Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit
(FS17 Studiengang Informatik)

Machine Learning for Speaker Clustering

Autoren	Patrick Gerber Sebastian Glinski-Haefeli
Hauptbetreuung	Thilo Stadelmann Oliver Dürr
Datum	07.06.2017

Erklärung betreffend das selbständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

……

Unterschriften:

……...

……

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Bachelorarbeiten zu Beginn der Dokumentation nach dem Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

Untersuchungen der letzten Jahre haben gezeigt, dass sich Neural Networks, im Speziellen die Unterart Convolutional Neural Networks (CNNs), für das Sprecherclustering eignen. So wurden die Fähigkeiten traditioneller Machine Learning Methoden wie dem Gaussian Mixture Model (GMM), kombiniert mit Mel-Frequency Cepstral Coefficients (MFCC), durch die der Neural Networks deutlich übertroffen. Eine weitere vor allem in der maschinengestützten Spracherkennung und Transkription eingesetzte Unterart der Neural Networks sind die Recurrent Neural Networks (RNNs). Allerdings finden diese im Problembereich des Sprecherclusterings bis anhin kaum Beachtung, obwohl sie die dafür geeignete Fähigkeit haben, Sequenzen aus Audiodaten mit zeitlichen Komponenten zu lernen.

Diese Bachelorarbeit befasst sich mit der Frage, ob sich RNNs und dessen Kombination mit anderen Unterarten von Neural Networks für das Sprecherclustering eignen. Dazu wurden zwei verschiedene Neural Network Architekturen untersucht. Eine der beiden Architekturen bestand aus einem Bidirectional Long Short-Term Memory (BLSTM) Neural Network, welches für die Sprecheridentifikation schon in vorhergehenden Arbeiten sehr gute Resultate zeigte. Die andere Architektur war eine Kombination aus CNN und Gated Recurrent Unit (GRU) Layern. Als Input Daten wurden den Neural Networks Audiosignale in Form von Spektrogrammen bereitgestellt. Die beiden Architekturen wurden mit dem TIMIT Datensatz trainiert und getestet.

Resultate der BLSTM Architektur ergaben ein fehlerfreies Clustering bei 40 Sprechern, was zum Zeitpunkt der Durchführung von keinem anderen Machine Learning Ansatz aus vergleichbaren Arbeiten erreicht wurde. Es konnte auch gezeigt werden, dass ein Clustering mit mehr als 40 Sprechern gute Resultate erzielt.

Abstract

In recently performed studies Neural Networks, in particular the subtype Convolutional Neural Networks (CNNs), have been shown to be suitable for speaker clustering. Therefore, Neural Networks have gained popularity over traditional machine learning models, such as Gaussian Mixture Model (GMM) combined with Mel-Frequency Cepstral Coefficients (MFCC). Another subtype of Neural Network, namely Recurrent Neural Networks (RNNs), is widely used for machine based speech recognition and transcription. However, despite their ability of learning sequences in audio data with temporal components, they have attracted little interest for speaker clustering problems.

This Bachelor's Thesis aims to determine, whether RNNs and the combination of RNNs with other subtypes of Neural Networks are suitable for the task of speaker clustering. For this purpose, two Neural Network architectures are examined. The first architecture consists of a Bidirectional Long Short-Term Memory (BLSTM) Neural Network, which has shown promising results in previous studies. The second architecture, in contrast, is a combination of CNN layers and Gated Recurrent Unit (GRU) layers. For both networks the input data are audio signals, which are provided as spectrograms. The pairwise KL-Divergence is used during the Neural Networks training process, in order to maximize the distance of two segments of the same speaker, as well as to maximize the segment distance to different speakers.

Results from the BLSTM architecture show a clustering without any error at the size of 40 speakers. To the best of the authors' knowledge, such results have not yet been achieved by any other Neural Network in comparable studies. The CNN GRU architecture showed similar results with a misclassification rate (MR) of 5%. Furthermore, clustering with more than 40 speakers also provided adequate results.

Vorwort

Patrick Gerber führte bereits im vergangenen Jahr die Projektarbeit "Speaker Identification mit Recurrent Neural Networks" unter der Betreuung von Thilo Stadelmann durch. Darauf folgte diese Bachelorarbeit, in der Sebastian Glinski-Haefeli, durch das Interesse im Bereich Machine Learning in Verbindung mit Audiodaten, dazu stiess. Betreut wurde diese Arbeit ebenfalls von Thilo Stadelmann und zusätzlich von Oliver Dürr.

Die Arbeit an einem Forschungsthema hat uns neue Einblicke in einen Bereich der Informatik ermöglicht, mit dem wir sonst nur selten in Berührung kommen.

Die Aufgabe stellte uns immer wieder vor neue Herausforderungen und war fordernd, dadurch waren die Erfolge die wir feiern konnte umso schöner. Die Zusammenarbeit im Team war aufgrund der knappen Terminpläne, welche wir als Teilzeitstudenten hatten, nicht immer einfach, allerdings waren die gegenseitigen Inputs und Diskussionen immer sehr fruchtbar.

Wir möchten uns bei Thilo Stadelmann und Oliver Dürr für die hervorragende Betreuung der Arbeit bedanken. Der Umgang war immer unkompliziert und ihr Feedback gab uns immer wieder wichtige und spannende Inputs. Ein grosser Dank geht auch an unsere Freundinnen bzw. Frau, die sehr Verständnisvoll waren und uns immer unterstützten.

Patrick Gerber & Sebastian Glinski-Haefeli

07.06.2017

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Sprechererkennung kann in vielen Bereichen verwendet werden. Sie kann für biometrische Sicherheitsverfahren, das automatische Protokollieren von Meetings oder auch in der Überwachung des öffentlichen Raums eingesetzt werden. Für das Problem des Sprecherclusterings, welches sich mit der Zuordnung von noch unbekannten Sprechern befasst, sind heutige Lösungen dem Menschen noch unterlegen. Ansätze mit Convolutional Neural Networks (CNN) haben aber bereits gezeigt, dass Neural Networks in diesem Bereich Verbesserungspotential bieten. Der Einsatz von Recurrent Neural Networks (RNN), welche im Gegensatz zu konventionellen Netzwerken fähig sind, zeitliche Zusammenhänge zu lernen, könnten ebenfalls zu einer Verbesserung des Sprecherclusterings beitragen.

1.2. Ausgangslage

Die Bachelorarbeit wurde unter der Betreuung von Dr. Thilo Stadelmann und Dr. Oliver Dürr am Institut für angewandte Informationstechnologie (InIT) der ZHAW School of Engineering durchgeführt. Die Arbeit baut auf der Projektarbeit des Co-Autors Gerber [1], zum Thema Sprecheridentifikation mit RNNs auf. RNNs wurden in den letzen Jahren mit grossem Erfolg im Bereich der Spracherkennung eingesetzt. So zeigten Sak et al. [2], dass sich Long Short-Term Memory (LSTM) Netzwerke mit einem zusätzlichen Projection Layer gut für Spracherkennung auf grossen Vokabularen eignen. Graves et al. [3] verwendeten Deep LSTM Netzwerke für Phonem Klassifizierung, welche State of the Art Ergebnisse erreichten. Für Sprechererkennung wurden LSTMs eingesetzt, um Sprecher in einem Video zu identifizieren [4]. Dieses Verfahren ist für reine audiobasierte Sprechererkennung jedoch nicht geeignet, da eine Kombination aus Video und Audiodaten verwendet wurde.

Lukic et al. [5] verwendeten CNNs um Sprecherclustering durchzuführen. Dieser Ansatz lieferte bessere Resultate als die traditionellen Verfahren, welche auf einem Gaussian Mixture Model (GMM) mit Mel Frequency Cepstral Coefficients (MFCC) basieren [6].

Der Einsatz von CNN-RNN Kombinationen auf dieser Problemstellung ist noch kaum erforscht. Allerdings wurde in der Studie von Sainath et al. [7] gezeigt, dass sich eine solche Architektur für die Spracherkennung eignet. Wobei man eine Architektur einsetzte, bei der mehrere CNN Layer mit einem RNN Layer und einem Dense Layer verbunden wurden. Ein ähnlicher Ansatz ohne Dense Layer wurde eingesetzt, um festzustellen welche Sprache gesprochen wird [8].

1.3. Zielsetzung

Das Ziel dieser Arbeit ist es, die Tauglichkeit von RNNs auf den Task des Sprecherclusterings zu überprüfen. Dazu werden diejenigen Ansätze aus der Projektarbeit von Gerber [1], die auf dem Task der Sprecheridentifizierung gute Resultate geliefert haben, aufgegriffen und weiterentwickelt. Zudem soll eine angepasste Version der CNN-RNN Architektur überprüft werden. Die Ergebnisse sollen anschliessend mit den aktuell besten Resultaten für Sprecherclustering aus der Bachelorarbeit von Lukic und Vogt [9], sowie der darauf basierenden Studie [10] verglichen werden.

1.4. Aufbau

- Kapitel 2 behandelt die zum Verständnis der Arbeit notwendigen Grundlagen. Es wird eine Übersicht über Neural Networks und eine Einführung in die Problemstellung des Sprecherclusterings gegeben.
- Kapitel 3 stellt das Vorgehen und die Methoden, sowie eingesetzten Netzwerkarchitekturen vor, welche verwendet wurden um die Experimente durchzuführen.
- Kapitel 4 und 5 zeigen die durchgeführten Experimente für die LSTM und CNN-RNN Netzwerke mitsamt deren Resultate.
- Kapitel 6 enthält das Fazit, welches aus den Resultaten der Experimente in Kapitel 4 und 5 gezogen wurde und gibt einen Ausblick auf weitere mögliche Arbeiten.

2. Theoretische Grundlagen

2.1. Sprecherclustering

Das Zuordnen von Audio-Sprachmaterial zu verschiedenen Sprechern (Sprecherclustering) zählt, neben Sprecherverifizierung (Speaker Verification) und Sprecheridentifizierung (Speaker Identification), zu dem übergeordneten Bereich der Sprechererkennung (Speaker Recognition). Verglichen mit dem Parallelbereich der Spracherkennung (Speech Recognition), in dem es darum geht zu erkennen *was* gesprochen wird, beschäftigt sich die Sprechererkennung damit *wer* etwas spricht [11][12].

Sprecherverifizierung ist die Aufgabe, die Identität eines Sprechers zu überprüfen und zu bestätigen, während es sich bei der Sprecheridentifizierung um die Zuordnung eines Sprechers zu einer dem System bekannten Identität handelt. Bei Letzterem unterscheidet man zusätzlich zwischen den beiden Varianten Closed-Set und Open-Set. Beim Closed-Set wird ein Sprecher in jedem Fall einer dem System bekannten Identität, nämlich der mit der grössten Übereinstimmung, zugeordnet. Beim Open-Set muss die Zuordnung eines Sprechers nicht zwingend erfolgen, sollte ein Mindestwert an Übereinstimmung mit einer der dem System bekannten Identitäten nicht erreicht werden [13].

Im Gegensatz zu Sprecherverifizierung und -identifizierung, die einen höheren Bekanntheitsgrad haben und in denen dadurch auch fortgeschrittenere Entwicklungen existieren [13], ist das Sprecherclustering durch Neural Networks ein noch sehr junges und weniger erforschtes Gebiet. Grund für diese Entwicklung ist mitunter auch, dass Sprecherclustering als die Komplexeste der drei Aufgaben im Bereich Sprechererkennung gilt [6]. Die Tatsache, dass bisherige Ansätze im Sprecherclustering eine geringere Zuverlässigkeit zeigen, wenn das Datenmaterial zu stark variiert [14][15][16], macht neue Entwicklungen und Fortschritte in diesem Bereich sehr attraktiv.

Damit das Audiomaterial eines Sprechers durch computergestützte Verfahren verarbeitet werden kann, ist es notwendig, dass dieses vorher in ein geeignetes Format umgewandelt wird. Hierbei eigenen sich vor allem Repräsentationen in Form von Grafiken, wie z.B. Spektrogrammen [17]. Verglichen mit alternativen Darstellungsarten, wie z.B. Waveforms, enthalten Spektrogramme nicht nur Informationen über die Signalstärke und den zeitlichen Verlauf, sondern auch über die Frequenz der unterschiedlichen Audiosignale.

Spektrogramme

Das digitale Generieren von Spektrogrammen aus Audiosignalen erfolgt unter Verwendung der Fast-Fourier-Transformation (FFT). Dabei wird die FFT pro Zeitschritt auf jedes Samplestück entlang der Zeitachse angewendet. Das Ergebnis der Berechnung eines Schrittes, sind die Amplitudenstärken der einzelnen Frequenzen innerhalb des betrachteten Frequensspektrums für den jeweiligen Zeitschritt [13].

Eine spezielle Form der Spektrogramme sind die sogenannten Mel-Spektrogramme, wie in Abbildung 2.1 dargestellt. Die Mel-Skala [18] stellt eine Messgrösse dar, welche die akustische Wahrnehmung von Frequenzen durch das menschliche Gehör beschreibt. Bei der Berechnung eines Mel-Spektrogramms wird die Frequenzachse auf die Mel-Skala mit Hilfe der Gleichung [19] abgebildet 2.1. f steht dabei für die umzuwandelnde Frequenz.

$$m(f) = 2595 \cdot \log_{10} \cdot \left(1 + \frac{f}{700}\right) \tag{2.1}$$

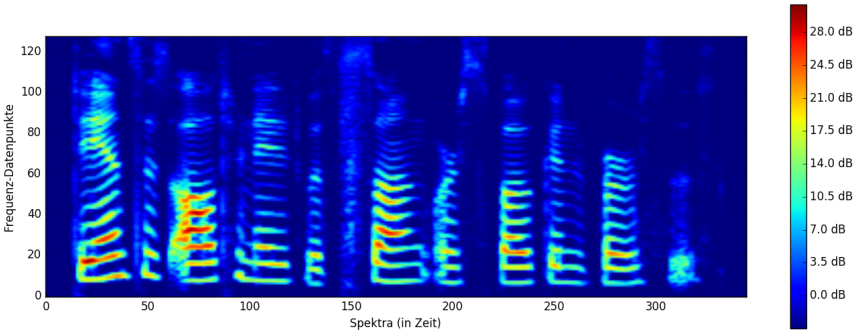


Abbildung 2.1.: Mel-Spektrogramm des Satzes "Dances alternated with sung or spoken verses."

2.2. Neural Networks

Die Grundstruktur eines Neural Networks (auch Artificial Neural Networks) besteht aus miteinander verbundenen Neuronen (auch Units genannt), die je nach Neuronentyp, einen Aufbau bestehend aus Eingängen x_z (Inputs), arithmetischen Operationen $\sum$, Aktivierungsfunktionen und Ausgängen (Outputs) y haben. Diese Neuronen agieren innerhalb unterschiedlicher Schichten, sogenannten Layern, die miteinander verbunden sind [20]. Der Begriff Deep Neural Network (DNN) beschreibt eine Architekturform die aus mehrerer solcher Layer aufgebaut ist. Dabei werden am Anfang ein Input Layer, danach ein oder mehrere Hidden Layers und am Ende ein Output Layer der Reihe nach miteinander verbunden.

Die Neuronenstruktur des Neural Networks wird oft auch als *Model* bezeichnet. Jede Verbindung zwischen den Neuronen ist mit einem Gewicht (*weight*) w_z versehen, das einem eingehenden Signal mehr oder weniger Bedeutung für das Neuron zuordnet. Anschliessend ist die Aktivierungsfunktion eines Neurons dafür verantwortlich, wie der Output des verarbeiteten Signals aussieht[20].

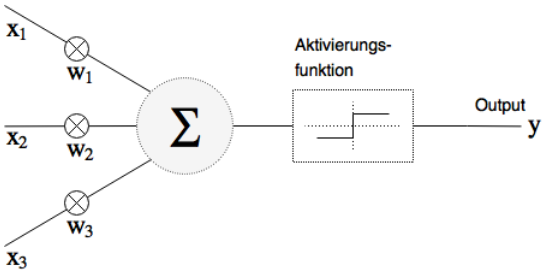


Abbildung 2.2.: Aufbau eines Neurons

Mit Hilfe unterschiedlicher Lernalgorithmen werden die Werte der Gewichte in den Neuronen trainiert. Ist eine ausreichend niedrige Fehlerrate des Netzwerks beim Lernen erreicht, wird das Training abgeschlossen und es können Daten auf dem Model evaluiert werden.

Das Training eines Neural Networks kann auf unterschiedliche Arten erfolgen. In dieser Arbeit wird dafür die Methode des *Supervised Learning* eingesetzt. Beim Supervised Learning lernt das Netzwerk mit einem Trainingsdatensatz, bei dem zu jedem Inputwert ein erwarteter Outputwert bekannt ist. Zu jedem vom Netzwerk berechneten Outputwert für einen zugehörigen Inputwert werden nach einer Trainingsiteration die Abweichungen zum erwarteten Outputwert berechnet und anhand dessen das Neural Network durch einen Optimierungsalgorithmus (kurz:*Optimizer*) justiert. Der Abweichungswert wird auch als *Fehler* oder *Loss* bezeichnet [21].

Das Berechnen des Loss während des Trainingsvorgangs übernimmt eine *Loss Funktion* (auch *Cost Function* genannt). Gleichung 2.2 ist ein Beispiel einer einfachen Loss Funktion, die Berechnung des

Mean Squared Errors [21, p. 134], die während eines Trainingsvorgangs für jede Iteration den Loss L ermittelt.

$$L = \frac{1}{m} \sum_{k=0}^{m} ||\hat{\nu}^{(k)} - \nu^{(k)}||^2 \tag{2.2}$$

Hierbei ist m die Anzahl Input-Werte im Trainingsdatensatz, $\hat{\nu}^{(k)}$ der vom Neural Network berechnete und $\nu^{(k)}$ der erwartete Output-Wert des korrespondierenden Inputwertes der k-ten Trainingsiteration [21, pp. 132–134].

Anschliessend nimmt ein Optimizer, wie z.B. der Adam Algorithmus [22], durch Verwendung des jeweiligen Loss das Feintuning an den Neuronengewichten vor. Dies geschieht durch Optimierungsmethoden wie Backpropagation [21], bei der der Optimizer, angefangen beim Output Layer, Rückwärts durch das Neural Network läuft und die einzelnen Werte der Gewichte anhand des errechneten Loss durch das Gradientenverfahren korrigiert.

Die mathematischen Operationen, die beim Training während jeder Iteration erneut ausgeführt werden, sind sehr rechenintensiv. Um Ressourcen zu sparen werden daher pro Iterationsdurchlauf mehrere Elemente des Datensatzes zu einem *Batch* (oder auch *Mini-Batch*) zusammengefasst und darauf die Operationen ausgeführt, anstatt jedes Element einzeln zu verarbeiten.

Wenn der Trainingsvorgang zu lange durchgeführt wird oder der Datensatz dafür nicht gross genug ist, kann es zu dem Effekt des *Overfittings* kommen. Dieser ist dadurch erkennbar, dass die Erkennungsrate bei den Trainingsdaten weiter zunimmt, während sie auf den Validierungsdaten schlechter wird [21, pp. 110–120]. Srivastava et al. [23] beschreibt eine Möglichkeit um diesem Problem durch das Hinzufügen eines sogenannten Dropout-Layers vorzubeugen.

Convolutional Neural Networks

CNNs sind eine Spezialform der Neural Networks. Dabei werden Convolution und Max–Pooling Layer anstelle von Dense Layern verwendet. Der Haupteinsatzbereich liegt in der Klassifizierung von Bildern [24]. Ein Convolution Layer besteht aus mehreren Feature Maps, wobei jede Feature Map eigene Gewichte besitzt. Die Neuronen in einer Feature Map sind nur mit einem kleinen lokalen Ausschnitt des Inputs verbunden, so können die gleichen Features in verschiedenen Bereichen des Bildes erkannt werden. Dies können beispielsweise Kanten in einem Bild sein, was der Objekterkennung dient [21]. Der Max-Pooling Layer generalisiert die Outputs des Convolution Layers, da die Tatsache, dass ein Feature vorkommt, wichtiger ist wie die Information wo es vorkommt [24].

2.3. Recurrent Neural Networks

Traditionelle Feed Forward Netzwerke sind nicht in der Lage, Informationen über zeitliche Relationen zu speichern. Um dies zu ermöglichen, wurden RNNs eingeführt. Es handelt sich um Netzwerke, welche durch einen Loop die zeitliche Information persistieren [25]. Dieser Loop lässt sich so erklären, dass eine RNN Zelle die Information zum Zeitpunkt t unter Berücksichtigung des vorhergehenden Resultats zum Zeitpunkt $(t-1)$ verarbeitet. Dabei wird der Output h_t einer RNN Zelle des ersten Zeitschrittes als zusätzlicher Input in $X_{(t+1)}$ im darauffolgenden Zeitschritt verwendet. Daher ist der Output h_t sowohl vom Input X_t als auch vom Output $h_{(t-1)}$abhängig, was in Abbildung 2.3 veranschaulicht wird. RNNs werden durch Backpropagation Through Time trainiert, die vergleichbar mit der normalen Backpropagation ist, mit dem Unterschied, dass der Gradient über die Zeitschritte hinweg berechnet wird. Backpropagation Through Time hat allerdings Schwächen, wenn ein Netzwerk über eine grössere Anzahl Zeitschritte hinweg trainiert werden soll. In solchen Fällen strebt der Gradient entweder gegen null, was als verschwindender Gradient bezeichnet wird, oder der Gradient strebt gegen unendlich, was als explodierender Gradient bezeichnet wird[26]. Daher werden heutzutage Weiterentwicklungen der RNN Architektur eingesetzt, die vor allem auf dem Konzept der LSTMs basieren. Im Folgenden wird

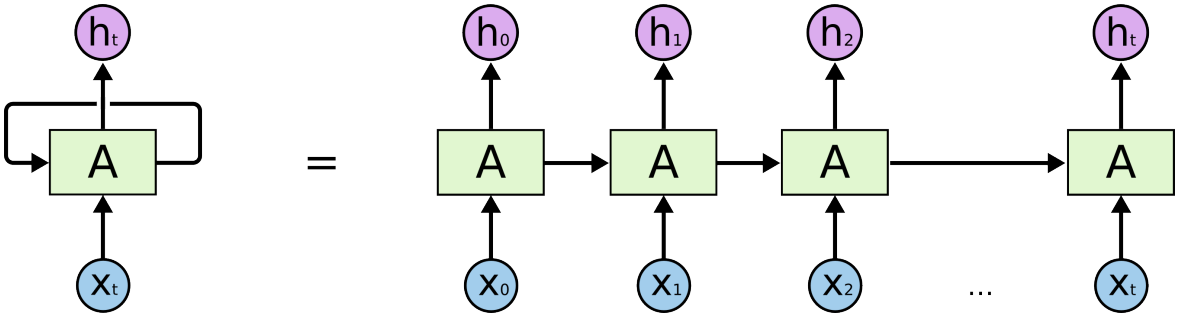


Abbildung 2.3.: Auseinandergefaltetes RNN Netzwerk [25]

auf die beiden in dieser Arbeit eingesetzten Netzwerkarten LSTM und das darauf aufbauende Gated Recurrent Unit (GRU) eingegangen.

2.3.1. Long Short-Term Memory

Um die zuvor erwähnten Probleme mit verschwindenden Gradienten zu beheben, wurde von Hochreiter et al. [27] die LSTM Architektur als Lösung vorgeschlagen. Bei LSTMs wird nicht mehr der Output des vorherigen Zeitschrittes als Input verwendet, sondern ein eigener separater Zellenzustand, welcher durch Gates beeinflusst werden kann. Gates einer LSTM Zelle bestehen daher aus mehreren Aktivierungsfunktionen. In Abbildung 2.4 ist der Zellzustand als obere horizontale Linie sichtbar, sowie die verschiedenen Gates und der Output, welcher als Hidden State h_t bezeichnet wird.

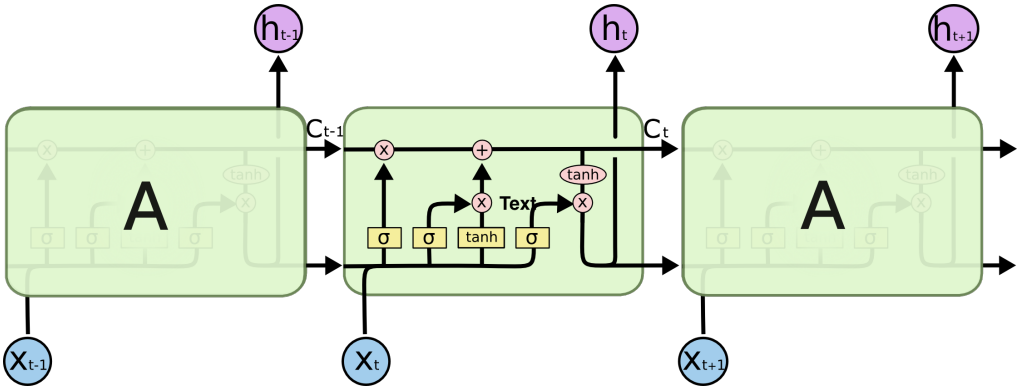


Abbildung 2.4.: LSTM Zelle [25]

2.3.2. Gated Recurrent Unit

Die im Jahr 2014 von Cho et al. [28] vorgestellten GRUs sind eine Variante der LSTMs. Die beiden Input- und Forget-Gates einer LSTM Zelle werden in einer GRU durch ein einziges Update-Gate ersetzt. Zudem besitzen GRUs gegenüber LSTM Zellen kein Gate vor dem Output mehr, sondern ein Reset Gate, welches direkt mit dem vorhergehenden Hidden State verbunden ist. Dies führt dazu, dass eine GRU im Vergleich zum traditionellen LSTM einen simpleren Aufbau hat, weshalb die GRU Architektur immer beliebter wird [25]. Zudem war das auch der Grund für dessen Verwendung in dieser Arbeit. Nachfolgend wird auf die Funktionsweise einer GRU Zelle genauer eingegangen.

Der Zellzustand C_t aus der LSTM Zelle, welcher in Abbildung 2.4 ersichtlich ist, verschwindet und es existiert nur noch der Hidden State h_t. Die erste Aktivierungsfunktion z_t hängt vom Input x_t, h_{t-1} und der Weight Matrix W_z ab. Zudem wird mit einer Sigmoid Funktion r_t berechnet, welche ebenfalls

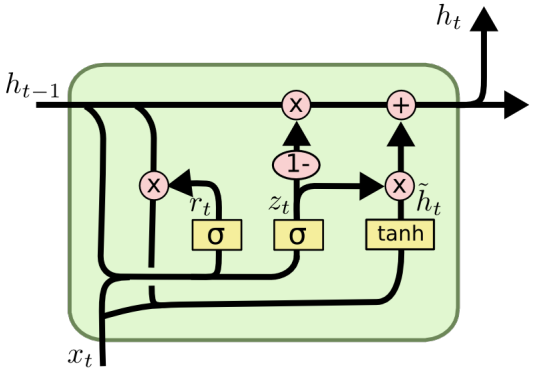


Abbildung 2.5.: GRU Zelle [25]

von x_t, h_{t-1} sowie der Weight Matrix W_r abhängt. Der Hidden State gelangt dann, wie in Abbildung 2.5 ersichtlich, durch die folgende Formel in den nächsten Zustand.

$$z_t = \sigma(W_z * [h_{t0}, x_t]) \tag{2.3}$$
$$z_t = \sigma(W_r * [h_{t-1}, x_t]) \tag{2.4}$$
$$\tilde{h_t} = \tanh(W * [r_t * h_{t-1}, x_t]) \tag{2.5}$$
$$h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h_t} \tag{2.6}$$

Aus Sicht der Performanz sind die Unterschiede zwischen den verschiedenen LSTM Varianten für die meisten Anwendungsfälle sehr gering [29]. Daher wurde für die reinen RNNs in dieser Arbeit weiterhin eine LSTM Implementation verwendet.

Bidirectional RNNs

Bidirectional RNNs verarbeiten die Input Sequenzen in beide Richtungen. Dies bedeutet, dass ein Layer die Input Daten vorwärts verarbeitet, d.h. $x_0, \ldots, x_t$, während der zweite Layer den Input rückwärts verarbeitet, d.h. $x_t, \ldots, x_o$. Die beiden Layer können als separate Netzwerke angesehen werden, zwischen denen keine Verbindung besteht [30]. Die beiden Outputs werden dann, wie in Abbildung 2.6 gezeigt, zusammengefügt. Durch die gleichzeitige Verarbeitung der Daten in beide Richtungen besitzt das Netzwerk zu jedem Zeitpunkt alle Informationen über vergangene und zukünftige Daten [30]. Die Zellen des RNNs können dabei alle möglichen Variationen beinhalten, es werden allerdings meist LSTM Zellen dafür verwendet [30].

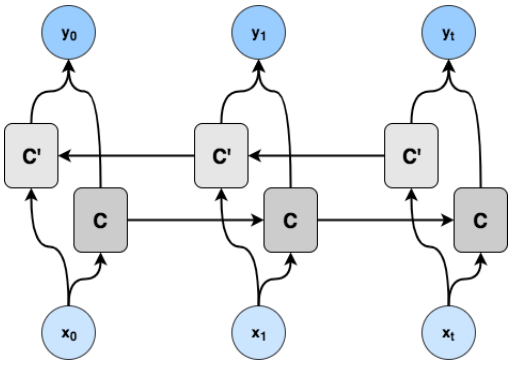


Abbildung 2.6.: Bidirectional RNN

Deep RNNs

RNNs sind von ihrer Architektur her Deep Neural Networks (DNN). Allerdings besteht die Tiefe hier über die Zeitschritte und nicht wie bei DNNs über mehrere Layer [31]. Um einem RNN mehr Tiefe zu verleihen, können ähnlich wie bei einem DNNs mehrere Layer übereinander gelegt werden. Dabei lernen die unterschiedlichen Layer über verschiedene Zeitintervalle, wie z.B. niedrigere Layer über kürzere Zeitintervalle. Dies bedeutet es werden nur Zusammenhänge über kurzer Zeitintervalle gespeichert. Bei höheren Layern wird das Zeitintervall länger [32]. Eine solche Architektur mit Bidirectional LSTMs (BLSTM) erzielt bessere Resultate als eine Architektur mit einem Layer [3].

2.4. Clustering

Zum Verständnis des Clustering Verfahrens, welches in dieser Arbeit zum Einsatz kam, werden Embeddings und das Prinzip des Hierarchical Clustering erklärt.

Embeddings

Von Embeddings, und im Falle dieser Arbeit von Sprecher Embeddings, wird gesprochen, wenn der Output eines Hidden Layers im Neural Network als Repräsentation des Sprechers verwendet wird [33]. Durch den Einsatz eines Hidden Layers können Embeddings extrahiert werden, die noch nicht vollständig auf den Identfikations Task trainiert sind und daher noch allgemeinere Features besitzen, welche zum Beispiel für das Clustering von Sprechern relevant sind.

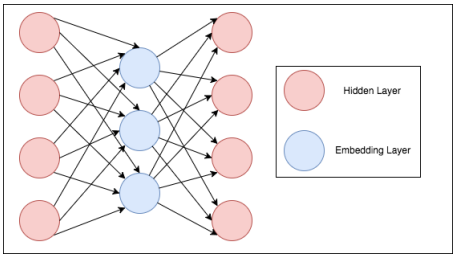


Abbildung 2.7.: Darstellung des Embedding Layer ohne Input und Output Layer

Hierarchical Clustering

Beim Hierarchical Clustering handelt es sich um eine Cluster Methode, in der die Cluster aufgrund eines Distanzmasses zwischen den einzelnen Embeddings gebildet werden. Ein solches Distanzmass ist die Euklidische Distanz, die in Gleichung 2.7 beschrieben ist.

$$d(\mathbf{p}\|\mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.7}$$

Die Variablen p und q sind dabei n dimensionale Vektoren, für welche die Distanz berechnet wird. Um nun die Distanz zwischen zwei Sets von Beobachtungen zu bestimmen, können verschiedene Metriken verwendet werden. Die in Gleichung 2.8 vorgestellte Methode wird Complete-Linkage Clustering genannt. Dabei werden aus zwei Sets A und B alle Vektoren mit der gewählten Distanz d paarweise verglichen und die maximale Distanz gewählt.

$$\max\{\, d(a,b) : a \in A, b \in B \,\} \tag{2.8}$$

Es wird mit den einzelnen Embeddings begonnen. Diese werden mit der obigen Metrik verglichen. Die beiden Embeddings mit der geringsten Distanz werden in einem Cluster zusammengefasst. Dies wird solange wiederholt, bis ein einziger Cluster existiert, welcher alle Embeddings enthält oder bis ein vorbestimmter Cut-off Point erreicht ist. Dies bedeutet, dass ein bestimmter Wert der Distanz erreicht wurde. Der daraus entstehende Cluster-Baum kann in einem Dendrogramm, wie in Abbildung 2.8, dargestellt werden.

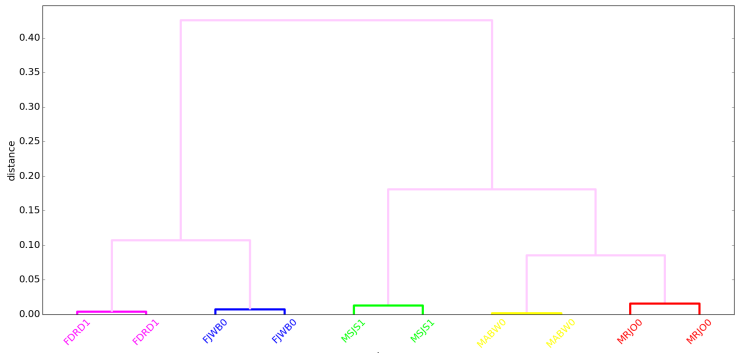


Abbildung 2.8.: Dendrogramm für Sprecherclustering auf 5 Sprechern mit jeweils 2 Aussagen.

2.5. Paarweise Kullback-Leibler Divergenz

Bei der paarweisen Kullback-Leibler Divergenz (PKLD) [34] wird der Output Vektor, der bei Klassifizierungsverfahren die Wahrscheinlichkeiten für die Klassenzugehörigkeit darstellt, als Wahrscheinlichkeitsverteilung, welche die verschiedenen Cluster darstellen, aufgefasst. Für zwei Segmente in der gleichen Klasse sollten die beiden Output Vektoren eine ähnliche Verteilung aufweisen, bei unterschiedlichen Klassen eine unterschiedliche Verteilung. Diese Idee wird folgendermassen umgesetzt [34].

Für ein gleiches Paar von Distributionen $\mathbf{P}$ und $\mathbf{Q}$, welche dem Output der Input Daten x_p und x_q für das Netzwerk f entsprechen, wird die KLD berechnet welche in der nachfolgenden Gleichung definiert ist:

$$KL(\mathbf{P}\|\mathbf{Q}) = \sum_i P_i \log \frac{P_i}{Q_i} \tag{2.9}$$

Für ein ungleiches Paar wird der Hinge-Loss verwendet, welcher, wie in Gleichung 2.10 dargestellt, die Distanz zwischen den Paaren maximiert. Die KLD wird weiterhin als Distanzmass eingesetzt.

$$HL(\mathbf{P}\|\mathbf{Q}) = \max(0, margin - KL(\mathbf{P}\|\mathbf{Q})) \tag{2.10}$$

Dies führt zu der Loss Funktion, welche in Gleichung 2.11 aufgeführt ist. Dabei ist I_s 1 für gleiche Paare (x_p, x_q) und 0 für ungleiche Paare. Das Gegenteil gilt für I_ds.

$$loss(\mathbf{P}\|\mathbf{Q}) = I_s \cdot KL(\mathbf{P}\|\mathbf{Q}) + I_ds \cdot HL(\mathbf{P}\|\mathbf{Q}) \tag{2.11}$$

Da der Loss sowohl für $(\mathbf{P}\|\mathbf{Q})$ als auch für $(\mathbf{Q}\|\mathbf{P})$ berechnet werden soll, ergibt sich die endgültige Loss Funktion.

$$L(\mathbf{P}, \mathbf{Q}) = loss(\mathbf{P}\|\mathbf{Q}) + loss(\mathbf{Q}\|\mathbf{P}) \tag{2.12}$$

Der $margin$ Parameter kann dabei frei gewählt werden. Allerdings wird von Hsu und Kira [34] ein Wert von 2 vorgeschlagen.

3. Vorgehen und Methoden

In diesem Kapitel werden die Ansätze vorgestellt, welche für diese Arbeit ausgewählt wurden. Zudem wird das Vorgehen zur Datengeneration, dem Training der Netzwerke und dem Testen der Ergebnisse dargelegt. Dazu werden auch die wichtigsten Messgrössen beschrieben.

3.1. LSTM Netzwerk Architektur

Für die LSTM Netzwerk Architektur wurde entschieden, sich an dem BLSTM Netzwerk von Gerber [1] zu orientieren, da dieses für die Sprecheridentifikation sehr gute Resultate erzielt hat. Allerdings wurden aufgrund der Experimente aus Abschnitt 4 dem BLSTM Layer noch zwei Dense Layer angefügt. Die beiden Dense Layer verwendeten 1000 respektive 500 Units und die ReLU Aktivierungsfunktion. Als Output des L3, dem letzten BLSTM Layer, wurde jeweils der letzte Output des vorwärts und rückwärts laufenden LSTMs genommen und zusammengehängt. So entstand ein Output Vektor der Länge 512. Dieser diente in den Experimenten auch als Embedding Layer für das Clustering. Das Netzwerk wurde mithilfe der Python-Bibliothek Keras [35] implementiert.

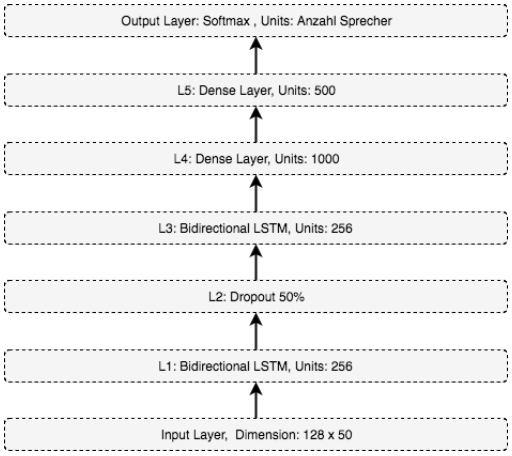


Abbildung 3.1.: Architektur des LSTM Clustering Netzwerks

3.2. CNN-GRU Architektur

Die CNN-GRU Architektur basierte auf Ansätzen aus verschiedenen Arbeiten. Der Aufbau der CNN Layer entsprach dem Identifikationsnetzwerk aus der Bachelorarbeit von Lukic und Vogt [9], in welcher gezeigt wurde, dass diese CNN Layer sprecherrelevante Features extrahieren können. Für die Verbindung des CNN Teils mit dem GRU Layer wurde der Ansatz von Harutyunyan and Khachatrian [8] verfolgt. Zudem kam für einige Experimente ein zusätzlicher Dense Layer nach dem GRU Layer zum Einsatz, da dies in der Arbeit von Sainath et al. [7] vorgeschlagen wurde. Der Aufbau des Netzwerkes wird in Abbildung 3.2 dargestellt.

Für die Verbindung der CNN Layer mit dem GRU Layer wurde der Output des letzten Max-Pooling Layers (L4) als 3D-Matrix mit den Dimensionen $[F, T, FM]$ aufgefasst. Die Frequenzrichtung F entsprach der Höhe, die Zeitrichtung T der Breite und die Feature Maps FM der Tiefe. Die Tiefendimension wurde mit der Frequenzdimension konkateniert, sodass dem Input des GRU Layers die Daten in den Dimensionen $[F \cdot FM, T]$ übergeben wurden. Der Transformationsprozess ist in Abbildung 3.3 veranschaulicht. Dies kann so aufgefasst werden, dass für jeden Zeitschritt alle Frequenzinformationen der Feature Maps verarbeitet werden. Die Netzwerk Architektur wurde direkt in TensorFlow umgesetzt.

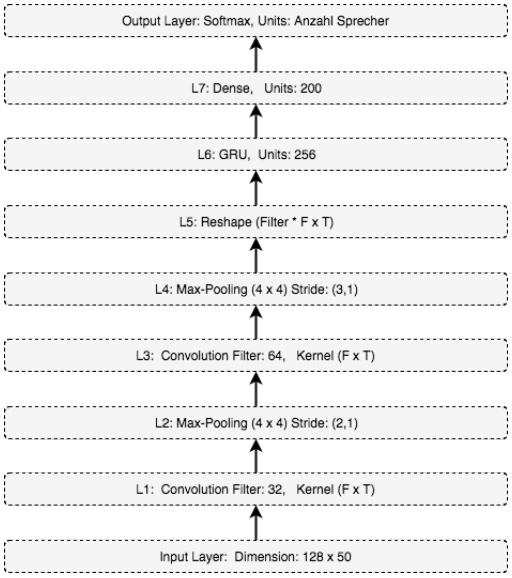


Abbildung 3.2.: Architektur des CNN-GRU Clustering Netzwerks

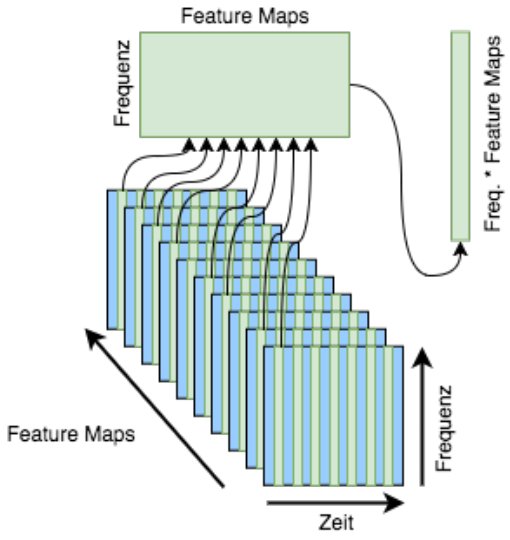


Abbildung 3.3.: Transformation des Max-Pooling Outputs in einen Input Vektor des GRU Layers

3.3. TIMIT Datensatz

Für die Durchführung der Experimente wurde der TIMIT Datensatz [36] verwendet, welcher Sprachaufnahmen von 630 Sprechern (438 männliche und 192 weibliche) in den acht häufigsten Dialekten der

USA enthält. Dieser Datensatz kam schon bei vorhergehenden Arbeiten von Lukic und Vogt [9], sowie in Lukic et al. [10] zur Anwendung. Dadurch war ein direkter Vergleich der Resultate möglich. Für die Trainigsläufe der Neural Networks kam ein Subset von 100 Sprechern zum Einsatz, welches wiederrum in 50 männliche und 50 weibliche Sprecher aufgeteilt war. Bei den Testläufen für das Clustering, wurden Datensätzen bestehend aus 40, 60 und 80 Sprechern verwendet. Die Sprecher der Testdatensätze waren nicht in den Trainingsdatensätzen vorhanden. Der Datensatz mit 40 Sprechern war identisch mit dem aus den Versuchen von Lukic und Vogt [9].

Um die Testdatensätze auszuwählen, wurden die Sprecher aus dem Testteil des TIMIT Datensatzes verwendet, wobei die ersten n Sprecher in lexikografischer Reihenfolge extrahiert wurden. Für diese Arbeit waren das die ersten 40, 60 bzw. 80 Sprecher. Somit waren alle Sprecher der kleineren auch eine Teilmenge der grösseren Testdatensätze. Auf diesem Weg waren die Resultate zwischen den Testdatensätzen vergleichbar, da ausgeschlossen wurde dass in den grösseren Datensätzen einfacher zu unterscheidende Sprecher vorkamen.

3.4. Daten Extraktion

Die Sprachaufnahmen lagen in Form von WAV-Dateien vor und mussten daher zuerst in Mel Spektrogramme umgewandelt werden, um dann als Input für die Netzwerke verwendet werden zu können. Dafür wurde dasselbe Extraktionvorgehen wie in den Arbeiten von Lukic und Vogt [9] und Gerber [1] angewandt. Nach der Extraktion lagen die Daten also in folgender Form vor:

$$X[Sprechersatz, Kanal, Frequenz, Zeit] \tag{3.1}$$
$$Y[Sprechersatz] \tag{3.2}$$

Dabei liegt die Dimensionsgrösse der $Frequenz$ bei 128 und die der $Zeit$ bei 800. Eine Sekunde entsprach also genau 100 Zeiteinheiten. Sätze die unter einer Länge von acht Sekunden lagen wurden durch das anfügen von Nullen auf die Länge vom 800 Zeiteinheiten angepasst. Die Dimension $Sprechersatz$ ergab sich aus $Anzahl\ Sprecher \cdot Anzahl\ Sätze\ pro\ Sprecher$. Die Daten wurden jeweils in einen Trainingsdatensatz mit acht und einen Testdatensatz mit zwei Sätzen pro Sprecher unterteilt.

3.5. Mini-Batchgenerierung

Um Trainingsdaten aus der vorhergehenden Extraktion zu bilden, wurden Mini-Batches nach dem Vorgehen aus der Arbeit von Lukic und Vogt [9] erstellt. Dafür ist jeweils ein zufälliger Sprechersatz ausgewählt worden und daraus wiederum ein zufälliges Segment extrahiert. Dies wurde solange wiederholt bis ein Mini-Batch voll war. Die Grösse eines Mini-Batches war auf 100 Segmentteile gesetzt. Die Segmentlänge variierte dabei je nach Experiment und Netzwerk. Die extrahierten Mini–Batches haben die Form:

$$XT[BatchGrösse, Zeit, Frequenz, Kanal] \tag{3.3}$$
$$YT[BatchGrösse, Sprecher] \tag{3.4}$$

Die Labels in YT werden dabei in ein One-Hot Encoding umgewandelt. Da für jeden Mini-Batch ein zufälliges Segment aus einem zufälligen Satz gewählt wurde, sah das Netzwerk sehr selten die gleichen Daten, was einem Overfitting entgegenwirkte. Die Daten des Trainingsdatensatzes waren wiederum in Trainings- und Validierungsdaten aufgeteilt, wobei 80% davon für das Training und 20% für die Validierung genommen wurde.

3.6. Messgrössen

Um den Erfolg des Trainings messen zu können, sind die beiden Messgrössen, Accuracy und Misclassification Rate (MR) verwendet worden. Verwendung und Funktionsweise der beiden Grössen werden im Folgenden erläutert.

Accuracy

Bei den Netzwerken, welche den Loss durch die Cross-Entropy Funktion berechneten, wurde die Accuracy zur Überprüfung des Lernfortschritts eines Netzwerks verwendet. Für die Berechnung der Accurracy wurde die in 3.5 dargestellte Funktion verwendet.

$$accuracy(t) = \frac{t}{N} \tag{3.5}$$

Dabei steht t für die Anzahl richtig klassifizierter Segmente und N für die Gesamtmenge der Segmente.

Misclassification Rate

Die Genauigkeit des Clusterings der Netzwerke wurde durch die MR bestimmt, welche wie folgt definiert ist:

$$MR = \frac{1}{N} \sum_{j=1}^{N_s} e_j \tag{3.6}$$

N ist die Gesamtmenge der Segmente, N_s steht für die Anzahl der Sprecher und e_j ist die Anzahl falsch zugewiesener Segmente eines Sprechers.

3.7. Experiment Ablauf

Im Folgenden wird auf die Vorgänge des Trainings und des Clusterings eingegangen, die auf die Netzwerke angewendet wurden. Die Vorgänge lehnten sich das Verfahren von Lukic et al. [5].

Training

Das Training der Netzwerke lief, wie in Abbildung 3.4 veranschaulicht, ab. Dies galt sowohl für die in Keras Implementierten BLSTM Netzwerke, als auch für die in TensorFlow implementierten CNN-GRU Netzwerke.

- **Traininsdaten laden:** Laden der Trainingsdaten aus Pickle-Dateien, welche in Abschnitt 3.3 definiert sind.
- **Netzwerk erstellen:** Erstellen des Netzwerks und setzen von Parametern wie z.B. Anzahl Units und Layer, sowie Anzahl der zu durchlaufenden Trainingsiterationen.
- **Training starten:** Das Training wurde gerstartet. In Keras kam hierfür `model.fit_generator` zum Einsatz. Für die TensorFlow Implementierung wurde hier eine Iterationsloop verwendet.
- **Mini-Batches generieren:** Generieren eines Mini-Batches mit der Elementgrösse 100, mithilfe der Funktion `batch_generator_lstm` aus der Python-Klasse `data_gen.py`.

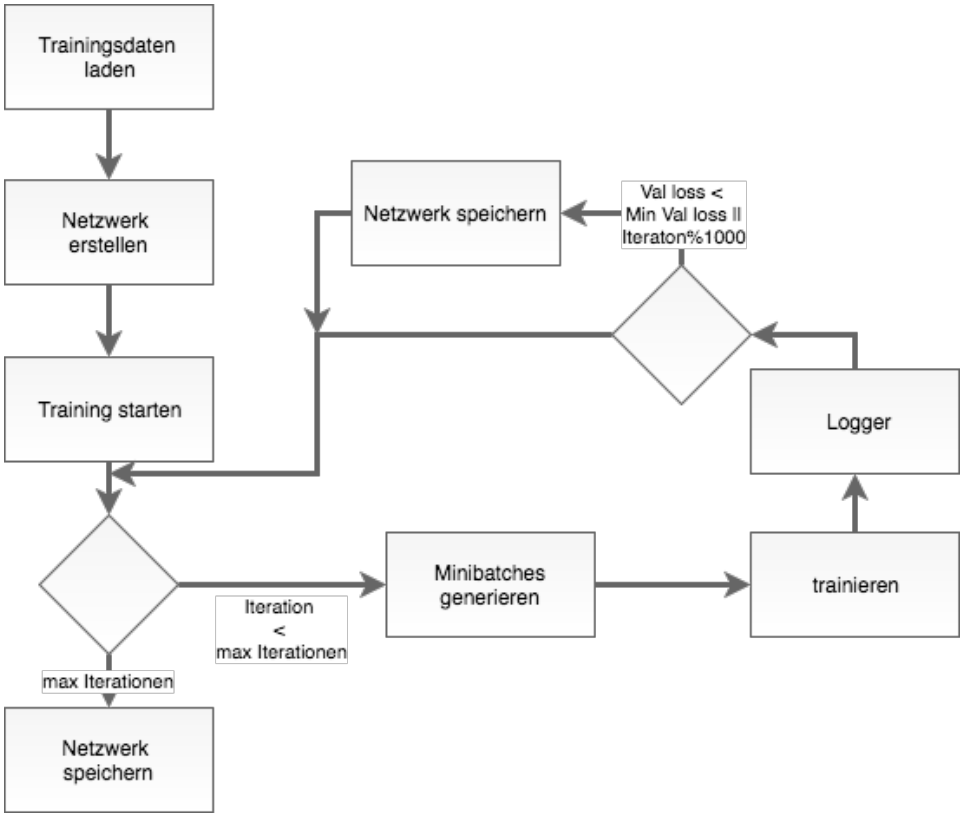


Abbildung 3.4.: Traininsablauf

- **trainieren:** Trainieren des Netzwerks durch den Adam Optimizer unter Verwendung der Cross-Entropy oder PKLD Funktion.
- **Logger:** Mitschreiben in eine Datei im CSV Format, sowie Ausgabe auf der Konsole der aktuellen Resulatete nach jedem 10. Mini-Batch.
- **Netzwerk speichern:** Alle 10 Mini-Batches wurde der Validation Loss überprüft und im Falle eines neuen Minimalwerts, sowie generell alle 1000 Iterationen, unabhängig des Validation Loss, gespeichert.
- **Endgültiges Netzwerk speichern:** Speichern der trainierten Netzwerke nach Beendigung der Trainingsiterationen. Bei Keras geschah dies im HDF5- und bei TensorFlow im Checkpoint-Format.

Optimizer

Für das Training der Netzwerke wurde der Adam Optimizer verwendet. Dieser vereint die Prinzipien der beiden Optimizer AdaGrad und RMSpro. Adam berechnet für jeden Parameter eine adaptive Lernrate und passt diese gemeinsam mit dem Momentum während des Trainings selbständig an [22]. Da der Optimizer die Lernrate selbstständig einstellt, war Feinjustierung der Lernrate nicht notwendig. Adam erreichte vergleichbare und teilweise geringfügig bessere Resultate als andere adaptive Optimizer [37]. Es wurden die von Kingma and Ba [22] für die Initialisierung des Optimizers vorgeschlagegenen Werte verwendet: $Learningrate = 0.001, \beta_1 = 0.09, \beta_2 = 0.999$ und $\epsilon = 10^{-8}$.

Loss Funktion

Neben der PKLD aus Abschnitt 2.5 kam die Cross-Entropy Funktion für die Berechnung des Loss zum Einsatz. Diese wurde verwendet, da mit ihr bereits gute Resultate auf dem Sprecheridentifizierungs Task in der Projektarbeit von Gerber [1] erzielt wurden. Die Cross-Entropy Funktion ist in Gleichung

3.7 definiert. Hierbei ist $y(x)$ das Label für den Input x, a der Output Vektor des Netzwerkes für den Input x und n die Anzahl Inputs.

$$C = -\frac{1}{n}\sum_{x}[y(x)\ln a + (1-y(x))\ln(1-a)] \tag{3.7}$$

Für die Cross-Entropy Funktion müssen die Labels als One-Hot Vektor kodiert sein. Diese Funktion hat gegenüber anderen Loss Funktionen den Vorteil, dass auch bei schlechter Initialisierung und grosser Abweichung zwischen dem Output und den Labels, die Lerngeschwindigkeit nicht abnimmt [38].

Clustering

Um das Clustering durchführen zu können, wurden disjunkte Segmente auf den trainierten Netzwerken evaluiert und die anschliessend erstellten Embeddings exportiert. Wichtig bei diesem Vorgang zu beachten war, dass die Segmente von Sprechern stammten, die nicht im Trainingsdatensatz vorkamen. Die Embeddings wurden in zwei Sets pro Sprecher aufgeteilt. Das erste Set bestand aus den Segmenten von acht Sätzen, dass zweite aus Segmenten von zwei Sätzen. Anschliessend wurde der Durchschnitt der Embeddings pro Set gebildet, wodurch genau zwei Embeddings (Aussagen) pro Sprecher entstanden, die anschliessend durch ein Hierarchisches Clustering zugeordnet wurden. Dabei wurde der Cut off Point für das Segmentpaar mit der besten MR gewählt. Wenn ein Cluster eine Aussage eines anderen Sprechers enthält, werden alle Aussagen innerhalb dieses Clusters als *falsch* gezählt. Aussagen werden immer als *falsch* gewertet, wenn sie keinem Cluster zugewiesen werden. Dieser Ansatz macht die Ergebnisse mit den vorhergegangenen Arbeiten von Lukic und Vogt [10] [9] vergleichbar. Die Qualität des Clusterings wird durch die MR gemessen.

3.8. Verwendete Software

Im Folgenden sind die Bibliotheken und deren Version aufgelistet, die für die Umsetzung der Versuche verwendet wurden. Besonders bei Keras [35] und TensorFlow war es entscheidend die richtigen Versionen zu verwenden, da das API dieser beiden Bibliotheken sich über die verschiedenen Versionen stark verändert hat.

Netzwerk

- `TensorFlow 1.0.1`
- `Keras 1.0`
- `NumPy`
- `Matplotlib 1.5.3`
- `h5py 2.6.0`

Spektrogramm Extraktion

- `Numpy 1.10.4`
- `SciPy 0.16.1`
- `Librose 0.4.3`

4. Experimente BLSTM

In diesem Abschnitt werden die Experimente, welche auf den BLSTM Netzwerken ausgeführt wurden, beschrieben. Dabei wurde zuerst mit der Netzwerkarchitektur aus der Projektarbeit von Gerber [1] begonnen. Alle Experimente verwendeten den Adam Optimizer, um die Netzwerke zu trainieren.

Für die Experimente wurde der Trainingsdatensatz mit 100 Sprechern verwendet, um die Netzwerke zu trainieren. Dieser ist in Abschnitt 3.3 beschrieben. Die Evaluationen der Netzwerke wurden mit Testdatensätze mit 40, 60 und 80 Sprechern durchgeführt. Alle Trainings- und Testdatensätze sind in Abschnitt 3.3 beschrieben. Die Sprecherlisten können im Abschnitt A.2.3 des Anhangs eingesehen werden.

4.1. BLSTM Segmentlänge der Input Daten

Wie in der Projektarbeit von Gerber [1] gezeigt wurde, eignen sich BLSTM Netzwerke für die Aufgabe der Sprecheridentifikation. Dieses Netzwerk mit zwei BLSTM Layern wurde nun verwendet und auf die Clustering Fähigkeit untersucht. In der Projektarbeit von Gerber [1] wurden die besten Resultate für das Sprecheridentifiaktionsexperiment mit einer Segmentlänge der Input Daten von 150 ms erreicht, wobei jedoch zu erkennen war, je kürzer einzelne Segmente wurden, desto schlechter fiel die Accuracy aus. Daher wurde das Netzwerk in dieser Arbeit auch mit Segmenten der Länge 300ms, 500ms, 750ms, und 1000ms trainiert.

Für das Testen das Clusterings wurde der Testdatensatz mit 40 Sprechern und als Loss die Cross-Entropy Funktion verwendet. Die Netzwerke wurden mit 1000 Iterationen trainiert.

Es wurde erwartet, dass die Netzwerke mit kürzeren Segmentlängen ein besseres Resultat liefern, da diese in der Projektarbeit von Gerber [1] die besten Resultate für die Sprecheridentifikation erzielten.

Auswertung

Entgegen der Erwartungen schnitten die Netzwerke mit den zwei kürzesten Segmentlängen am schlechtesten ab. Die besten Resultate wurden, wie in Tabelle 4.1 zu sehen ist, mit einer Segmentlänge von 500ms erreicht. Dies konnte dadurch erklärt werden, dass zwar die Accuracy in den Identifikationsexperimenten[1] über alle Segmente mit geringerer Segmentlänge stieg, die Accuracy pro Segment jedoch abnahm. Daher wurde angenommen, dass die Accuracy pro Segment für das Clustering eine grössere Rolle spielte.

Segmentlänge	MR
150 ms	0.225
300 ms	0.175
500 ms	**0.100**
750 ms	0.125
1000 ms	0.125

Tabelle 4.1.: MR der verschiedenen Segmentlängen

4.2. BLSTM mit zusätzlichen Dense Layer

In der Bachelorarbeit von Lukic and Vogt [9] wurde gezeigt, dass die Embeddings eines Layers, welcher weiter vom Output Layer entfernt ist, sich besser für das Clustering eignen. Daher wurde eine Architektur des 2-Layer BLSTM Netzwerkes um zwei zusätzlichen Dense-Layer mit 1000 und 500 Units erweitert, wie in Abschnitt 3.1 dargelegt.

Das Netzwerk wurde, wie im vorherigen Experiment, mit 10 000 Iterationen trainiert. Dabei hatten die Segmente eine Länge von 500ms, da diese im vorherigen Experiment die besten Resultate erzielten. Für das Clustering kam der Testdatensatz mit 40 Sprechern zu Einsatz.

Es wurde erwartet, dass das Clustering besser wird, je weiter ein Layer vom Output entfernt war. Zudem wurde erwartet, dass alle Embedding Layer bedeutend besser abschneiden als der Output Layer, da dieser vollständig auf die Sprecheridentifizierung trainiert wurde.

Auswertung

Die Resultate in Tabelle 4.2 zeigen, dass der Output Layer (L6), wie erwartet die schlechteste MR erreichte. Die beiden Dense Layer (L5, L4) erreichen eine MR von 0.075. Der LSTM Output erreichte die beste MR von 0.025. Damit wurde die in der Arbeit von Lukic und Vogt [9] erreichte MR von 0.05 unterboten. Somit hatte diese Architektur einen neuen Bestwert für das Clustering auf 40 Sprechern erreicht.

Layer	**MR**
L6: Softmax	0.275
L5: Dense 2	0.075
L4: Dense 1	0.075
L3: LSTM out	0.025

Tabelle 4.2.: MR der verschiedenen Embedding Layer

4.3. BLSTM im PKLD Loss Funktion

In diesem Experiment wurde untersucht, ob die PKLD eine Verbesserung des Clusterings bringen würde. Dazu wurde dieselbe Netzwerkarchitektur wie im vorherigen Experiment verwendet.

Das Netzwerk wurde mit 10 000 Iterationen trainiert. Das Clustering wurde ebenfalls auf denselben 40 Sprechern ausgeführt, welche bereits in den vorherigen Experimenten zum Einsatz kamen.

Es wurde erwartet, dass sich die Clustering Performanz im Vergleich zum vorherigen Experiment weiter verbessert, da die PKLD, wie in Abschnitt 2.5 beschrieben, einen Loss berechnet, der das Netzwerk besser für den Clustering Task trainiert. Auch wurde in der Arbeit von Lukic und Vogt [9] die PKLD erfolgreich verwendet. Zudem wurde auch erwartet, dass der BLSTM Output Layer am besten abschneidet, da dieser am weitesten vom Output Layer entfernt war.

Auswertung

Es konnte im Vergleich zum Netzwerk, welches mit der Cross-Entropy Funktion trainiert wurde, keine Verbesserung der MR erreicht werden. Der BLSTM Output Layer hatte die gleiche MR von 0.025 erreicht. Wie die Resultate in Tabelle 4.3 zeigen, erreichten die beiden Dense Layer allerdings ein schlechteres Resultat. Vermutung lagen darauf, dass dies aufgrund der Zufallseinflüsse zustande kam. Diese wurden durch das zufällige Initialisieren der Gewichte zu Beginn des Trainings hervorgerufen. Ein weiterer Faktor für diese Schwankungen war, dass die Mini-Batches immer aus zufällig ausgewählten Segmenten bestanden. So wurden in jedem Testlauf leicht andere Daten verwendet.

Layer	MR
L6: Softmax	0.300
L5: Dense 2	0.125
L4: Dense 1	0.125
L3: LSTM Out	0.025

Tabelle 4.3.: MR der verschiedenen Embedding Layer für das PKLD Netzwerk

Wie Abbildung 4.1 dargestellt ist, fand nach ca. 6000 Iterationen ein leichtes Overfitting statt. Wie die Auswertung des Clusterings beim Netzwerk mit dem niedrigsten Validation Loss, welcher bei 6100 Iterationen erreicht wurde, zeigte, hatte dieses Overfitting keinen negativen Einfluss auf das Clustering, da auch dort eine MR von 0.025 erreicht wurde.

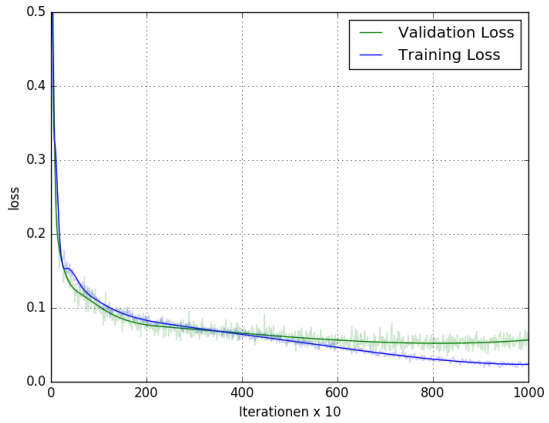
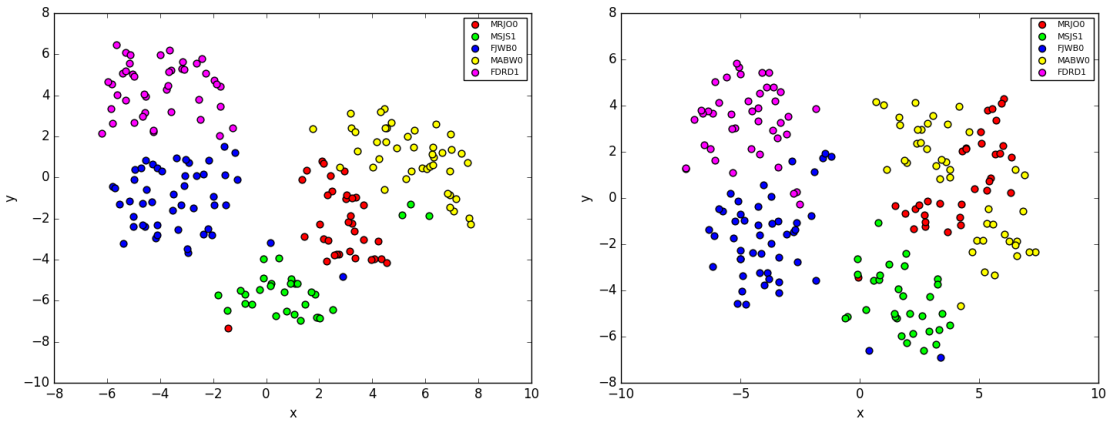


Abbildung 4.1.: Architektur des LSTM Clustering Netzwerks

Da mit Cross-Entropy Loss die gleiche MR auf 40 Sprechern erreicht werden konnte, wurden die Embeddings von fünf Sprechern für die beiden Netzwerke analysiert, da diese im vorherigen Experiment die besten Resultate erzielten. Wie Abbildung 4.2 zeigt, war das Clustering für das Netzwerk mit der PKLD auf den Embeddings deutlich besser. Daher wurde beschlossen für die weiteren Experimente die PKLD zu verwenden.



(a) PKLD Netzwerk Embeddings Layer 3 (b) Cross-Entropy Netzwerk Embeddings Layer 3

Abbildung 4.2.: Embeddings für fünf Sprecher auf dem LSTM Output Layer

4.4. Clustering mit mehr Sprechern

Das Netzwerk aus dem vorherigen Experiment sollte nun dahingehend überprüft werden, ob ein Clustering für 60 und 80 Sprecher ebenfalls gute Resultate lieferte. Aufgrund der in Abschnitt 4.3 beschriebenen Zufallseinflüsse wurden noch weitere Testläufe durchgeführt.

Es wurde erwartet, dass die MR mit zunehmender Sprecheranzahl steigt, da es immer anspruchsvoller wurde, Sprecher auseinander zu halten. Dies würde auch die Resultate aus der Studie von Lukic et al. [5] bestätigen.

Auswertung

Wie die Tabelle 4.4 dargestellt, variierten die Ergebnisse der drei Testläufe. Allerdings lagen die Ergebnisse relativ nahe beieinander. Bei den Testläufen 1 und 3 war das erwartete Verhalten eingetreten: je mehr Sprecher geclustert wurden, desto höher fiel auch die MR aus. Der 2. Testlauf erreichte allerdings eine tiefere MR für 60 Sprecher, was ebenfalls mit den Schwankungen während des Trainings erklärt werden kann.

	MR		
Anzahl Sprecher	**Run 1**	**Run 2**	**Run 3**
40	**0.025**	0.050	0.050
60	0.083333	**0.033333**	0.066667
80	0.1125	**0.075**	0.125

Tabelle 4.4.: MR für 3 Testläufe des KLD Netzwerkes für 40, 60 und 80 Sprecher

Für den Testlauf 1 wurde ein Dendrogramm erstellt, um zu analysieren welche Sprecher für den Fehler verantwortlich waren. Wie in Abbildung 4.3 zu sehen ist, entstand der Fehler, da für die Sprecherin FJEM0 die Distanz zu gross war und der optimale Cut-off Point darunter lag. Das manuelle Reinhören in die Samples von FJEM0 brachte jedoch keinen Aufschluss darüber, warum hier eine hohe Distanz existierte. Für das Gehör der Autoren klangen alle Sätze sehr ähnlich. Allgemein war festzustellen, dass mit Ausnahme einer weiblichen Sprecherin vor allem Aussagen von männlichen Sprechern eine grosse Distanz zueinander aufwiesen.

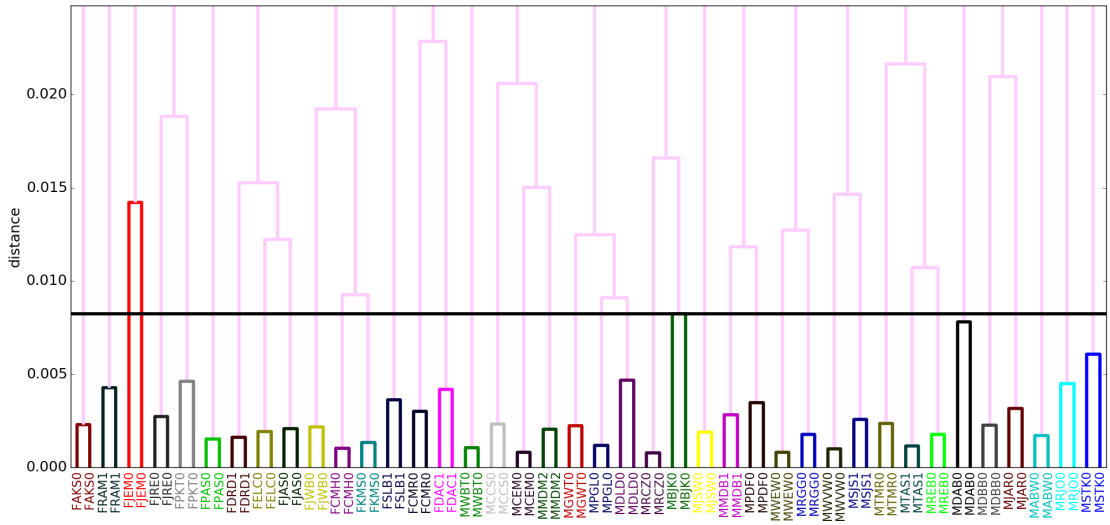


Abbildung 4.3.: Dendrogram mit 40 Sprecher, für das Netzwerk aus dem ersten Testlauf , "F- weiblich / M = "Männlich"

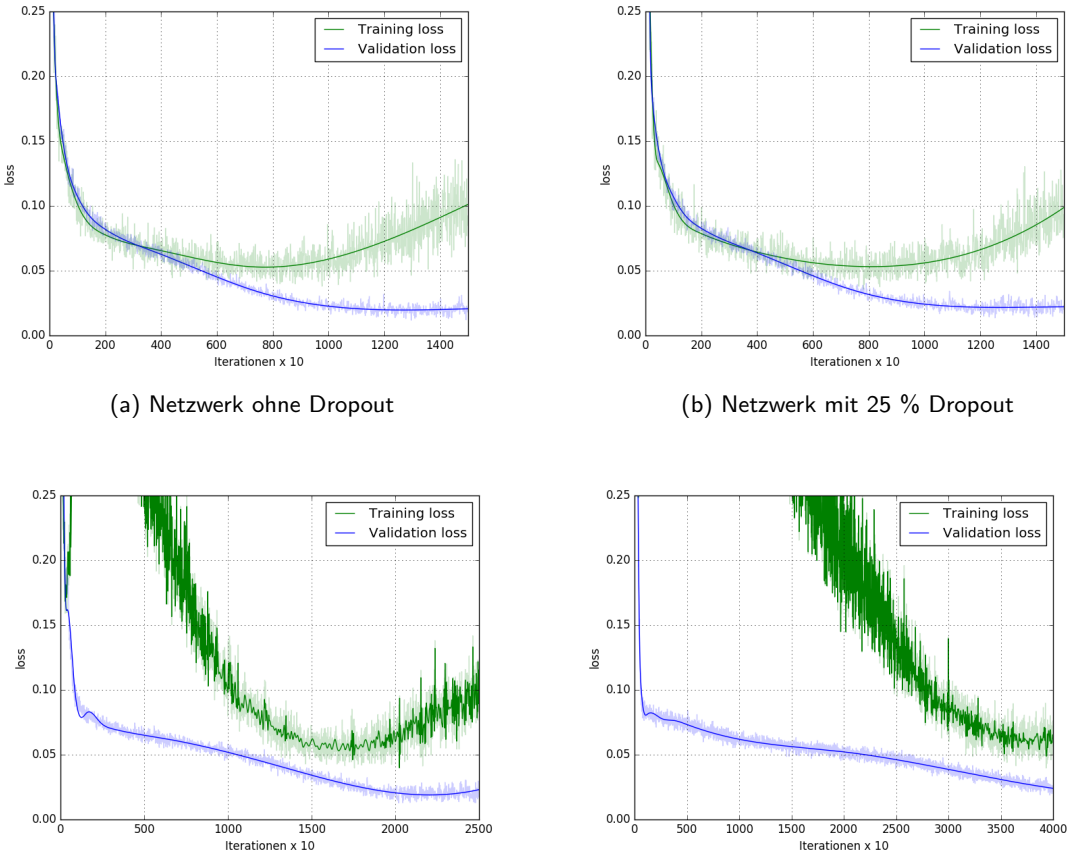
4.5. Zusätzlicher Dropout Layer

Wie in Experiment 3 zu sehen ist, fand ein starkes Overfitting während des Trainings stattfindet. Dies schien durch die beiden Dense Layer, welche im Experiment aus Abschnitt 4.2 hinzugefügt wurden, verursacht zu werden. Aus diesem Grund wurde nun die Auswirkung eines zusätzlichen Dropout Layers zwischen den beiden Dense Layern untersucht. Ein Dropout Layer setzt eine spezifizierte Anzahl Gewichte nach jedem Batchdurchlauf auf 0 zurück. Wie die Studie von Srivastava et al. [23] zeigt, verhindert oder minimiert dieses Vorgehen das Overfitting.

Es wurden Netzwerke mit einem Dropoutfaktor von 50%, 40% und 25% getestet. Die Evaluation der Resultate fand erneut auf den Testdatensätzen von 40, 60 und 80 Sprechern statt.

Es wurde erwartet, dass die Dropout Layer das Overfitting vermindern. Als Nebeneffekt wurde aber auch erwartet, dass die Netzwerke langsamer lernen, da ein Teil der gelernten Weights wieder zurückgesetzt wurde.

Auswertung



(a) Netzwerk ohne Dropout

(b) Netzwerk mit 25 % Dropout

(c) Netzwerk mit 40% Droput

(d) Netzwerk mit 50% Dropout

Abbildung 4.4.: Lernverhalten der Netzwerke mit unterschiedlichen Dropout stärke zwischen den Dense Layern

Wie die Trainingsverläufe in Abbildung 4.4 zeigen, verbesserten die Dropout Layer den Validation Loss nicht. Es fand auch bei allen Testläufen ein Overfitting statt. Die Netzwerke mit 40% und 50% Dropout benötigten zudem bedeutend mehr Iterationen, um den besten Validation Loss zu erreichen. Ein zusätzliches Problem war die Tatsache, dass die Netzwerke bereits ein Overfitting aufwiesen bevor

der Trainings Loss die tiefsten Resultate erreichte. Da das Lernverhalten für die Netzwerke mit 40 und 50% Dropout nicht vielversprechend war und auch das Clustering schlechter als im Netzwerk mit 25% Dropout rauskam, wurde entschieden für diese beiden Konfigurationen nur einen Testlauf durchzuführen.

	25 %			**40%**	**50 %**
Anzahl Sprecher	**Run 1 15k**	**Run 2 10k**	**Run 3 10k**	**Run 1 20k**	**Run1 40k**
40	0.025	**0**	0.025	0.050	0.150
60	**0.033333**	0.083333	0.050	0.066667	0.250
80	**0.100**	0.125	0.125	0.1375	0.325

Tabelle 4.5.: MR für Netzwerke mit 25 %, 40%, 50% Dropout, mit Angabe der trainierten Iterationen

Die Werte in Tabelle 4.5 zeigen, dass das Netzwerk mit 25% Dropout die besten Resultate erreichte. Auf 40 Sprechern erreichte der zweite Run sogar ein fehlerfreies Clustering. Dies unterbot die bis anhin erreichten Bestwerte. Auch wenn der zusätzliche Dropout Layer dem Overfitting kaum entgegenwirken konnte, wurde ein besseres Resultat erzielt wie das ohne Dropout Layer. Daher wurde in nachfolgenden Experimenten das Netzwerk mit dem zusätzlichen Dropout Layer mit 25% Dropout verwendet.

4.6. Anpassung des Margin Parameters

Wie in der Bachelorarbeit von Lukic und Vogt [9] vorgeschlagen, wurde in diesem Experiment der Margin Parameter der PKLD untersucht. Es wurde bis anhin immer mit einem Wert von 2 getestet, welcher in der Studie von Hsu and Kira [34] empirisch bestimmt worden ist. Allerdings wurde dieser für das Training auf dem MNIST Datensatz [39], welcher handschriftliche Ziffern beinhaltet, verwendet. Deshalb wurde nun untersucht, ob für das Training auf Spektrogrammen ein anderer Margin Parameter bessere Resultate lieferte.

Dafür wurden die Werte 1.5, 2.5 und 3.0 getestet. Das Training betrug 15000 Iterationen, wobei alle 1000 Iterationen ein Netzwerk gespeichert und das beste verwendet wurde.

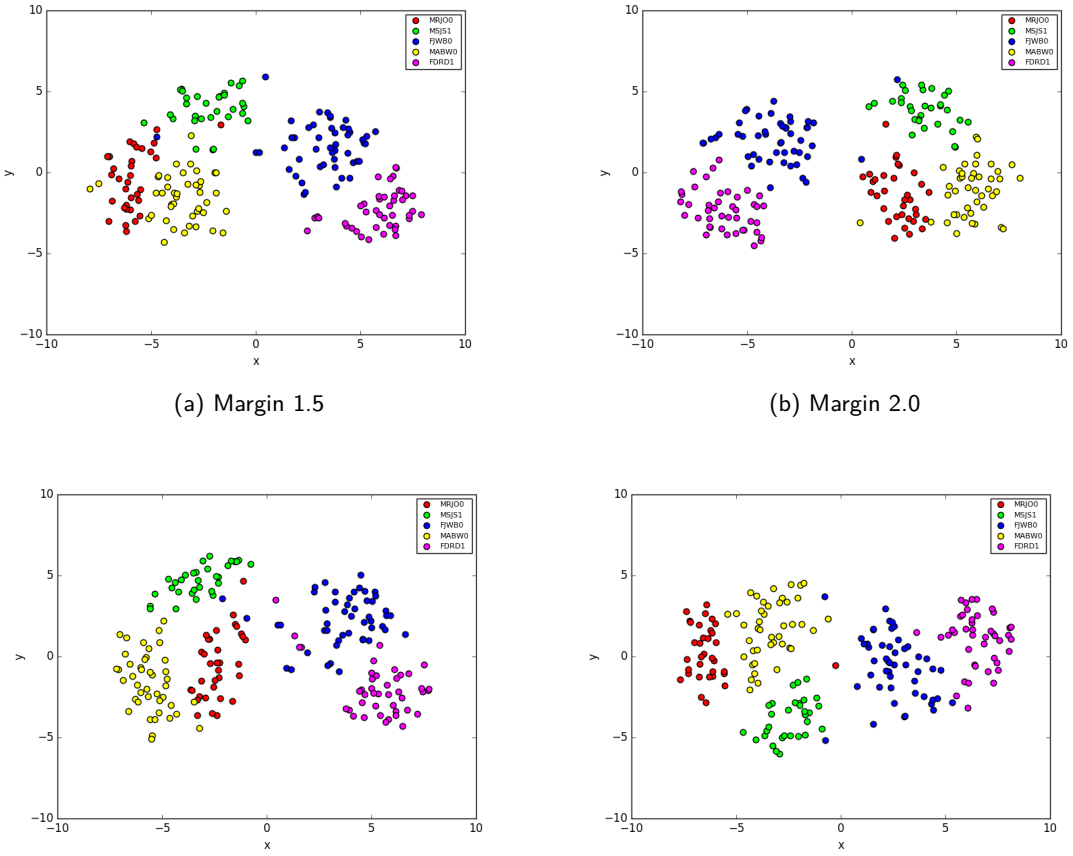
Die Erwartungen waren, dass sich für grössere Margin Werte der Abstand zwischen den Clustern vergrössert, da der Hinge-Loss wie in Gleichung 2.10, die Interclusterdistanz beeinflusst. Dies sollte sich positiv auf das Clustering auswirken.

Auswertung

	Margin 1.5		**Margin 2.5**		**Margin 3**	
Anzahl Sprecher	**Run 1 10k**	**Run 2 10k**	**Run 1 10k**	**Run 1 10k**	**Run 1 10k**	**Run 2 12k**
40	0.075	0.500	0.250	0.500	**0**	0.250
60	0.116667	0.100	0.016667	0.500	0.050	**0.016667**
80	0.175	0.1125	0.1	0.100	**0.0875**	0.100

Tabelle 4.6.: MR für die Netzwerke Trainiert mit verschiedenen Margin Werten.

Wie in der Tabelle 4.6 zu sehen ist, schnitten die Netzwerke mit Margin 2.5 und Margin 3 bedeutend besser ab als das Netzwerk mit Margin 1.5. Dies war auch zu erwarten, da eine kleinere Interclusterdistanz das Auseinanderhalten der Cluster erschweren sollte. Wie die Visualisierung der Embeddings für 5 Sprecher in Abbildung 4.5 zeigen, waren die Unterschiede rein optisch sehr gering. Dabei konnte jedoch erkannt werden, dass die Cluster bei Margin 1.5 näher zusammen waren. Allerdings zeigten die Resultate, dass die Netzwerke mit 2.5 und 3.0 Margin leicht bessere Resultate lieferten. Das Netzwerk



(a) Margin 1.5

(b) Margin 2.0

(c) Margin 2.5

(d) Margin 3.0

Abbildung 4.5.: Lernverhalten der Netzwerke mit unterschiedlichen Dropout stärke zwischen den Dense Layern

mit Margin 3 erreichte ein perfektes Clustering für 40 Sprecher und eine MR von 0.5, respektive 0.875 auf 60 und 80 Sprecher. Damit lag der Wert für 60 Sprecher auf gleichem Niveau wie in der Studie von Lukic et al. [5] und der für 80 Sprecher deutlich darunter.

4.7. Fazit

Es konnte gezeigt werden, dass sich BLSTMs sich für die Aufgabe des Sprecherclusterings eignen. So erreichte schon das BLSTM Identifikationsnetzwerk aus Abschnitt 4.1 eine MR von 10% für 40 Sprecher, bei einem Trainingsdatensatz von 100 Sprechern. Als das Netzwerk um zusätzliche Dense Layer erweitert wurde, konnte die MR nochmals verbessert werden, was auch den Beobachtungen aus der Arbeit von Lukic und Vogt [9] entsprach.

Der Einsatz der PKLD als Loss Funktion brachte nochmals kleine Verbesserungen hervor. Allerdings war dieser Effekt geringer als erwartet. Die visuelle Analyse der Embeddings zeigte jedoch, dass die PKLD ein besseres Clustering erzeugen konnte. Es wurde gezeigt, dass die Erhöhung des Margin Parameters des PKLD Losses eine weitere Verbesserung der Clustering Resultate erreichte. So wurden 40 Sprecher fehlerfrei geclustert. Für 60 Sprecher konnte eine MR von 5% erreicht werden, was dem Resultat aus der Studie von Lukic et al. [5] entsprach, welche bis dahin die besten Resultate erreichte und für 80 Sprecher wurde das Resultat mit einer MR von 8.75% unterboten. In Abbildung 4.6 ist das Dendrogramm für das beste Netzwerk aus Experiment 6 zu sehen. Verglichen mit dem Dendrogramm in Abbildung 4.3, zeigte sich, dass wieder die Sprecherin FJEM0 die grösste Distanz im Cluster aufwies. Dieselbe

Sprecherin wies auch schon bei den CNN Ergebnissen [9] die grösste Distanz zwischen den Aussagen auf. Bei anderen Sprechern war kein Muster auszumachen. Da starke Varianzen im Training auftraten, wurde auch noch das Dendrogramm des zweiten Testlaufs mit Margin 3 analysiert. Selbst bei komplett gleichen Netzwerken unterschieden sich die Distanzen zwischen den Aussagen der Sprecher zwischen den Testläufen.

Das Training dieser Netzwerke dauerte ca. 5 Stunden für 10 000 Iterationen. Dies war bedeutend schneller, verglichen mit den Netzwerken aus der Arbeit von Lukic und Vogt [9], welche für die gleiche Anzahl Iterationen 24 Stunden benötigten und deren beste Resultate erst nach 30 000 Iterationen erreichten.

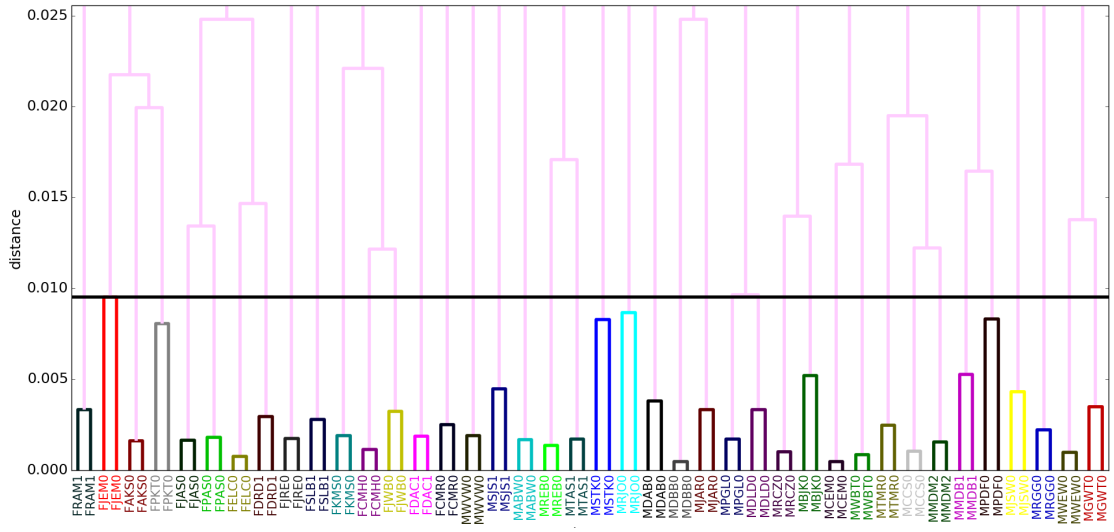


Abbildung 4.6.: Dendrogram mit 40 Sprecher, für das Netzwerk mit einem Margin von 3, “F”=weiblich / “M”=Männlich

5. Experimente CNN-GRU

Die folgenden Experimente zeigen Untersuchungen des CNN-GRU Netzwerks, das in Abschnitt 3.2 beschrieben wurde. Mit der beschriebenen Architektur wurden die Fehler der Implementation der Projektarbeit von Gerber [1] behoben und daher erwartet, dass das Netzwerk gute Resultate für das Sprecherclustering liefert. Als Optimierungs Funktion kam der Adam Optimizer und als Loss Funktion die Cross-Entropy, sowie die PKLD zum Einsatz.

5.1. Lernfähigkeit

Um eine Basis zu schaffen, sollte zuerst festgestellt werden, ob das Netzwerk aus Abschnitt 3.2 überhaut lernfähig war. Dazu wurde das Netzwerk mit der in Tabelle 5.1 angegebenen Konfiguration verwendet. Der GRU Layer bestand aus 128 Units. Das Netzwerk wurde mit dem Trainingsdatensatz von 100 Sprechern trainiert und kein Clustering Testdatensatz verwendet, da es nur darum ging, Festzustellen, ob das Netzwerk überhaupt etwas lernt. Die Erwartung war, dass das Netzwerk trainiert, um weitere Experimente mit dem Netzwerk durchgeführt zu können.

	CNN/Max-Pooling Layer 1	**CNN/Max-Pooling Layer 2**
Filter	32	64
Kernel	8×1	8×1
Pool	4×4	4×4
Stride	2×2	2×2

Tabelle 5.1.: Experiment-Aufbau Lernfähigkeit

Auswertung

Wie in Abbildung 5.1 zu sehen ist, nahm der Trainings Loss als auch der Validation Loss ab. Daraus wurde geschlossen, dass die Architektur lernfähig und somit weitere Experimente sinnvoll waren.

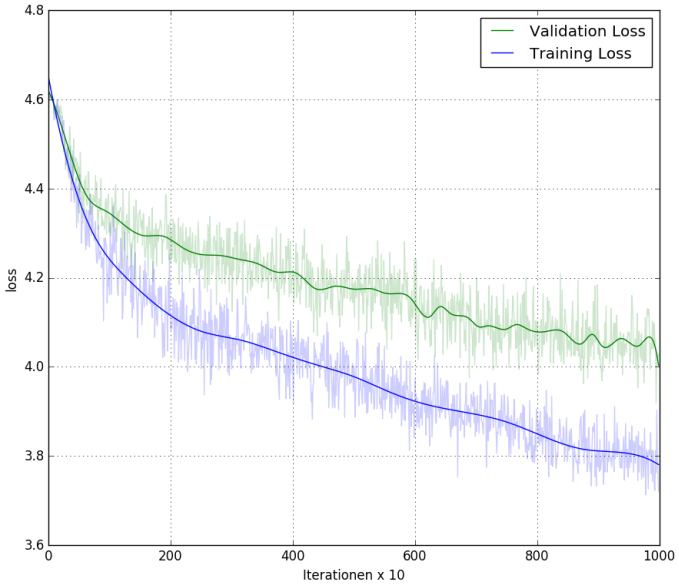


Abbildung 5.1.: Lernfähigkeit CNN-GRU Netzwerk

5.2. Parameter der CNN und Max-Pooling Layer

Nachdem sich zeigte, dass die CNN-GRU Kombination lernhfähig war, sollte nun untersucht werden, ob die Parameter der CNN und Max-Pooling Layer gegenüber der übernommenen Parameter aus Projektarbeit von Gerber [1] optimiert werden konnten.

Inspiriert durch die Arbeit von Harutyunyan and Khachatrian [8], wurden in diesem Experiment auch Versuche mit vier CNN/Max-Pooling Layern durchgeführt, welche in Tabelle 5.3 aufgeführt sind.

In der Bachelorarbeit von Lukic und Vogt [9] wurde gezeigt, dass Convolution in Zeitrichtung nicht sinnvoll ist. Dies machte auch für die hier durchgeführten Experimente Sinn, da der zeitliche Zusammenhang nach der Convolution noch vorhanden sein sollte, damit der GRU Layer über die zeitliche Komponenten Lernen konnte. Daher wurde, bis auf einzelne Ausnahmen, der Fokus dieser Untersuchungen auf folgende Werte in Frequenzrichtung gelegt, wobei auch das Pooling in den meisten Experimenten nur in Frequenzrichtung war:

#	**CNN/Max-Pooling Layer 1**				**CNN/Max-Pooling Layer 2**			
	Filter	*Kernel*	*Pool*	*Stride*	*Filter*	*Kernel*	*Pool*	*Stride*
exp_0	32	4×1	2×2	2×1	64	6×1	3×3	3×1
exp_1	32	7×7	3×3	2×2	64	5×5	3×3	2×2
exp_2	32	8×1	4×1	2×1	64	6×1	3×1	2×1
exp_3	32	8×1	4×4	2×1	64	6×1	3×3	2×1
exp_4	32	8×1	4×1	2×1	64	8×1	4×1	2×1
exp_5	32	8×8	4×4	2×2	64	8×8	4×4	2×2

Tabelle 5.2.: Experiment-Aufbau 2 CNN/Max-Pooling-Parameter

Die Parameter der Experimente *exp_0* und *exp_1* in Tabelle 5.2, sowie derer in Tabelle 5.3 orientierten sich an einer CNN-GRU Kombination aus der Arbeit von Harutyunyan and Khachatrian [8].

#	**CNN/Max-Pooling Layer 1**				**CNN/Max-Pooling Layer 2**			
	Filter	*Kernel*	*Pool*	*Stride*	*Filter*	*Kernel*	*Pool*	*Stride*
exp_6	16	7×7	3×3	2×1	32	5×5	3×3	2×1
exp_7	32	8×1	4×4	2×1	64	6×1	3×3	2×1
exp_8	64	16×1	4×1	4×1	64	8×1	4×1	2×1
#	**CNN/Max-Pooling Layer 3**				**CNN/Max-Pooling Layer 4**			
	Filter	*Kernel*	*Pool*	*Stride*	*Filter*	*Kernel*	*Pool*	*Stride*
exp_6	32	3×3	3×3	2×1	32	3×3	3×3	2×1
exp_7	64	4×1	2×2	2×1	128	2×1	2×2	2×1
exp_8	128	4×1	2×1	2×1	256	4×1	2×1	2×1

Tabelle 5.3.: Experiment-Aufbau 4 CNN/Max-Pooling-Parameter

Das Netzwerk wurde mit dem Trainingsdatensatz von 100 Sprechern trainiert. Zur Überprüfung des Clusterings kam derselbe Testdatensatz zum Einsatz wie auch schon bei den LSTM Experimenten.

Alle Netzwerke besassen einen GRU Layer mit 256 Units. Diese Anzahl Units hat sich in den Experimenten von Harutyunyan and Khachatrian [8] bewährt und wurde deshalb als guter Startpunkt für die Experimente angesehen.

Es wurde erwartet, dass die Netzwerke, welche sowohl eine Convolution als auch ein Pooling nur in Frequenzrichtung besassen, die besten Resultate liefern.

Auswertung

Wie in Tabelle 5.4 zu sehen ist, erreichte die Konfiguration *exp_3* die niedrigste MR und somit das beste Resultat hinsichtlich des Clusterings. Wie erwartet, war dies eine Konfiguration, in der die Convolution nur in Frequenzrichtung durchgeführt wurde. Der Pooling Kernel deckte zwar mit der Dimension 4×4, respektive 3×3, auch die Zeitrichtung ab, jedoch dadurch, dass sich der Kernel nur in Frequenzrichtung verschoben hat, wurde die zeitliche Komponente nicht zerstört, sondern nur komprimiert. Es zeigte sich ebenfalls, dass die beiden Experimente *exp_2* und *exp_5*, welche sowohl Convolution als auch Pooling auf beide Dimensionen durchführten, die schlechtesten Resultate der Netzwerke mit 2 Convolutional Layern erreichten.

Die Netzwerke mit 4 Convolutional Layern erreichten allesamt eine schlechte MR und wurden deshalb nicht weiterverfolgt. Es schien, als hätte die zusätzliche Tiefe des Netzwerkes keine Verbesserung gebracht und die Performanz sogar verschlechtert.

Konfiguration	**MR**
exp_0	0.125
exp_1	0.200
exp_2	0.175
exp_3	**0.100**
exp_4	0.200
exp_5	0.300
exp_6	0.225
exp_7	0.275
exp_8	0.300

Tabelle 5.4.: Resultate CNN/Max-Pooling Parameter

Aufgrund dieser Resultate wurde entschieden, dass die Parameter-Konfiguration *exp_3* auf die CNN/Max-Pooling Layer in den folgenden Experimenten weiterhin angewendet werden soll.

5.3. Anzahl GRU Neuronen

Nachdem eine gute Grundlage durch die beiden vorhergehenden Versuche geschaffen wurde, sollte in diesem Experiment untersucht werden, wie sich verschiedene Werte für die Anzahl Neuronen im GRU-Layer auf die Cluster Performanz des Netzwerks auswirken.

Es kam das Netzwerk *exp_03* aus dem vorherigen Experiment zum Einsatz, wobei die Anzahl Units im GRU Layer auf die Werte 64, 128, 256 und 512 gesetzt wurden. Dies selben Test- und Trainingsdaten aus dem vorherigen Experiment wurden wieder eingesetzt.

Da der Wert von 256 Units bereits in der CNN-GRU Kombination [8], sowie auch im BLSTM aus Abschnitt 4 erfolgreich verwendet wurde, war die Erwartung, dass sich diese Anzahl Units auch in diesem Experiment durchsetzt.

Auswertung

Die Resultate in Tabelle 5.5 zeigen, dass sich, wie erwartet, das Netzwerk mit 256 Units im GRU Layer durchgesetzt hat.

Anz. Neuronen	MR
64	0.150
128	0.300
256	0.100
512	0.275

Tabelle 5.5.: Resultate Anzahl GRU Neuronen

Überraschendweise war das zweit-beste Netzwerk, dasjenige mit nur 64 Units. Es wäre zu erwarten gewesen, dass die anderen Netzwerke eine besseres Resultat liefern. Jedoch konnte keine Erklärung für dieses Resultat gefunden werden.

5.4. Zusätzliche Dense Layer

In der Arbeit von Lukic und Vogt [9] wurde festgestellt, dass Dense Layer zwischen dem Output Layer und dem Embedding Layer, das Clustering verbessern.

Es sollte nun geklärt werden, ob der Einsatz eines oder mehrerer Dense Layer für die CNN-GRU Kombination von Nutzen ist. Dazu wurden Testäufe, mit einem und zwei Dense Layer zwischen dem GRU Layer und dem Output Layer, durchgeführt. Die verwendeten Konfigurationen sind in Tabelle 5.5) ersichtlich. Die Entscheidung den zweiten Dense Layer grösser zu wählen kam daher, dass diese Konfiguration in den BLSTM Experimenten gut funktioniert hat.

#	Dense 1	Dense 2
den_0	500	-
den_1	500	1000
den_2	200	-
den_3	200	250

Tabelle 5.6.: Experiment Dense Layer Units

Es wurde erwartet, dass die zusätzlichen Dense Layer, aufgrund der beschriebenen Eigenschaften, das Clustering auf dem GRU Layer verbessern.

Auswertung

Wider den Erwartungen zeigte sich eine bedeutende Verschlechterung bei *den_1* und *den_1*. Was die Ursache für solch einen drastischen Einbruch der MR bei diesen beiden Läufen war, lies sich zum Zeitpunkt dieser Arbeit leider nicht klären. Es wurde vermutet, dass die zusätzliche Komplexität das Lernen unmöglich machte, denn weder der Validation Loss, noch der Trainings Loss sind während des Trainings gesunken. Diese Vermutung wurde dadurch gestützt, dass die Netzwerke mit nur einem zusätzlichen Dense Layer bessere MR Werte errichten, wie in Tabelle 5.7 ersichtlich ist. Der Testlauf *den_2* erreichte mit einer MR von 0.05 ein sehr gutes Resultat, dass bis zum damaligen Zeitpunkt nur durch die BLSTM Experimente unterboten wurde.

#	MR
den_0	0.250
den_1	0.950
den_2	**0.050**
den_3	0.925

Tabelle 5.7.: Resultate Anzahl Dense Layer Units

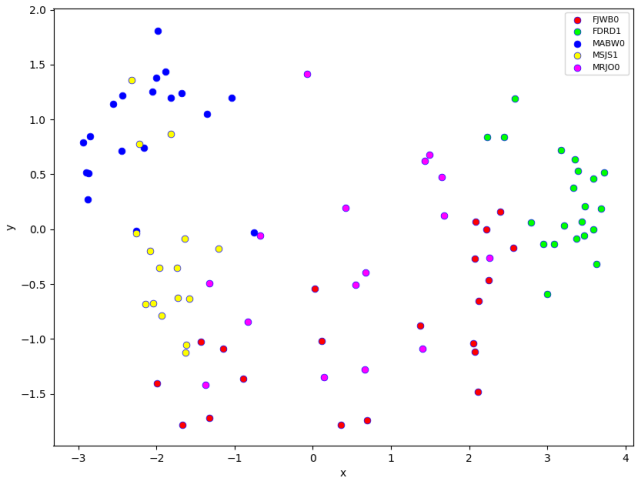


Abbildung 5.2.: Evaluation des Clustering von 5 Sprechern des Netzwerkes *den_2*

Obwohl nach dem Trainingsvorgang die Evaluation mit Sätzen von 40 Sprechern durch das *den_2* Model eine vielversprechende MR erreichte, zeigte die grafische Auswertung in Abbildung 5.2 des Clusterings mit 5 Sprechern durch das Model ein schlechtes Ergebnis. Es war auch nicht möglich den Testlauf zu reproduzieren. Alle anderen Tesläufe erreichten schlechtere Resultate.

5.5. Training mit PKLD Loss

Das beste Netzwerk aus dem letzten Experiment sollte nun noch mit der PKLD Loss Funktion trainiert werden, da ihr Einsatz in den BLSTM Experimenten (vgl. Abschnit 5.4) zu einer Verbesserung der MR führte.

Das Training wurde mit dem gleichen Datensatz durchgeführt wie in den anderen Experimenten. Das Netzewerk wurde für 8000 Iterationen trainiert. Das Clustering war primär auf dem Testdatensatz mit 40 Sprechern evaluiert.

Es wurde erwartet, dass die PKLD eine Verbesserung des Clusterings bringt.

Auswertung

Der Einsatz der PKLD Loss-Funktion brachte nicht die erhofften Verbesserung für das Clustering. Da die MR für 40 Sprecher 0.075 betrug, konnte das Resultat von 0.05 aus dem vorherigen Experiment nicht verbessert werden. Das Training des Netzwerkes für 8000 Iterationen dauerte ca. 4 Tage. Zudem benötigte das Netzwerk mit längerer Trainingsdauer immer mehr Zeit pro Iteration, wodurch die ersten 10 Iterationen am Start des Trainings ca. 250 Sekunden benötigten, jedoch die letzen 10 Iteratonen bei 594 Sekunden lagen. Der Grund für das immer langsamer werdende Training konnte nicht gefunden werden. Zwei weitere Trainingsversuche stürzten nach 5000 respektive 6000 Iterationen mit einem Out-of-Memory Fehler ab.

Netzwerk	**MR**	
	Run 1	**Run 2**
PKLD	0.075	-
PKLD/Cross-Ent. (Run 1)	0.200	0.225

Tabelle 5.8.: MR der PKLD Netzwerke

Aufgrund der langen Trainingsdauer wurde entschieden, das Netzwerk zuerst mit 8000 Iterationen mit der Cross-Entropy Funktion und anschliessend mit 2000 Iterationen mit der PKLD zu trainieren. Dadurch konnte die Trainingszeit auf 24 Stunden verringert werden. Wie in Tabelle 5.8 zu sehen ist, verschlechterte dieser Ansatz die Resultate enorm. Es wurde angenommen, dass das vorgängige Trainieren mit der Cross-Entropy einen negativen Einfluss auf das Clustering hatte.

5.6. Fazit

Die Experimente mit den CNN-GRU Netzwerken erbrachten nicht die beabsichtigten Ergebnisse. Es wurde erwartet, dass die Kombination der CNN Layer, welche gut zur Extraktion von Featues für das Sprecherclustering geeignet sind [9], mit dem GRU Layer, der die zeitlichen Zusammenhänge erkennen kann, gute Resultate lieferte. Entgegen dieser Erwartungen lieferten alle getesteten Netzwerkkonfigurationen eine schlechtere MR verglichen mit den BLSTM Netzwerken aus Kapitel 4. Die beste MR von 0.05 auf 40 Sprechern war zwar vielversprechend, allerdings zeigte die Visualisierung der Embeddings von 5 Sprechern in Abbildung 5.2 kaum eine Struktur, was auf ein vom Zufall behaftetes Ergebnis schliessen lies, da es ebenfalls nicht reproduzierbar war.

Dennoch konnte gezeigt werden, dass eine Kombination von Convolution und Pooling in Frequenzrichtung die besten Resultate lieferte. Zudem wurde festgestellt, dass zusätzliche Convolution Layer die Performance weiter verschlechtert hatten.

Ein Problem bei der Implementierung der Netzwerke könnte die Verbindung zwischen CNN und GRU gewesen sein, da die Input Vektoren für den GRU Layer sehr gross und dadurch für das beste Netzwerk, welches die MR von 0.5 erreichte, eine Länge von 1984 hatten.

Das Training mit der PKLD brachte im Gegensatz zu den Experimenten Kapitel 4 und der Arbeit von Lukic und Vogt [9] keine Verbesserung. Zudem dauerte das Training für 10 Iterationen zwischen 250 und 600 Sekunden, was um den Faktor 50 länger war als das Training mit der Cross-Entropy Funktion. Daher konnten aus Zeitmangel nur wenige Experimente damit durchgeführt werden.

6. Fazit und Ausblick

6.1. Fazit

Wie in den Experimenten mit dem BLSTM demonstriert werden konnte, eignen sich diese Neural Networks sehr gut für das Sprecherclustering. So wurde im Experiment aus Abschnitt 4.6, durch erhöhen des Margin Parameters der PKLD, auf dem Testdatensatz mit 40 Sprechern eine MR von 0 erreicht. Dies bedeutet, dass alle Segmente den richtigen Sprechern zugeordnet werden konnten. Dies unterbot den bisherigen Bestwert von 0.05 auf 40 Sprechern, welcher in der Arbeit von Lukic et al. [5] erreicht wurde. Auf 60 und 80 Sprecher wurde eine MR von 0.05 und 0.0875 erreicht, dies unterbot die Resultate für 80 Sprecher von 0.1375 um in Lukic et al. [5] deutlich und erreichte das gleiche Resultat für 60 Sprecher. Wie die Experimente dieser Arbeit zeigten, hatten andere Neural Network Konfigurationen noch bessere MRs auf 60 Sprecher erreicht, allerdings ohne ein fehlerfreies Clustering auf einem Datensatz mit 40 Sprechern.

Das CNN-GRU Netzwerk konnte die Erwartungen, eine mindestens so niedrige MR und somit ein mindestens so gutes Clustering wie das BLSTM zu erreichen, nicht erfüllen und schnitt, bis auf einen Testlauf, bedeutend schlechter ab, als ursprünglich vermutet. Das beste Ergebnis des CNN-GRU Netzwerks war in Abschnitt 5.4 mit einer MR von 0.05 auf 40 Sprechern. Allerdings zeigte die visuelle Darstellung in Abbildung 5.2 zu diesem Ergebnis ein bedeutend schlechteres Clustering als das Beste des BLSTM Netzwerks (vlg. Abbildung 4.5). Auch der zusätzliche Einsatz der PKLD lieferte keine besseren Resultate. Da die Trainingsdauer für 10 Mini-Batches 300s - 600s betrug, konnten nur wenige Testläufe durchgeführt werden.

Der Fokus auf 2 verschiedene Netzwerkarchitekturen hat sich nachteilig auf die verfügbare Zeit für die Evaluation des CNN-GRU Netzwerks ausgewirkt, welches eine grosse Anzahl Hyperparameter hatte und deshalb eine detailliertere Untersuchung sinnvoll gewesen wäre.

6.2. Ausblick

Weitere Analyse der BLSTM Architektur

Die BLSTM Architektur lieferte sehr gute Resultate. Allerdings trat während den Versuchen schon sehr früh ein Overfitting ein. Experimente, welche die Auswirkungen eines Batchnormalization Layers untersuchen, scheinen hier sinnvoll zu sein, da so dem Overfitting entgegengewirkt werden kann.

Weitere Analyse der CNN-RNN Architektur

Die Resultate der CNN-RNN Architektur waren im Vergleich zu denjenigen des BLSTM mit Dense Layer schlechter. Dennoch ist intuitiv die Kombination der beiden Netzwerke sinnvoll, da die CNN Layer relevante Features aus den Input Daten extrahieren und die RNN Layer danach die zeitlichen Zusammenhänge lernen können. Eine mögliche Schwäche der Implementation, welche für diese Arbeit verwendet wurde, ist die Grösse der Feature Vektoren mit einer Länge von 1984, die dem GRU Layer als Input diente. Ein möglicher Lösungsansatz den zu untersuchen sich lohnen würde, ist, einen zusätzlichen Layer einzufügen, der diesen Feature Vektor verkleinert, wie es in [7] beschrieben wurde.

Verwendung von Recurrent Highway Networks

Eine neue Entwicklung im Bereich der RNNs sind die Recurrent Highway Networks (RHN) Zilly et al. [40]. RHNs sind Flexibler hinsichtliche der Teile des Inputs, die sie transformieren oder an den nächsten Layer weiterleiten. Dies führt dazu, dass sie bessere Resultate als herkömmliche Stacked RNNs liefern. Ein solches Netzwerk könnte anstatt der in dieser Arbeit eingesetzten BLSTM Architektur verwendet werden.

Loss Funktion

Wie in den Experimenten gezeigt wurde, ist die Differenz der Resultate zwischen Cross-Entropy Loss und PKLD relativ gering. Daher wäre es sinnvoll, andere Loss Funktionen zu testen. In [41] wird eine Loss Funktion beschrieben, die ähnlich wie bei der PKLD die Distanz zwischen den Output Vektoren verwendet. Allerdings werden hier 3 Segmente eingesetzt, von denen eines aus der vorherigen Iteration kommt und als Anker dient.

Verwendung von Audiodaten

Bei allen bisherigen Arbeiten zum Thema Sprecherclustering wurden Mel-Spektrogramme als Input verwendet. Neural Networks sind fähig, relevante Features eigenständig zu lernen und die notwendigen Transformationen durchzuführen. Es wäre daher interessant, reine Audiodaten wie zum Beispiel WAV-Dateien als Input für die Neural Networks einzusetzen. Dabei scheint es sinnvoller, dies auf die reine LSTM Architektur anzuwenden. Da CNNs stärker von Spektrogrammen, welche als Bilder aufgefasst werden könne, profitieren sollten.

Datenmenge

In den in dieser Arbeit durchgeführten Experimente wurde der TIMIT Datensatz verwendet, mit 630 Sprechern mit jeweils 10 Sätzen von 3 bis 8 Sekunden relativ klein ist. Ein möglicher Ansatz wäre, ein Netzwerk auf einem grösseren Datensatz zu trainieren. Dies würde bedeuten, sowohl einen Datensatz mit mehr Sprechern als auch längeren Sätzen respektive mehr Daten pro Sprecher zu verwenden.

Datenqualität

Der TIMIT Datensatz beinhaltet Daten in Studioqualität. Dies eliminiert viele Variablen wie Hintergrundgeräusche und andere Einflussfaktoren einschliesslich qualitativ unterschiedliche Mikrofone und Lautstärkeschwankungen zwischen den einzelnen Segmenten. Es wäre nun interessant zu sehen, ob die Netzwerke auch mit imperfekten Daten umgehen können und vergleichbar gute Resultate liefern. Insbesondere wäre es interessant zu untersuchen, ob CNN-RNN Kombinationen in einem solchen Szenario bessere Resultate, als die in dieser Arbeit verwendete, liefern würden, da CNNs laut den Erkenntnissen von Pascanu et al. [31] Störfaktoren ausgleichen können.

7. Verzeichnisse

Literaturverzeichnis

[1] P. Gerber, "Projektarbeit HS16Speaker Identification mit Reccurent Neural Networks," 2016.

[2] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," *arXiv preprint arXiv:1402.1128*, no. Cd, 2014.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[4] J. Ren, Y. Hu, Y.-W. Tai, C. Wang, L. Xu, W. Sun, and Q. Yan, "Look, Listen and Learn - A Multimodal LSTM for Speaker Identification," in *Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016)*. AAAI, 2016, pp. 3581–3587.

[5] Y. Lukic, C. Vogt, O. Dürr, and T. Stadelmann, "Learning Embeddings for Speaker Clustering based on Voice Equality," *unpublished*, 2017.

[6] T. Stadelmann and B. Freisleben, "Unfolding Speaker Clustering Potential : A Biomimetic Approach," in *Proceedings of the 17th ACM international conference on Multimedia*. ACM, 2009, pp. 185–194.

[7] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4580–4584.

[8] H. Harutyunyan and H. Khachatrian. (2016). [Online]. Available: https://yerevann.github.io/2016/06/26/combining-cnn-and-rnn-for-spoken-language-identification/{#}combinations-of-cnn-and-rnn

[9] Y. Lukic and C. Vogt, "Bachelorarbeit FS16 Studiengang Informatik Automatische Stimmerkennung mit Deep Learning," 2016.

[10] Y. Lukic, C. Vogt, O. Dürr, and T. Stadelmann, "Speaker Identification and Clustering Using Convolutional Neural Networks," in *Proceedings of IEEE International Workshop on Machine Learning for Signal Processing*. Salerno: IEEE, 2016, pp. 13–16.

[11] M. Kotti, V. Moschou, and K. Constantine, "Speaker segmentation and clustering in meetings," *arXiv preprint arXiv:1511.05939*, 2007.

[12] J. P. Campbell, "Speaker recognition: A tutorial," *Proceedings of the IEEE*, vol. 85, no. 9, pp. 1437–1462, 1997.

[13] H. Beigi, *Fundamentals of Speaker Recognition*. New York, Dordrecht, Heidelberg, London: Springer, 2011.

[14] S. Zhang, W. Hu, T. Wang, J. Liu, and Y. Zhang, *Speaker Clustering Aided by Visual Dialogue Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 693–702.

[15] K. J. Han, S. Kim, and S. S. Narayanan, "Strategies to improve the robustness of agglomerative hierarchical clustering under data source variation for speaker diarization," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 16, no. 8, pp. 1590–1601, 2008.

[16] D. A. Reynolds and P. Torres-Carrasquillo, "Approaches and applications of audio diarization," in *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, vol. 5. IEEE, 2005, pp. 953–956.

[17] T. Stadelmann, "Voice Modeling Methods for Automatic Speaker Recognition," Ph.D. dissertation, Philipps-Universität Marburg, 2010.

[18] S. S. Stevens, J. Volkmann, and E. B. Newman, "A Scale for the Measurement of the Psychological Magnitude Pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[19] D. O'Shaughnessy, *Speech Communications: Human and Machine*. Wiley-IEEE Press, 1999, vol. 2 edition.

[20] K. Gurney, *Introduction to neural networks*. London: UCL Press Limited, 1996, vol. 6, no. 2.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge: The MIT Press, 2016.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980 [cs.LG]*, pp. 1–15, 2014.

[23] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014.

[24] Y. LeCun, Y. Bengio, G. Hinton, L. Y., B. Y., and H. G., "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[25] C. Olah. (2015). [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[26] Y. Bengio, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[27] S. Hochreiter, , and J. Schmidhuber, "Long short-term memory." *Neural computation*, vol. 9, no. 8, pp. 1735–80, 1997.

[28] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv preprint arXiv:1406.1078*, 2014.

[29] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[30] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[31] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to Construct Deep Recurrent Neural Networks," *arXiv preprint arXiv:1312.6026*, pp. 1–13, 2014.

[32] M. Hermans and B. Schrauwen, "Training and Analyzing Deep Recurrent Neural Networks," in *Advances in neural information processing systems*, 2013, pp. 190–198.

[33] M. Rouvier, P.-M. Bousquet, and B. Favre, "Speaker diarization through speaker embeddings," in *Signal Processing Conference (EUSIPCO), 2015 23rd European*. IEEE, 2015, pp. 2082–2086.

[34] Y.-C. Hsu and Z. Kira, "Neural network-based clustering using pairwise constraints," *arXiv:1511.06321*, pp. 1–12, 2015.

[35] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[36] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallet, N. L. Dahlgren, and Z. Victor, "TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1," 1993.

[37] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, pp. 1–12, 2016.

[38] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[39] Y. LeCun, C. Cortes, and C. Burges, "The MNIST Database of Handwritten Digits," 1998.

[40] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent Highway Networks," *arXiv preprint arXiv:1607.03474*, 2016.

[41] O. Rippel, M. Paluri, P. Dollar, and L. Bourdev, "Metric Learning with Adaptive Density Discrimination," *arXiv preprint arXiv:1511.05939*, no. Dml, pp. 1–15, 2015.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

A. Anhang

A.1. Aufgabenstellung

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Machine Learning for Speaker Clustering
BA17_stdm_1

BetreuerInnen:	Thilo Stadelmann, stdm Oliver Dürr, dueo
Fachgebiete:	Bildverarbeitung (BV) Datenanalyse (DA) Software (SOW)
Studiengang:	IT / WI
Zuordnung:	Institut für angewandte Informationstechnologie (InIT)
Interne Partner:	Institut für Datenanalyse und Prozessdesign (IDP)
Gruppengrösse:	2

Kurzbeschreibung:
Automatische Stimmerkennung ist eine wichtige Basistechnologie in verschiedenen kommerziellen Bereichen (z.B. biometrische Authentifizierung, Gesprächsanalyse für Teledienste, Medienmonitoring). Trotzdem liefern automatische Verfahren deutlich schlechtere Ergebnisse als menschliches Hören. Forschung der Betreuer hat kürzlich einen Weg gezeigt, diese Lücke zu schliessen: Mittels Verfahren des maschinellen Lernens, wie sie in der Analyse von Bildern zum Einsatz kommen, lässt sich der zeitliche Verlauf von Sprache (der "Klang") extrahieren und in einem für die Stimme charakteristischen Modell abspeichern. Insbesondere Deep Convolutional oder Recurrent Neural Networks (CNNs / RNNs) bieten sich hier an, die in den letzten Jahren zu einem Milliardengeschäft und heissesten Eisen in der Machine Learning Forschung geworden sind.

Ziel

Ziel dieser Bachelorarbeit ist es, "Speaker Clustering" weiterzuentwickeln. Hierzu ist ein geeigneter Ansatz aus der Literatur gemeinsam mit den Betreuenden auszuwählen, zu implementieren und experimentell mit dem wissenschaftlichen State of the Art zu vergleichen.

Vorgehen

- Kennenlernen des Hintergrunds von Speaker Clustering
- Einarbeiten in die relevante Literatur; einen Einblick liefert Rippel et al., "Metric Learning with Adaptive Density Discrimination", 2016
- Implementieren eines geeigneten Ansatzes (Auswahl zusammen mit Betreuern)
- Geeignetes Wählen von Einstellungen anhand der Leistung auf gegebenen Testdaten
- Evaluieren anhand systematischer Experimente und Vergleich mit existierendem Baseline Ansatz
- Darstellung und Diskussion der eigenen Ergebnisse im Licht des Standes der Forschung (BA-Bericht, Vortrag)

Voraussetzungen:
Die Arbeit setzt kein Vorwissen im Bereich Machine Learning oder Stimmerkennung voraus. Freude am Programmieren und Experimentieren sowie Lust auf ein Auseinandersetzen mit einer wissenschaftlichen Fragestellung sind hingegen wichtig. So kann die Arbeit als Einstieg in forschungsnahes Arbeiten genutzt werden.

In vorangegangenen Projekt- und Bachelorarbeiten wurden bereits Ansätze positiv evaluiert und Ergebnisse international durch Studierende präsentiert. Eine experimentelle Umgebung für Stimmerkennung mit CNNs steht zur Verfügung, in der bekannte Datensätze geladen, analysiert und Ergebnisse ausgewertet werden können.

Die Arbeit ist vereinbart mit:
Patrick Gerber (gerbepat)
Sebastian Glinski (glinsseb)

Sonntag 4. Juni 2017 23:3

A.2. Weiteres

A.2.1. Elektronische Daten

Die der Arbeit, auf einer CD, beiliegenden Elektronischen Daten haben die folgende Form.

- **01_Dokumentation:** Enthält die Bachelorarbeit im PDF Format
- **02_Code:** Enthält den Python Code und die Verzeichnissstruktur um alle Funktionen des Codes Verwenden zu können.
- **03_Experimente:** Enthält eine Auswahl der Besten Netzwerke. sowie deren Logs.

A.2.2. Implementation Paarweise KL-Divergenz

Da bis anhin noch keine Implementationen der Paarweise KL-Divergenz für Tensorflow oder Keras Existieren und die Implementation aus der Arbeit von Lukic und Vogt [9] auf Theano basierte, wurde diese in Tensorflow implementiert. Tensorflow bietet die Möglichkeit an While Loops mit Hilfe der Funktion while_loop zu implementieren. Die Funktion nimmt folgende Parameter entgegen:

- **cond:**Funktion welche die Abbruchbedingung des Loops bestimmt. Die Funktion nimmt genau die Anzahl Parameter Entgegen welche dem while_loop im Parameter loop_vars übergeben wird.
- **Body:** Funktion welche die Anweisungen im Loop ausführt. Die Funktion nimmt genau die Anzahl Parameter entgegen welche dem while_loop im Parameter loop_vars übergeben wird.
- **Loop_vars:** Tupel von Variablen welche im Loop verarbeitet werden.
- **parallel_iterations:** Anzahl Iterationen welche parallel ausgeführt werden.
- **Back_prop:** Boolean welcher bestimmt ob Backpropagation für diesen Loop ermöglicht wird. Für die Berechnung des Gradients muss der Wert auf True gesetzt werden.
- **Swap_memory:** Boolean welcher darüber entscheidet ob der Speicher vom GPU auf den Hauptspeicher geschrieben werden kann. Dies kann notwendig werden falls der GPU nicht über genügend Speicher verfügt.

Der Folgende Code zeigt die Implementation der Loops.

```
1  def outerLoop(x, tf_l, predictions, labels, margin):
2      with tf.device('/cpu:0'):
3          def innerLoop(y, x, tf_l, predictions, labels, margin):
4              tf_l = tf.add(tf_l, tf.cond(tf.greater(y, x),lambda:
                   loss_with_kl_div(predictions[x], labels[x],
                   predictions[y], labels[y], margin) , return_zero))
5              y = tf.add(y, tf.constant(1))
6              return y, x, tf_l, predictions, labels, margin
7
8          def innerLoop_cond(y, x, tf_l, predictions, labels, margin):
9              return tf.less(y, tf.shape(predictions)[0])
10
11         y = tf.constant(0)
12         res = tf.while_loop(innerLoop_cond, innerLoop, [y, x, tf_l,
               predictions, labels, margin], parallel_iterations = 10 ,
               name='innerloop')
13         return tf.add(x, 1), res[2], predictions, labels, margin
14
15 def outerLoop_condition(x, tf_l, predictions, labels, margin):
16     return tf.less(x, tf.shape(predictions)[0])
17
18 def pairwise_kl_divergence(labels, predictions):
```

```
19      with tf.device('/cpu:0'):
20          x = tf.constant(0)
21          sum_loss = tf.while_loop(outerLoop_condition, outerLoop, [x,
                tf_l, predictions, labels, margin], parallel_iterations =
                10 , name='outerloop')
22          n = tf.to_float(tf.shape(predictions)[0])
23          pairs = tf.multiply(n, tf.divide(tf.subtract(n, tf.constant
                (1.)), tf.constant(2.)))
24          loss = tf.divide(sum_loss[1], pairs)
25          return loss
```

Listing A.1: Implementation der PKLD in TensorFlow

Es werden 2 Loops verwendet. Beiden Loops werden alle Predictions und alle Labels übergeben. im ersten wird über die Predictions, also den Outputs des Netzwerkes Iteriert. Im 2. Loop wird nochmals über alle Predictions mit einem Index grösser als die Aktuelle Prediction iteriert. Und mit den entsprechenden Predictions und Labels die KLD berechnet. Es hat sich nach mehreren Versuchen gezeigt, dass eine Erhöhung des parrallel_iterations Parameter auf über 10 die Performance der Funktion wieder verschlechtert. Daher kann hier auch der Standardwert verwendet werden. Es wird empfohlen den swap_memory Parameter auf True zu setzen da ansonsten Gefahr gelaufen wird, dass es zu einem out of Memory Fehler kommen kann. Es konnte keine Performance Verbesserung festgestellt werden, wenn der Parameter auf False gelassen wird.

A.2.3. Sprecherlisten

Trainingsdatensatz 100 Sprecher

FCJF0	FDAS1	FCKE0	MKLS0
FDAW0	FDNC0	FCMG0	MKLW0
FDML0	FDXW0	FDFB0	MMGG0
FECD0	FEAC0	FDJH0	MMRP0
FETB0	FHLM0	FEME0	MPGH0
FJSP0	FJKL0	FGCS0	MPGR0
FKFB0	FKAA0	FGRW0	MPSW0
FMEM0	FLMA0	FJLG0	MRAI0
FSAH0	FLMC0	FJLR0	MRCG0
FSJK1	FMJB0	FLAC0	MRDD0
FSMA0	FMKF0	FLJD0	MRSO0
FTBR0	FMMH0	FLTM0	MRWS0
FVFB0	FPJF0	MCPM0	MTJS0
FVMH0	FRLL0	MDAC0	MTPF0
FAEM0	FSCN0	MDPK0	MTRR0
FAJW0	FSKL0	MEDR0	MWAD0
FCAJ0	FSRH0	MGRL0	MWAR0
FCMM0	FTMG0	MJEB1	MARC0
FCYL0	FALK0	MJWT0	MBJV0

MCEW0	MDMT0	MJAE0	MJMD0
MCTM0	MDPS0	MJBG0	MJPM0
MDBP0	MDSS0	MJDE0	MJRP0
MDEM0	MDWD0	MJEB0	MKAH0
MDLB0	MEFG0	MJHI0	MKAJ0
MDLC2	MHRM0	MJMA0	MKDT0

Testdatensatz mit 40 Sprecher für Clustering

MPGL0	MGWT0	MTAS1	FCMH0
MSTK0	FDAC1	MSJS1	MRJO0
FCMR0	FPKT0	FJEM0	FPAS0
MWBT0	FJAS0	MWEW0	FJWB0
MCEM0	MCCS0	MREB0	FDRD1
MRGG0	MMDB1	FJRE0	MJSW0
MBJK0	MDLD0	MWVW0	FKMS0
MPDF0	MJAR0	FRAM1	FSLB1
MDAB0	FAKS0	MRCZ0	MTMR0
MMDM2	FELC0	MABW0	MDBB0

Testdatensatz mit 60 Sprechern

FAKS0	FJWB0	MPGL0	MGLB0
FDAC1	FPAS0	MRCZ0	MHPG0
FELC0	FRAM1	MRGG0	MJBR0
FJEM0	FSLB1	MTAS1	MJES0
MDAB0	MABW0	MTMR0	MJJG0
MJSW0	MBJK0	MWEW0	MJMP0
MREB0	MCCS0	MWVW0	MJVW0
MRJO0	MCEM0	FCMH0	MKCH0
MSJS1	MDBB0	FKMS0	MLNT0
MSTK0	MDLD0	FPKT0	MMAB0
MWBT0	MGWT0	MBDG0	MMDH0
FCMR0	MJAR0	MBWM0	MMJR0
FDRD1	MMDB1	MCSH0	MMWH0
FJAS0	MMDM2	MCTW0	MRTK0
FJRE0	MPDF0	MGJF0	MTAA0

Testdatensatz mit 80 Sprechern

FAKS0	MBJK0	MBDG0	MTDT0
FDAC1	MCCS0	MBWM0	MTHC0
FELC0	MCEM0	MCSH0	MWJG0
FJEM0	MDBB0	MCTW0	FADG0
MDAB0	MDLD0	MGJF0	FCFT0
MJSW0	MGWT0	MGLB0	FCRH0
MREB0	MJAR0	MHPG0	FDMS0
MRJO0	MMDB1	MJBR0	FEDW0
MSJS1	MMDM2	MJES0	FGJD0
MSTK0	MPDF0	MJJG0	FJLM0
MWBT0	MPGL0	MJMP0	FJMG0
FCMR0	MRCZ0	MJVW0	FLBW0
FDRD1	MRGG0	MKCH0	FLKD0
FJAS0	MTAS1	MLNT0	FMAF0
FJRE0	MTMR0	MMAB0	FMCM0
FJWB0	MWEW0	MMDH0	FNMR0
FPAS0	MWVW0	MMJR0	FREW0
FRAM1	FCMH0	MMWH0	FRNG0
FSLB1	FKMS0	MRTK0	FSEM0
MABW0	FPKT0	MTAA0	MBNS0