Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Projektarbeit
HS 17 Studiengang Informatik

Deep Learning für Speaker Clustering

Autoren	Savin Niederer Benjamin Heusser
Hauptbetreuung	Thilo Stadelmann
Nebenbetreuung	Oliver Dürr

Datum	22.12.2017

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

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

Mit der Abgabe dieser Projektarbeit 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 Projektarbeit 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 Projektarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Zusammenfassung

In den vergangenen Jahren wurden an der ZHAW mehrere Arbeiten auf dem Thema Speaker Clustering durchgeführt. Sprecher Clustering ist das Gruppieren von Audiosegmenten, welche dem gleichen Sprecher gehören. Zur Ausführung der Aufgabe wurden in den Arbeiten verschiedene Arten von Neuronalen Netzwerken trainiert und deren Eignung überprüft. Es wurde in allen Arbeiten gute Erfolge erzielt, welche jedoch nicht vollständig untereinander vergleichbar waren. Der Zeitaufwand, um die Trainingsdaten aufzuarbeiten und das Clustering und die Analyse zu programmieren, was keine Forschungsarbeit per se. ist, ist sehr umfangreich.

Diese Projektarbeit soll eine vereinheitlichte Basis für zukünftige Forschungen auf dem Gebiet des Speaker Clusterings mit Neuronalen Netzwerken erstellen und die Prozesse, welche keine Forschungsarbeit sind, vereinfachen und vereinheitlichen. Die Aufbereitung der Trainingsdaten wird vereinheitlicht, die Auswertung der Clusteringperformance wird von dieser Arbeit übernommen und die Einarbeitungszeit durch eine saubere Dokumentation des Codes sowie einer Bedienungsanleitung verkürzt.
Die bestehenden Ansätze werden in diese Projektarbeit eingebunden, ihre Ergebnisse verifiziert und mit einer neuen, vereinheitlichten Metrik untereinander verglichen.

Die Vorgaben konnten umgesetzt werden und diese Projektarbeit ist zum folgenden Schluss gekommen:
Die Netzwerkarchitektur von Patrick Gerber und Sebastian Glinski, ein paarweise vergleichendes Bi-LSTM, konnte für 40, 60 und 80 Sprecher die besten Resultate erzielen und setzte somit den State of the Art.

Abstract

In the previous years there have been multiple thesis on the topic of speaker clustering. Speaker clustering is the grouping of audio segments which belong to the same speaker. To fulfil this task, the thesis have taken approaches with training different kinds of artificial neural networks and evaluating their fitness for this it. All thesis have managed to yield good results, but they were not completely comparable with each other. The time spent on preprocessing the training data, coding the clustering and analysis of the network results was unpleasantly long, for it is non-research time spent on a research task.

This project thesis should create a uniform basis for future research on the topic of speaker clustering with neural networks and unify processes which are no considered research tasks. The preprocessing of the training data and the clustering as well as analysis is being taken care of in this project and the time spent for getting to know the project is reduced by creating a clean documentation of the code and and adding a usage guide.
The already existing works are being fully integrated into this project, the results of their thesis should get verified and finally compared against each other with a new and uniform metric.

The given goals have been met and we concluded this project thesis with the following:
The network architecture of Patrick Gerber and Sebastian Glinski, a pairwise comparing bi-lstm, was able to yield the best results for 40, 60 and 80 speakers and thus set the state of the art.

Vorwort

Wir wollten eine Projektarbeit im Bereich der KI (Künstlichen Intelligenz) durchführen und dabei Software schreiben, die auch nach unserer Arbeit genutzt wird. Die Begeisterung und Erklärungsbereitschaft von Thilo Stadelmann und Oliver Dürr, sowie die Möglichkeit einer wissenschaftlicher Publikation, hat uns dazu gebracht, diese Projektarbeit zu wählen. Die Änderung der Aufgabenstellung in der ersten Sitzung war für uns eine Überraschung. Diese war aber insofern eine willkommene, da wir so die Chance bekommen haben, mehr Zeit in die Weiterverwendbarkeit des Codes zu investieren und saubere Software zu erstellen. Der wissenschaftliche Aspekt der Arbeit litt unter dieser Änderung ein wenig, ging dennoch aber nicht vollständig verloren. Im Rückblick sind wir auf die erfolgreiche Projektarbeit und die gelungene Speaker Clustering Suite sehr stolz.

Die Teamarbeit lief im Allgemeinen gut, was wohl darauf zurückzuführen ist, dass wir bereits einige Projekte als Team bearbeitet haben. Gegen Ende der Arbeit entstanden jedoch einige Kommunikations- und Abstimmungsprobleme. Aus diesen Herausforderungen konnten wir allerdings für die gemeinsame Bachelorarbeiten lernen, wie wir besser miteinander kommunizieren können.

Wir möchten unseren Betreuern, Thilo Stadelmann und Oliver Dürr, für ihre großartige Unterstützung bei der Arbeit und ihren Input danken. Außerdem möchten wir uns bei den vier Projektteams, deren Arbeiten in diese einflossen, für ihre raschen und hilfreichen Antworten zu ihren einzelnen Implementierungen bedanken. Als Letztes möchten wir uns noch bei Benjamin Meier für die zur Verfügung gestellte Implementierung der MR und seinen Fragen zur Verwendung unserer Arbeit, die wir für deren Verbesserung verwenden konnten, bedanken.

Benjamin Heusser & Savin Niederer
17. Dezember 2017, Winterthur

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

Die Projektarbeit wurde an der ZHAW am Institut für angewandte Informationstechnologien (In-IT) unter der Betreuung von Thilo Stadelmann und Oliver Dürr durchgeführt.
Auf dem Thema des Speaker Clustering gibt es bereits vier Arbeiten an der ZHAW, zwei von Lukic und Vogt[1][2], eine weitere von Gerber und Glinski[3] und eine von Gygax und Egli[4], worauf diese Arbeit aufbaut.

Die Arbeiten verwenden in ihren Lösungsansätzen Neuronale Netzwerke (im speziellen Convolutional Neural Networks[5]), welche auf dem TIMIT-Datensatz trainiert werden. Die Arbeit von Gerber et al. hat ein perfektes Clustering bei 40 Sprechern erreicht, was zu diesem Zeitpunkt (vermutlich) dem State of the Art entspricht.

1.2 Motivation

Das Thema Speaker Clustering erweist sich als sehr ergiebig, es gibt ungeklärte Fragen aus den Vorgängerarbeiten und es können noch zahlreiche Experimente durchgeführt werden. Die vier Arbeiten legen eine Basis, mit der eigene Erfolge verglichen werden könnten, doch fehlt dazu ein einheitliches Maß.

Die allgemeine Metrik zur Auswertung der Clusteringleistung ist die *minimale misclassification Rate* (*MR*), jedoch wurde diese von jeder Arbeit selbst implementiert und so haben Gerber et al. [4] festgestellt, dass sich die Arbeiten nicht direkt vergleichen lassen. Die Frage, welcher MR Wert oder welcher in einer Arbeit vorgestellte Prozess nun State of the Art ist, konnte nicht mit Sicherheit beantwortet werden.

Der Vorgang der Datenaufbereitung und Auswertung ist für jede der Arbeiten fast identisch, jedoch wurde diese jedes Mal neu erstellt. Mit einer einheitlichen Aufbereitung und Auswertung könnte der Forschungsaufwand ins Zentrum gestellt und den Aufwand für nicht der Forschung dienender Arbeiten minimiert werden.

1.3 Zielsetzung

Es wird eine Speaker Clustering Suite erarbeitet, welche die Aufbereitung von Sprecherdaten und Auswertung von Clustering-Netzwerken vereinheitlicht. Die vier Arbeiten an der ZHAW zum Thema Speaker Clustering werden eingebunden, deren Leistung verifiziert und miteinander verglichen. Die Schnittstellen zwischen Aufbereitung - Netzwerk und Netzwerk - Auswertung werden dokumentiert und für zukünftige Netzwerke wenn möglich vereinfacht.

Die Schnittstellen sollen Teil einer Architektur sein, die auf Erweiterung, Modularität und Adaption ausgelegt ist. Die Einarbeitungszeit in die Suite soll angenehm und schnell vonstattengehen und neue Netzwerke sollen zeiteffizient eingebunden werden können.
Im Allgemeinen sollen spätere Nutzer auf einen gut dokumentierten Code zurückgreifen können.

1.4 Aufbau

Um die Leseführung zu vereinfachen wird hier eine kurze Beschreibung des Aufbaus gegeben.

- Kapitel 2 führt den Leser / die Leserin in die für das Verständnis der Projektarbeit wichtigen Grundlagen ein. Der Bereich Speaker Clustering und die dazu verwendeten Methoden werden vorgestellt.
- Kapitel 3 beschreibt das Vorgehen und die Methodik welche in der Projektarbeit verwendet wurden.
- Kapitel 4 vergleicht die bestehenden Ansätze, verifiziert deren Ergebnisse und stellt diese graphisch im Verhältnis zueinander dar.
- Im Anhang befindet sich eine Bedienungsanleitung der Speaker Clustering Suite und legt offen, was benötigt wird, um sie einzusetzen und weiterzuentwickeln.

2 Grundlagen

2.1 Neuronale Netzwerke

In dieser Arbeit werden verschiedene Arten von Neuronale Netzwerke verwendet, daher wird nur auf die Grundlage eingegangen. Erklärungen, welche zu einem der verwendeten Neuronalen Netzwerken in die Tiefe gehen, sind in den jeweiligen Arbeiten zu finden.

Layering Neuronale Netzwerke bestehen aus mehreren Layern von Neuronen. Jedes Neuron eines Layers ist mit jedem Neuron der beiden benachbarten Layern verbunden. Der erste Layer wird als Input-Layer bezeichnet, der letzte als Output-Layer. Die restlichen Layer werden als Hidden-Layers bezeichnet. Jede Verbindung zwischen den Neuronen besitzt ein Gewicht w_z, welche mit dem Output des vorangegangenen Neurons multipliziert wird. Dadurch wird bestimmt, wie viel jedes Neuron eines Layers zum Wert eines jeden Neurons in der folgenden Schicht beiträgt. Die einzelnen Input-Werte x_i werden im Neuron aufsummiert und einer Aktivierungsfunktion übergeben. Der Output y dieser Funktion ist wiederum der Input des nächsten Layers oder einer der Outputs des Neuronalen Netzwerks im Falle des Output-Layers. Als Aktivierungsfunktion wird oft eine Funktion verwendet die die Output-Werte zwischen 0 und 1 oder -1 und 1 festlegt. Eine solche Funktion kann zum Beispiel die Sigmoit Funktion sein.

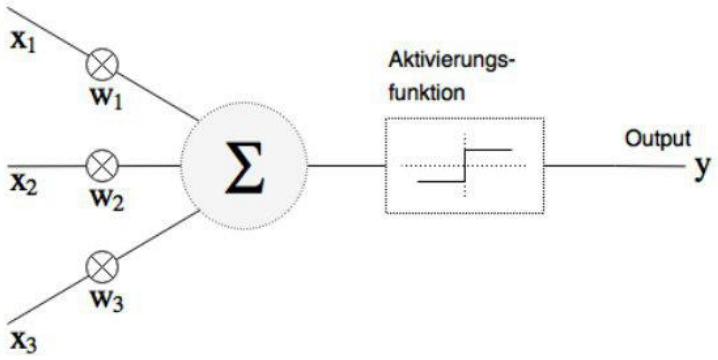


Abbildung 1: Berechnung eines Outputs eines Neurons. x_i Inputs, x_i Gewichte, y Output [3]

Supervised Learning Es existieren mehrere unterschiedliche Trainingsvarianten, wobei im Folgenden die Methode des Supervised Learning erklärt wird. Beim Supervised Learning ist für jeden Inputwert der erwartete Outputwert bekannt. Beim Training werden über mehrere Iterationen die Gewichte der einzelnen Verbindungen so verändert, dass der Output näher an den erwarteten Outputwert rückt. Dieser Prozess wird als Backpropagation bezeichnet, und durch einen Optimierungsalgorithmus, dem Optimizer, am Ende jeder Iteration durchgeführt. Eine Iteration ist die Verarbeitung eines ganzen Mini-Batches. Ein Mini-Batch ist eine zufällig ausgewählte Gruppe von Trainingsdaten. Die Abweichung zwischen dem erwarteten und dem tatsächlichen Output werden als Fehler oder Loss bezeichnet und durch eine sogenannte Loss-Funktion berechnet. Die Berechnung des Loss wird normalerweise als eine Matrixoperation durchgeführt. Einige der verwendeten Netzwerke verwenden allerdings eine elementweise Berechnung, zum Beispiel das Netzwerk flow_me von Gygax et al.[4].

Zeitaufwand Die meisten für Neuronale Netzwerke verwendeten Bibliotheken bieten eine große Anzahl verschiedener Loss-Funktionen und Optimizer, aus denen ausgewählt werden kann. Die in den einzelnen Arbeiten verwendeten Loss-Funktionen werden in den jeweiligen Arbeiten erklärt. Das Training von Neuronalen Netzen ist durch den hohen Rechenaufwand bei der Backpropagation sehr ressourcen- und zeitintensiv und kann je nach Konfiguration, Größe des Netzwerks und gewünschter Loss-Funktion mehrere Tage oder Wochen dauern. Der Gewinn ist jedoch eine sehr geringer Rechen- und Zeitaufwand bei der Auswertung von Inputs nach dem Training.

Checkpoints Wie bereits erwähnt dauert ein Training zuweilen mehrere Tage lang. Um den Fortschritt eines Trainings zu speichern, werden sogenannte "Checkpoints" eingesetzt. Im Falle eines Trainingsfehlers, wenn das Training abbricht oder der Loss größer werden sollte, kann auf die bereits gespeicherten Checkpoints zurückgegriffen werden. Ein Checkpoint kann entweder die Form eines voll funktionsfähigen Netzwerkes haben, quasi eine vollwertige Sicherungskopie, oder es wird nur die zur Wiederherstellung des Trainings notwendigen Informationen abgelegt.

Embeddings Ein Embedding ist im Zusammenhang dieser Arbeit der Output eines Hidden-Layers in einem Neuronalen Netzwerk, manchmal wird auch von einem Metric-Embedding geredet. Es wird einer der Hidden-Layer als Output-Layer verwendet, da keines der verwendeten Neuronalen Netze dafür trainiert wurde den Task des Speaker Clusterings auszuführen. Die Hoffnung ist jedoch in einem der Hidden-Layer eine gute Repräsentation eines beliebigen Sprechers zu finden. Jedes Embedding entsteht aus einem Spektrogramm, das aus den Samples generiert wird. Für das Clustering werden jedoch Embeddings pro Sprecher verwendet. Um diese zu erhalten müssen alle Embeddings eines Sprechers in ein einzelnes Embedding mit den gleichen Dimensionen umgewandelt werden. Dazu wird aus allen Embeddings eines Sprechers der Mittelwert gebildet und dieser dann im Clustering verwendet.

2.2 Sprecherclustering

Die Aufgabe von Sprecherclustering (engl. *speaker clustering*) ist ein Teilbereich von Sprechererkennung (engl. *speaker recognition*) und behandelt das gruppieren von Audio-Segmenten gleicher Sprecher, wobei vorab nicht bekannt ist, wie viele Sprecher beteiligt noch wer diese Sprecher sind. Somit handelt es sich um ein Mapping-Problem von Schwierigkeit $m : n$, m Samples zu n Sprechern. Verwandte Aufgaben im Bereich der Sprechererkennung sind die Sprecherverifizierung (engl. *speaker verification*) und Sprecheridentifizierung (engl. *speaker identification*) [6].

Sprechererkennung muss von der ähnlich klingenden Problemstellung der Spracherkennung (engl. speach recognition) unterschieden werden, welche sich der Übersetzung von gesprochener Sprache in geschriebenen Text widmet.

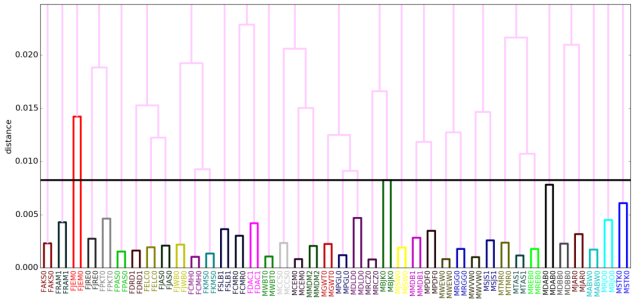
Sprecherclustering könnte in der Praxis zur Auswertung von Aufnahmen in Konferenzen oder Podiumsdiskussionen dienen. Hand in Hand mit der Übersetzung von Gesprochenem in Text, welche durch Spracherkennung gewonnen wird, und Aufteilen der Äußerungen mittels *Speaker Segmentation*, könnten zukünftig die Aussagen von gleichen Sprechern gruppiert und annotiert werden. Des Weiteren wäre somit denkbar, in Videos automatisch Untertitel zu erstellen und diese auch spezifischen Sprecher zuzuordnen.

2.3 Clustering

In dieser Arbeit wird hierarchisches Clustering verwendet. Der erste Schritt beim hierarchischen Clustering ist die Distanz zwischen den Embeddings zu berechnen. Für die Distanzberechnung wird die "cosine" Distanz verwendet, siehe Gleichung 1.

$$d(p||q) = 1 - \frac{u \cdot v}{||u||_2 ||v||_2} \tag{1}$$

Diese Distanzen werden dann verwendet, um das Clustering durchzuführen. Auch bei diesem Schritt existieren verschiedene Methoden. Bei der verwendeten "complete" Methode wird die maximale Distanz zwischen zwei Clustern als Distanzwert verwendet. In diesem Zusammenhang kann ein Cluster auch die Größe eins haben, in diesem Fall besteht ein Cluster aus einem einzelnen Element. In jedem Schritt werden jeweils die beiden Cluster zu einem verbunden, die die geringste Distanz zueinander aufweisen. Dies kann so lange fortgesetzt werden bis nur noch ein einzelner Cluster existiert oder bei einem bestimmten Distanzwert, dem Threshold, abgebrochen wird. In der Abbildung 2 entspricht dies der schwarzen Linie. In dieser Arbeit wird für jeden möglichen Threshold das Clustering durchgeführt und ausgewertet.



[3]

Abbildung 2: Darstellung eines hierarchischen Clusterings

2.4 Spektrogramme

Um Audiosignale in einem computergestützten Verfahren, z.B. einem Neuronalen-Netzwerk, verarbeiten zu können, kann auf Spektrogramme, wie Abbildung 3, zurückgegriffen werden. Ein Spektrogramm stellt ein Audiosignal in drei Dimensionen dar:

1. Zeitlicher Kontext in Sekunden
   Wert in der X-Achse des Spektrogramms
2. Frequenz in Hertz
   Wert auf der Y-Achse des Spektrogramms
3. Lautstärke in Db
   Wert dargestellt als Farbe von blau (leise) zu rot (laut)

Die Umrechnung vom Audiosignal zum Bild wird über eine FastFourier-Transformation (FFT) erreicht, welche über die Zeitachse angewendet wird. [7]

Wenn in der weiteren Arbeit von Spektrogrammen gesprochen wird, ist tatsächlich das Mel-Spektrogramm gemeint. Dies ist eine spezielle Form des Spektrogramms die nur die Frequenzen abdeckt, welche durch das menschliche Gehör wahrgenommen werden kann. Dies geht zurück auf die Mel-Skala[8], auf welcher die Frequenzen des Spektrogramms abgebildet werden mit Hilfe der folgenden Gleichung[9]:

$$m(f) = 2595 * log_{10}(1 + f/700) \tag{2}$$

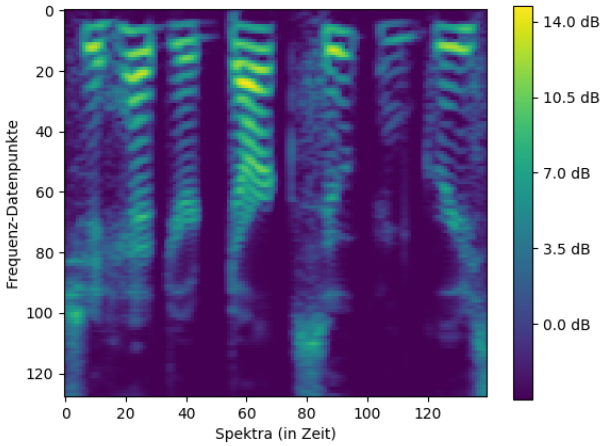


Abbildung 3: Ein Spektrogramm eines Sprechers des TIMIT Datensatzes

2.5 Eingebundene Arbeiten und Netzwerke

In der Speaker Clustering Suite werden vier Netzwerke mit eingebunden, die den Ansatz des Speaker Clusterings implementieren. In den dazugehörigen Arbeiten wird tiefer darauf eingegangen, wie ihr Vorgehen bei der Entwicklung und der Aufbau der Netzwerke ist.

Die Analyse der einzelnen Arbeiten ist nicht Teil dieser Arbeit, aber die wichtigsten Punkte und Merkmale der Arbeiten werden im Folgenden aufgeführt. Speziell erwähnt wurde, unter welchen Aspekten sich die Implementationen voneinander unterscheiden und welche Ergebnisse erreicht wurden.

Namensgebung der Netzwerke

Um Missverständnissen vorzubeugen und die Diskussion über einzelne Netzwerke / Arbeiten zu erleichtern, wurde jeder Arbeit, die in die Suite eingebunden wurde, ein kurzer Name zugewiesen, mit dem sie in den folgenden Texten auch identifiziert werden:

- HS 16 PA Lukic-Vogt[1] luvo
- FS 17 BA Lukic-Vogt[2] pairwise_kldiv
- FS 17 BA Gygax-Egli[4] flow_me
- FS 17 BA Gerber-Glinksi[3] pairwise_lstm

Diese Namen werden auch in der Speaker Clustering Suite verwendet, weshalb es auch wichtig war diese Namen kurz zu halten.

Luvo, Lukic et. al[1]

Diese Arbeit ist die erste Untersuchung an der ZHAW zum Thema Speaker Clustering, hier wurden die Grundlagen zur Verwendung von CNNs[5] erkundet. Das benutzte Netzwerk ist ein reines CNN mit Convolution und Batchnorm Layern, sowie einer Softmax-Aktivierung.
Trainiert wird das Netzwerk über den Surrogate-Task der Sprecher-Identifikation mit 590 Sprechern.

Die Sprecher-Embeddings wurden aus dem siebten Layer des Netzwerkes extrahiert und zum Clustering verwendet. Bei dem Test von 20 Sprechern wurde eine MR von 0.1 und bei 40 Sprechern eine von 0.05 erreicht. Dies kam en-par mit einem MFCC-GMM Ansatz, ohne die Notwendigkeit von händisch erarbeiteten Features. [1]

Die Arbeit hat keine herausragend speziellen Merkmale, weshalb der Name "**Luvo**" auf die Namen **Lu**kic und **Vo**gt zurückgeht.

Pairwise_kldiv, Lukic et. al[2]

Die zweite Arbeit von Lukic et. al verbessert den Ansatz aus der vorhergehenden Arbeit. Hierzu verwendeten sie wiederum ein CNN, vergleichbar mit dem Aufbau aus dem ersten Ansatz, aber diesmal trainierte das Netzwerk über den Surrogate-Task der Stimmengleichheit. Die Daten waren nur noch schwach gelabelt, was heißt, dass beim Training nicht bekannt war, wer der Sprecher war, sondern nur, ob das Paar vom gleichen Sprecher stammt oder nicht.

Für die Cost-Funktion, die invertierte Loss-Funktion, wurde die KL-Divergenz benutzt [10].
Die Sprecher-Embeddings wurden wiederum aus dem siebten Layer extrahiert und mit dem neuen Trainingstask waren nur noch 100 anstatt den 590 Sprechern zum Training notwendig.

Somit konnte mit weniger Sprechern fürs Training der gleiche MR Wert bei 40 Sprechern erreicht werden. [2]

Der Name "**Pairwise_kldiv**" setzt sich zusammen aus der paarweise vergleichenden (engl. **pairwise**) Natur des Netzwerkes und der verwendeten Loss-Funktion (KL-Divergenz[10]).

Flow_me, Gygax et. al[4]

Die Arbeit von Gygax und Egli basierte ebenfalls auf einem klassischen 2 Layer CNN, welches in TensorFlow implementiert wurde. Der Aufbau des Netzwerkes ist identisch mit dem von Lukic et al.
Wo sich die beiden arbeiten unterscheiden ist die Trainingsherangehensweise und Loss-Funktion. Flow_me trainiert das Netzwerk wie der erste Ansatz von Lukic et al. mit dem Surrogate-Task der Sprecher Identifikation und verwendet eine Loss-Funktion von Hoffer et al [11]. Das Netzwerk optimiert sein Training über die Adadelta Funktion und entnimmt seine Sprecher Embeddings aus dem siebten Layer.[4]

'Flow_me" wurde als Name gewählt, zusammengesetzt aus der Implementierung in Tensor**Flow** und den Metric Embeddings (**me**), auf welchen das Clustering gemacht wird.

Pairwise_lstm, Gerber et. al[3]

Gerber und Glinski habe für das Netzwerk auf einen neuen Ansatz gesetzt, die von ihnen gewählte Netzwerkarchitektur war ein LSTM, ein Long Short Term Memory Netzwerk. Wie der Name des Netzwerkes suggeriert, kann dieses besser auf zeitliche Zusammenhänge schließen und ist somit für Stimmen gut geeignet. Umgesetzt wurde dies mit der Bibliothek Keras.

Dieses Netzwerk hat ebenfalls mit schwach gelabelten Daten gearbeitet und paarweise Sprecher verglichen und diese mit der KL-Divergenz als Cost-Funktion ausgewertet. Als Optimizer wurde Adam[12] gewählt.

Mit dieser Architektur konnte für 40 Sprecher ein MR von 0 erreicht werden und unterbot die bisherigen Netzwerke auch bei einer größeren Anzahl von Sprechern.[3]

Der Name "**Pairwise_lstm**" steht für die paarweise (engl. **pairwise**) vergleichenden Natur des Netzwerkes und dem verwendeten Netzwerktyp, **LSTM**.

2.6 MR Definition

In den vier Arbeiten wurde die *misclassification rate (MR)*[13] und der Minimalwert davon als Vergleich zur Clusteringleistung benutzt. Die Arbeit von Gygax und Egli hat in ihrem Fazit festgestellt, dass die *MR* verschieden implementiert wurde. Jede Arbeit besaß eine leicht andere Implementierung vom *MR*, dadurch waren sie nicht untereinander vergleichbar. In der Arbeit flow_me wurde eine neue MR Definition vorgeschlagen, die konkrete Implementierung stammt von Meier et al.[14], welche sich daran anlehnt. Der Ausgabewert des *MR* liegt zwischen 0.0 und 1.0, wobei in diesem Fall 0.0 anzustreben ist. Die Definition des *MR* lautet folgendermaßen:

"Für jeden Sprecher wird der passende Cluster gesucht. Ein Cluster wird als passend bezeichnet, wenn die folgenden zwei Bedingungen erfüllt werden:

- Die meisten Samples eines Sprechers müssen in diesem Cluster beinhaltet sein.
- Innerhalb des Clusters darf es keinen anderen Sprecher geben, der mehr Samples hat.

Ist eine dieser Bedingungen nicht erfüllt, wird mit dem Cluster, welcher die nächst kleinere Anzahl an vorkommenden Samples besitzt, fortgefahren. Es kann vorkommen, dass ein Sprecher keinem Cluster und ein Cluster keinem Sprecher zugewiesen wird. Ist ein Sample nicht im Cluster seines Sprechers, wird es als falsch klassifiziert gewertet."[4]

Im Folgenden wird der Ablauf der *MR* Berechnung erklärt.

1. Finde für jeden vorhergesagten einen Zielcluster:
   (a) Finde für jeden vorhergesagten Cluster den Zielcluster, der die meisten korrekt zugeordneten Elemente besitzt.
   (b) Wenn mehr als 50% des Clusters korrekt zugeordnet sind, weise diesen vorhergesagten Cluster dem Zielcluster zu.
   (c) Wenn diese Definition auf mehrere Zielcluster zutrifft, wähle denjenigen Zielcluster, der die meisten korrekten Zuordnungen hat.
   (d) Wenn weiterhin mehrere Zielcluster in Frage kommen, wähle den kleinsten Zielcluster.
   (e) Ordne alle noch nicht zugewiesenen Cluster in der Reihenfolge ihrer Entropie.
   (f) Sollten zwei Cluster dieselbe Entropie haben, ordne sie nach der Anzahl nicht zugeordneter Elemente.
   (g) Sollten immer noch zwei Cluster den gleichen Platz erhalten, ordne sie nach einem beliebigen nummerischen Wert.
   (h) Weise jeden vorhergesagten Cluster demjenigen Zielcluster zu, der die größte Übereinstimmung mit dem vorhergesagten Cluster hat und noch nicht zugeordnet wurde.
2. Dividiere die falsch oder nicht zugewiesenen Elemente durch die Anzahl aller Elemente.

Um diesen Ablauf zu verdeutlichen, folgen noch einige Beispiele. Zu beachten ist, dass die Labels der Cluster nicht die eindeutige Bezeichnung des Clusters ist.

Zielcluster	[0, 0, 1, 1]
Vorhergesagte Cluster	[a, a, b, b]
Zuweisung mit Punkten 1.a) und 1.b)	[0, 0, 1, 1]
MR Berechnung Punkt 2.	$0/4 = 0$

Zielcluster	[0, 0, 0, 1, 1, 1, 1, 2, 2]
Vorhergesagte Cluster	[a, a, b, b, c, c, a, d, d]
Zuweisung mit Punkten 1.a) und 1.b)	[0, 0, b, b, c, c, 0, 2, 2]
Punkt 1.h)	[0, 0, b, b, 1, 1, 0, 2, 2]
Falsch oder nicht Zugewiesen	[0, 0, **b**, **b**, 1, 1, **0**, 2, 2]
MR Berechnung Punkt 2.	$3/9 = 1/3$

2.7 Softwarepatterns

Controller Pattern

Der Controller, auch Front Controller genannt, ist die zentrale Schnittstelle einer Applikation. Oft wird dieses Pattern im Zusammenhang mit Web-Anwendungen gebraucht, um z.B. alle Requests an eine Webseite zu verarbeiten. [15]

Der Controller zentralisiert die Aufrufung eines (Sub-)Systems wie in Abbildung 4 und kapselt diesen somit von der Außenwelt ab. Bei einem Aufruf initialisiert er die benötigten Objekte, Daten und Prozesse, ruft die (Sub-)Systeme auf und gibt die fertige Antwort zurück an den Benutzer.
Die Kapselung ermöglicht es die dahinterliegenden Systeme beliebig zu verändern, ohne dem Nutzer dies mitteilen zu müssen. So lange der Aufruf des Controllers gleich bleibt bemerkt der Nutzer nichts.

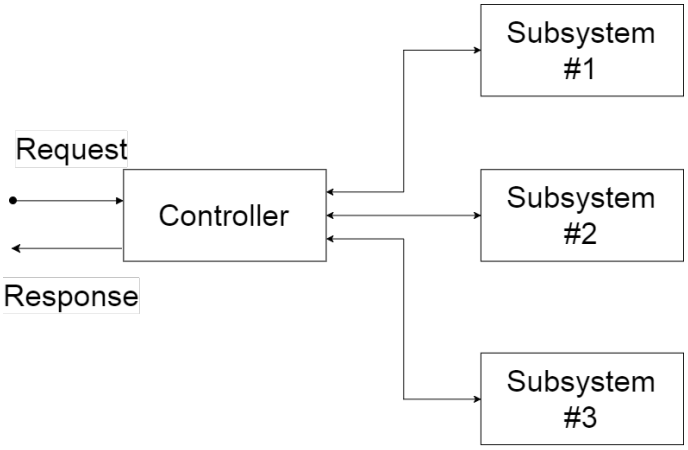


Abbildung 4: Die Anwendung eines Controllers zur Kapselung von Subsystemen

Komposit Pattern

Das Komposit Pattern beschreibt, wie eine abstrakte Klasse entweder ein einzelnes Objekt oder eine Kollektion von Objekten repräsentiert, wie in Abbildung 5 zu sehen ist. So können Objekte oder Komposite von Objekten einheitlich benutzt werden. [15]

Ein Beispiel hierfür ist das Dateisystem im Betriebssystem Linux, welches Files (einzelne Objekte) gleichbehandelt wie Ordner (Komposite von Files/Ordnern). Alle Befehle wie z.B. die Zugriffssteuerung können auf gleiche Weise für beide Typen verwendet werden.
Der Vorteil hierbei ist einerseits die einheitliche Benutzung und andererseits, dass der Unterschied zwischen einzelnen und Kompositobjekten vom Anwender verborgen wird.

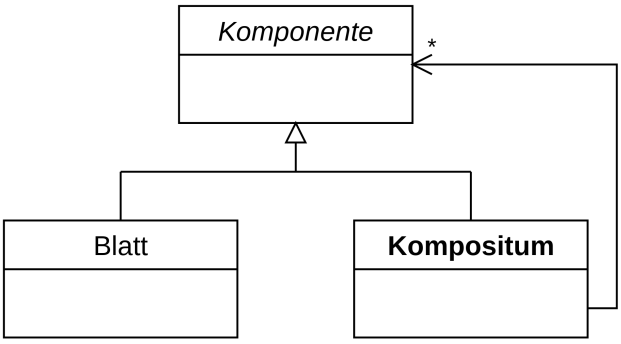


Abbildung 5: Die Klassenstruktur eines Komposit Patterns in UML notiert

3 Vorgehen und Methoden

3.1 Auswertung des Aufbaus der Arbeiten

Der Aufbau jeder Arbeit wurde analysiert und deren Funktionsweise im Ansatz ausgewertet. Im Anschluss darauf wurde ausgewertet, welche Teile vereinheitlicht werden sollen und welche nicht.

In der Datenselektion wird als erstes entschieden, welche der 630 Sprecher als Trainings-, Validations- und Testdaten dienen sollen. Es werden verschiedene Sprecherlisten erstellt und aufgrund von Geschlecht und Dialekt durchmischt, damit ein möglichst guter Trainingsertrag erzielt werden kann.

Als zweites werden die Daten in der Vorprozessierung extrahiert. Das heißt, die Wav-Files zu den zugehörigen Namen werden aus den Sprecherlisten ausgelesen und in einem binären Format, der Pickle Datei, auf der Festplatte abspeichert.

Im dritten Schritt wird mit den Trainingsdaten ein Neuronales Netzwerk antrainiert. Anschließend wird in einem Durchgang die Testdaten durch das Neuronale Netzwerk verarbeitet und es werden Sprecher Embeddings erzeugt. Dieser Teil ist für jedes Netzwerk einzigartig.

Im Abschluss werden diese Sprecher Embeddings geclustert und die Leistung über die *misclasification rate* bewertet.

Die Datenselektion und Vorprozessirrung, sowie das Clustering und die Analyse können in diesem Aufbau vereinheitlicht werden.

3.2 Softwarearchitektur

Da die Suite aus mehreren Netzwerken besteht, welche einerseits gemeinsam und als einzelne angesteuert werden können müssen, wurde eine Architektur gewählt, welche das Controller- und Komposit-Pattern[15] verbindet.

Die abstrakte Klasse NetworkController übernimmt die Verantwortung für das Aufrufen vom Training, Testing und Plotten der Ergebnisse. Um die Kapselung der Netzwerke zu verbessern, implementiert jedes Netzwerk seinen eigenen Netzwerk-Controller, der den Zugriff auf das jeweilige Netzwerk zentralisiert. Die Ansteuerung kann so vereinheitlicht und vereinfacht werden und der Code besser geteilt werden, so wird zum Beispiel die Analyse in der abstrakten Klasse aufgerufen.

Ihrerseits ist die gesamte Applikation über einen Front-Controller ansteuerbar, welcher der einzige Kontaktpunkt der Suite mit dem Benutzer / der Benutzerin ist und ebenfalls die Klasse NetworkController implementiert. Über diese Front-Schnittstelle wird die Suite aufgesetzt und die einzelnen Netzwerke angesprochen.

Das Ziel war es auch, eine grösstmögliche Kapselung zu erreichen und andere Grundprinzipien von Softwarearchitektur umzusetzen, wie das Prinzip der wenigsten Verantwortung oder DRY - ”Don’t repeat yourself”. Hierzu wurde der Netzwerke Code von dem allgemeinen Code getrennt, was die Aufgabe der einzelnen Netzwerke auf ihren Kern reduziert.

3.3 Datensatz

Die Wahl des Datensatzes fiel aus Kompatibilitätsgründen auf TIMIT, welches vom LDC stammt. Die Daten wurden bereits in [3], [2], [1], [4] verwendet, um Speaker Clustering zu trainieren.
Der Datensatz beinhaltet 630 Sprecher mit zehn Sätzen pro Sprecher, welche in Studioqualität aufgenommen sind. Zudem ist jeder Satz phonetisch notiert, um damit zum Beispiel *Speach Recognition* trainieren zu können, was in der Suite aber keine Verwendung findet. Unter den 630 Sprechern sind 192 weibliche und 438 männliche Stimmen, welche ihrerseits in die acht Hauptdialekte der Vereinigten Staaten von Amerika unterteilt sind.[16]

Die Sätze sind von maximal acht Sekunden Länge, können aber auch kürzer sein. Die Gesamtzeit an Audiosignalen beträgt:

$$630 Sprecher * 10 Sätze * 8 Sekunden = 50400 Sekunden$$

Der Datensatz besitzt eine Größe von 1.2GB auf der Festplatte und besteht aus 32'000 Dateien.

3.4 Daten Selektion und Vorprozessierung

In der Arbeit von Lukic et al.[1] wurden Sprecherlisten erstellt, welche die Sprecher in nicht überlappende Gruppe unterteilen und so möglichst geeignete Trainings- und Testdaten generieren. In luvo[1] wurden zum Training 590 Sprecher verwendet, in allen anderen Netzwerken der Suite wurden nur 100 Sprecher verwendet. Die Testdaten bestehen aus 40, 60 oder 80 Sprechern, welche im Falle von 100 Trainingssprechern keine überlappenden Sprecher besitzen.

Spezielles Augenmerk wurde bei der Auswahl der 100 Sprecher gelegt, diese unterteilen sich in 50 Männer und 50 Frauen, um die Netzwerke möglichst gleichwertig zu trainieren.

Den Sprecherlisten wurden in einem nächsten Schritt die zugehörigen Soundfiles vom TIMIT Datensatz ausgelesen und ausgewertet. Die Trainingsdaten wurden nach [4], in folgenden Format generiert:

$$X = [Sprechersatz, Kanal, Frequenz, Zeit]$$
$$y = [Sprecher - Nummer]$$

Bei der Aufbereitung wurde herausgefunden, dass der Kanal bei allen Instanzen überall den Wert '0' enthält. Er wird nicht mehr gebraucht in unserem Code.

3.5 Netzwerke

Architektur und Interfaces

Die Architekturen und das Vorgehen waren durch die bestehenden Arbeiten gegeben, die Anpassungen hierbei bestanden im Vereinheitlichen der Ein- und Ausgabewerte der Netzwerke.

Eingabewerteformat:

$$X = [Sprechersatz, Kanal, Frequenz, Zeit]$$

Ausgabewerteformat:

$$Embeddings = [Netzwerk/Checkpoint, Listof Embeddings]$$

Die Netzwerke übernehmen die Batchgenerierung selbst, dabei wird bei allen das Spektrogramm (Frequenz und Zeit) aus den Eingabewerten gelöst und eventuell noch weiterverarbeitet.

Alle Netzwerke wurden so eingestellt, dass sie das erfolgreichste Experiment ihres Papers / der Arbeit replizieren und somit auch theoretisch die bestmöglichen Ergebnisse erreichen sollten. Im trainierten Zustand verarbeiten die Netzwerke die Testdaten und geben ein *Speaker Embedding*[17] zu jedem Sprecher aus. Mithilfe dieser Speaker Embeddings kann das Speaker Clustering durchgeführt werden.

Training

Thilo Stadelmann hat in einer der Projektsitzungen erwähnt, dass beim Training der verwendeten Netzwerke bisher bei der Auswertung von Netzwerken vermutlich Cherrypicking betrieben wurde. Das heißt, dass manchmal mehrere Netzwerke mit derselben Einstellung trainiert wurde, aber nur der beste Wert in die Arbeit aufgenommen und dokumentiert worden ist. Dies wurde im Ansatz versucht zu ändern.

Hierzu wurden die Netzwerke nicht nur einmal mit der "geerbten" Voreinstellung trainiert, sondern so oft wie es der Zeitraum dieser Arbeit erlaubte. Aus den Resultaten der Trainingsdurchläufe wird ein Konfidenzintervall definiert, welches in der Auswertung mit den anderen Netzwerken verglichen wird. Dieses Konfidenzintervall beschreibt die Streuung, in welcher sich die Analysewerte der trainierten Netzwerke befinden.

3.6 Clustering der Embeddings

Wenn sich auch der Vorgang vom Clustering der Embeddings nicht von den Vorgängerarbeiten unterscheidet, war die Implementierung doch leicht abweichend in jedem Netzwerk. Das Clustering folgt dem Algorithmus:

- Die Distanz der Embeddings zueinander wird mit der Metrik "**cosine**" errechnet.
- Aus der Distanz wird eine Linkage Matrix erstellt mit der Linkagemethode = "**complete**"
- Aus der Linkage und den gewonnenen Thresholds errechnen sich die vorhergesehenen Cluster.

Embedding Metrik cosine und Linkagemethode complete haben sich in den Arbeiten von [2] [3] [1] [4] bewährt.

Da es sich bei der Clusteringmethode um ein Hierarchisches Clustering handelt und aus den zwei verwendeten Datensätzen jeweils ein Embedding entsteht, gibt es bei N Sprechern jeweils maximal $2N$ Cluster. In den vier referenzierten Arbeiten wurde jeweils nur der MR-Wert errechnet, welcher am niedrigsten ist und die idealste Threshold aufweist. Dies eignet sich zwar, um eine generelle Aussage über die Performanz des Netzwerkes zu machen, aber erlaubt keine detaillierte Analyse der restlichen Werte. Im Unterschied wurden die MR-Werte vom gesamten Hierarchischen Clustering nicht nur errechnet, sondern auch ausgewertet.

3.7 Analyse

Im Schritt der Analyse wird die Performance des Clusterings ausgewertet. Um ein definitives Maß zu definieren, wie gut eine Clustering ist, wurde der Minimalwert der MR verwendet. Dies wurde in den bestehenden Arbeiten bereits so gehandhabt.

Um den Minimalwert der misclassification rate in Perspektive zu rücken, wurden die ganze Wertereihe für jedes Netzwerk abgespeichert. Zusätzlich helfen zwei weitere Metriken zur Einschätzung der Clusteringleistung. Diese wurden im Endergebnis unterhalb der MR geplottet und zeigen unterschiedliche Aspekte des Clusterings auf.

3.7.1 Completeness Score

Der Completeness Score oder auf Deutsch "Vollständigkeitswert" beschreibt, zu wieviel Prozent die einzelnen Cluster vollständig sind. Wenn alle Sätze eines Sprechers zusammen geclustert werden, gilt das Cluster als vollständig und ihm wird die Score 1.0 gegeben. Dieser Wert ist anzustreben. Das untere Ende der Skala liegt bei 0.0.

Formel: [18] 3 4 5

$$c = \begin{cases} 1 & if H(K,C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & else \end{cases} \tag{3}$$

wo

$$H(K|C) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{k=1}^{|K|}a_{ck}} \tag{4}$$

$$H(K) = -\sum_{k=1}^{|K|}\frac{\sum_{c=1}^{|C|}a_{ck}}{n}\log\frac{\sum_{c=1}^{|C|}a_{ck}}{n} \tag{5}$$

Beispiel:

$$completeness\_score(expected\_clusters, actual\_clusters)$$
$$1 - completeness\_score([0,0,1,1],[1,1,0,0]) = 1.0$$
$$2 - completeness\_score([0,1,2,3],[0,0,1,1]) = 1.0$$
$$3 - completeness\_score([0,0,1,1],[0,1,0,1]) = 0.0$$

1) Alle Cluster sind korrekt zugeordnet.
2) Nicht perfektes Clustering, verschiedene Samples werden demselben Cluster zugeordnet, ist dennoch complete.
3) Samples desselben Clusters werden falsch zugeordnet

Die Implementierung dieser Metrik wurde aus der Bibliothek sklearn entnommen und folgt der obigen Formel beschrieben in [18].
In Abbildung 6 ist der angewandte Completeness Score auf einem trainierten Luvo Netzwerk. Auf der X-Achse sind die Anzahl Cluster und auf der Y-Achse der Completeness Score (oben 1.0, unten 0.0). Im Falle eines Clusters sind alle zusammengehörigen Samples in einem Cluster, was den guten Anfangswert beschreibt.

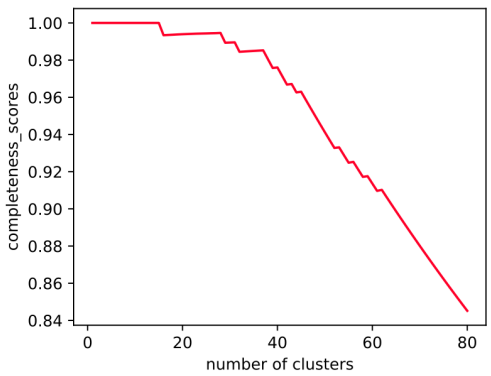


Abbildung 6: Der Completeness Score im Beispielfall vom Luvo-Netzwerk

3.7.2 Homogeneity Score

Der Homogeneity Score bewertet die Homogenität eines Clusters, sprich wie hoch der Anteil an Fremden Sprechern in einem Cluster ist. Diese Metrik richtet sich nach der Konvention das 1.0 als Ergebnis anzustreben ist und 0.0 der schlechteste Fall darstellt. 1.0 steht für ein perfekt homogenes Resultat.

Formel: [18] 6 7 8

$$h = \begin{cases} 1 & if H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & else \end{cases} \tag{6}$$

wo

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{c=1}^{|C|}a_{ck}} \tag{7}$$

$$H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{ck}}{n}\log\frac{\sum_{k=1}^{|K|}a_{ck}}{n} \tag{8}$$

Beispiel:

$$homogeneity\_score(expected\_clusters, actual\_clusters)$$
$$1 - homogeneity\_score([0,0,1,1],[1,1,0,0]) = 1.0$$
$$1 - homogeneity\_score([0,0,0,0],[1,1,0,0]) = 1.0$$
$$2 - homogeneity\_score([0,0,1,1],[0,1,0,1]) = 0.0$$

1) Ein Clustering erreicht dann 1.0, wenn all Samples desselben Sprechers im selben Cluster zugeordnet werden.
2) Auch wenn alle Samples von verschiedenen Sprechern einem Cluster Zugeordnet werden entsteht ein Score von 1.0. 3) Hier wurden die verwandten Samples komplett falsch zugeordnet, was in einer Score von 0.0 ersichtlich wird.

Die Implementierung der Homogeneity Score wurde ebenfalls aus der sklearn Bibliothek entnommen und nach den obigen Formeln implementiert.
Unten in Abbildung 7 ist der angewandte Homogeneity Score auf einem trainierten Luvo Netzwerk ersichtlich. Auf der X-Achse sind die Anzahl Cluster und auf der Y-Achse der Homogeneity Score (oben 1.0, unten 0.0) zu sehen. Gegen 80 Cluster konvergiert der Wert auf 1.0, was bei einem jedem Clustering war, dann sind alle Samples einem einzigen Cluster zugewiesen, was sie vollkommen homogen macht.

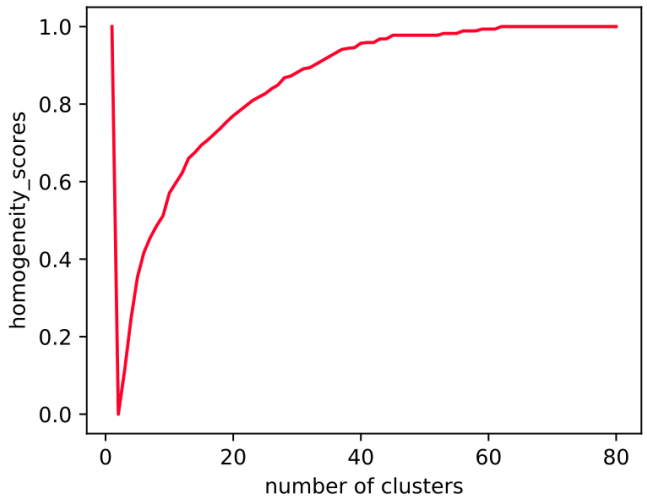


Abbildung 7: Der Homogeneity Score im Beispielfall vom Luvo-Netzwerk

3.8 Verwendete Software und Hardware

Software

Als Basis für die Entwicklung der Projektarbeit wurde Python 3.5.2 verwendet und PyCharme von Jetbrains[1] als IDE benutzt. Folgende Python-Packages (und ihre Sub-Packages) kamen zum Einsatz:

- tensorflow(-gpu) 1.3
- numpy
- scipy
- theano 0.9.0
- scikit-learn 0.0
- lasagne 0.2dev
- keras 2.0.6
- librosa 0.5.1
- nolearn
- pandas 0.20.3
- munkres 1.0.7

Als Packagemanager für Python wurde mit Anaconda[2] gearbeitet. Wenn keine Versionsangabe gemacht wurde, ist die Arbeit mit der neusten und zukünftigen Versionen kompatibel.

Docker

Docker ist eine virtuelle Maschine, die das Ausführen von Programmen in separaten vor-kompilierten Containern erlaubt. Ein solcher Container erlaubt eine plattformunabhängige Entwicklung und Ausführung von Programmen. Zusätzlich sind in einem Container alle Abhängigkeiten vorhanden, so ist es nicht notwendig auf jedem Rechner alle Programme zu installieren. Zusätzlich können Fehler vermieden werden, die plattformabhängig oder installationsbedingt auftreten.

Für diese Arbeit war die Verwendung von Docker essentiell, da der zur Verfügung stehende GPU-Cluster lediglich mit Dockercontainern ausgeführt werden kann. Außerdem sind die plattformunabhängige Ausführbarkeit des Codes und das Umgehen einer Installation aller notwendigen Programme und Packages große Vorteile.
Von Tensorflow, einer Neuronalen Netzwerk Bibliothek, gibt es eine CPU und GPU Ausführung, welche sich in der Installation unterscheiden. Durch einen passenden Dockercontainer kann zwischen den beiden einfach gewechselt werden, ohne z.B. die Entwicklungsumgebung neu einrichten zu müssen.

Zusätzlich bietet jeder Container sein eigenes Filesystem was es ermöglicht das Training zu parallelisieren und so zu beschleunigen.

Hardware

Das Trainieren der Netzwerke wurde auf dem GPU-Cluster der ZHAW ausgeführt. Für Netzwerke [3] [2] [4] mit GPU-Support wird jeweils eine GPU, die NVIDIA Titan X, und zwei CPUs, 2.10 GHz Intel(R) Xeon(R) Prozessor mit 6 Kernen, alloziert. Für das andere Netzwerk [1] wurde eine GPU und vier CPUs alloziert.

[1] PyCharme: https://www.jetbrains.com/pycharm/
[2] Anaconda: https://anaconda.org/

4 Vergleich und Verifizierung bestehender Ansätze

Ein Teil der Aufgabenstellung der Suite war es, die vier Netzwerke neu zu trainieren, die in den Arbeiten erwähnten Ergebnisse zu replizieren und sie mit einem einheitlichen Maß zu vergleichen. Die Auswertung und der Vergleich wurden mit 40 Sprechern gemacht.

4.1 Training und Verifikation

Die vier Netzwerke wurden in der Suite neu antrainiert und im ersten Schritt mit der MR Definition der jeweiligen Arbeiten, des jeweiligen Papers ausgewertet.

Tabelle 1: Vergleich und Verifikation

Networks / Metriken	Arbeiten	Alter MR
Luvo	0.45	0.625
Pairwise_kldiv	$0.08^{+0.12}_{-0.04}$	0.12
Flow_me	0.044	0.125
Pairwise_lstm	$0.017^{+0.008}_{-0.017}$	0.025

Luvo

Die erste Netzwerkarchitektur von Lukic et al.[1] hat in unseren Tests deutlich schlechter abgeschnitten, als im Paper erwähnt. Bei 590 trainierten Sprechern und der Embeddingentnahme auf Layer 7 (Dense-Layer 2) wurden nicht die im Paper beschriebenen 0.45 MR erreicht, sondern 0.625.

Aus unerklärlichen Gründen hat das erste Training im ersten Durchlauf Ergebnisse geliefert, als ob es wie anfangs "random" auswählen würde, weshalb dieses Training aus der Auswertung ausgeschlossen wurde. Die weiteren Trainings haben zwar nach der neuen Berechnung Werte "in der Norm" geliefert, zeigten jedoch noch andere Probleme. So lässt sich die Auswertung der trainierten Netzwerke nur mit dem gleichen Dockercontainer durchführen. Aufgrund der sehr langen Trainingszeit konnten sie nicht nochmals Trainiert werden. Die durchschnittliche Epochendauer am Anfang war ~200 Sekunden, jedoch kam es durchaus vor, dass pro Epoche zwischen 700 und 1700 Sekunden gebraucht wurden.

Pairwise_kldiv

Die Ergebnisse von Lukic et al.[2] konnte mit ihren eigenen Metriken repliziert werden und ist in allen Trainingseinheiten in der im Paper beschriebenen Konfidenzintervall von $0.08^{+0.12}_{-0.04}$ geblieben. Dieser Ansatz von Lukic et al.[2] hat, mit der neuen MR Berechnung, wesentlich bessere Ergebnisse geliefert als der erste Ansatz.

Flow_me

In dem Netzwerk von Gygax et al.[4] konnte im ersten Trainingsdurchgang nur ein um 0.081 höherer MR als in der BA erreicht werden, die weiteren Trainingseinheiten erreichten ebenfalls den in der Arbeit beschriebenen Wert nicht. Unsere Ergebnisse wurden dadurch erzeugt, dass aus den Durchläufen jeweils der Trainingscheckpoint mit dem minimalen MR ausgelesen wurde.

Pairwise_lstm

Die Ergebnisse von Gerber et al.[3] konnte mit ihren eigenen Metriken repliziert werden und ist in allen Trainingseinheiten in der im Paper beschriebenen Konfidenzintervall von $0.0^{+0.025}_{-0.0}$ geblieben. Die neue Implementation des MR's liefert konstant niedrigere MR Werte als die Implementation von Gerber et al.

4.2 Vergleich nach Vereinheitlichung

Die Zeit erlaubte es die Netzwerke mehrere Male zu trainieren. Aus den Durchläufen konnte ein Konfidenzintervall erstellt werden. Im Nachfolgenden präsentieren wir die Untersuchungsergebnisse der Vergleiche.

Tabellenbeschrieb In den Tabellen 3, 4 und 5 wollten wird darauf eingehen, ob und wie die Netzwerke bei mehreren Trainings streuen. Der neue minimale MR wird berechnet aus dem Mittelwert aller Trainings und die Standartabweichung zeigt das Ausmaß der Streuung auf. In der letzten Spalte sind noch die konkreten Ergebnisse aufgelistet.
Das Luvo Netzwerk konnte als einziges nur zwei Mal trainiert werden, was an dem immensen Zeitaufwand im Training lag (siehe Tabelle 2). Außerdem sind Resultate des Luvo Netzwerks bei 60 und 80 Sprechern nicht so aussagekräftig wie die der anderen Netzwerke, da einige der Testdaten bereits in den Trainingsdaten des Luvo Netzwerks verwendet wurden.
Es konnte nicht herausgefunden werden weshalb das Netzwerk flow_me in den Runs 2 und 4 identische Werte besitzt. Da wurde überprüft, dass nicht zwei Mal die gleichen Checkpoints ausgewertet wurden und es ist zu vermuten, dass das Training ähnlich verlief und deshalb die gleichen Werte entstanden.

Metriken Interpretation Die Hauptmetrik ist der minimale MR, sprich der tiefste Punkt auf der MR Wertelinie. Ein minimaler MR ist dann speziell gut, wenn er sich auf der X-Achse nahe der Sprecheranzahl befindet. Dies lässt sich auch in der Completeness und Homogeneity score ablesen, beide sollten in den Graphiken bei der Zielanzahl von Clustern (40, 60, 80) 1.0 erreichen, um ein perfektes Clustering zu zeigen. Der MR sollte bei der Zielanzahl an Clustern möglichst 0.0 sein.

Graphiken Beschrieb Die Vergleichsgraphiken 8, 9 und 10 zeigen die besten Checkpoints der Netzwerke ausgewertet bei 40, 60 und 80 Sprechern. In jeder Graphik gibt es drei Subplots, in welchen auf der X-Achse die Anzahl von Clustern abzulesen ist. Im oberen Subplot ist der MR Wert auf der Y-Achse aufgezeichnet, die wichtigste Vergleichsmetrik der Netzwerke. Der Subplot unten links zeigt die Completeness-Score, und unten rechts ist der Homogeneity-Score.

Zeitvergleiche im Training Aus der Tabelle in 2 kann festgestellt werden, dass die Trainingszeit auf keinem Fall mit den erreichten MR Werten korreliert. Viel mehr ist zu sehen, dass das LSTM Netzwerk bei exzellenter Performance eine kurze Trainingszeit aufweist.
Pairwise_kldiv weist ebenfalls gute Ergebnisse auf, aber braucht fast fünfmal so lange, um vollständig trainiert zu sein. Dies kann erklärt damit werden, dass für das Pairwise_kldiv Netzwerk 30'000 Mini-Batches trainiert wurden um eine leichte Performanzsteigerung zu 10'000 Mini-Batches zu erreichen. In einem kleinen Experiment war der Unterschied bei 0.04 MR doch ersichtlich. Die Trainingszeit hängt auch von der Zahl der Sprecher ab, mit denen trainiert wird. Im Falle von Luvo wurde mit den gesamten 590 Sprechern trainiert, was das Training ebenfalls verlangsamt.

	Trainingszeit	**# Training**
Luvo	~120-170 h	590 Sprecher
Pairwise_kldiv	~111 h	100 Sprecher
Flow_me	~17 h	100 Sprecher
Pairwise_lstm	~20 h	100 Sprecher

Tabelle 2: Zeitvergleich im Training

Der Zeitfaktor spielt in dieser Arbeit keine wichtige Rolle, es hatte trotzdem Auswirkungen auf diese Arbeit. Wegen der langen Trainingszeit vom Luvo Netzwerk konnte dieses nur zwei Mal und das Pairwise_kldiv Netzwerk nur drei Mal trainiert werden. Vorgesehen waren vier Trainings für alle Netzwerke.

4.2.1 Vergleiche mit 40 Sprechern

	Neuer minimal MR	Standardabweichung	Ergebnisse
Luvo	0.088	0.017	0.075 0.1
Pairwise_kldiv	$0.03\overline{3}$	0.014	0.05 0.025 0.025
Flow_me	0.096	0.033	0.125 0.125 0.075 0.0625
Pairwise_lstm	**0.01875**	**0.007**	0.025 0.0125 0.0125 0.025

Tabelle 3: Vergleich neue minimale MR mit 40 Sprechern

Die Auswertung von vier Trainingsdurchläufen für Flow_me und Pairwise_lstm, drei für Pairwise_kldiv und zwei Durchläufen für Luvo ergeben, dass das Netzwerk **Pairwise_lstm** mit einer MR von **0.01875** über 40 Sprecher am besten abschneidet. Alle MR-Werte im Konfidenzintervall liegen unterhalb der Werte anderer Netzwerke. Mit einer Standartabweichung von **0.007** ist auch dies der beste Wert der vier Netzwerke.

Interessant zu beobachten ist auch, dass Luvo zwar im minimal MR Wert von 0.088 recht gut abschneidet, jedoch bei der Betrachtung von Completeness und Homogeneity konstant weniger performant ist als die anderen Netzwerke.

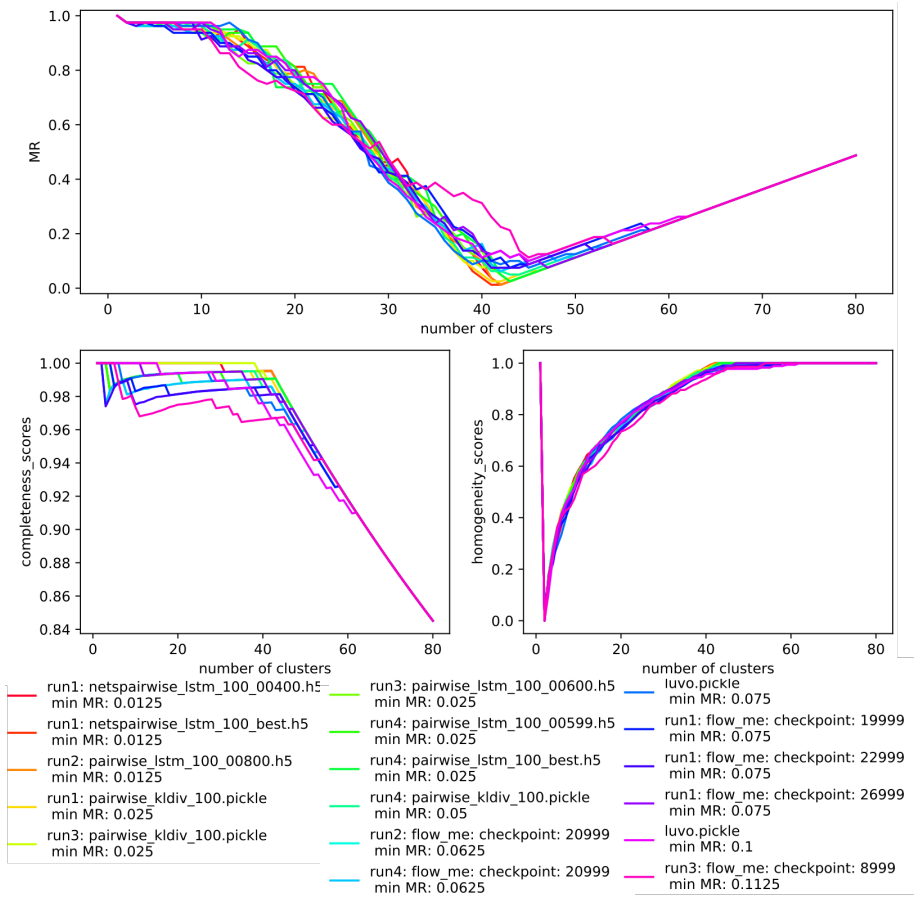


Abbildung 8: Die komplette Vergleichsgraphik der besten Netzwerke für 40 Sprecher.

4.2.2 Vergleiche mit 60 Sprechern

	Neuer minimaler MR	Standardabweichung	Ergebnisse
Luvo	0.066	0.012	0.075 0.058
Pairwise_kldiv	0.024	0.015	$0.01\overline{6}$ $0.01\overline{6}$ 0.042
Flow_me	0.135	0.023	0.108 $0.13\overline{3}$ $0.16\overline{6}$ $0.13\overline{3}$
Pairwise_lstm	**0.0205**	**0.0052**	$0.01\overline{6}$ $0.01\overline{6}$ 0.025 0.025

Tabelle 4: Vergleich neue minimale MR mit 60 Sprechern

Auch mit 60 Sprechern schließt das Pairwise_lstm Netzwerk mit 0.0205 MR am besten ab, der Wert unterscheidet sich nur marginal vom erreichten Wert bei 40 Sprechern. Die Streuung hat sich noch verkleinert, was aber für alle Netzwerke zutrifft.
Zu beobachten ist ebenfalls eine wesentliche Verschiebung der Kurve der schlechteren Netzwerke, die Unterschiede sind hier wesentlich besser sichtbar. Es kann argumentiert werden, dass die paarweise vergleichenden Netzwerke auch bei einer höheren Anzahl von Sprechern in ihrem Resultat konstant bleiben.

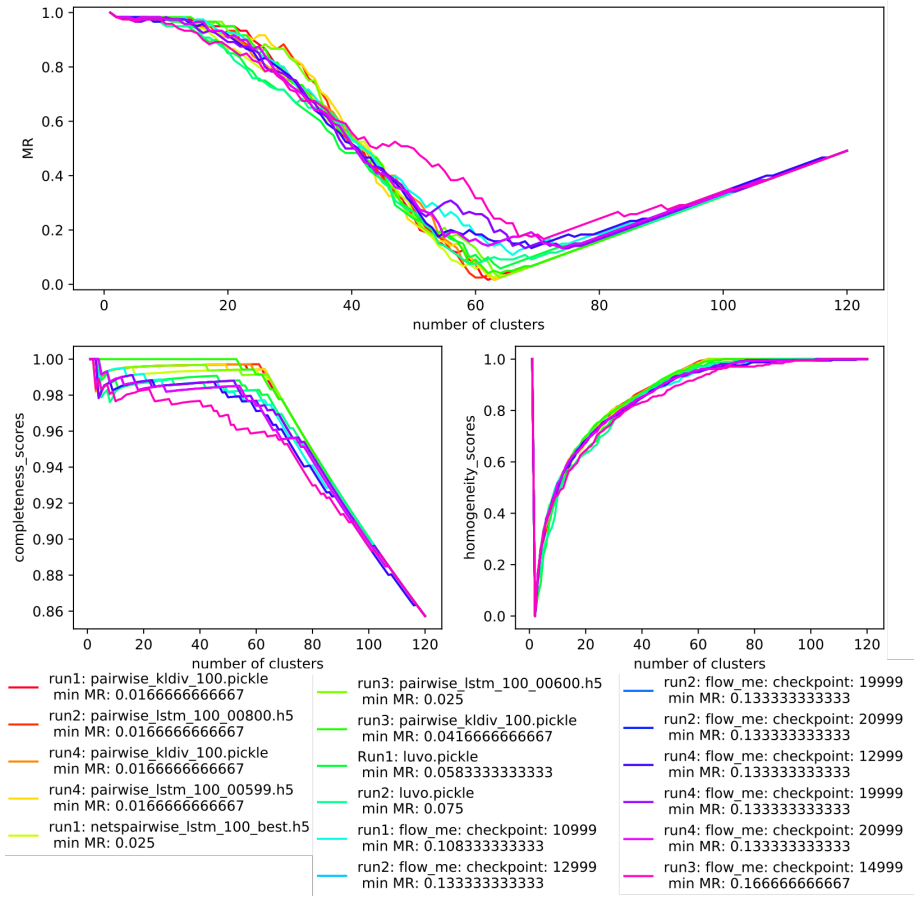


Abbildung 9: Die komplette Vergleichsgraphik der besten Netzwerke für 60 Sprecher.

4.2.3 Vergleiche mit 80 Sprechern

	Neuer minimaler MR	Standardabweichung	Ergebnisse
Luvo	0.053	0.013	0.043 0.062
Pairwise_kldiv	0.049	0.011	0.043 0.043 0.069
Flow_me	0.149	0.015	0.156 0.1375 0.168 0.1375
Pairwise_lstm	0.048	0.006	0.043 0.043 0.05 0.056

Tabelle 5: Vergleich neue minimale MR mit 80 Sprechern

Die beiden Netzwerke pairwise_kldiv und pairwise_lstm erreichen bei 80 Sprechern mit den Werten 0.049 und 0.048 nahe beieinanderliegenden Resultate. Pairwise_lstm besitzt aber auch hier die geringste Standardabweichung. Das Netzwerk flow_me performt mit einem MR Wert von 0.149 und einer Standardabweichung von 0.015 signifikant schlechter als alle anderen. Aufgrund der sinkenden MR-Werte über alle drei Analysen und der im Paragraf Tabellenbeschrieb 4.2 erwähnten Einschränkungen ist das gute Abschneiden des luvo Netzwerks wohl ein systematischer Fehler.

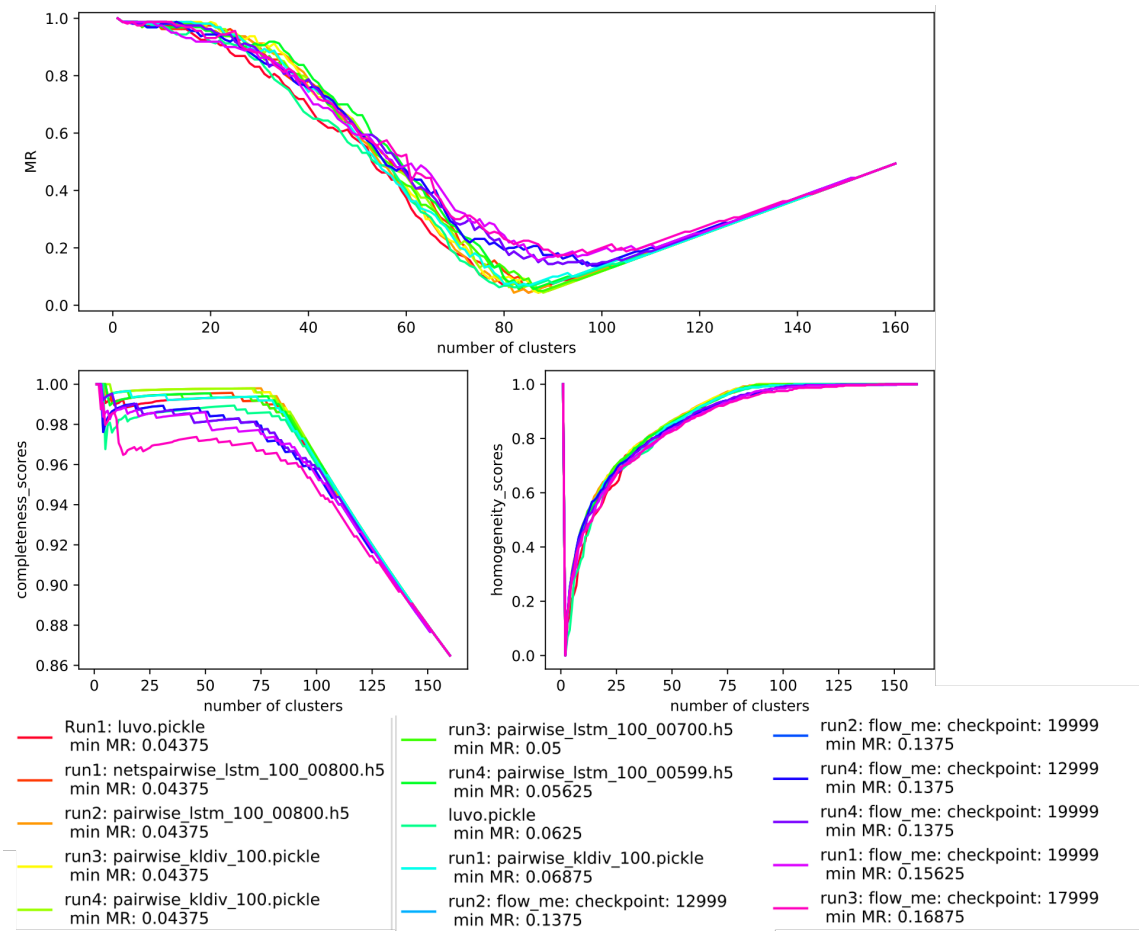


Abbildung 10: Die komplette Vergleichsgraphik der besten Netzwerke für 80 Sprecher.

5 Fazit und Erkenntnisse

5.1 Aufgabenstellung

Die Aufgabe einer Speaker Clustering Suite, so wie sie zu Anfang der Arbeit gedacht war, konnte erfolgreich umgesetzt werden.

- Mehr Teilbereiche als erwartet konnten vereinheitlicht werden.
  Workflow, Aufbereitung der Sprecherdaten, Clustering, Analyse, Darstellung/Plotten der Ergebnisse sind alle nur noch einmal vorhanden.
- Die vier Netzwerke konnten wie geplant eingebunden werden und es wurde zudem ein gemeinsames Interface definiert, was den Umfang der Netzwerke auf die Kernaufgabe reduzierte.
- Die Ergebnisse/Arbeiten konnten verifiziert werden, wenn sie auch nicht immer die gleich guten Werte lieferten.
- Der Vergleich wurde nicht auf einzelnen Werten, sondern auf Konfidenzintervallen ausgewertet. Somit konnte auf eventuelle Schwankungen in der Netzwerkleistung eingegangen werden.
- Die Ziele der Architektur wurden erreicht. Die Suite ist auf denkbare Änderungen vorbereitet und modulare Erweiterungen durch neue Netzwerke sind leicht durchführbar.
- Die Einarbeitungszeit von einer neuen Person konnte durch Dokumentation in dieser Arbeit, sowie docstrings im Pythoncode und einer schnell verständlichen Codeflow verbessert werden. In allem Sourcecodefiles der Arbeit, welche im Ordner common der Arbeit sind, wurden auf die Regeln von ”Cleancode” und andere Codeconventions geachtet.

Abschließend kann festgehalten werden, dass die Ziele der Arbeit erreicht und mehrheitlich übertroffen wurden.

5.2 Feststellungen beim Vergleich der Netzwerke

Die Vermutung liegt nahe, dass der Surrogate-Task beim Training die Performanz beim Clustering beeinflusst. Die beiden Netzwerke, welche über den Task ”Speaker Identification” trainiert haben [1] [4], wiesen, bei 40 Sprechern, im Schnitt einen 0.05 höheren MR Wert auf als die paarweise trainierten Netzwerke[2] [3]. Bei 60 und 80 Sprechern fällt das flow_me Netzwerk noch weiter zurück. Da sich die Trainings und Testdaten im Falle des luvo Netzwerks überschneiden, können die Resultate dieses Netzwerks nicht in diese Betrachtung einfließen.

5.3 State of the Art in Speaker Clustering

Das Pairwise_lstm Netzwerk von [3] konnte bei 40 Sprechern eine MR von 0.01875 mit einer Standardabweichung von 0.007 erreichen. Zudem kann eine Aussage darüber getroffen werden, dass Pairwise_lstm von allen vier Netzwerken bei der Anzahl an Zielclustern (40 Cluster bei 40 Sprechern) ebenfalls den tiefsten MR Wert erreicht hat. Auch bei 60 und 80 Sprechern ist Pairwise_lstm das beste Netzwerk, auch wenn der Unterschied zu Pairwise_kldiv geringer wird. Die Streuung von Pairwise_lstm bleibt in allen Analysen am kleinsten, es scheint das konstanteste der vier Netzwerke zu sein.
Gegeben, dass sich der Threshold bei 40 Sprechern durch Heuristik oder ebenfalls maschinelles Lernen herausfinden ließe, würde dieses Netzwerk auch in der Praxis mit einem MR von ca. 0.1 sehr gut abschneiden. Die gesamte Analyse befindet sich im Bild 11.

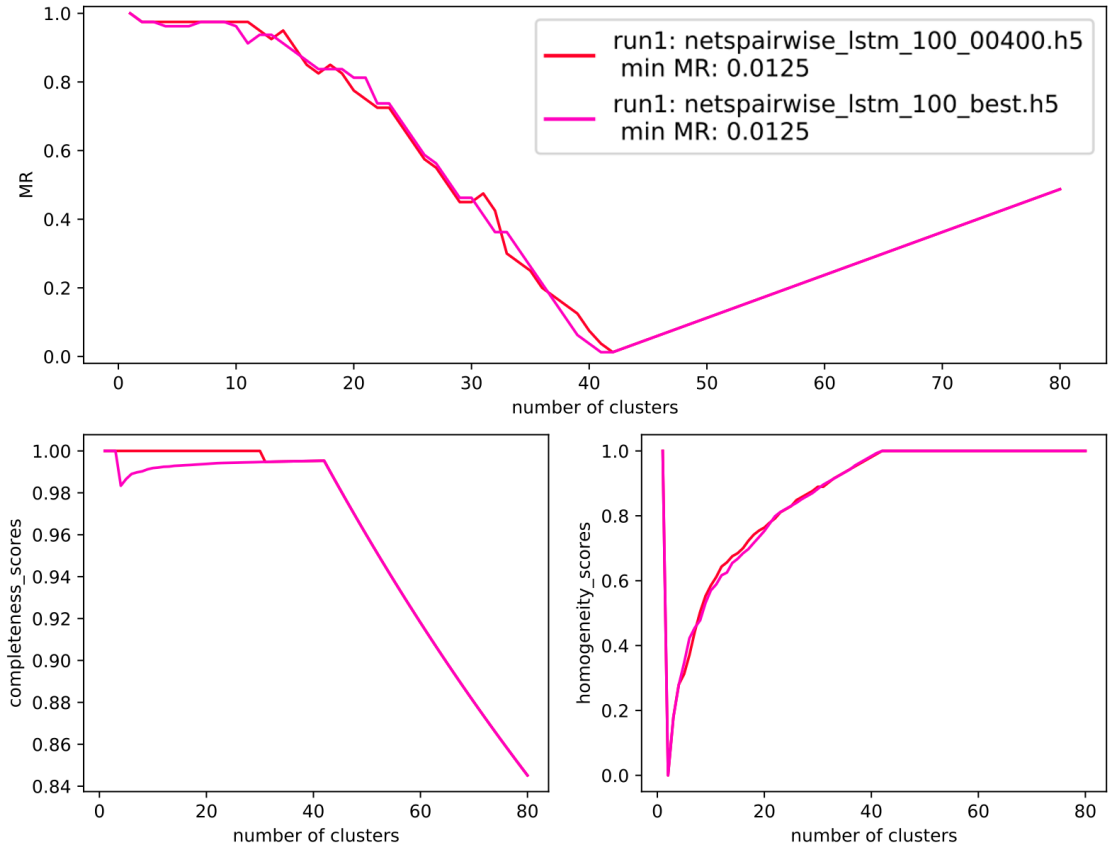


Abbildung 11: Das LSTM Netzwerk, welches nun State of the Art ist.

Die obere Graphik besitzt die gleichen Eigenschaften wie in Abbildung 8, X-Achse Anzahl Cluster und Y-Achse MR (oben), Completeness- (unten links) und Homogeneity Score (unten rechts). Wir betrachten hier ein Clustering von 40 Sprechern. Im Unterschied zu 8 zeigt Abbildung 11 nur die Resultate der besten Versionen vom Netzwerk Pairwise_lstm. Markant ist ebenfalls, dass der minimale MR bei 42 Clustern erreicht wird, was ebenfalls für das Netzwerk spricht. Die Completeness liegt bei 40 Sprechern auf 0.99 und die Homogeneity auf 1.0.

Bei den drei Auswertungen mit 40, 60 und 80 Sprechern ist aufgefallen, dass jeweils ein anderes Training am besten abgeschnitten hat. Im Schnitt liegen diese zwar nahe beieinander, doch ist es ein merkbarer Unterschied von einigen Promillen MR zu beobachten.

6 Ausblick

Nach Beendigung der Arbeit sehen wir weiteres Potenzial, um darauf aufzubauen.

Auswechseln der Trainingsdaten Weiterführend könnte an der Möglichkeit geforscht werden, die Trainingsdaten auszutauschen. Die TIMIT Daten können nun, da das Setup vereinheitlicht ist, durch einen anderen Datensatz getauscht werden. Sofern das Format der Trainingsdaten gleich bleibt wie in 3.4 beschrieben, ist dies problemlos möglich.

An welchen Stellen die Suite angepasst werden müsste, um dies zu ermöglichen steht im Anhang der Arbeit unter Weiterentwicklungsoptionen.

Erweiterung der Suite Die Suite bietet einiges an Komfort, an einigen Stellen können aber noch Verbesserungen vorgenommen werden. So wäre es komfortabel den "-val#" Parameter ebenfalls mit dem Argument "all" aufrufen zu können.
Im Moment werden mehrere Checkpoints, die den gleichen minimalen MR haben, im File "best" gespeichert. Um nur noch je einen Checkpoint pro Netzwerk zu erhalten, könnte eine der beiden anderen Metriken, der minimale MR bei der optimalen Clusteranzahl oder die Differenz der Clusteranzahl beim minimalen MR und der optimalen Clusteranzahl, verwendet werden.

LSTM Weiterentwicklung und Finetuning Das LSTM-Netzwerk von [3] hat gezeigt, dass ihre Architektur einem CNN überlegen ist. Zu überprüfen wäre, ob durch weiteres Finetuning der Hyperparameter oder der Netzwerkarchitektur eine Verbesserung der erreichten MR Werte möglich wäre.

Konkrete Implementierung Diese Arbeit bezieht sich immer auf ein Theoretisches Optimum der Clusteringleistung, legt aber keine konkrete Implementierung eines Anwendungsfalles offen. Die Überlegung, auf die eingegangen werden könnte, ist auf die Zuverlässigkeit von den paarweise vergleichenden Netzwerken zurück zu kommen.

Wir nehmen an, ein Audiostream einer Konferenz ist unser Ausgangsmaterial und die Aufgabenstellung ist es zu bestimmen, zu welcher Zeit welcher Sprecher spricht. In der Datenaufbereitung wird der Audiostream in 150ms-Stücke unterteilt und von jedem Abschnitt wird ein Spektrogramm erstellt. Im ersten Durchgang wird jedes Spektrogramm mit dem zeitlich zuvorkommenden Spektrogramm verglichen. Die Logik besagt, dass es häufig der Fall sein wird, dass derselbe Sprecher immer noch am Sprechen ist. Die beiden Spektrogramme werden über ein paarweise Trainiertes Netzwerk verglichen und wenn es der gleiche Sprecher ist, wird den beiden dieselbe Nummer zugewiesen und sie werden zusammen gruppiert. Ist es nicht der gleiche Sprecher wird die Sprechernummer inkrementiert.

Nach dem ersten Durchgang besitzen wir eine Sammlung von Zeitstrecken, in denen jeweils eine Person gesprochen hat und deren zugehörigen Spektrogramme. Im zweiten Durchgang wird jeweils eine Anzahl von Samples der einen Sammlung mit allen anderen Sammlungen paarweise verglichen. So wird ermittelt, welche der Sammlungen zum selben Sprecher gehören.

Es könnte interessant sein diesen Ansatz in einer Praxisorientierten PA oder BA aufzugreifen und ein lauffähiges Produkt zu erstellen. Die Basistechnologie, eine Trainingsumgebung und eine geeignete Netzwerkarchitektur sind bereits vorhanden.

Appendix

A Bedienungsanleitung

Um die Benutzung der Speaker Clustering Suite (SCS) möglichst angenehm zu gestalten, beinhaltet dieses Kapitel alle nötigen Schritte, um die Suite aufzusetzen und die Netzwerke zu trainieren und evaluieren.

A.1 Voraussetzungen

Um mit der SCS arbeiten zu können, muss der Benutzer / die Benutzerin im Besitz des TIMIT Datensatzes sein. Dieser ist zum Verkauf erhältlich bei der LDC[3]. Die von der SCS generierten Trainings- und Evaluationsdaten basieren auf den TIMIT Daten und können in der Suite aufbereitet werden.

Wenn die SCS nicht nur benutzt, sondern auch weiterentwickelt werden soll beachten sie den Abschnitt "Aufsetzen der Entwicklungsumgebung".

Die SCS ist über zwei Wege erhältlich:

1. Die Suite als Source-Code über Github herunterladen[4]
2. Die Suite als Dockercontainer vom zentralen Dockerrepository herunterladen[5]

TIMIT-Daten Aufbereitung

Die von der LDC erhaltenen TIMIT Daten sind roh in unserer Suite nicht zu gebrauchen. Es erfordert leider einige Schritte, um diese für die Neuronalen Netzwerke zur Verfügung zu stellen.

Die Daten aus TIMIT sind Sprecher zugeordnete Wav Dateien mit einem speziellen Header, der zur Benutzung in der Suite entfernt werden muss. Dies wird über das Tool sph2pipe[6] erreicht.

Nach Beendigung der Headerentfernung muss der TIMIT Ordner in den Dockercontainer gemountet werden, womit die konvertierten TIMIT Daten im Container/Source-Code an den richtigen Ort kopiert werden. Dies geschieht mit der Option "-v" beim ausführen des "docker run" Befehls. Der Runbefehl sieht in diesem Fall Folgendermaßen aus

"docker run --name my_container Containername -v /host/TIMIT/directory:/usr/local/scs/ common/data/training/TIMIT"

Internes Setup der Suite

Um die Vorbereitungen abzuschließen, müssen nun die TIMIT Daten verarbeitet werden, so dass diese später von den Neuralen Netzwerken verwendet werden können. Dies wird über das Bashscript "setup_suite.sh" erledigt, welches ebenfalls über Docker aufgerufen werden kann.

A.2 Wahl des Containers

Es gibt vier Container der SCS:

1. empty-cpu — Verpackt nur die Python Installation für CPU
2. empty-gpu — Verpackt nur die Python Installation für GPU
3. with-code-gpu — Python Installation für **GPU** und allen Code um lauffähig zu sein
4. with-code-cpu — Python Installation für **CPU** und allen Code um lauffähig zu sein

[3]LDC Homepage: https://www.ldc.upenn.edu/
[4]Github Url: https://github.com/stdm/ZHAW_deep_voice
[5]Docker Url: https://hub.docker.com/r/stdm/zhaw_deep_voice/
[6]sph2pipe: https://www.ldc.upenn.edu/language-resources/tools/sphere-conversion-tools

Wird als Docker Host ein Nvidea-Docker verwendet ist der Dockercontainertag "with-code-gpu" zu empfehlen. Hier müssen mindestens die TIMIT Daten gemountet werden. Zusätzlich kann zur leichteren Verwendung zusätzlich der Ordner "common/data/experiments/results" gemountet werden.

Steht nur der normale Docker Host zur Verfügung sollte der Tag "with-code-gpu" verwendet werden. Das restliche Vorgehen ist identisch zur Verwendung des Containers "with-code-gpu".

A.3 Anwendung auf dem GPU Cluster der ZHAW

Auf dem GPU Cluster der ZHAW sollte der "leere" Container verwendet werden, um die Verwendung möglichst einfach zu gestalten. Um das Projekt damit laufen lassen zu können, müssen alle Daten vom lokalen Computer auf das Cluster geladen werden.
Dazu muss eine Bash-Shell geöffnet werden (Note: Windows Shell funktioniert nicht) und der Befehl:

scp -r /c/some/local/dir/SCS-Folder [username]@gpulogin.cloudlab.zhaw.ch:/cluster/home/[username]/SCS-Folder

Nun pullen Sie das docker-image mit dem Befehl:

shifterimg pull stdm/zhaw_deep_voice:empty_gpu

Um auf eine neue Session vorzubereiten und das Training im Hintergrund laufen lassen zu können wird ein neuer Screen geöffnet:

screen -S username-session

Nach dem ausführen des Screen Befehls sind Sie in dem screen drin. Nun starten Sie das Image und den controller mit

srun --pty --ntasks=1 --cpus-per-task=2 --mem=64G --gres=gpu:1 --qos=student-long shifter --image=[imagename] python /cluster/home/[username]/SCS-Folder/controller.py -n all -train -test -plot

Nun hat das Training begonnen. Mit "**Ctrl+A, D**" wird die Konsole vom Screen getrennt. Zu einem späteren Zeitpunkt kann auf den screen zurückgekehrt werden mit dem Befehl:

screen -r username-session

Während der Arbeit kam es mehrfach dazu, dass beim Wiederherstellen eines Screens der Fehler "bad file descriptor" auftauchte oder der Konsolenoutput der Netzwerke nicht angezeigt wurde. Beim Fehler "bad file descriptor" kann es helfen sich die vorhandenen screens mit dem Befehl **"screen -ls"** anzeigen zu lassen und den Befehl zur Wiederherstellung des Screens mehrfach auszuführen.

Wenn der Konsolenoutput des Netzwerks nicht mehr zu sehen ist, sind wahrscheinlich mehrere Screens mit demselben Namen vorhanden. Dies kann ebenfalls über den Befehl **"screen -ls"** überprüft werden oder es wird direkt ein Fehler geworfen, der die verschiedenen offenen Screens mit der zugehörigen ID aufführt. In beiden Fällen müssen alle Screens getestet werden um denjenigen zu finden, der den Konsolenoutput enthält. Der Befehl, um aus mehreren gleichnamigen Screens einen bestimmten zu öffnen lautet:

screen -r sessionID.username-session

Die gesa mmte Anleitung befindet sich auf der Webseite des Datalab-Clusters[7]

[7] Datalab: https://gpu-cluster.cloudlab.zhaw.ch/

A.4 Aufrufen der Suite

Die gesamte Suite kann über das File controller.py gesteuert und alle verfügbaren Netzwerke trainiert und getestet werden. Die Parameter für das ausführen controller.py sind:

controller.py [-h] [-setup] [-n NETWORK] [-train] [-test] [-plot] [-clear] [-debug] [-best] [-val# NR]

- -h, --help Hilfe zu diesen Parametern aufrufen
- -setup Setup der Suite aufrufen und Sprecher Daten erzeugen
- -n Netzwerk Das Netzwerk, welches getestet/trainiert wird
  Verfügbare Netzwerke:pairwise_kldiv, pairwise_lstm, flow_me, luvo, all
- -train Flag ob das Netzwerk trainiert werden soll
- -test Flag ob das Netzwerk getestet werden soll
- -plot Flag ob die Ergebnisse geplottet werden sollen
- -clear Directories von logs und plots löschen vor dem Durchgang
- -debug LogLevel auf debug setzen für TensorFlow und den Logger
- -best Es werden nur die besten Resultate des Netzes im Befehl -plot verwendet.
- -val# NR Angabe der Sprecherzahl, die im Befehl -test verwendet werden soll.

Beispielaufrufe:

- python controller.py -setup
- python controller.py -n pairwise_kldiv -train -test -val# 60 -plot -best
- python controller.py -n pairwise_lstm -test -debug

Generelle Bemerkungen zur Bedienung

Der Controller sollte beim ersten Durchgang mit dem Setup-Flag laufen gelassen werden, dadurch werden alle benötigten Trainings- und Testdaten aus dem TIMIT Datensatz extrahiert und für die Verwendung in den Netzwerken aufbereitet.

A.5 Konfidenzintervall in einem Plot

Um das Konfidenzintervall eines Netzwerkes zu erhalten, können die verschiedenen "netzwerkname_best.pickel" Files nummeriert, "netzwerkname_best_1.pickel", im Ordner common/data/ experiments/results abgespeichert werden und der Controller mit dem Befehl "python controller.py -n netzwerkname -plot -best" aufgerufen werden. Dadurch werden nur die besten Resultate dieses Netzwerks in einem Plot gespeichert. Der niedrigste MR befindet sich immer am Anfang der Legende und der schlechteste am Ende.

B Erweitern der Speaker Clustering Suite

Die SCS ist auch dafür gedacht, dass neue Ansätze zeiteffizient entwickelt, getestet und verglichen werden können. Um die Suite zu erweitern, müssen ein paar andere Schritte vorgenommen werden als bei der reinen Benutzung.

B.1 Aufsetzen der Entwicklungsumgebung

Die Voraussetzungen aus der Benutzung bleiben gleich, bis darauf, dass hier der Source-Code heruntergeladen werden **muss**. Es empfiehlt sich hier das Github-Repository zu forken und die Files per Git zu clonen.

Python

Um die Suite von der Source her laufen lassen zu können, muss die Python Umgebung strickte nach folgender Anleitung aufgesetzt werden. Es empfiehlt sich dazu die Verwendung von Anaconda[8].

1. Miniconda herunterladen und installieren
   Von der Miniconda Webseite[9] die für euer System korrekte Version herunterladen.
2. Miniconda prompt starten
3. Ein neues conda Environment kreieren
   »conda create -n yourname python=3.5.2
4. Das neu kreierte Environment aktivieren
   »activate yourname
5. Folgende Packages in dieser Reihenfolge installieren:
   (a) »pip install --ignore-installed --upgrade tensorflow(-gpu)
   (b) »conda install numpy scipy theano=0.9.0 scikit-learn
   (c) »pip install --upgrade https://github.com/Lasagne/Lasagne/archive/master.zip
   (d) »conda install -c conda-forge keras=2.0.6
   (e) »conda install -c conda-forge librosa=0.5.1
   (f) »pip install https://github.com/dnouri/nolearn/archive/master.zip#egg=nolearn
   (g) »conda install pandas=0.20.3
   (h) »conda install -c omnia munkres=1.0.7
   (i) »conda install matplotlib=2.1.0

TensorFlow auf GPU

Die offizielle TensorFlow page[10] ist zumindest für die Installation in der Windows Version nicht spezifisch genug und kann einem schnell in die Irre führen. Die Version von TensorFlow, die für die Suite benutzt wird ist 1.3, deswegen beziehen sich folgende Tipps auf diese spezifische Version.
Der Blog "Tensorflow-gpu für Windows 10"[11] ist eine gute Ausgangslage für die Installation von TensorFlow-gpu auf Windows. Folgende Ergänzungen kommen zu diesem Blog-Artikel dazu:

- Installieren Sie exakt die Version 8.0 des Cuda Toolkits, welche nur erreichbar ist im cuda-toolkit-archiv[12]
- Installieren Sie die cuDNN Version 6.0 anstatt der im Artikel vorgeschlagenen 5.1
- Eventuell muss noch die Visual c++ Distribution 2015 installiert werden. Die 2017 Distribution sollte ebenfalls genügen[13].

[8] Anaconda Webseite: https://anaconda.org/
[9] Miniconda Webseite: https://conda.io/miniconda.html
[10] TensorFlow Installations Page: https://www.tensorflow.org/install/
[11] Tensorflow-gpu für Windows 10: http://blog.nitishmutha.com/tensorflow/2017/01/22/TensorFlow-with-gpu-for-windows.html
[12] https://developer.nvidia.com/cuda-toolkit-archive
[13] V C++ 2015: https://www.microsoft.com/de-ch/download/details.aspx?id=52685

Docker

Docker benötigt unter Windows das Feature Hyper-v. Dieses ist bei Windows 8, 8.1 und 10 nicht in den Home Versionen enthalten. Zum jetzigen Zeitpunkt gibt es keinen bekannten Weg dies zu umgehen.

Es existieren zwei Dockerfiles, diese installieren eine unterschiedliche Version von TensorFlow. Die verwendete TensorFlow Versionen sind: "1.3.0-py3" und "1.3.0-gpu-py3". Andere Versionen können verwendet werden, indem im Dockerfile der Tag durch den passenden Tag der Offiziellen Tensorflow Dockerseite [14] ersetzt wird.
Sollen die TIMIT Daten direkt im Dockercontainer enthalten sein, müssen noch die passenden Einträge im dockerignore File angepasst werden. Alternativ können diese auch gemountet werden, siehe dazu Abschnitt "TIMIT-Daten Aufbereitung".

Um einen neuen Dockercontainer zu builden kann der Befehl
'docker build -t my_container path/to/dockerfile_folder' ausgeführt werden.
Der Name muss kleingeschrieben werden. Beim Ausführen eines Docker Containers ist zu beachten, dass der Befehl "docker run --name my_container Containername" jeweils einen neuen Dockercontainer startet. Um einen bereits gestarteten Container zu stoppen, wird der Befehl "docker stop my_container" verwendet. Ein gestoppter Container kann mit "docker start my_container" mittels "docker exec -t -i id /bin/bash" wird die Konsole des Dockercontainers geöffnet.

B.2 Common Bereich des Codes

Code oder Ressourcen, welche für alle Netzwerke von Relevanz sind befinden sich im Ordner *commons*. Es lohnt sich die Methoden dieser Packages zu überfliegen, um Codeaufwand und Duplikation zu vermeiden. Hier eine Kurzbeschreibung der Unterpackages

- Data: Alles, was keine Python Files sind, sollte hier liegen
  - Experiments: Logs, Plots, Resultate und Netzwerke
  - Sprecher Listen: Listen mit Sprechernamen
  - Trainingsdaten: TIMIT, in wav und pickle Form
- Analysis: Methoden für die Bewertung von MR und anderen Metriken
- Clustering Methoden zur Generierung und dem Clustering von Embeddings
- Extrapolation: Methoden für die Aufbereitung von TIMIT Daten
- Spectrogram: Methoden für die Spektrogramerzeugung und Verarbeitung
- Utils: Verschiedene Utilitymethoden, wie z.B. *Paths*

B.3 Neues hinzufügen oder bestehendes Verändern

Ein neues Netzwerk hinzufügen

Jedes Netzwerk in der SCS besitzt seinen eigenen Ordner unter "networks", der über einen Network-Controller angesteuert wird. Außerhalb des Packages wird nur dieser NetworkController verwendet.

Der abstrakte Klasse NetworkController ist in **common/network_controller.py** zu finden und besitzt zwei Abstrakte Methoden, welche implementiert werden müssen.

1. train_network:
   Bringt das Training des Netzwerkes in Gange. Dies ist verantwortlich für das Loggen von Aktivitäten, Plotten von Trainings/Validationsloss und bei Beendigung des Trainings auch für das Abspeichern vom Netzwerk.

[14] Tensorflow Dockerseite: https://hub.docker.com/r/tensorflow/tensorflow/

2. get_embeddings:
   Lässt die im NetworkController gespeicherte Validationsdaten durch das trainierte Netzwerk und liefert die Embeddings zurück, welches aus einem der Layer extrahiert wurden.

Im controller.py muss der neue NetworkController importiert werden und in der Methode "generate_controllers" ein Dicteintrag erstellt werden. Der Rest wird von der Suite übernommen. Die Trainingsdaten stehen zur Benutzung bereit, die Analyse und das Plotten der Ergebnisse wird aufgrund der Embeddings automatisch ausgeführt.

Neuer Datensatz verwenden

Um die bestehenden oder neuen Netzwerke mit einem neuen Datensatz trainieren zu können, müssen verschiedene Systeme angepasst werden. Die Suite ist bereits sehr gut darauf vorbereitet einen weiteren Datensatz einzubinden:

- **common/data/training**
  In diesem Ordner muss ein neuer Unterordner erstellt werden, welcher den Datensatz enthält.
- **common/extrapolation/speaker.py**
  Das Setup der Suite ist auf das Extrahieren von "Sprechern" ausgelegt. Die Klasse Speaker besitzt eine Variable **dataset**, welche anders als TIMIT übergeben werden müsste. Zudem müsste für die Verarbeitung der neuen Daten eine neue Methode implementiert und in der Methode **extract_data_from_speaker** eingebettet werden.
- **common/extrapolation/speaker_factory.py**
  In der Factory müssen die neuen Speaker aufgeschrieben werden, damit sie beim nächsten Ausführen von "controller.py -setup" durchgeführt werden.
- **networks**
  Je nach Anspruch kann nun im Netzwerkcode einer der neuen Speakerpickle als Trainingsdaten oder Testdaten ausgewählt werden.
- **controller.py**
  Falls es sich um Testdaten handeln soll, können diese im init Teil des Controllers zum dict "validation_data" hinzugefügt werden.

Clustering Metrik oder Embedding-Linkage Methode ändern

In Zukunft könnte es von Vorteil sein, die Clustering Metrik (welche Momentan "cosine" ist) und/oder die Embedding-Linkage Methode zu ändern (im Moment "complete"). Hierzu kann auf zwei Wege zugegriffen werden:

- **Statisches Verändern**
  Im File **common/clustering/generate_clusters.py** steht eine einzige Methode deren Defaultparameter für "metric" und "method" verändert werden können. Diese kann nach Belieben angepasst werden.
- **Dynamische Einstellung**
  Die oben genannte Methode wird aus dem NetworkController aufgerufen und könnte dort die optionalen Parameter setzen. In **controller.py** könnte dem Argumentparser neue Argumente hinzugefügt werden, welche ein Feld im Controller setzen. Dieser kann die Felder in der Methode "generate_controllers" den angesteuerten NetworkControllern übergeben.

C Glossar

Suite - In Software wird von einer Suite gesprochen, wenn ein Zusammenschluss von mehreren Programmen in einer Applikation besteht. Wie eine Musikalische Suite aus mehreren Stücken besteht, sind hier die Komponenten eigenständige Softwareprogramme. Der Zweck bei Software ist oft die Bündelung von Stärken und das minimieren von Organisatorischem Aufwand.

Cherrypicking - Beschreibt ein Vorgang, meistens in der Datenanalyse, vom herauslesen der besten Resultate einer Testreihe.
Wenn ursprünglich 1000 Samples ausgewertet wurden, aber nur 300 davon werden für die Studie oder die finale Auswertung betrachtet, weil diese auf das gewünschte Ergebnis schließen lassen, bezeichnet man diesen Vorgang als Cherrypicking.

IDE - Integrated Developement Environment, Integrierte Entwicklungsumgebung.
Ein Programm, welches ein Codeeditor ist und es erlaubt den geschriebenen Code zu verwalten und zu kompilieren. Oft ein nützliches Instrument der Softwareentwicklung, denn es bietet angenehme Funktionen, die das Entwickeln von Software beschleunigen oder vereinfachen. Vor allem für große Projekte hilft eine IDE beim Suchen oder/und lesen des Codes.

Hyperparameter - In einem Neuronalen Netzwerk werden einige Parameter vom Training verändert und andere nicht. Jene, die "händisch" gesetzt werden, werden Hyperparameter genannt und auch während des Trainings nicht geändert.

GPU / CPU - Graphik Processing Unit / Central Processing Unit, Graphische Recheneinheit / Zentrale Recheneinheit.
Die CPU ist eine Allzweckrecheneinheit, welche Arithmetik und Logik beherrscht. Die GPU auf der anderen Seite ist eigentlich dafür gedacht, bei graphischen Prozessen zu unterstützen. Diese wird aber oft in Neuronalen Netzwerken gebraucht, weil sie eine große Anzahl Gleitpunktkommazahloperationen in einer Sekunde berechnen kann.

Screen (Kontext GPU Cluster) - Ausführungsumgebung auf einem Linuxsystem. Ein Screen ist eine Shell, die eigenständig läuft und im Hintergrund bleibt, wenn der Benutzer es nicht braucht. So kann ein Experiment laufen gelassen werden, ohne dass dieses ständig überwacht werden muss.

Surrogate-Task - Ein Task, der anstelle eines anderen Tasks verwendet wird, im Zusammenhang mit dieser Arbeit zum Beispiel der Task der Sprecher-Identifikation anstelle des eigentlichen Task, dem Sprecher-Clustering.

D Verzeichnisse

Literatur

[1] Y. Lukic, C. Vogt, O. Dürr, and T. Stadelmann, "speaker identification and clustering using convolutional neural networks," *MLSP*, pp. 1–6, 2016.

[2] Y. Lukic, C. Vogt, O. Dürr, and T. Stadelmann, "learning embeddings for speaker clustering based on voice equality," *MLSP*, pp. 1–6, 2017.

[3] P. Gerber and S. Glinski, "Machine learning for speacker clustering," *BA Thesis*, pp. 5–57, 2017.

[4] T. Gygax and J. Egli, "Speaker clustering mit metric embeddings," *BA Thesis*, vol. 0, pp. 5–49, 2017.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, p. 2278–2324, 1998.

[6] T. Stadelmann, "Voice modeling methods for automatic speaker recognition," *PhD Thesis*, 2010.

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

[8] 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, p. 185–190, 1937.

[9] D. O'Shaughnessy, *Speech Communications: Human and Machine*, vol. 2. 1999.

[10] S. Kullback and R. A. Leibler, *Annals of Mathematical Statistics*, vol. 22. 1951.

[11] E.Hoffer and N.Ailon, "Semi-supervised deep learning by metric embedding,"

[12] D. Kingma and J. Ba, "Adam: A method for stochastic optimization,"

[13] M. Kotti, V. Moschou, and C. Kotropoulos, "Speaker segmentation and clustering," *Signal processing*, vol. 88, no. 5, p. 1091–1124, 2008.

[14] B. Meier, T. Stadelmann, and O. Dürr, "Learning to cluster - investigations in deep learning for end-to-end clustering.".

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Entwurfsmuster*. München: Addison-Wesley Verlag, 2004.

[16] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "Timit acoustic-phonetic continuous speech corpus,"

[17] M. Rouvier, P.-M. Bousquet, and B. Favre, "Speaker diarization through speaker embeddings," *Signal Processing Conference (EUSIPCO)*, vol. 86, no. 23rd European, p. 2082–2086, 2015.

[18] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," 1999.

Abbildungsverzeichnis

Tabellenverzeichnis

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

Deep Learning für Speaker Clustering
PA17_stdm_3

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

Kurzbeschreibung:
Hintergrund und Ziel

Seit mehreren Jahren führen die Betreuer mit grossem Erfolg studentische Arbeiten im Gebiet der automatischen Stimmererkennung durch (siehe das Ergebnis einer PA im Link unten, vorgestellt durch die Studierenden auf einer internationalen Machine Learning Konferenz). Im Rahmen dieser Arbeit soll unser State-of-the-Art Ansatz zum Clustern von Stimmen mittels Deep Neural Networks (DNNs) nochmals verbessert werden. Verschiedenen Methoden bieten sich hierfür an und werden zu Beginn der Arbeit mit den Betreuern genauer eingegrenzt:

- Generelles Ende-zu-Ende Clustern mit einem DNN in "einem Aufwasch" durch "k-Means Layer"
- Verwendung einer optimierten (z.B. tiefen, mehrdimensionalen) RNN Architektur (Recurrent Neural Network)
- Evaluation optimierter Loss-Funktionen (z.B. [Romanov & Rumshisky, 2017])
- Einbeziehung von unüberwachtem Lernen (z.B. [Meyer et al., 2017])

Vorgehen

- Einarbeiten in den aktuellen Ansatz zum Speaker Clustering (Python / TensorFlow) und DNNs im Allgemeinen
- Implementieren und evaluieren des gewählten Verbesserungsansatzes
- Herausarbeiten von Argumenten für die beobachteten Resultate
- Verortung der Ergebnisse im Stand der Forschung

Voraussetzungen:
Es wird kein Vorwissen über Deep Learning oder TensorFlow vorausgesetzt. Alles kann im Laufe der Arbeit erlernt werden. Notwendig sind gute Programmierkenntnisse sowie Lust und die Fähigkeit, sich eigenständig in wissenschaftliche Literatur einzuarbeiten, diese zu verstehen und umzusetzen. Am wichtigsten ist Freude und Leidenschaft für das Thema. Ein Besuch Data Science-relevanter Vorlesungen wie STDM, IE1/2 oder KI ist von Vorteil.

Die Arbeit ist vereinbart mit:
Benjamin Heusser (heussben)
Savin Niederer (niedesav)

Weiterführende Informationen:
https://www.zhaw.ch/no_cache/de/forschung/personen-publikationen-projekte/detailansicht-publikation/publikation/210537/