.

Hochreiter, Sepp (2004). „The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions." In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.2, S. 107–116.

Hunter, J. D. (2007). „Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3, S. 90–95.

Kim, Yoon (2014). „Convolutional Neural Networks for Sentence Classification." In: *CoRR* abs/1408.5882.

Kumar, Piyush, Jyoti Verma und Shitala Prasad (2012). „Hand data glove: a wearable real-time device for human-computer interaction". In: *International Journal of Advanced Science and Technology* 43.

Le Cun, Yann u. a. (1990). „Handwritten zip code recognition with multilayer networks". In: *Pattern Recognition, 1990. Proceedings., 10th International Conference on*. Bd. 2. IEEE, S. 35–40.

Lecun, Y. u. a. (1990). „Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems 2*. Hrsg. von D. S. Touretzky. Morgan Kaufmann, S. 396–404.

Lobo, Jorge und Pedro Trindade (2011). „Distributed accelerometers for gesture recognition and visualization". In: *Doctoral Conference on Computing, Electrical and Industrial Systems*. Springer, S. 215–223.

– (2013). „Interactive demonstration of InerTouchHand – iTH a glove device with distributed inertial sensors and vibro-tactile feedback". In: *2013 2nd Experiment@ International Conference (exp.at'13)*, S. 178–179.

Poole, David und Alan K. Mackworth (2010). *Artificial Intelligence - Foundations of Computational Agents*. Cambridge University Press, S. I–XVII, 1–662.

Rayat, Pritpal u. a. (2016). *Breast Screening Programme England Statistics for 2014/-15*.

Roggen, D. u. a. (2010). „Collecting complex activity datasets in highly rich networked sensor environments". In: *2010 Seventh International Conference on Networked Sensing Systems (INSS)*, S. 233–240.

Sasaki, Yutaka u. a. (2007). „The truth of the F-measure". In: *Teach Tutor mater* 1.5.

Shoemake, Ken (1985). „Animating Rotation with Quaternion Curves". In: *SIGGRAPH Computer Graphics* 19.3, S. 245–254.

Walt, Stéfan van der, S. Chris Colbert und Gaël Varoquaux (2011). „The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science & Engineering* 13.2, S. 22–30.

Wheeler, K. R. und C. C. Jorgensen (2003). „Gestures as input: neuroelectric joysticks and keyboards". In: *IEEE Pervasive Computing* 2.2, S. 56–61.

Yao, Rui u. a. (2017). „Efficient Dense Labeling of Human Activity Sequences from Wearables using Fully Convolutional Networks". In: *CoRR* abs/1702.06212.

Zeiler, Matthew D und Rob Fergus (2014). „Visualizing and understanding convolutional networks“. In: *European conference on computer vision*. Springer, S. 818–833.

2 Online-Quellen

[1] Brownlee, Jason (2016). *Supervised and Unsupervised Machine Learning Algorithms*. URL: http://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (besucht am 05. 05. 2017).

[2] Reid, Stuart (2014). *10 misconceptions about Neural Networks*. URL: http://www.turingfinance.com/misconceptions-about-neural-networks/#blackbox (besucht am 05. 05. 2017).

[3] Apotact Labs (2017). *Gest*. URL: https://gest.co/ (besucht am 30. 04. 2017).

[4] Open Source Robotics Foundation, Inc. (2017). *ROS.org*. URL: http://www.ros.org/ (besucht am 30. 04. 2017).

[5] Battenberg, Eric u. a. (2015). *Lasagne*. URL: http://lasagne.readthedocs.io/en/latest/ (besucht am 12. 06. 2017).

3 Abbildungsverzeichnis

4 Tabellenverzeichnis

5 Datenträger-Verzeichnis

Dieser Arbeit ist ein digitaler Datenträger beigelegt. Darauf finden Sie die Arbeit in digitalem Format, sowie begleitende Materialen:

- diese Arbeit, sowie die Arbeit von Paul Bienkowski
- den Quelltext der Handschuh-Firmware
- den Quelltext der Softwarekomponenten
- diverse Tools und Skripte zur Datenanalyse
- das Exposee, welches unserem Projekt zugrunde liegt
- die Folien unseres Kolloquiumvortrags (in englischer Sprache)
- Schaltplan und Leiterbahnlayout des „Shields"
- Fotos und Filme des fertigen Handschuhs

Nicht enthalten sind aus Speicherplatzgründen die Aufzeichnungen der Experimente. Diese können bei Interesse vom Projektverzeichnis auf Github heruntergeladen werden:

`https://github.com/imutype/bachelor`

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 12.06.2017 Carolin Konietzny

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 12.06.2017 Carolin Konietzny