Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Projektarbeit
HS16 Studiengang Informatik

Speaker Identification mit Reccurrent Neural Networks

Autor Patrick Gerber

Hauptbetreuung Thilo Stadelmann

Datum 22. Dezember 2016

Zusammenfassung

In den letzten Jahren wurden grosse Forschritte im Bereich Sequence Learning, insbesondere mit Recurrent Neural Networks (RNNs) erzielt.Es werden kontinuirlich neue Anwendungsgebiete für den Einsatz dieser Netzwerke gefunden. Daher wird in dieser Arbeit In dieser Arbeit wird die Tauglichkeit von Reccurent Neural Networks, im Speziellen Long Short Term Memory Networks und deren Kombination mit Convolutional Neural Networks (CNNs) auf den Problembereich der Speaker Recognition untersucht.

Wärend in den Bereichen Speach Recognition, Language Translation und Speech Pocessing sowie Image Captioning RNNs erfolgreich eingesetzt werden, gibt es für die Anwendung im Bereich der Speaker Recognition noch kaum Forschungsarbeiten.

Die Untersuchungen beziehen sich auf das Problem der Speaker Identification. Die Arbeit zeigt das Long Short Term Memory Networks (LSTMs) sehr gute Ergebnisse liefern bezüglich der Speaker Identification. Mit einer Segmentlänge von 150 Millisekunden erreicht das beste Netzwerk eine Genauigkeit von 100% auf dem TIMIT Datensatz mit 100 Sprechern. Somit wird eine Accuracy erreicht welche auf dem Niveau von bewährten statistischen Verfahren ist, ohne diese Netzwerke und Eingabedaten ausgiebig zu tunen. Die Architektur welche für die Kombination von CNNs und LSTMs verwendet wurde, brachte nicht den gewünschten Erfolg. Diese ezielen schlechtere Resultate als das LSTM und erreichen auch die Werte von CNNs aus vorherigen Arbeiten nicht. Mit dieser Arbeit wird ein gute Grundlage für die Verwendung von LSTMs im Problembereich der Speaker Recognition gelegt.

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 Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Inhaltsverzeichnis

1. Einleitung

1.1. Ausgangslage

Die Projektarbeit wird an der ZHAW am Institut für angewandte Informationstechnologien(InIT) unter der Betreuung von Thilo Stadelmann durchgeführt. Die vorherigen Arbeiten zu diesem Thema beschäftigten sich mit der Anwendung von Convolutional Neural Networks auf die Problemstellung der Sprechererkennung. Im Bereich der Sprechererkennung mit Convolutional Neural Networks haben Lukic und Vogt mit ihren Arbeiten sehr gute Resultate erzielt [15] [17] [16]. Für die Sprechererkennung mit Recurrent Neural Networks gibt es bis anhin nur wenige Arbeiten. J.Larrson [13] verwendete in seiner Arbeit Long Short Term Memory Networks und Audiodaten aus Audiobüchern um Speakeridentification zu erforschen, Die Test Accuracy lag dabei bei 94%, wurde jedoch ein anderes Audiobuch verwendet, konnten die Sprecher nicht mehr erkannt werden.

1.2. Motivation

Speaker Recognition, auf Deutsch Sprechererkennung, kann in vielen Bereichen eingesetzt werden. Sei es für biometrische Zugangskontrollen, automatisches Protokollieren von Meetings oder Telefonkonferenzen. Auch im Bereich der Überwachung des öffentlichen Raumes können Sprechererkennungssysteme verwendet werden. Bis anhin wurden vor allem statistische Methoden, sowie Convolutional Neural Networks (CNNs) für die maschinelle Sprechererkennung eingesetzt. Der Erfolg von Reccurent Neural Networks (RNNs) im Sequence Learning motivierte uns dazu, diese auf das Problem der Speaker Recognition anzuwenden. Da Sprache eine zeitliche Komponente besitzt, sollten RNNs, welche zeitliche Zusammenhänge erkennen können, eine Bereicherung für den Problembereich sein.

1.3. Zielsetzung

Es wird die Tauglichkeit von Sequence Learning mit Reccurent Neural Networks (RNN), im speziellen Bidirectionl Long Short Term Memory Networks (LSTM), im Bezug auf die Problemstellung der Speaker Identification untersucht. Zu diesem Zweck werden verschiedene LSTM Architekturen, sowie eine Implementation der Kombination von CNNs und LSTMs untersucht.

1.4. Aufbau

Zum beseren Verständnis der Arbeit wid hier eine kurze Beschreibung des Aufbaus gegeben.

- Kapitel 2 stellt die zum Verständnis der Arbeit wichtigen Grundlagen vor . Es wird der Problembereich der Speaker Recognition vorgestellt, sowie eine Einführung in die Welt der Neural Networks gegeben.
- Kaptiel 3 behandelt das Vorgehen und Methodik welche bei der ausführung der Experimente angewandt wird. Des weiteren werden die Architekturen der verwendeten Neural Networks erläutert und es wird auf die verwendeten Technologien verwiesen.

- Kapitel 4 legt die einzelnen Experimente dar. Die Ausgangslage und die Erwartungen werden erläutert. Die erhaltenen Resultate werden aufgezeigt und analysiert.
- Im Kapitel 5 wird ein Fazit über die ausgeführten Experimente und deren Resulate gezogen.
- Kapitel 6 gibt einen Ausblick auf mögliche Verbesserungen der verwendeten Netzwerke und weitergehende Untersuchungen.

2. Grundlagen

2.1. Speaker Recognition

Der Problembereich der Speaker Recognition beschäftigt sich mit der Sprechererkennung. Im Gegensatz zu Speech Recognition ist es nicht von Interesse was gesagt wird, es geht nur darum welcher Sprecher etwas gesagt hat. Das Feld der Speaker Recognition kann in drei grundlegende Kategorien unterteilt werden [10].

- **Speaker Verfication:** Beschäftigt sich mit der Verifikation eines Sprecher, dies kommt zum Beispiel bei Stimmverifikationen zur Anwendung, bei denen entschieden werden muss, ob die Stimme zur Person gehört.
- **Speaker Identification:** Bei Speaker Identification versucht man die Stimme eines Sprechers einer beliebigen Anzahl dem System bereits bekannter Sprecher zuzuordnen.
- **Speaker Classification:** Das Problem der Speaker Classification kann als der Versuch, eine noch nie gehörte Stimme einer Gruppe zuzuordnen, beschrieben werden. Problembereiche hier wären, das Zuordnen des Geschlechts eines Sprechers, oder die Stimme einer Altersgruppe zu zuweisen. Auch das Speaker Clustering, bei welchem versucht wird verschiedene Stimmen auseinander zu halten, gehört in diese Problembereiche.

Um die Stimme für ein computergestütztes Verfahren, verständlich zu machen, muss diese zuerst in brauchbare Daten umgewandelt werden. Eine Möglichkeit Audiosignale für den Computer verwertbar zu machen ist die Waveform. Ein Diagramm einer solchen Waveform ist in der Abbildung 2.1 zu sehen. Sie repräsentiert den vier Sekunden langen Satz „She had your dark suit in greasy wash water all year“. Da diese Repräsentation von Audiodaten, die für die Sprechererkennung wichtigen Merkmale in nur zwei Dimensionen enthält ist es sinvoll eine andere Repräsentation der Daten zu verwenden [25]. Es ist sinnvoll

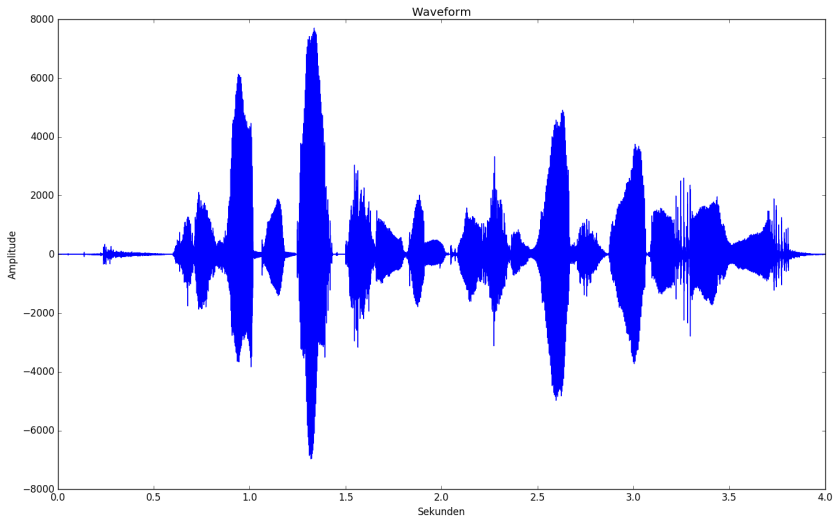


Abbildung 2.1.: Waveform des Satzes “She had your dark suit in greasy wash water all year”, mit Samplerate 16 Kilohertz

die verschiedenen Aspekte der Sprache in mehrere Dimensionen zu extrahieren. Um Frequenz, Lautstärke und zeitliche Verteilung in einer Repräsentation zu erhalten, können Spektrogramme verwendet werden.

Spektrogramme

Spektrogramme benutzen die Fourier-Transformation, um aus der Waveform die Frequenz und ihre Lautstärke zu extrahieren. Um die zeitliche Komponente zu bewahren wird die Fourier-Transformation für kleine Zeitfenster angewandt, diese werden danach zusammengefügt [25] (S.39-40). Spektrogramme sind gut dazu geeignet Audiodaten zu visualisieren. Sie ermöglichen es, Frequenz, Amplitude und zeitliche Information in einem zweidimensionalen Array darzustellen. In vorherigen Arbeiten kamen oft sogenannte Mel-Spektrogramme zum Einsatz. Die Mel-Skala bildet ab, wie das menschliche Gehör Frequenzen wahrnimmt [25] (S.52). Bei Mel-Spektrogrammen wird die Frequenzachse auf die Mel-Skala projiziert. Die dafür verwendete Formel ist wie folgt:

$$M(f) = 1125 * ln(1 + \frac{f}{700}) \tag{2.1}$$

Dabei steht f für die Frequenz. Das Mel-Spektrogramm für den Satz „She had your dark suit in greasy wash water all year“. Ist in Abbildung 2.2 zu sehen.

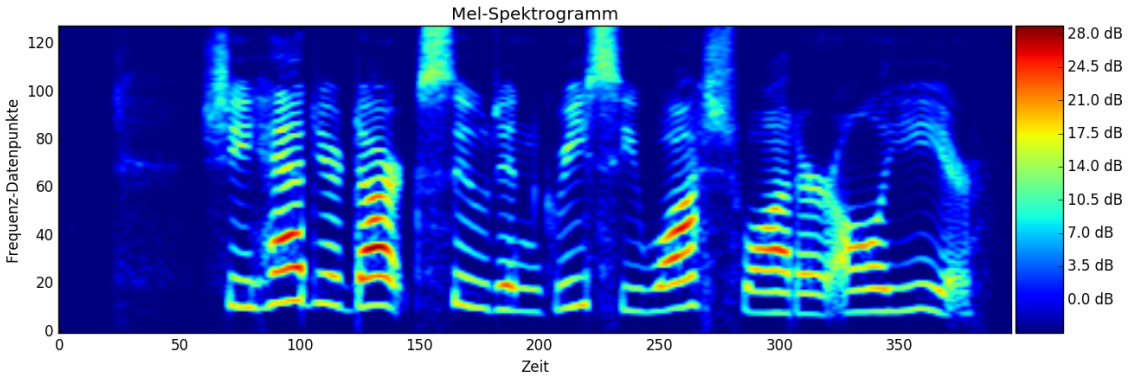


Abbildung 2.2.: Mel-Spektrogramm des Satzes “She had your dark suit in greasy wash water all year”.

2.2. Deep Learning und Neural Networks

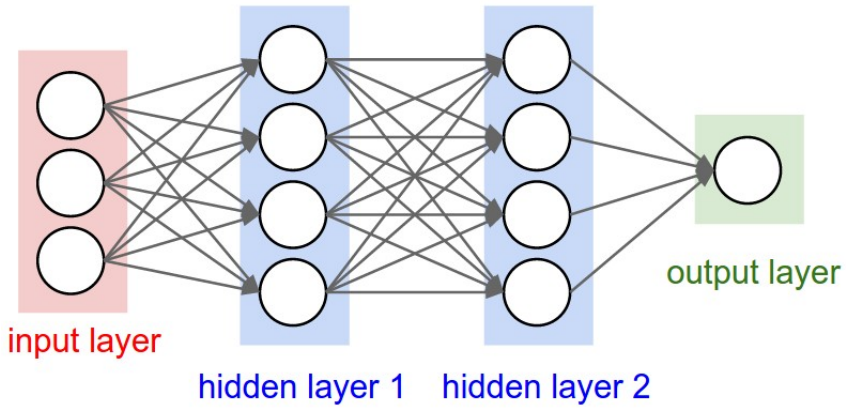


Abbildung 2.3.: Neural Network mit 2 Hidden layers[11]

Als Deep Learning werden Machine Learning Verfahren bezeichnet, welche mehrere Repräsentationsebenen, sogenannte Layers, aufweisen[14]. Dabei werden die einzelnen Layers nicht von Ingenieuren entwickelt, sondern lernen durch Anwendung von generellen Lernalgorithmen. Deep Learning wird oft mit Neural Networks gleichgesetzt, da diese eine sehr verbreitetete Implementation des Konzepts sind. Da für die nachfolgenden Ausführungen gewisse Grundlage betreffend dem Verständnis von Neural Networks notwendig sind, erfolgt hier eine Einführung in die Thematik. Ein Neural Network besteht aus einem Input Layer , $1....n$ Hidden Layers und einem Output Layer. Die in Abbildung 2.3 dargestellten Hidden Layers werden auch als Dense bzw. Fully Connected Layer bezeichnet. Jeder Layer besitzt mehrere Neuronen, welche Inputs für jedes Neuron des vorhergehenden Layers besitzen, sowie Outputs zu den Neuronen des nachfolgenden Layers.

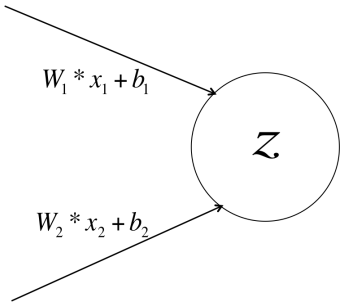


Abbildung 2.4.: Neuron mit 2 Inputs

Jede Verbindung zwischen Neuronen hat ein Gewicht und einen Bias, welche durch eine Weightmatrix W und den bias b dargestellt werden. Daher ergibt sich für z :

$$z = \sum_{i \in Input} W_i * x_i + b_i \tag{2.2}$$

Jedes Neuron besitzt eine Aktivierungsfunktion. Es gibt verschiedene Varianten von Aktivierungsfunktionen. Die in den letzten Jahren am häufigsten verwendete Funktion ist das Rectified Linear Unit (ReLU)[14]

$$f(z) = max(0, z) \tag{2.3}$$

Eine weitere klassische Aktivierungsfunktion ist die Sigmoid Funktion, welche of bei Output Layern zur Anwendung kommt.[14] .

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

Training

Für das Training von Neural Networks können zwei verschiedene Strategien eingesetzt werden [5] (S.105):

- **Supervised Learning:** Das Netzwerk lernt über einen Datensatz. Zu jedem Trainingsbeispiel existiert ein Label, im Falle von Speaker Identification wäre dies der zum Satz gehörende Sprecher. Das Netzwerk versucht den Input einem dieser Label zuzuweisen. Dieses Verfahren wird, wie im Falle der Speaker Identification, oft für Klassifizierungaufgaben verwendet.
- **Unsupervised Learning:** Das Netzwerk lernt über einen Datensatz, und versucht selber relevante Features des Datensatzes zu extrahieren. Im Gegensatz zum Supervised Learning erhält das Netzwerk bei Anwendung dieser Strategie keine Information über die Art der Eingabedaten. Diese Strategie wird zum Beispiel bei Clusteringverfahren eingesetzt.

Die nachfolgend beschriebenen Konzepte beziehen sich auf das Supervised Learning Model.

Accuracy

Die Accuracy gibt an wie gut ein Netzwerk die Inputs klassifiezieren kann. Eine Accuracy nahe 1 bedeutet, dass das Netzwerk fast alle Inputs der richtigen Klasse zugewiesen hat. Die Funktion ist im Folgenden definiert, wobei k die Anzahl richtig klassifizierter Inputs definiert und N die Gesamtmenge aller Inputs.

$$accuracy = \frac{k}{N} \tag{2.5}$$

Cost Function

Die Cost Function, auch Loss Function genannt, kann so verstanden werden, dass sie die Cross Entropy der Datenverteilung des Datensatzes und die Verteilung der berechneten Output Daten darstellt [5] (S. 178-179). Etwas einfacher ausgedrückt, kann die Funktion als Differenz zwischen dem Erwarteten Output Vektor und dem vom Netzwerk generierten Output Vektor aufgefasst werden. Eine relativ simple Loss Function ist im Folgenden aufgeführt:

$$C = \frac{1}{2n} \sum_{x} (y(x) - a)^2 \tag{2.6}$$

In dieser Funktion steht n für die Menge aller Trainings Inputs, x ist ein Input Vektor, $y(x)$ ist der erwartete Output Vektor für x und a ist der effektive Output Vektor den das Netzwerk für Input x liefert. Diese Funktion wird als Mean Squared Error bezeichnet. Das Ziel des während des Trainings ist es nun diese Cost Function zu minimieren. Da der Output Vektor von allen Weights im Netzwerk abhängt, wird durch das Ändern dieser Weights eine Minimierung angestrebt.

Backpropagation

Während der Berechnung des Outputs eines Netzwerks fliesst die Information immer vorwärts vom Input bis zum Output. Um das Netzwerk trainieren zu können, muss nun der Wert der Cost Function dem Netzwerk zurückgegeben werden. Dies geschieht durch den Backpropagation Algorithmus, welcher den Wert der Cost Function rückwärts durch das Netzwerk fliessen lässt und so für jede Verbindung zwischen den einzelnen Neuronen den Gradient berechnet. Dieser kann dann verwendet werden, um mithilfe eines Gradient-Descent Algorithmus die Weights anzupassen [5] (S.204).

Overfitting

Das Phänomen des Overfittings tritt in allen Machine Learning Bereichen auf. Es beschreibt die Tatsache, dass der Validierungsfehler immer grösser wird, während der Trainingsfehler weiter abnimmt oder gleich bleibt. Im Extremfall nimmt der Validierungsfehler nie ab [5] (S. 111). Eine Methode Overfitting in Neural Networks zu verhindern sind Dropout-Layer, welche nach jedem Trainingsdurchgang zufällige Verbindungen auf Null setzen. [24].

Batch

Für das Training eines Neural Nets werden die Input Daten in Batches (auch Mini Batches genannt) aufgeteilt. Die Cost Function wird dann jeweils für die im Batch enthaltenen Daten berechnet und Backpropagation wird erst durchgeführt, nachdem alle Daten des Batches verarbeitet wurden. Von einer Epoche wird gesprochen wenn alle Daten des Trainingsdatensatzes durch das Netzwerk verarbeitet wurden. Wenn man also einen Trainingsdatensatz mit 320 Inputs hat und eine Batchgrösse von 32, werden die 320 Inputs in 10 Batches aufgeteilt und nach jedem Batch werden die Weights im Netzwerk angepasst. Wenn alle Batches verarbeitet wurden, ist eine Epoche abgeschlossen.

2.3. Convolutional Neural Networks

In diesem Abschnitt werden Convolutional Neural Networks (CNNs) vorgestellt, welche grosse Fortschritte im Problembereich der Klassifizierung bringen. [14]. Diese Problembereiche umfassen die Object Recognition in Bildern, Speaker Identification [15] und Speaker Clustering [17]. Die Architektur der CNNs besitzt zwei wichtige Layer, welche sich von den zuvor vorgestellten Dense Layern unterscheiden, den Convolutional Layer und den Pooling Layer. Nachfolgend werden die Funktionsweise von Convolution und Pooling erläutert.

Convolution

Convolution ist eine Operation, bei welcher über eine Input Matrix M eine Matrix K, in der Literatur meist Kernel genannt, gelegt wird. Wie Abbildung 2.5 zeigt, werden die korrespondierenden Elemente von M und K miteinander multipliziert und alle Werte aufsummiert. Danach wird der Kernel verschoben und die Operation wiederholt.

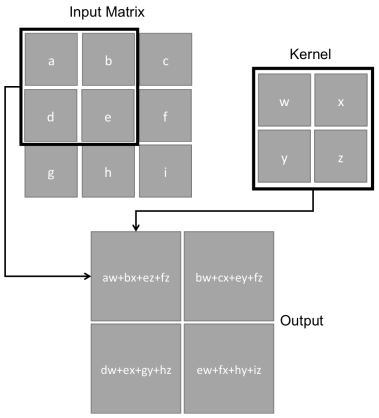


Abbildung 2.5.: Beispiel einer Convolution auf einer $3x3$ Matrix mit einem $2x2$ Kernel

Convolution wird in der Bildverarbeitung als Filter verwendet um zum Beispiel Kanten im Bild hervorzuheben. Convolution dient in CNNs der Extraktion von Features. Man kann sich vorstellen, dass die

Hervorhebung der Kanten nützlich ist, um Objekte zu erkennen und diese von der Umgebung abzugrenzen [5].

Pooling

Beim Pooling wird eine Matrix in nicht überlappende Teilmatrizen eingeteilt. für eine $4x4$ Matrix M könnten so Rechtecke der Form $2x1, 2x2, 1x2, 1x4$ oder $4x1$ gewählt werden. Für CNNs wird meist das Maxpooling verwendet, welches den Maximalwert der sich im Ausschnit befindenden Werte nimmt. Die Abbildung 2.6 veranschaulicht das Verfahren.

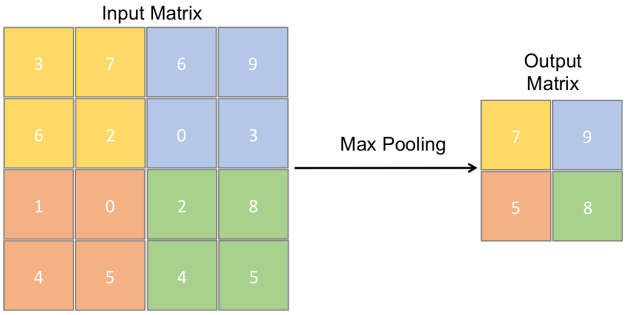


Abbildung 2.6.: Maxpooling einer $4x4$ Matrix mit enem $2x2$ Filter

Aufbau von CNNs

Der im Folgenden beschriebene Aufbau eines Beispiel CNNs erfolgt in Anlehnung an Nielsen [19] („Neural Networks and Deep Learning“, Kapitel 3). Zur Veranschaulichung dient Abbildung 2.7. Als Eingabe wird ein Graustufenbild mit der Grösse $28x28$ verwendet. Dieses wird in einen Convolutional Layer gespeist, im dargestellten Fall, mit einer Kernelgrösse von $5x5$ und einer Schrittweite 1. Nun wird die Convolution Operation auf das Input Bild angewendet. Die als Output generierte Matrix besitzt die Form $24x24$ und wird als Feature Map bezeichnet. Ein Convolutional Layer hat mehrere solcher Feature Maps, im Beispiel besitzt der Layer drei solche Feature Maps. Am Ende des Prozesses wird üblicherweise die ReLU Aktivierungsfunktion auf den Layer angewendet. Dieser Output dient als Input für den Pooling Layer. Auf jede $24x24$ Feature Map wird nun ein Maxpooling mit einem $2x2$ Kernel angewandt, dabei überlappen sich die Abschnitte nicht. Der Output hat die Dimension $3x12x12$, was drei Feature Maps mit jeweils $12x12$ Neuronen entspricht. Dieser Output wird nun an einen normalen Dense Layer übergeben. In der Praxis eingesetzte CNNs haben oft mehrere Schichten von Convolutional und Pooling Layers sowie mehrere Dense Layers.

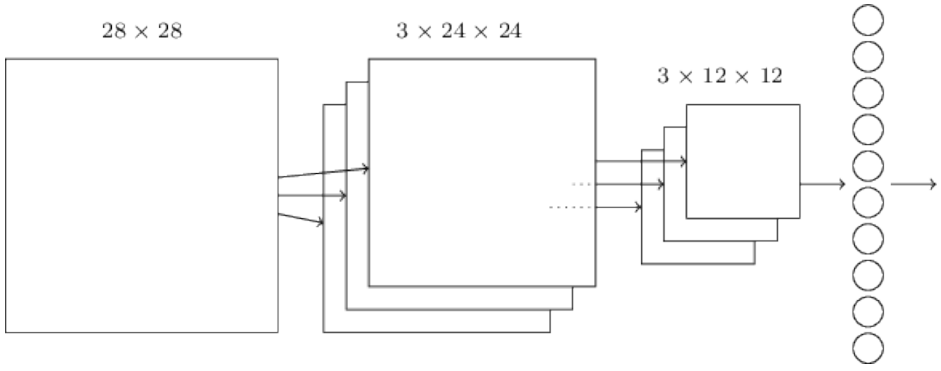


Abbildung 2.7.: Aufbau eines CNN mit einem Convolutional und Pooling Layer sowie Output Layer [19]

2.4. Reccurrent Neural Networks

Die bisher vorgestellten Neural Networks sind nicht in der Lage, Information über zeitliche Relationen zu speichern. Reccurent Neural Networks (RNN) sind Netzwerke, welche diese Schwäche lösen sollen. Es handelt sich um Netzwerke, welche einen Loop besitzen, damit die vorherige Information persistiert werden kann [20]. Die Abbildung 2.8 zeigt ein ausgerolltes RNN, wobei jedes Netzwerk A einen Timestep beinhaltet. Man sieht also, dass der Ausgang sowohl vom Eingang X_t als auch vom Ausgang des vorherigen Netzwerks A_{t-1} abhängig ist. Für das Training solcher Netzwerke wird Backpropagation Through Time angewandt, welche mit der Backpropagation für normale Neural Networks vergleichbar ist, aber über die Timesteps den Gradient berechnet. [20]

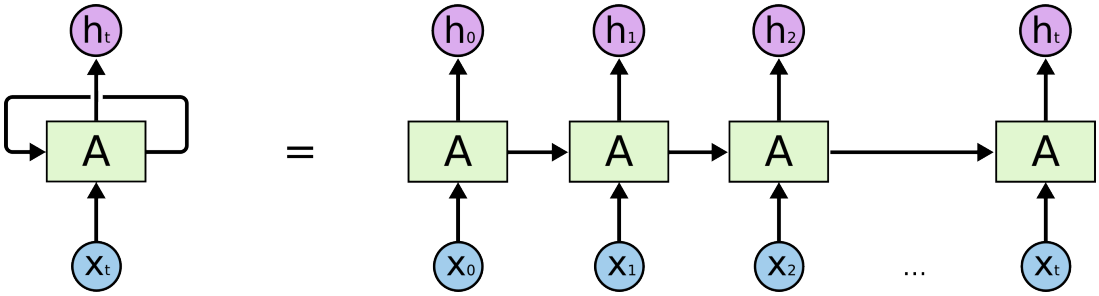


Abbildung 2.8.: Ausgerolltes RNN [20]

RNNs haben jedoch einige Schwachstellen. Wenn ein RNN viele Timesteps besitzt, funktioniert Backpropagation Through Time nicht mehr, da der Gradient verschwindet und das Netzwerk dadurch nicht lernfähig ist [1]. Es gibt mehrere theoretische Lösungsansätze zur Verhinderung dieses Problems. Diese theoretischen Lösungsansätze versuchen, den Backpropagation Algorithmus anzupassen, beziehungsweise zu ersetzen [1]. In der Praxis werden heute allerdings meist Long Short Term Memory Networks eingesetzt.

2.5. Long Short Term Memory

Da traditionelle RNNs, wie zuvor dargelegt, Probleme mit dem Lernen über viele Timesteps haben, wurde von Hochreiter et al. das sogenannte Long Short Term Memory (LSTM) als Lösungsansatz vorgeschlagen [9]. Im Gegensatz zu RNNs, bei welchen eine Zelle nur eine einfache Aktivierungsfunktion besitzt, verfügen LSTMs über einen Cellstate, welcher durch sogenannte Gates beeinflusst werden kann. Gates bestehen aus einem Sigmoid Neural Net Layer und einer elementweisen Multiplikation. Wie Abbildung 2.9 zeigt, geht der Cellstate C_t durch die Zelle.

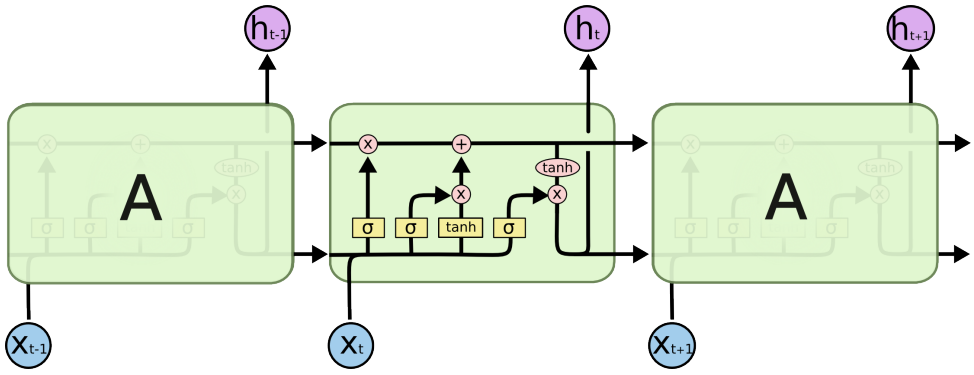


Abbildung 2.9.: LSTM Zelle mit Aktivierungsfunktionen, der Cellstate ist der obere vertikale Pfeil. [20]

Das erste Gate, welches den Cellstate beeinflusst, wird als Forget Gate bezeichnet. Die Aktivierungsfunktion des Forget Gates wird in Gleichung 2.7 angegeben, wobei W und b die Weightmatrix respektive den Bias darstellen. Die Aktivierung hängt vom Input x_t der Zelle ab und wird zusätzlich vom Output der vorherigen Zelle h_{t-1} beeinflusst. Da es sich um eine Sigmoid Aktivierung handelt, liegt der Output zwischen 0 und 1. Weil der Output mit dem Cellstate multipliziert wird, bedeutet ein Wert nahe 0, dass alle vorherigen Informationen vergessen werden.

$$f_t = \sigma(W_f * [h_t - 1, x_t] + b_f) \tag{2.7}$$

Nach dem Forget Gate folgen zwei Neural Net Layer, die darüber entscheiden, welche Informationen in den Cellstate übernommen werden. Die dafür verantwortlichen Aktivierungen sind in den Funktionen 2.8 und 2.9 beschrieben. Der Output der Multiplikation der beiden Aktivierungen wird dann zum Cellstate addiert. Dies ist aus Funktion 2.10 ersichtlich. Der erste Teil dieser Funktion zeigt die Multiplikation mit dem Forget Gate.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{2.8}$$

$$\tilde{C_t} = tanh(W_c * [h_{t-1}, x_t] + b_c) \tag{2.9}$$

$$C_t = f_t * C_{t-1} + \tilde{C_t} * i_t \tag{2.10}$$

Die LSTM Zelle besitzt den Ausgang h_t, dieser setzt sich zusammen aus o_t ,welches wie in 2.12 gezeigt mit dem Cellstate multipliziert wird.

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{2.11}$$

$$h_t = o_t * tanh(C_t) \tag{2.12}$$

Die vorbeschriebene Zellenkonfiguration erlaubt es einem LSTM über mehr als 1000 Timesteps zu lernen [9]. LSTMs sind sehr erfolgreich in den Bereichen Speech Processing [6] und Speech Recognition [23]. Im Allgemeinen eignen sich LSTMs hervorragend für die Verarbeitung von sequentiellen und zeitabhängigen Daten. Seit der Einführung von LSTMs von Hochreiter [9] wurden weitere Modifikationen an den Zellen vorgenommen. Einige dieser Modifikationen werden im Folgenden vorgestellt. Es ist zu erwähnen, dass die Performanceunterschiede dieser verschiedenen Varianten sehr gering sind und daher für die meisten Datensätze kaum von Relevanz. [8].

LSTM mit Peephole connections

Peephole LSTMs wurden 2000 von Gers und Schmidhuber vorgestellt [4]. Das Peephole ermöglicht es, den Cellstate nicht nur durch den Output h_{t-1} zu verändern sondern es kann direkt auf den Cellstate der vorhergegangenen Zelle C_{t-1} zugegriffen werden. Dies kann je nach Implementation allen Zellen ermöglicht werden oder nur bei einzelnen Zellen zugelassen werden.

Gated Recurrent Units

In der Arbeit "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation" [2] stellten Cho et al. eine neuartige Architekuter für LSTMs vor. Diese Variante fügt das Forget und Input Gate sowie den Hidden State und Cellstate zusammen. Dies vereinfacht die Architektur der LSTM Zellen und führt zu einer stetig steigenden Beliebtheit dieser Variante [20].

Bidirectional LSTM

Die Idee von Bidirectional LSTMs ist es, die Trainingsdaten sowohl vorwärts als auch rückwärts in ein LSTM zu geben. Dabei werden zwei separate LSTM Networks verwendet, welche mit einem Output Layer verbunden sind, aber keine Verbindungen untereinander aufweisen [7]. In das vorwärts laufende LSTM werden also die Daten von x_0 bis x_i gespeist. Gleichzeitig werden in das rückwärts laufende LSTM die Daten von x_i bis x_0 gespeist. Dies verdeutlicht Abbildung 2.10. Ein solches Netzwerk besitzt daher zu jedem Zeitpunkt alle Informationen über vergangene und zukünftige Daten [7].

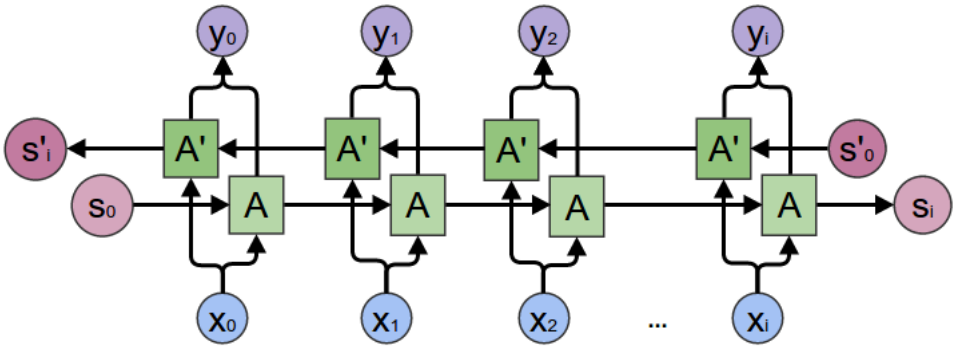


Abbildung 2.10.: Bidirectional LSTM [20]

Deep (Bidirectional) LSTMs

Wie bei normalen Neural Networks und CNNs können auch bei LSTMs und Bidirectional LSTMs mehrere Layer übereinander gelegt werden. In diesem Fall werden die Output Sequenzen des unteren Layers als Input für den oberen Layer verwendet [6]. Diese Verwendung von Bidirectional LSTMs im Bereich der Phenome Recognition [6] führten zu deutlichen Verbesserungen im Vergleich zu Versuchen mit Bidirectional LSTMs mit nur einem Layer [3].

3. Vorgehen / Methoden

Die Autoren unterteilten das Projekt in mehrere Phasen. Zuerst erfolgte der Wissensaufbau durch Literaturrecherche, anschliessend wurden die ersten Prototypen erstellt, um die Implementation zu testen. In der nächsten Phase wurden durch Iteration die verschiedenen Experimente durchgeführt und entsprechend der Ergebnisse Anpassungen an den Versuchen vorgenommen.

3.1. Datensatz

Als Datensatz kommt der TIMIT Datensatz zur Anwendung, welcher 630 amerikanische Sprecherinnen in den 8 verbreitetsten Dialekten umfasst. Von jeder Sprecherin sind 10 Sätze aufgenommen worden. Die Sprecherinnen sind in 438 männliche und 192 weibliche Sprecher aufgeteilt. Der Datensatz wurde bereits in den Untersuchungen zur Speaker Identification und Clustring verwendet [15] . Das ermöglicht es die Resultate zu vergleichen. Für die Experimente werden auch Teildatensätze mit 100 Sprechern verwendet, bei diesem Datensatz sind es 69 männliche und 31 weibliche Sprecherinnen. Dies spiegelt die Verteilung der Geschlechter im gesamten Datensatz wieder. Für die erste Evaluation der Implementationen wurde auch ein Datensatz mit 10 Sprechern verwendet, dieser kommt bei den Experimenten allerdings nicht zum Einsatz, da die Datenmenge zu klein ist, um aussagekräftige Resultate zu erhalten.

3.2. Daten Extraktion

Für die Extraktion der Daten aus den TIMIT WAV Dateien wird, die schon in der Arbeit von Lukic und Vogt [16] zur Anwendung gekommene Implementation, verwendet. Die Datenverarbeitung wir mit Hilfe der Python Bibliothek Librosa durchgeführt. Durch die Verwendung der folgenden Parameter, von 1024 Samples als FFT-Window und 160 Samples Hop Length ergibt sich ein Mel-Spektrogramme mit 100 Datenpunkte in Zeitrichtung für eine Sekunde und 128 Elemente in Frequenzrichtung. Für den erwarteten Sprecher wird ein Vektor generiert, wobei der Sprecher als Integer $0...anz.Sprecher$ angegeben wird. Es wird für jeden Satz ein Spektrogramm der Länge 800 generiert, da die längsten Sätze acht Sekunden nicht überschreiten. Da nicht alle Sätze acht Sekunden lang sind, wird für die fehlenden Daten das Spektrogramm mit Nullen aufgefüllt.

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

$$y[anz\_Sätze] \tag{3.2}$$

Die Frequenz und Zeitachse weisen die Grösse 128 respektive 800 auf. Die Daten werden zusätzlich in einen Trainingsdatensatz und Testdatensatz aufgeteilt. Mit dem Split 80/20 hat der Trainingsdatensatz somit acht Sätze pro Sprecher und der Testdatensatz zwei Sätze pro Sprecher.

3.3. Testdaten Generation

Um die extrahierten Daten in verwendbare Batches umzuwandeln, verwenden wir dasselbe Prinzip welches Lukic und Vogt in ihrer Bachelorarbeit [16] beschrieben haben. Bei diesem wird jeweils für einen zufällig ausgewählten Sprecher ein zufällig ausgewähltes Segment extrahiert. Das geschieht bis ein Batch voll ist. Die Batchgrösse beträgt in unseren Experimenten immer 128. Da unsere Implementation in Keras mit Tensorflow Backend erfolgt, verwenden wir einen selbstgeschriebenen Batchgenerator, welcher

die Batches generiert. Die Segmentlänge kann als Parameter übergeben werden. Diese Änderung ist notwendig da Segmente verschiedener Länge für die Experimente verwendet werden. Da die von uns verwendete Lossfunktion einen Label Input in der Form $YT[BatchGrösse, Sprecher]$ erwartet, wird der in Abschnitt 3.2 beschriebene Datensatz so transformiert, dass für jeden Eintrag in Y ein Vektor mit Nullen kreiert wird, ausser an der Stelle welche dem Wert von Y entspricht, hier wird eine 1 gesetzt wird. Der Trainigsdatensatz, bestehend aus acht Sätzen pro Sprecher, wird in einen Trainingsdatensatz von sechs Sätzen und einen Validierungsdatensatz von zwei Sätzen pro Sprecher aufgeteilt.

3.4. LSTM Architektur

Für die Experimente welche die Tauglichkeit von LSTMs untersuchen, wird die in Abbildung 3.1 dargestellte Grundarchitektur verwendet. Der Input ist dabei das Spektrogramm. Die Zeitachse des Spektrogramms sind die einzelnen Timesteps. Dies bedeutet, dass für jeden Timestep ein Vektor mit der Länge 128 als Input dient. Die Anzahl Timesteps variiert dabei je nach Experiment. Die Bidirectional LSTM Layer geben jeweils wieder eine Sequenz als Output, welche die Timesteps des nächsten Layers bilden. Zudem werden Architekturen mit mehr als zwei Layern verwendet, sowie Varianten mit einem Dropout Layer.

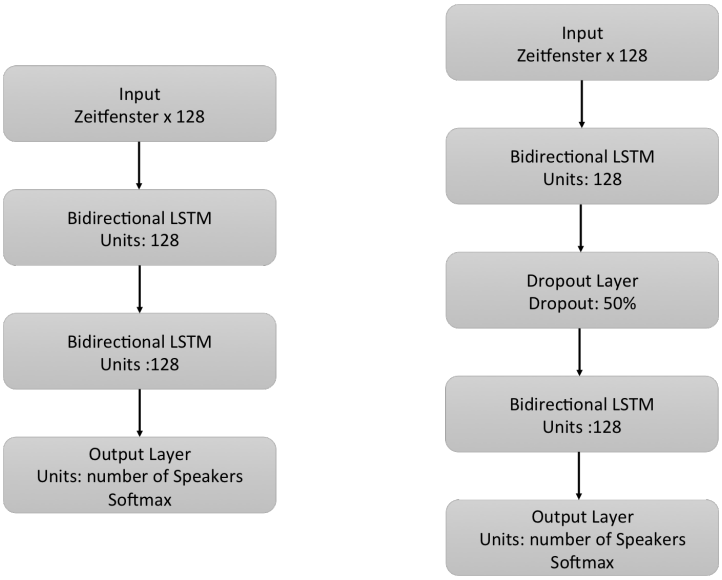


Abbildung 3.1.: Architektur des LSTM Networks

3.5. LSTM-CNN Architektur

Für das kombinierte Netzwerk wird ein Aufbau, wie Abbildung 3.2 dargestellt, verwendet. Wir lehnen uns dabei an das CNN von Lukic und Vogt [17] an, da sie in ihren Arbeiten gezeigt haben, dass ihr CNN für Speaker Identifikation relevante Features extrahiert. Die Beiden LSTM Layers werden an den Layer 7 angebracht. Die einzelnen Outputs des Dense Layers sind die einzelnen Inputs für den LSTM Layer. Dabei dient jedes Neuron des Dense Layer als Timestep. Es wird zudem noch eine Konfiguration verwendet, in welcher das Bidirectional LSTM auf den Layer 5 aufgesetzt wird.

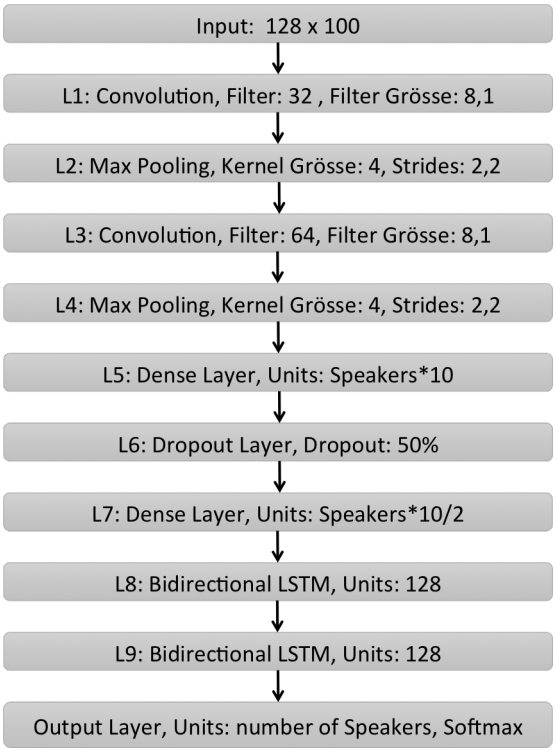


Abbildung 3.2.: Architektur des kombinierten CNN und LSTM Networks

3.6. Training des Netzwerks

Das Training des Netzwerkes läuft für alle Experimente gleich ab und wird zum besseren Verständnis in Abbildung 3.3 dargestellt und nachfolgend die einzelnen Schritte erläutert:

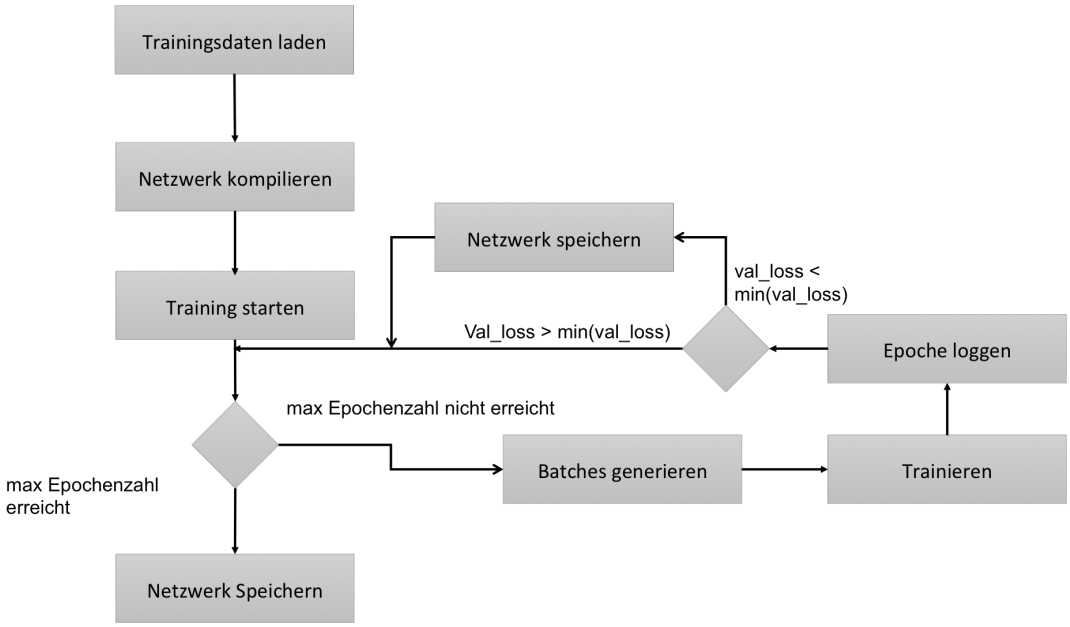


Abbildung 3.3.: Trainingsablauf

- **Trainingsdaten laden:** Die in Abschnitt 3.2 beschriebenen Trainingsdaten werden geladen.
- **Netzwerk kompilieren:** Die Netzwerke für alle Experimente werden mit Hilfe der Python-Bibliothek Keras erstellt und kompiliert.
- **Training starten:** Mit der Funktion `model.fit_generator` der Keras Bibliothek wird das Training gestartet.
- **Batches generieren:** Die Batches werden mit dem `batch_generator` bzw. `batch_generator_LSTM`, in der Datei `data_gen.py` generiert.
- **Trainieren:** Das Netzwerk wird mit einem Adam Optimizer trainiert und verwendet Categorical Cross Entropy als Loss Funktion.
- **Epoche loggen:** Die Resultate jeder Epoche werden über einer Callback Funktion `CSVLogger`, welche von Keras zur Verfügung gestellt wird, in eine csv Datei geschrieben.
- **Netzwerk speichern:** Falls der Validation Loss in der Epoche einen neuen Minimalwert erreicht hat, wird das Netzwerk in einer HDF5 Datei gespeichert. Dazu wird die Keras Funktion `ModelCheckpoint` verwendet.
- **Netzwerk speichern:** Das Netzwerk wird mittels der von Keras zur Verfügung gestellten Funktionen `model.save` im Dateiformat HDF5 gespeichert.

Eine genaue Anleitung zur Verwendung des Codes kann im Anhang unter Abschnitt A.2 nachgeschlagen werden.

Optimizer

Als Optimizer wird Adam, welcher 2014 von Kingma und Ba vorgestellt wurde [12], verwendet. Adam als Optimizer besitzt den Vorteil, dass die Learning Rate nicht selber eingestellt werden muss, da er für jeden Parameter eine adaptive Learning Rate berechnet. Zudem benötigen auch die Hyperparameter kaum Tuning. Adam erreicht leicht bessere Resultate als Optimizer, welche ähnliche Methoden zur Optimierung verwenden [22]. Wir verwenden die von Kingma vorgeschlagenen Parameter: $Learningrate = 0.001$, $\beta_1 = 0.09$, $\beta_2 = 0.999$ und $\epsilon = 10^{-8}$.

Loss Funktion

Als Loss Funktion wird die von Keras bereitgestellte Funktion Categorical Cross Entropy verwendet. Die Funktion ist dabei wie folgt definiert:

$$C = -\frac{1}{n}\sum_{x} ln[y(x) * lna + (1 - y(x))ln(1 - a)] \tag{3.3}$$

Dabei steht n für die Anzahl Inputs, y steht für den erwarteten Output Vektor des Input Vektors x und a steht für den Output Vektor des Netzwerks für Input x. Für eine genauere Beschreibung empfehlen wir das Kapitel 3 aus dem Onlinebuch "Neural Networks and Deep Learning"von Michael Nielsen [19]

3.7. Testen der Resultate

Um die Ergebnisse des Trainings zu überprüfen wird der Testdatensatz verwendet welcher zwei Sätze pro Sprecher besitzt. Dabei werden die Sätze wieder in Segmente unterteilt. Es wird nicht wie bei den Trainingsdaten ein zufälliger Ausschnitt des Satzes extrahiert, sondern jeder Satz wird in nicht überlappende Segmente unterteilt. Die Segmente besitzen die gleiche Länge, wie die zum Training verwendeten Inputs. Die Accuracy wird mit zwei Varianten gemessen. Zum Einen wird die Accuracy für jedes Segment ermittelt. In der 2. Variante werden alle Segmente vom Netzwerk verarbeitet. Die

erhaltenen Output-Vektoren für jeden Sprecher werden dann mit dem arithmetischem Mittel verarbeitet.

$$\tilde{x} = -\frac{1}{n}\sum_{i=n}^{n} x_i \tag{3.4}$$

x_i ist der Wert für den Sprecher x im Output-Vektor i und n die Anzahl Output-Vektoren. Für diese Accuracy wird die Bezeichnung Mean Accuracy verwendet. Diese Berechnung wird verwendet, da mit mehreren Output-Vektoren pro Sprecher eine genauere Klassifikation der Sprecher erfolgen kann. Diese Variante macht die Resultate auch mit bereits existierenden Arbeiten vergleichbar [15].

3.8. Verwendete Libraries

Für die Implementierung der Netzwerke und der Spektrogrammextraktion wurden zusätzliche Python Bibliotheken verwendet, welche nachfolgend aufgelistet werden.

Netzwerk:

- Tensorflow 0.11
- Keras 1.1.2
- NumPy 1.10.4
- Matplotlib 1.5.3
- h5py 2.6.0

Spektogramm Extraktion:

- NumPy 1.10.4
- SciPy 0.16.1
- Librosa 0.4.3

4. Experimente

In diesem Kapitel werden die ausgeführten Experimente dargelegt. Es wird die Tauglichkeit von Bidirectional LSTMs und CNN-LSTM Kombinationen auf die Problemstellung der Speaker Identification geprüft.

4.1. Testdaten

Als Testdaten kam ein Datensatz mit 100 Sprechern zum Einsatz, welcher 69 männliche und 31 weibliche Sprecher beinhaltet. Wir verwenden diese Aufteilung, da diese der Verteilung von weiblichen und männlichen Sprechern über den gesamten Datensatz entspricht. Für die genauere Validation steht zudem das komplette Sprecherset mit 630 Sprechern zur Verfügung, welches jeweils auf die am vielversprechendste Architektur angewendet wird, um diese mit vorherigen Ergebnissen zu vergleichen.

4.2. Experiment 1 – Anzahl Units Pro Layer

4.2.1. Ausgangslage

In diesem Versuch wird untersucht, welchen Einfluss die Anzahl Units auf ein Bidirectional LSTM mit 2 Layern hat. Dabei werden für beide Layer die gleiche Anzahl Units verwendet. Es werden LSTMs mit 32, 64, 128, 256 Units getestet. Als Testdaten verwenden wir das Sprecherset mit 100 Sprechern wie in 4.1 erwähnt. Wir trainieren jedes Netzwerk für 1000 Epochen. Wir erwarten, dass Netzwerke mit 128 und 256 Layern ein besseres Ergebnis liefern, da jeder Input Vektor die Länge 128 aufweist.

4.2.2. Resultate

Wie in Abbildung 4.1 zu sehen ist, lernen alle Netzwerke. Allerdings sind, wie vermutet, die Netzwerke mit 128 und 256 Units pro Layer besser als die kleineren Netzwerke mit 32 respektive 64 Units. Interessant ist der Verlauf des Netzwerkes mit 256 Units, dieses scheint starken Schwankungen zu unterliegen. Wir würden hier von Overfitting ausgehen, aber das Netzwerk erholt sich jeweils nach einigen Epochen wieder und verbessert seine Genauigkeit. In Abbildung 4.2 sind der Validation Loss und Train Loss des Bidirectional LSTM mit 256 Units zusehen, auch hier sind die Einbrüche bei beiden Metriken zu erkennen. Wir haben keine Erklärung für dieses Phänomen.

In der Tabelle 4.1 ist die Accuracy der getesteten LSTMs aufgelistet. Dabei steht Accuracy für die Accuracy über jedes Segment und Mean Accuracy für das arithmetische Mittel wie in Abschnitt 3.7 beschrieben. Diese Verwendung der Begriffe wird für alle Experimente verwendet. Wie zu erkennen ist schneiden die Netzwerke mit 128 und 256 Units am besten ab. Interessant ist, dass die Mean Accuracy beim Netzwerk mit 256 Units höher liegt obwohl die Segment Accuracy tiefer ist. Wir vermuten, dass mit weiteren Trainingsläufen das Resultat leicht schwanken kann und daher entscheiden wir uns das Netzwerk mit der besseren Accuracy pro Segment als Grundlage für die weiteren Experimente zu verwenden.

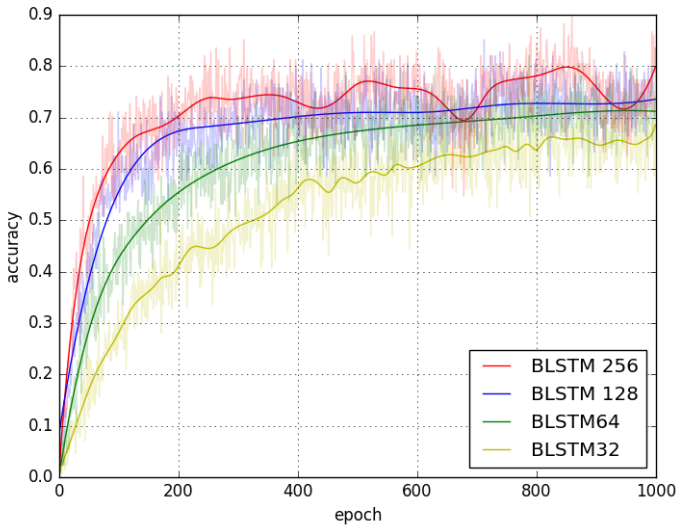


Abbildung 4.1.: Validation Accuracy der Bidirectional LSTMs mit 32, 64, 128 und 256 Units

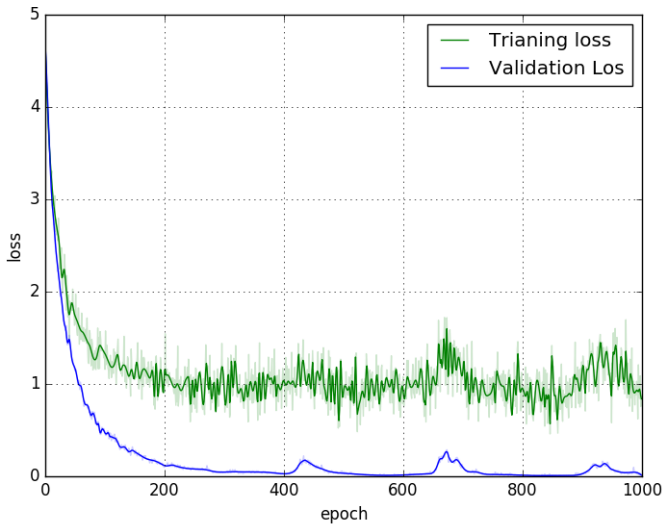


Abbildung 4.2.: Training und Validation Loss des Bidirctional LSTM mit 256 Units

Netzwerk	Accuracy	Mean Accuracy
BLSTM_32	59.66%	88%
BLSTM_64	67.54%	91%
BLSTM_128	73.55%	94%
BLSTM_256	69.42%	95%

Tabelle 4.1.: Accuracies für Bidirectionl LSTMs mit 32, 64, 128 und 256 Units.

4.3. Experiment 2 - Anzahl Layers

4.3.1. Ausgangslage

Wie im vorhergehenden Experiment gezeigt, sind Bidirectionale LSTMs in der Lage, relevante Informationen für die Aufgabe der Speaker Identification zu lernen. Da in vielen Anwendungen tiefere Neural Network Architekturen bessere Resultate liefern, soll in diesem Experiment soll der Einfluss der Anzahl Layer auf die Performance der LSTMs untersucht werden. Dafür benutzen wir Layers mit 128 Units, da diese in Experiment 1 die besten Ergebnisse erzielten. Es werden Netzwerke mit 2, 3, 4 und 5 Layern getestet, erneut kommt der Datensatz mit 100 Sprechern zur Anwendung.

4.3.2. Resultate

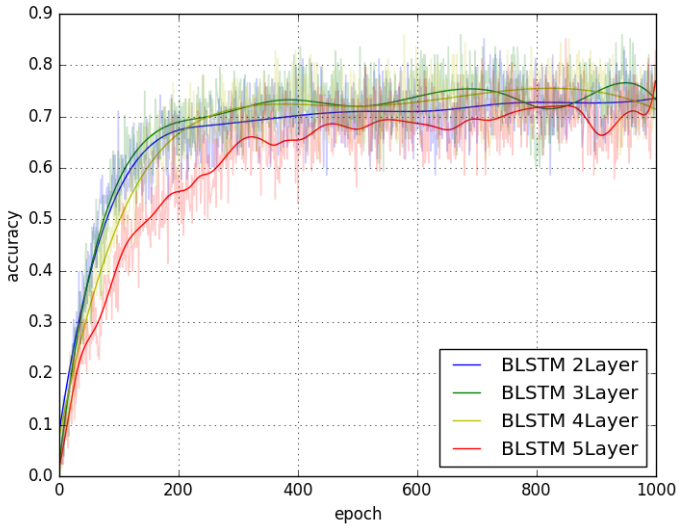


Abbildung 4.3.: Validation Accuracy für die Netzwerke mit 2 bis 5 Layern

Es wurde vor der Ausführung der Experimente erwartet, dass die Netzwerke mit mehr Layern besser abschneiden würden. Die Unterschiede im Lernverhalten der Netzwerke sind jedoch wider Erwarten sehr gering. Wie in Abbildung 4.3 gezeigt erreichen die Netzwerke mit 2, 3 und 4 Layern schon nach 200 Epochen 65 % Genauigkeit auf dem Validation Datensatz. Das Netzwerk mit 5 Layern benötigt dafür 300 Epochen. Die Accuracy der 3 und 5 Layer Netzwerke unterliegt relativ starken Schwankungen. Nach 1000 Epochen erreichen aber alle Netzwerke eine ähnliche Validation Accuracy. Wie in Tabelle 4.2 ersichtlich, erreichte das Netzwerk mit 2 Layern aus dem vorherigen Experiment die höchste Accuracy pro Segment, und nur das Netzwerk mit 4 Layern konnte eine höhere Mean Accuracy erreichen was wieder der Trainingsvarianz zugeschrieben wird. Daher wird für die weiteren Experimente das Netzwerk mit 2 Layern und 128 Units verwendet.

Netzwerk	Accuracy	Mean Accuracy
2 Layer	73.55%	94%
3 Layer	70.73%	94%
4 Layer	71.11%	95%
5 Layer	69.42%	93%

Tabelle 4.2.: Test Accuracy der Netzwerke mit 2 bis 5 Layern

4.4. Experiment 3 - Dropout Layer

4.4.1. Ausgangslage

Es soll geprüft werden, welchen Einfluss der Dropout Layer auf das Bidirectional LSTM hat. Dazu wird ein Netzwerk mit zwei Bidirecectional LSTM Layern verwendet und ein Dropout Layer mit 20% respektive 50% Dropout. Als Vergleichsnetzwerk dient wieder das Netzwerk mit 128 Units und zwei Layern. Es wird erneut der Datensatz mit 100 Sprechern verwendet um das Experiment zu prüfen.

4.4.2. Resultate

Die beiden Dropout Layer haben nur einen geringen Einfluss auf das Lernverhalten des Netzwerks. Ohne Glättung wäre in Abbildung 4.4 kaum ein Unterschied zwischen den drei getesteten Netzwerken zu erkennen. Allerdings ist zu erkennen, dass das Netzwerk mit 50% Dropout starke Schwankungen in der Validitation-Accuracy aufweist.

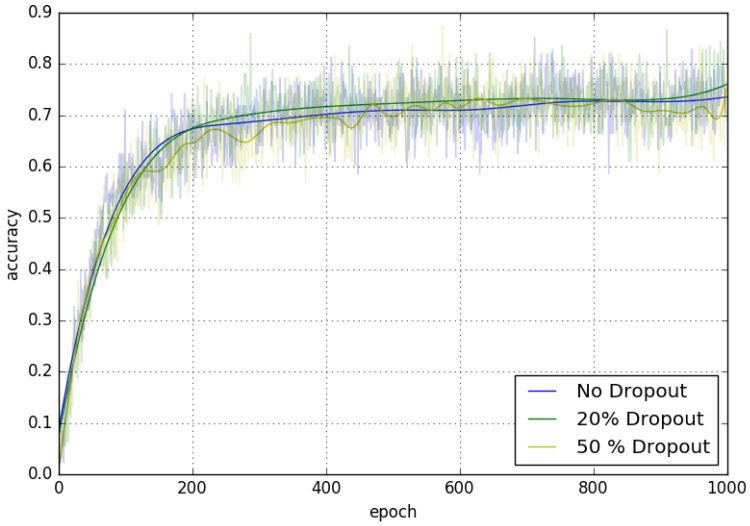


Abbildung 4.4.: Validation Accuracy für die Netzwerke mit 20% und 50% dropout

4.5. Experiment 4 - Zeitfenster der Eingabedaten

4.5.1. Ausgangslage

In diesem Versuchen untersuchen wir den Einfluss des Zeitfensters auf die Performance des Bidirectional LSTMs. Dies wird untersucht, da die Wahl der Zeitfenster einen Einfluss auf die Sprechererkennung hat[25]. Es werden dazu Zeitfenster von 500, 300, und 150 Millisekunden(ms) verwendet. Dazu werden anstatt der ein Sekunden Segmenten jeweils die Segmente so gekürzt, dass sie den gewünschten Zeitfenstern entsprechen. Die Form der Eingabedaten kann der Tabelle 4.3 entnommen werden. Zur Anwendung kommt erneut das Netzwerk mit 2 Layern und 128 Units pro Layer. Es wird der Testdatensatz mit 100 Sprechern verwendet.

Zeitfenster	Dimension
1000 ms	$100x128$
500 ms	$50x128$
300 ms	$30x128$
150 ms	$15x128$

Tabelle 4.3.: Format der Eingabedaten für verschiedene Zeitfenster

4.5.2. Resultate

Anhand der Abbildung 4.5 ist zu erkennen, dass eine kleinere Segmentgrösse zu einer Abnahme der Validation Accuracy führt. Die Lernrate des Netzwerkes sinkt ebenfalls. Bei der Auswertung der Testdaten zeigt sich das gleiche Bild für die Accuracy über jedes Segment. Die Mean Accuracy für jeden Sprecher steigt allerdings für die kleineren Segmente. Das Netzwerk mit 150 ms Zeitfenstern erreicht eine hundertprozentige Genauigkeit. Das Zeitfenster von 150 ms ist sehr nahe an dem von Thilo Stadelmann vorgeschlagenem Zeitfenster von 130 ms[25] (S. 125). Daher ist zu vermuten, dass in diesem Zeitfenster die relevantesten Features zur Speaker Identification extrahiert werden können. Einen weiteren Einfluss könnte die Anzahl Segmente haben welche für jeden Sprecher vorhanden sind. Diese ist bei einem Zeitfenster von 150 ms um ein 6.5 Faches höher als bei einem Zeitfenster von einer Sekunde.

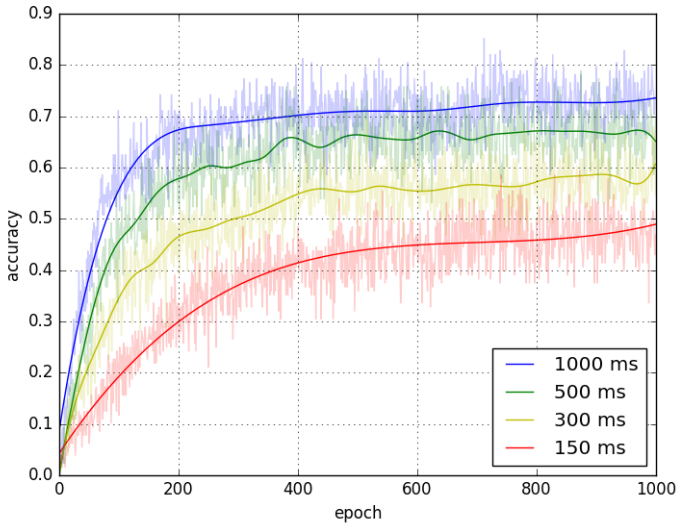


Abbildung 4.5.: Validation Accuracy für die Netzwerke 1000, 500, 300, 150 ms Zeitfenstern

Die 100% Mean Accuracy auf einem Datensatz von 100 Sprechern übertrifft die erreichten Resultate vorheriger Arbeiten mit CNNs[15]. Um das Ergebnis zu validieren, muss ein Test mit dem vollen Datensatz von 630 Sprechern durchgeführt werden.

Zeitfenster	Accuracy	Mean Accuracy
1000 ms	67.54%	94%
500 ms	59.83 %	97%
300 ms	53.83%	99%
150 ms	45%	100%

Tabelle 4.4.: Test Accuracies für die verschiedenen Zeitfenster

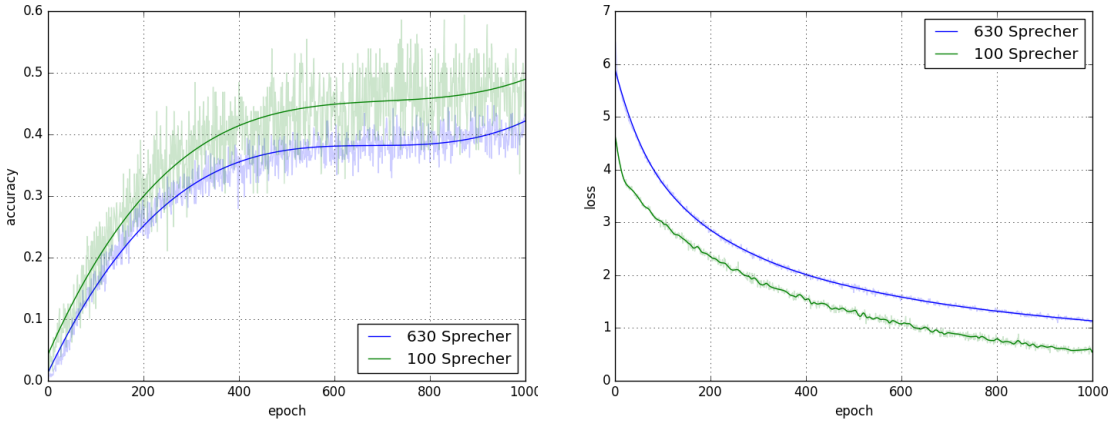
4.6. Experiment 5 - Gesamter TIMIT Datensatz

4.6.1. Ausgangslage

Um die Resultate aus dem vorherigen Experiment mit existierenden Arbeiten vergleichbar zu machen, wird das gleiche Netzwerk wie in Experiment 4 mit einem Zeitfenster der Eingabedaten von 150 ms verwendet. Für dieses Experiment wird der komplette TIMIT Datensatz von 630 Sprechern verwendet. Wir erwarten, dass die Genauigkeit des Netzwerks nachlassen wird, da es nun bedeutend mehr Sprecher verarbeiten muss.

4.6.2. Resultate

Das Netzwerk weist mit 630 Sprechern eine geringere Lernrate auf als noch mit 100 Sprechern, was in Abbildung 4.6a verdeutlicht wird. Die Validation Accuracy liegt ebenfalls tiefer. Das Training für 1000 Epochen benötigte nun etwa 2.3 Stunden im Vergleich zu den Experimenten mit 100 Sprechern welche etwa 40 Minuten benötigten. Es ist auch festzustellen, dass der Trainings Loss nach 1000 Epochen noch immer sinkt, und die Accuracy steigt.



(a) Validation Accuracy für die Netzwerke mit 100 respektive 630 Sprecher

(b) Trainings Loss für die Netzwerke mit 100 respektive 630 Sprecher

Abbildung 4.6.: Darstellung der Validation Accuracy in (a) und des Training Loss in (b)

Auf dem Testdatensatz mit 630 Sprechern angewandt erreicht dass Netzwerk nun 99,68 % Accuracy, mit der Mean Accuracy Methode. Dies bedeutet, dass das Netzwerk zwei Sprecher nicht richtig klassifizieren konnte. Bei den beiden falsch klassifizierten Sprechern war der Sprecher jeweils unter den 4 am höchsten bewerteten Sprechern. Mit einer Accuracy von 99.68% erreicht das Netzwerk ein besseres Resultat als das CNN von Lukic und Vogt [15] welches 97% erreichte. Es ist allerdings zu erkennen, dass die Validation Accuracy bei 1000 Epochen noch ansteigt, daher wird noch ein weiteres Experiment durchgeführt, in dem das Netzwerk länger trainiert wird.

4.7. Experiment 6 - Längere Trainingsdauer

4.7.1. Ausgangslage

Da im vorherigen Experiment zu erkennen war, dass die Accuracy noch steigt und der Trainings Loss sinkt, wird in diesem Experiment das gleiche Netzwerk für 4000 Epochen trainiert. Es wird erwartet,

dass die Accuracy weiter ansteigt, dass der Trainings und Validation Loss weiter sinken werden und eine bessere Accuracy auf den Testdaten erreicht werden kann. Da bei längeren Trainingseinheiten das Risiko von Overfitting besteht, wurde zusätzlich noch ein Netzwerk mit einem Dropout Layer, welcher 50% Dropout besitzt, getestet.

4.7.2. Resultate

Wie die Resultate in den Abbildungen 4.8b und 4.7 zeigen, setzt bereits kurz nach dem Erreichen von 1000 Epochen, Overfitting ein und der Trainingsloss steigt wieder an. Auch die Validation Accurracy steigt nicht mehr weiter und nimmt leicht ab. Dies wiederspiegelt sich auch in den Testergebnissen für dieses Netzwerk, da es schlechter abschneidet als das Netzwerk im vorherigen Experiment.

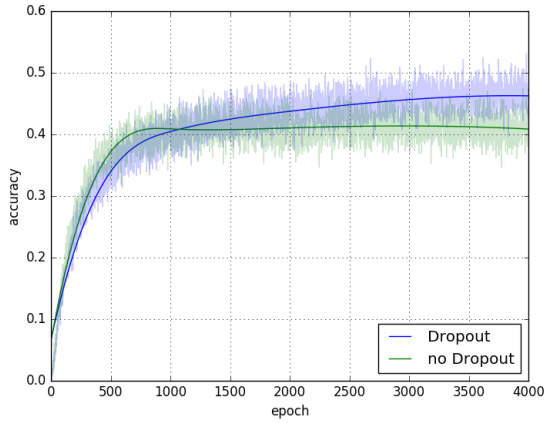
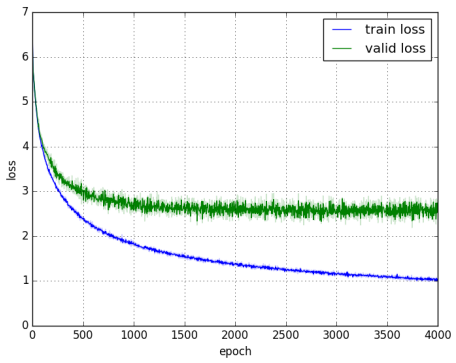
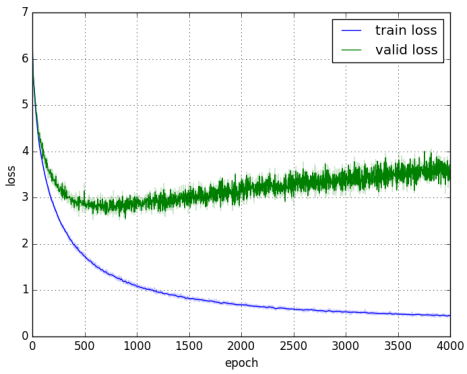


Abbildung 4.7.: Validation Accuracy für die Netzwerke mit 20% und 50% droput



(a) Validation Accuracy für das Netzwerke mit Dropout Layer



(b) Trainings Loss für das Netzwerk ohne Dropout Layer

Abbildung 4.8.: Trainings und Validation Loss für das Netzwerk mit Dropout Layer (a) und das Netzwerk ohne Dropout Layer(b).

Der Dropout Layer erzielt den erwünschten Effekt und verhindert die Abnahme des Validation Loss, allerdings ist festzustellen, dass der Validation Loss nach 1500 Epochen nicht weiter abnimmt. Die Validation Accuracy steigt noch langsam an bis rund 3000 Epochen erreicht werden. Wie in Tabelle 4.5 zu sehen ist erreicht das Netzwerk mit Dropout Layer höhere Accuracy auf den einzelnen Elementen als das Netzwerk im vorherigen Experiment und erreicht mit der Mean Accuracy Methode 100%. Es

konnte also alle Sprecher richtig klassifizieren. Wir erreichen damit nahezu die gleiche Accuracy wie statistische State of the art Verfahren [18] [21]

Netzwerk	Accuracy	Mean Accuracy
ohne Dropout	39.24%	99.52%
mit Dropout	44.68%	100%

Tabelle 4.5.: Accuracy und Mean Accuracy für das Netzwerk mit und ohne Dropout Layer

4.8. Experiment 7 - CNN-LSTM Layer

4.8.1. Ausgangslage

Für das erste Experiment wird das Netzwerk aus der Projektarbeit von Lukic und Vogt übernommen [15]. Es wird geprüft, ob das Aufsetzen des Bidirectional LSTMs auf den Layer 5 oder Layer 7 bessere Resultate erzielt. Es wird der Datensatz mit 100 Sprechern verwendet und beide Netzwerke werden für 1000 Epochen trainiert. Das Ergebnis wird mit den erreichten Resultaten in der vorhin erwähnten Arbeit verglichen, in welcher 98% Accuracy auf dem gleichen Datensatz erreicht wurde.

4.8.2. Resultate

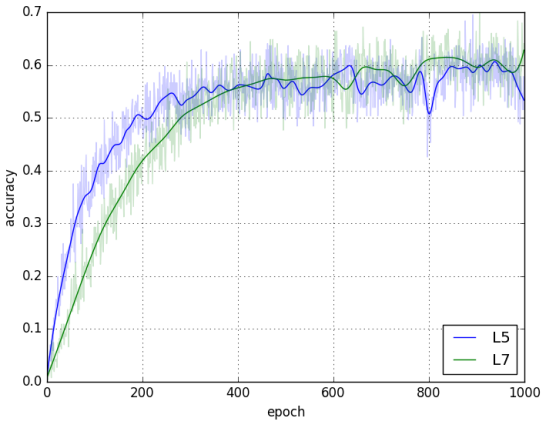


Abbildung 4.9.: Validation Accuracy für die Netzwerke mit LSTM auf Layer 5 respektive Layer 7

Die Performance beider Kombinationen erreicht nicht die 98%, welche das reine CNN erreichte. Während die Validation Accuracy der beiden Netzwerke recht ähnlich ausfällt, wie in Abbildung 4.9 gezeigt wird, weisen die Ergebnisse für den Testdatensatz grosse Unterschiede auf. So erreicht das Netzwerk mit Layer 7 eine Accuracy von 92%, das Netzwerk mit Layer 5 aber nur 78%. Es ist auch festzustellen, dass die Validation Accuracy beim Netzwerk, welches den Dense Layer 5 verwendet, sehr starken Schwankungen unterworfen ist.

Netzwerk	Accuracy	Mean Accuracy
Layer 5	60.90%	92%
Layer 7	55.45%	78%

Tabelle 4.6.: Test Accuracy für die Netzwerke mit LSTM auf Layer 5 respektive Layer 7

4.9. Experiment 8 - CNN-LSTM Variation der Pooling Kernel

4.9.1. Ausgangslage

Nach den eher entäuschenden Resultaten der CNN-LSTM Kombination in Experiment 7, soll nun überprüft werden ob eine Anpassung des Pooling Kernels eine Verbesserung bringen könnte. Es wird angenommen, dass ein Pooling, welches nur auf der Frequenzachse erfolgt, eine Verbesserung der Accuracy bringen könnte. Dafür werden die Experimente in Tabelle 4.7 ausgeführt. Es wird die Netwerkarchitektur mit dem Dense Layer 7 verwendet da diese im vorherigen Experiment bessere Resultate lieferte. Der Datensatz mit 100 Sprechern kommt wieder als Trainingsdatensatz zur anwendung.

Experiment	Kernel Size	Step size
Exp 1	4 x 4	2 x 2
Exp 2	4 x 1	2 x 1
Exp 3	6 x 1	3 x 1
Exp 4	8 x 1	4 x 1

Tabelle 4.7.: Experimente für Pooling size

4.9.2. Resultate

In Tabelle 4.8 ist zu erkennen, dass die Experimente mit der Pooling Size keine Verbesserung der Accuracy bringen konnten. Nur das Netzwerk mit $4x1$ Pooling konnte die Segment Accuracy verbessern, es wurde aber keine Verbesserung der Mean Accuracy erreicht.

Experiment	Accuracy	Mean Accuracy
Exp 1	60.9%	92%
Exp 2	62.78%	91%
Exp 3	57.52%	89%
Exp 4	49.44%	80%

Tabelle 4.8.: Test Accuracies für die Pooling size Experimente

Wir schliessen daraus, dass die Architektur welche für die Experimente verwendet wird, keinen zusätzlichen Nutzen bringt. Berücksichtigt man, dass die Trainingsdauer einer Epoche bei 40s liegt und die des reinen CNNs bei 20 Sekunden, ist ein Einsatz dieser Architektur nicht sinvoll. Da eine Epoche für 630 Sprecher etwa 2000 Sekunden dauert, haben wir uns auch dazu entschlossen, auf einen Test über den vollen Datensatz zu verzichten.

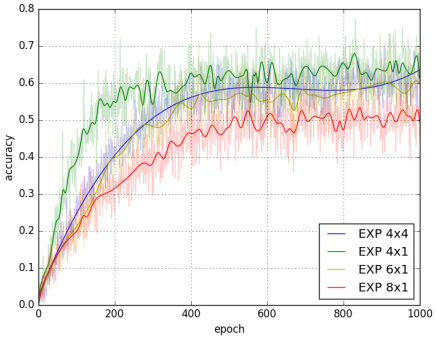


Abbildung 4.10.: Validation Accuracy der Netzwerke

5. Fazit

Wir konnten eine gute Grundlage für weitere Arbeiten im Bereich der Speaker Recognition mit LSTMs legen und erreichten mit LSTMs sehr gute Ergebnisse auf dem Task der Speaker Identification. Nachfolgend ziehen wir ein detailierteres Fazit über die Ergebnisse der LSTM Experimente und den Kombinationen von CNN und LSTMs.

5.1. Fazit der LSTM Experimente

Durch unsere Experimente konnten wir zeigen, dass LSTMs für den Problembereich der Speaker Identification gute Resultate liefern. Wir konnten zeigen, dass kürzere Zeitsegmente eine bessere Accurracy erreichen und das die Accuracy bei Zeitsegmenten von 150 ms am höchsten ist. In der Arbeit von Thilo Stadelmann wurden 130 ms als optimales Zeitfenster für die Speaker Identification angenommen [25] (S.125). Unsere verwendetes Zeitfenster ist sehr nahe an diesem Wert und bestätigt daher auch diese Beobachtung. Die einzelenen Segmente können bei diesem Zeitfenster weniger gut einem Sprecher zugeordnet werden als bei längeren Segmenten. Dies erklären wir uns mit der Tatsache, dass das Netzwerk immer nur einen kleinen Teil der Daten sieht pro Batch und daher die Analyse einzelner Segmente schwer fällt. Es war auch festzustellen, dass das Netzwerk ab 1500 Epochen Overfitting aufwies. Dies konnten wir mit einem Dropout Layer beheben, welcher es erlaubte das Netzwerk über 4000 Epochen zu trainieren ohne dass der Validation loss abnahm, allerdings ist nach 1500 Epochen kaum noch eine Verbesserung festzustellen. Komplexere Netzwerke mit mehr als zwei Layern und mehreren Dropout Layern konnten ebenfalls keine Verbesserung des Trainingsverhaltens erzeugen.

Das Netzwerk in Experiment 6 im vorherigen Abschnitt, konnte auf dem TIMIT Datensatz alle Sprecher richtig identifizieren. Somit erreicht das LSTM eine Accuracy von 100%, welche die Performance von CNNs auf dem Datensatz übertrifft [15]. Wir erreichen gleichwertige Resultate wie die besten traditionellen statistischen Methoden, und dies ohne an den Input Daten und dem Verhalten des Netzwerkes zu feilen.

5.2. Fazit der CNN-LSTM Experimente

Wir erwarteten, dass die Kombination der Stärken beider Netzwerke gute Resultate liefern sollten, da CNNs sehr stark in der Feature Exctraction sind und LSTMs zeitliche Abhängigkeiten gut modellieren können. Die Resultate der Experimente zeigten aber, dass unsere Implementation keine Verbesserung der bisher verwendeten CNN Netzwerke [15] bringt. Das beste Netzwerk welches die Grundkonfiguration, der vorangegangenen Projektarbeit verwendet, erreicht nur 92% Accuracy auf den Testdaten von 100 Sprechern. Dies ist deutlich weniger als das CNN, welches 98% erreichte und unser LSTM welches 100% erreichte. Das Netzwerk skaliert sehr schlecht da für den kompletten TIMIT Datensatz eine Trainingszeit von 2000 Sekunden pro Epoche festzustellen war. Das erklärt sich aus der Grösse des Dense Layers, welcher für 630 Sprecher eine Grösse von 3150 aufweist, was 3150 Timesteps für das LSTM bedeutet. Verlgleicht man dies mit den knapp 40 Sekunden des CNNs und den 10 Sekunden des LSTM Netzwerkes ist es nicht zu empfehlen ein solches Netzwerk zu verwenden.

Als Grund für die schlechtere Performance vermuten wir, dass die zeitliche Information, welche für das LSTM entscheidend ist, verloren geht. Das geschieht wahrscheinlich be der Verbindung des letzten Maxpooling Layers mit dem Dense Layer, da eine Verbindung von allen Neuronen im Maxpooling Layer mit allen Neuronen im Dense Layer stattfinden. Daher wären in Zukunft Ansätze zu wählen, welche das LSTM direkt auf den Feature Maps des CNN operieren lassen.

6. Ausblick

6.1. LSTMs

In diesem Abschnitt werden mögliche weiterführende Untersuchungen für LSTMs und RNNs vorgeschlagen.

Clustering

Wir konnten zeigen, dass LSTMs für den Problembereich der Speaker Identification relevante Features extrahieren können. Vorhergehende Arbeiten zeigen auf, dass CNNs welche für Speaker Identification einsetzbar sind, sehr gute Resultate im Bereich Speaker Clustering erzielen [17] [16]. Es wäre deshalb interessant zu untersuchen, welche Resultate mit einem LSTM Netzwerk für die Aufgabe des Speaker Clustering erzielt werden können.

Untersuchung von LSTM Abwandlungen

In dieser Arbeit wurden nur Bidirectional LSTM Implementierungen untersucht. Verschiedene andere Variationen von LSTMs beziehungsweise RNNs wie z.B. Gated Recurrent Units und LSTMs mit Peephole Connection könnten auf dem Problemgebiet angewandt werden.

Untersuchung Lernverhatlen

Wie in den Experimenten 1 und 2 gesehen, gibt es gewisse Konfigurationen von Bidirectional LSTMs, bei welchen es immer wieder zu kurzen, relativ starken Einbrüchen beim Trainings sowie Validation Loss kommt. Dies waren vor allem die Konfigurationen mit zwei Layern und jeweils 256 Units, sowie diejenige mit 128 Units und fünf Layern. Es könnte aufschlussreich sein zu untersuchen, was die Ursache dieser Einbrüche ist.

6.2. CNN – LSTM Kombination

Unsere Implementation von kombinierten CNN-LSTM Netzewerken, erreicht zwar schlechtere Resultate als reine CNN Architekturen, trotzdem sehen wir in dieser Architekturform noch Potential. Da wir vermuten, dass die Verwendung eines oder mehrerer Dense Layern zwischen dem CNN und dem LSTM die zeitliche Information vernichtet, wären in zukünftigen Arbeiten die Verbindung des LSTMs mit den Feature Maps des Maxpooling-Layers zu untersuchen.

Variante mit einem LSTM

Eine mögliche Anpassung der Architektur wäre das LSTM direkt mit dem letzten MaxPooling-Layer zu verbinden. Dabei würde jede Featuremap in einen Vektor umgewandelt und die einzelnen Feature-Maps würden als Zeitschritte verarbeitet. Da mit einer Convolution in Frequenzrichtung die zeitliche Information erhalten bleibt, scheint dieser Ansatz sinvoll.

Variante mit mehreren LSTMs

Eine weitere Möglichkeit wäre mehrere LSTMs zu verwenden. Dies würde bedeuten, dass für jede Feature Map des CNNs ein eigenes LSTM verwendet wird. Dabei würde jede Featuremap wie ein eigenese Spektrogramm behandelt. Die Ausgänge der LSTMs müssten nach der Verarbeitung wieder zusammengeführt werden.

Es wäre auch denkbar eine Architektur mit mehr als zwei Convolutional Layern zu testen, da tiefere Netzwerke bereits in anderen Problembereichen erfolgreich eingesetzt werden.

6.3. Form der Input Daten

Veränderung der Spektrogramme

Eine mögliche Veränderung der Form der Input Daten wäre es, die Spektrogramme in einer höheren zeitlichen Auflösung zu generieren. Da die jetztigen Spektrogramme aber bereits eine sehr kurze Hop Length haben, wäre es durchaus möglich, dass bei einer weiteren verkürzung, zuviele redundante Informationen vorhanden wären.

Andere Art von Spektrogrammen

Momentan werden Mel-Spektrogramme als Inputs verwendet. Diese komprimieren die Frequenzinformation des Spektrogramm bereits. Es wäre Interessant zu sehen ob ein Netzwerk mit einem normalen Spektrogramm, welches das volle Frequenzspektrum umfasst bessere Resultate liefern könnte.

Verwendung von Audiodaten

Da Neural Networks selber lernen können welche Features wichtig sind für einen Problembereich und die entsprechenden Transformationen der Daten selber vornehmen können, wäre es interessant zu erforschen wie ein LSTM sich verhalten würde, wenn man ihm „rohe“ Audiodaten als Input geben würde. Bis anhin greifen alle getesteten Ansätze auf Mel-Spektrogramme als Input zurück. Diese Variante scheint nur sinnvoll in der Kombination mit LSTMs/RNNs, da nach unserem Verständnis die CNNs die Spektrogramme als Bilder auffassen und für diese Betrachtungsweise sollten Spektrogramme relevantere Daten beinhalten.

Grösse des Datensatzes

Der TIMIT Datensatz eignet sich wie, sich in mehreren Arbeiten gezeigt hat, gut um Ansätze auf die Tauglichkeit im Bereich Speaker Recognition zu prüfen. Doch könnte ein Grösserer Datensatz mit mehr Sprechern interessant sein. Auch wäre es Interessant zu Untersuchen wie die Netzwerke mit Audio Daten umgehen können auf denen noch Umgebungsgeräusche vorhanden sind. Interessant wäre es auch zu Untersuchen ob ein Sprecher, welcher die Sprache wechselt wieder erkannt werden kann.

7. Verzeichnisse

Literaturverzeichnis

[1] Yoshua Bengio. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166, 1994. ISSN 19410093. doi: 10.1109/72.279181.

[2] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014. ISSN 09205691. doi: 10.3115/v1/D14-1179.

[3] Santiago Fernández, Alex Graves, and Juergen Schmidhuber. Phoneme recognition in TIMIT with BLSTM-CTC. *Arxiv preprint arXiv08043269*, abs/0804.3:8, 2008.

[4] F.A. Gers and J. Schmidhuber. Recurrent nets that time and count. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, 1:189–194 vol.3, 2000. ISSN 1098-6596. doi: 10.1109/IJCNN.2000.861302.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Abrufdatum: 16.12.2016, 2016. URL http://www.deeplearningbook.org.

[6] A. Graves, A.-R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (6): 6645–6649, 2013. ISSN 1520-6149. doi: 10.1109/ICASSP.2013.6638947.

[7] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM networks. *Proceedings of the International Joint Conference on Neural Networks*, 4:602–610, 2005. ISSN 08936080. doi: 10.1109/IJCNN.2005.1556215.

[8] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2016. ISSN 21622388. doi: 10.1109/TNNLS.2016.2582924.

[9] Sepp Hochreiter, S Hochreiter, Jürgen Schmidhuber, and J Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–80, 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

[10] Beigi Homayoon. *Fundamentals of Speaker Recognition*. Springer, 2011. ISBN 9780387775920.

[11] Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition, 2016. URL http://cs231n.github.io/neural-networks-1/. Abrufdatum: 16.12.2016.

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs.LG]*, pages 1–15, 2014.

[13] Joel Larsson. recognition using an LSTM neural network Master Thesis in Robotics. 2014.

[14] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Lecun Y., Bengio Y., and Hinton G. Deep learning. *Nature*, 521(7553):436–444, 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

[15] Yanick Lukic and Carlo Vogt. Projektarbeit HS15 Studiengang Informatik Sprechererkennung mit Convolutional Neural Networks, 2015.

[16] Yanick Lukic and Carlo Vogt. Bachelorarbeit FS16 Studiengang Informatik Automatische Stimmerkennung mit Deep Lear- ning. 2016.

[17] Yanick Lukic, Carlo Vogt, Oliver Dürr, and Thilo Stadelmann. Speaker Identification and Clustering Using Convolutional Neural Networks. pages 13–16, 2016.

[18] Andrew Morris, Dalei Wu, and Jacques Koreman. GMM based Clustering and Speaker Separability in the TIMIT Speech Database. (Saar-IP-08-08-2004):1–13, 2004.

[19] Michael A. Nielsen. Neural Networks and Deep Learning. chapter Chapter 3. Determination Press, 2015. URL http://neuralnetworksanddeeplearning.com/. Abrufdatum: 10.12.2016.

[20] Christopher Olah. Understanding LSTM Networks, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Abrufdatum: 16.12.2016.

[21] Douglas A. Reynolds. Speaker identification and verification using Gaussian mixture speaker models. *Speech Communication*, 17(1-2):91–108, 1995. ISSN 01676393. doi: 10.1016/0167-6393(95)00009-D.

[22] Sebastian Ruder. An overview of gradient descent optimization algorithms. *Web Page*, pages 1–12, 2016.

[23] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *Neural and Evolutionary Computing*, (Cd), 2014. doi: arXiv:1402.1128.

[24] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR)*, 15:1929–1958, 2014. ISSN 15337928. doi: 10.1214/12-AOS1000.

[25] Thilo Stadelmann. *Voice Modeling Methods for Automatic Speaker Recognition*. PhD thesis, 2010.

Abbildungsverzeichnis

Tabellenverzeichnis

A. Anhang

A.1. Projektmanagement

A.1.1. Aufgabenstellung

Abstract

Since 2014, subsequent Bachelor student's thesis projects (PA/BA) have researched the topic of automatic speaker (voice) recognition. This resulted in a first scientific publication in spring 2016, presented by the students at an international research conference in Italy. In this PA, our successful speaker recognition approach shall be extended to incorporate new ideas for sequence modelling, especially based on recurrent neural networks (RNN). This requires a literature review, the creation of hypotheses of what might work, and the experimental evaluation of these ideas (which typically involves "hacking togetherßome software). The end product is a verified or justifiably rejected approach to deep sequence learning-based speaker recognition.

Prerequisites

Some knowledge of machine learning methods and experimentation is beneficial, but not required. Speech processing know-how is not required. Comparable projects in the last several semesters have shown that interested students with curiosity, good programming skills and an experimental mindset do very well in submitting excellent theses in the end. Example source code and experimental setups already exist and can be used. This topic can be seen as an entry point to research-oriented work in machine learning and image processing. For example, students from HS2015 just submitted their first paper that resulted directly from their PA. The project will typically be conducted and documented in German. Programming can be done e.g. using the Python language and Lasagne/Theano library stack.

Further information

https://dublin.zhaw.ch/ stdm/?page_id=77

A.2. Anleitung

Dieser Abschnit liefert Anleitung für die Verwendung des Codes, welcher für die Datenextraktion und -generation, sowie dem Trainieren der Netzwerke verwendet werden kann. Voraussetzung ist eine Filestruktur wie in Abbildung A.1.

A.2.1. Daten extrahieren

Der TIMIT Datensatz muss unter *codetraining TIMIT* abgelegt sein, unter emphcodetrainingspeaker_list sind die Sprecherlisten abgelegt, für welche die Extraktion durchgeführt wird. Die Extraktion wird mit dem Python script *main_data_extractor.py* ausgeführt. Die extrahierten Spektrogramme werden im Pickle Dateiformat unter *code/data/training/TIMIT_extracted* gespeichert. Dabei wird jeweils der komplette Satz in ein Spektrogramm abgespeichert wie in Abschnitt 3.2 beschrieben. Um die so extrahierten Dateien in unserem Framework weiterverwenden zu können muss der Trainingsdatensatz mit dem prefix *train_* versehen sein. Für den Testdatensatz muss das Präfix *test_* verwendet werden.

A.2.2. Netzwerke trainieren

Die Netzwerke sind im Ordner *src/nets* abgelegt, jedes Netzwerk ist eine eigene Klasse. Ausgeführt werden sie über das Skript *network_runner.py* im Ordner *src*. Im Folgenden ist eine Beispiel Signatur für ein LSTM Network aufgeführt.

```
1 class bilstm_2layer(object)
2   def __init__(self, name, training_data, n_hidden1=128 , n_hidden2
        =128, n_classes=630, n_epoch=1000, segment_size=15, frequency
        =128):
```

Wie zu erkennen ist sind für alle Parameter ausser dem Namen und den Trainingsdaten bereits Standardwerte vorhanden, welche überschrieben werden können. Es sind die Folgenden Netzwerke vorhanden:

- *bilstm_2layer.py*
- *bilstm_2layer_dropout.py*
- *bilstm_3layer.py*
- *bilstm_4layer.py*
- *bilstm_5layer.py*
- *cnn_lstm_layer7.py*
- *cnn_lstm_layer5.py*

Ein Aufruf eines Solchen Netzwerks kann wie folgt aussehen:

```
1 import core.bilstm_2layer as bilstm2
2
3 bilstm2.bilstm_2layer.('netzwerk_name_1', 'train_data_100.pickle',
    n_classes=100, n_epoch =10)
```

In diesem Fall wird die Anzahl Klassen (Sprecher) überschrieben und die zu trainierenden Epochen auf 10 gesetzt. Es wird vorausgesetzt, dass die Trainingsdaten sich im Verzeichnis *cod/data/TIMIT_extractet* befinden Der Name wird verwendet um die csv Datei, das Netzwerk selber und die Plots zu speichern. Die csv Datei wird im Verzeischnis *code/data/experiments/logs* abgespeichert das netzwerk unter *code/data/experiments/nets* und die Plots in *code/data/experiments/plots*. Für die Plots wird die *src/core/plot_saver.py* verwendet, das Skript erstellt jeweils einen Accurcy und Loss Plot. Falls eine andere Verzeichnisstruktur verwendet werden soll, können die Pfade für die oben erwähnten Funktionalitäten im script *src/core/settings.py* angepasst werden. Am ende des Trainings wird automatisch eine

Evaluation ausgeführt mit Hilfe von *src/analysis/data_analysis.py* welches die Resultate in die Datei *test_scores.txt* schreibt.

Batch Generators

Die Batchgenerators befinden sich im Skript *src/core/data_gen.py* es existieren 2 Batch Generators einer für LSTMs und einer für CNNs. im folgenden sind die beiden Signaturen aufgeführt:

```
1 def batch_generator_lstm(X, y , batch_size, segment_size=15)
2 def batch_generator(X, y, batch_size, segment_size=100)
```

Dabei sind X die Trainingsdaten und y die Labels.

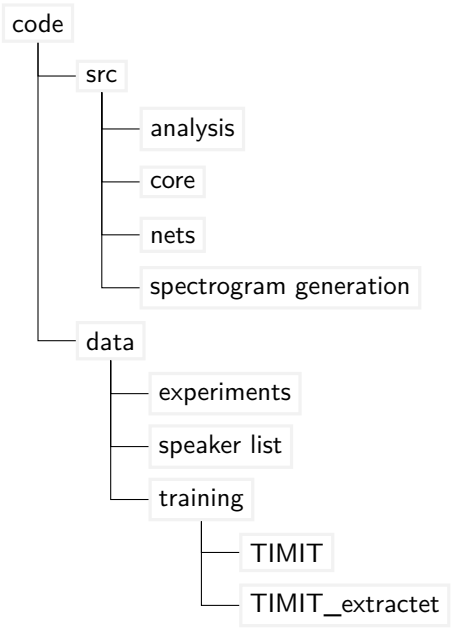


Abbildung A.1.: Verzeichnisstruktur der Applikation

A.2.3. Netzwerk testen

Die Netzwerke werden automatisch am ende jedes Trainings getestet. Falls Bedarf vorhanden ist eine Evalutation nochmals durchzuführen, kann dies über das Skript *src/accuracy_runner.py* ausgeführt werden. Es wird wie beim Training die Funktion *calculate_test_accuracies* aus der Datei *src/analysis/-data_analysis* verwendet. Die Signatur der Funktion is wie folgt:

```
1 def calculate_test_accuracies(network_name, test_data, one_file,
      write_to_file, is_LSTM, segment_size= 15):
```

Dabei sucht das Skript, das Netzwerk mit *network_name* im Verzeichnis *code/data/experiments/-nets*.

A.3. Weiteres

A.3.1. Elektronische Daten

Die Elektronischendaten, welche sich auf dem beigelegten USB Stick befinden, haben die folgende Form:

- **01_Dokumentation**: Enthält die Projektarbeit im PDF Format.
- **02_Experimente**: Enthält die logs der Experimente und Plots. Zusätzlich sind noch einige Netzwerke im HDF5 Format vorhanden.
- **03_Code**: Enthält den Python Code sowie, die TIMIT Daten sowie die Verzeichnissstruktur um alle Funktionen des Codes verwenden zu können.

A.3.2. Sprecherlisten

100 Sprecher:

MSES0	MBWM0	MAJP0	MPFU0
FJSK0	FJEM0	MRMB0	FJHK0
FSCN0	FGRW0	FKDW0	MJXL0
MCHL0	FMMH0	MZMB0	MCEW0
MJRP0	MREB0	FMBG0	MBNS0
FDAW0	MRAB0	MLJC0	MTLB0
MLEL0	MGXP0	FJSP0	FJCS0
MJLG1	MMWB0	MKRG0	FKDE0
MLIH0	FHEW0	MRAI0	MGLB0
MRGS0	MJAC0	MRPP0	MCDD0
FCMH0	MGMM0	MMWS0	FMEM0
MKLT0	MJMD0	MKLS0	FCRZ0
FMCM0	FDTD0	FSJG0	FSBK0
MGAR0	FMML0	MRJS0	MDVC0
MJFC0	FLAS0	MGWT0	FMJU0
FKLC1	FMLD0	MGSH0	MRML0
MTQC0	MMWS1	MMWH0	MJSR0
MWDK0	MMGG0	FPKT0	MCTH0
FDML0	FJRP1	MBBR0	FREH0
MAKB0	FPAF0	MMJB1	MHXL0
MCCS0	FSGF0	MSVS0	FETB0
MWAC0	FAPB0	MMDS0	MJWT0
FJRE0	MLSH0	FBMH0	FGDP0
FCLT0	FBAS0	FCAL1	MDMA0
MGRP0	MRCS0	MJLB0	MRJT0

10 Sprecher:

FJRE0	MREB0	FSLB1	MABW0
FAKS0	MRGG0	FELC0	
FCMH0	MJAR0	MWEW0	