

EBERHARD-KARLS-UNIVERSITÄT TÜBINGEN
Wilhelm-Schickard-Institut für Informatik
Lehrstuhl Kognitive Systeme

Bachelorarbeit

Audiosignal-Separierung mittels Rekurrenter Neuronaler Netze (RNNs)

Paul Stöckle

Prüfer: Prof. Dr. rer. nat. Andreas Zell
Wilhelm-Schickard-Institut für Informatik

Betreuer: Sebastian Otte
Wilhelm-Schickard-Institut für Informatik

Begonnen am: 1. März 2016

Beendet am: 1. August 2016

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Tübingen am 1. August 2016

Paul Stöckle

Kurzfassung. Die Audiosignal-Separierung beschreibt den Vorgang, aus einem gegebenen Musikstück (Audiodatei), einzelne Instrumente oder Komponenten zu isolieren und stellt ein Problem in der digitalen Signalverarbeitung dar. Anwendungsbereiche sind Störgeräusch-Unterdrückung, Nachbearbeitung einzelner Instrumente, Remixing oder Sampling. In der vorliegenden Bachelorarbeit sollen Methoden der Separierung, basierend auf rekurrenten neuronalen Netzwerken (RNNs), insbesondere Long-Short-Term-Memory Netzen (LSTM), erprobt werden. Aufgrund der zyklischen Struktur von RNNs sind diese besonders für die Verarbeitung von Sequenzen und dem Erkennen zeitlicher Zusammenhänge und Verläufe geeignet.

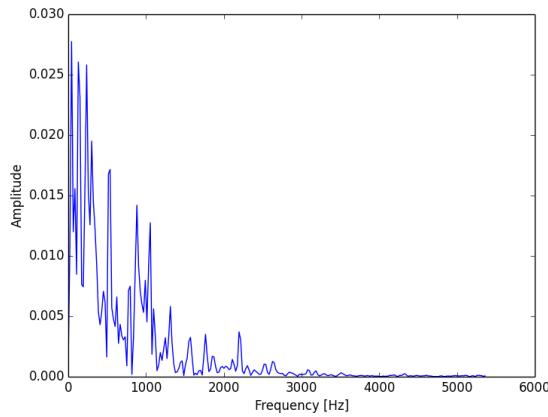
Inhaltsverzeichnis

1	Einführung	1
2	Methoden	3
2.1	Kurzzeit-Fourier-Transformation	3
2.1.1	Komplexe Zahlen	3
2.1.2	Diskrete Fourier-Transformation	4
2.1.3	Kurzzeit Fourier-Transformation	4
2.1.4	Fensterfunktionen	8
2.1.5	Segmentierte Faltung (Overlap-Add)	9
2.2	Künstliche Neuronale Netze (ANNs)	9
2.2.1	Mehrschichtiges Perzeptron (Multilayer-Perzeptron)	10
2.2.2	Rekurrente Netze	12
2.2.3	LSTM-Netze	14
2.2.4	Gradientenabstiegsverfahren	17
2.3	Methode der Audiosignal-Separierung	18
2.3.1	Netzeingabe	19
2.3.2	Trainingsphase	21
2.3.3	Anwendungsphase	24
2.3.4	Netzstruktur	24
2.3.5	Implementierung	26
3	Ergebnisse	27
3.1	Training - Teil 1	28
3.2	Training - Teil 2	38
4	Diskussion	46
	Literaturverzeichnis	48

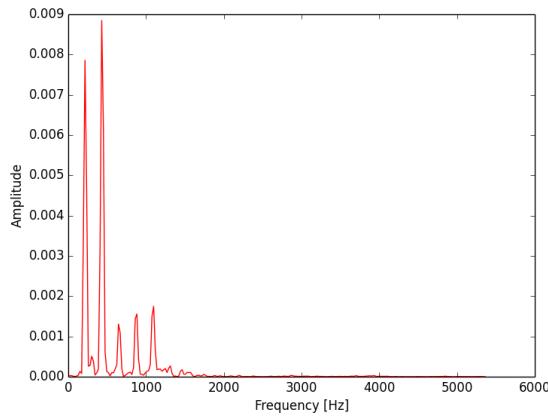
1 Einführung

Das menschliche Gehör besitzt die Fähigkeit, selektiv zu hören. Aus einem Gemisch von Schallquellen kann dabei eine bestimmte Schallquelle fokussiert und durch das Unterdrücken anderer Schallquellen extrahiert werden. Das Extrahieren einer Schallquelle wird in der Signalverarbeitung als Audiosignal-Separierung bezeichnet. Bisherige Lösungsstrategien für die Separierung einer Tonquelle aus komplexen Musikstücken, konnten keine qualitativ hochwertigen Ergebnisse liefern. Damit stellt die Audiosignal-Separierung, nach wie vor, ein Problem in der digitalen Signalverarbeitung dar. Anwendungen finden sich unter anderem in der Musikproduktion bei der Störgeräusch- Unterdrückung oder der Nachbearbeitung einzelner Audiospuren. Die Grundlage der Audiosignal-Separierung eines digitalen Signals bietet die Möglichkeit, Schwingungen in ihrem zeitlichen Verlauf, als auch dem zugehörigen Frequenzspektrum analysieren zu können. Die Bearbeitung eines Signals in der Frequenzdomäne ermöglicht, im Gegensatz zur Verarbeitung in der Zeitdomäne, eine direkte Analyse und Anpassung der Frequenzen. Dadurch werden Filter wie der Hoch- oder Tiefpass in der Signalverarbeitung realisiert. Zudem ist eine sequentielle Verarbeitung der einzelnen Abtastwerte in der Zeitdomäne äußerst zeittintensiv. Unter der Annahme, dass bestimmte Klangerzeuger, wie beispielsweise Stimmbänder, Schlaginstrumente, Streichinstrumente, etc. , charakteristische Bereiche und Verläufe in ihren Frequenzspektren aufweisen, kann eine Separierung mittels Manipulation der Frequenzen erfolgen. Ziel ist es, das Frequenzspektrum eines aus mehreren Quellen gemischten Signals so zu manipulieren, dass das manipulierte Signal, dem im Anwendungsfall unbekannten Signal der zu isolierenden Tonquelle bestmöglich gleicht. Dieses Problem ist als blind audio source Separierung (BASS) bekannt. Für die Verarbeitung in der Frequenzdomäne wird ein Signal in Abschnitte zerlegt und dieses via Kurzzeit-Fourier-Transformation in den Frequenzraum transformiert. Erfolge konnten bereits durch die Verwendung eines neuronalen Netzes für die Erzeugung binärer Masken erzielt werden [SRP15]. Binäre Masken werden mit den Amplituden der Frequenzen aus dem Frequenzspektrum multipliziert. Dadurch können bestimmte Frequenzen ausgelöscht werden, um eine Näherung an das Frequenzspektrum des Quellsignals zu erzielen. Für die Erzeugung einer binären Masken durch das neuronale Netz wird als Eingabe lediglich ein lokaler Ausschnitt des zu separierenden Signals verwendet. Aufgrund der verwendeten, vorwärtsgerichteten Netzstruktur, existiert damit bei dieser Methode kein zeitlicher Kontext bei der Signalverarbeitung. Des Weiteren ist die Anpassung der Frequenzen durch die binären Masken stark eingeschränkt.

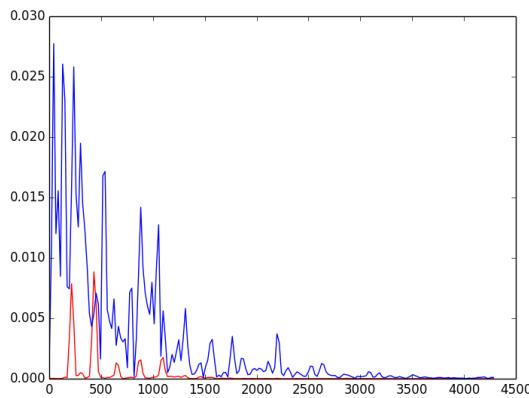
In dieser Arbeit soll eine Verbesserung der Audiosignal-Separierung durch die Verwendung eines rekurrenten, neuronalen Netzes ermöglicht werden. Dadurch wird durch rekurrente Strukturen der Verarbeitung einer Eingabesequenz ein zeitlicher Kontext hinzugefügt. Als rekurrente Strukturen sollen dabei insbesondere sogenannte Long-Short-Term-Memorys [HS97] (LSTMs) verwendet werden. Diese besitzen die Fähigkeit, zeitliche Zusammenhänge und Verläufe bei der Verarbeitung von Sequenzen erkennen zu können. Zudem sollen anstelle binärer Masken die Frequenzspektren des Ausgabesignals durch das neuronale Netz erzeugt werden.



(a) Frequenzspektrum Mix



(b) Frequenzspektrum Gesang



(c) Frequenzspektrum Mix+Gesang im Frequenzbereich
0-4500Hz

Abbildung 1.1: Frequenzspektren einer Audiosequenz von 2048 Abtastwerten mit einer Abtastrate von 44100 s^{-1}

Die Amplituden der einzelnen Tonquellen überlagern sich hier besonders im niederen Frequenzbereich. Niedere Frequenzen sind im Gesang mit geringerer Amplitude als im Mix vorhanden. Durch Anpassung der Frequenzen soll eine Näherung an das Frequenzspektrum (b) bei gegebenem Frequenzspektrum (a) erreicht werden.

2 Methoden

In diesem Kapitel werden zunächst alle nötigen Grundkenntnisse und Methoden der Audioseparierung vermittelt. In Abschnitt 2.1 wird die Kurzzeit-Fourier-Transformation erläutert, mit welcher ein Signal aus dem Zeitbereich in den Frequenzbereich transformiert werden kann. In Abschnitt 2.2 werden die einzelnen Strukturen neuronaler Netze beschrieben und die verwendete LSTM-Struktur definiert.

2.1 Kurzzeit-Fourier-Transformation

2.1.1 Komplexe Zahlen

Die Fourier-Transformation arbeitet auf komplexen Vektoren und liefert die komplexen Fourierkomponenten eines Signals. Aus den Fourierkomponenten können die Amplituden der enthaltenen Frequenzen bestimmt werden. Im Folgenden werden die für die komplexen Zahlen benötigten Formeln definiert.

Im Allgemeinen besitzt eine komplexe Zahl die Form

$$z = a + bi , \quad (2.1)$$

wobei a als Realteil und b als Imaginärteil von z bezeichnet werden. Zudem werden die Notationen $a = Re(z)$ und $b = Im(z)$ verwendet. Der reelle Betrag einer komplexen Zahl berechnet sich zu

$$|z| = \sqrt{a^2 + b^2} \quad (2.2)$$

und das Argument $\arg(z)$ zu

$$\arg(z) = \begin{cases} \arctan\left(\frac{b}{a}\right) & \text{für } a > 0 \\ \arctan\left(\frac{b}{a}\right) + \pi & \text{für } a < 0, b \geq 0 \\ \arctan\left(\frac{b}{a}\right) - \pi & \text{für } a < 0, b < 0 \\ \frac{\pi}{2} & \text{für } a = 0, b > 0 \\ -\frac{\pi}{2} & \text{für } a = 0, b < 0 . \end{cases} \quad (2.3)$$

Mit dem Betrag und dem Argument einer komplexen Zahl, lässt sich diese auch mit $r = |z|$ und $\varphi = \arg(z)$ in der Exponentialdarstellung

$$z = r \cdot e^{i\varphi} \quad (2.4)$$

oder der trigonometrischen Darstellung

$$z = r \cdot (\cos(\varphi) + i \cdot \sin(\varphi)) \quad (2.5)$$

der Polarform darstellen.

2.1.2 Diskrete Fourier-Transformation

Die Diskrete Fourier-Transformation ist eine wichtige Methode in der Signalverarbeitung, um ein endliches, periodisches, zeitdiskretes Signal auf das zugehörige diskrete Frequenzspektrum abzubilden. Die Transformation liefert die komplexen Koeffizienten, mit welchen das Signal als Summe komplexer Sinus-Schwingungen dargestellt werden kann. Diese bestehen aus reellen Amplituden und Phasen.

Wir definieren die diskrete Fourier-Transformation (DFT) zu:

$$y_k \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} x_n \exp^{-in\omega_k}, \quad k = 0, \dots, N-1. \quad (2.6)$$

Dabei ist N die Anzahl der Elemente des Eingangssignals, x_n der komplexe Eingabewert an der Stelle n , y_k der komplexe Ausgabewert (der komplexe Koeffizient) an der Stelle k und ω die Kreisfrequenz mit $\omega_k = 2\pi k/N$. Zudem definieren wir die inverse diskrete Fourier-Transformation (IDFT) zu:

$$x_n \stackrel{\text{def}}{=} \frac{1}{N} \cdot \sum_{k=0}^{N-1} y_k \exp^{in\omega_k}, \quad n = 0, \dots, N-1. \quad (2.7)$$

Dabei ist y_k der komplexe Wert an der Stelle k des Frequenzspektrums und x_n der Wert an der Stelle n des zeitdiskreten Ausgangssignals. In der Anwendung werden die DFT und IDFT durch den Algorithmus der schnellen Fourier-Transformation (FFT) [CT65] bzw. der inversen schnellen Fourier-Transformation (IFFT) berechnet. Mit einer Laufzeit von $T(N) = \mathcal{O}(N \cdot \log(N))$ wird damit eine effiziente Berechnung der DFT und IDFT möglich.

2.1.3 Kurzzeit Fourier-Transformation

Die Kurzzeit-Fourier-Transformation (STFT) ist die Aneinanderreihung der Frequenzspektren eines in Abschnitte unterteilten Signals. Damit bildet die STFT eine Möglichkeit, die zeitliche Änderung, der im Signal enthaltenen Frequenzen, darzustellen.

Akustische Instrumente besitzen charakteristische Frequenzbereiche und Frequenz-Verläufe. Deshalb bietet sich die diskrete Fourier-Transformation unter Anwendung der Kurzzeit Fourier-Transformation an, um ein Audiosignal vor dessen Eingabe in ein neuronales Netz, in seine Frequenzspektren zu zerlegen. Die aus den Frequenzspektren berechenbaren Amplituden der einzelnen Frequenzen können parallel an das Netz angelegt werden. Dies ermöglicht ein effizientes Berechnen des Netzes. Damit das Erkennen zeitlicher, charakteristischer Verläufe der Frequenzen möglich ist, wird das Eingangssignal zuvor in kurze Blöcke zerlegt (im Training wurden Blöcke mit jeweils 2048 Abtastwerten verwendet) und mit einer Fensterfunktion gewichtet. Da die einzelnen Blöcke abgeschnitten werden, kann das im Block enthaltene Signal meistens nicht unendlich periodisch fortgesetzt werden.

Dies führt bei der Transformation zum sogenannten Leck-Effekt, durch welchen das errechnete Spektrum breit und ungenau wird. Dadurch entstehen Fehler bei der Rücktransformation des Signals durch die IDFT. Um den Effekt zu mindern, wird auf die Blöcke eine Fensterfunktion angewandt, welche das Signal am Anfang und Ende des Blocks ein- und ausblendet. Dadurch wird die Amplitude der Anfangs- und Endfrequenz mit dem Faktor 0 gewertet, wodurch das Signal periodisch fortgesetzt werden kann.

Wir definieren die STFT für ein zeitdiskretes Eingangssignal wie folgt [AR77, Smi11]:

$$X_{km} \stackrel{\text{def}}{=} \sum_{n=-\infty}^{\infty} [x_n w_{n-mR}] \exp^{-in\omega_k}. \quad (2.8)$$

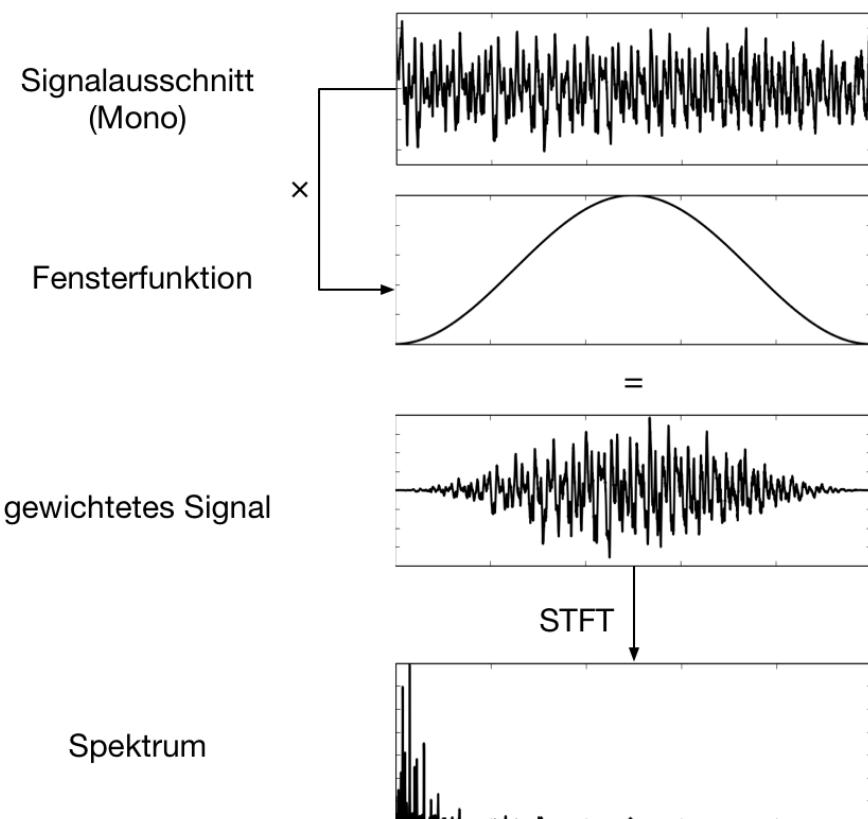


Abbildung 2.1: Transformation von Signalausschnitt zum Frequenzspektrum
 Ein nicht periodisch fortsetzbares, abgeschnittenes Audiosignal, welches mit der Fensterfunktion 2.3(a) gewichtet wurde und mit der FFT in den Frequenzbereich transformiert wurde.

Dabei ist x_n der Abtastwert an der Stelle n des Eingabesignals x , w_{n-mR} ist der Wert der Fensterfunktion w mit der Breite M (Abtastwerte), an der Stelle $n - mR$, wobei m der Index der DFT bzw. der n -te Sprung der Fensterfunktion mit der Sprungweite R ist.

Zudem definieren wir die inverse Kurzzeit Fourier-Transformation (iSTFT) durch:

$$x_n \stackrel{\text{def}}{=} \frac{1}{N} \sum_{-\infty}^{\infty} \sum_{k=0}^{N-1} X_{km} \exp^{jn\omega_k}. \quad (2.9)$$

Kupfmüllersche Unschärferelation

Die Kurzzeit-Fourier-Transformation besitzt die Eigenschaft, dass mit zunehmender Auflösung im Frequenzbereich durch eine Vergrößerung der Fensterbreite, die Auflösung im Zeitbereich sinkt. Ebenso sinkt die Auflösung im Frequenzbereich mit einer zunehmenden Auflösung im Zeitbereich. Das Produkt von Frequenz und der Zeit ergibt dabei stets einen konstanten Wert. Diese Relation wird als die Kupfmüllersche Unbestimmtheitsrelation bezeichnet. In Abbildung 2.2 ist ein kurzes Signal mit einer Dauer von einer Sekunde, bestehend aus 2 Schlägen einer Bassdrum, mit unterschiedlichen Auflösungen im Frequenzbereich dargestellt. Bei einer kleinen Fensterbreite und damit einer hohen Auflösung im Zeitbereich, folglich einer niedrigen Auflösung im Frequenzbereich, kann der Zeitpunkt des Schlags mit großer Genauigkeit festgestellt werden. Mit einer Vergrößerung der Fensterbreite wird der Frequenzbereich zwar höher aufgelöst, allerdings kann der genaue Zeitpunkt des Schlags nicht mehr festgestellt werden (breiter Balken im Spektrogramm).

Spektogramme

Die zeitliche Änderung des Frequenzspektrums eines Signals, kann durch ein Spektrogramm graphisch dargestellt werden. Die Änderung des Frequenzspektrums bezieht sich dabei auf die Änderungen der Amplituden der einzelnen Frequenzen über die Zeit. Die Amplituden-Werte werden in der logarithmischen Größe für den Schallpegel in Dezibel [dB] angegeben und werden meist farblich kodiert, um das Spektrogramm in einem 2-dimensionalen Graphen darzustellen. Die Zeit wird dabei auf der horizontalen und die Frequenz auf der vertikalen Achse abgebildet. Die verwendeten Spektogramme wurden mit Audacity[Tea08] erstellt. Die Amplitudenwerte sind wie folgt farbkodiert:

- weiß für Werte über -20 dB
- rot bis weiß für Werte von -40 dB bis -20 dB
- magenta bis rot für Werte von -60 dB bis -40 dB
- dunkelblau bis magenta für Werte von -80 dB bis -60 dB
- hellblau bis dunkelblau für Werte von -100 dB bis -80 dB
- grau für Werte unter -100 dB

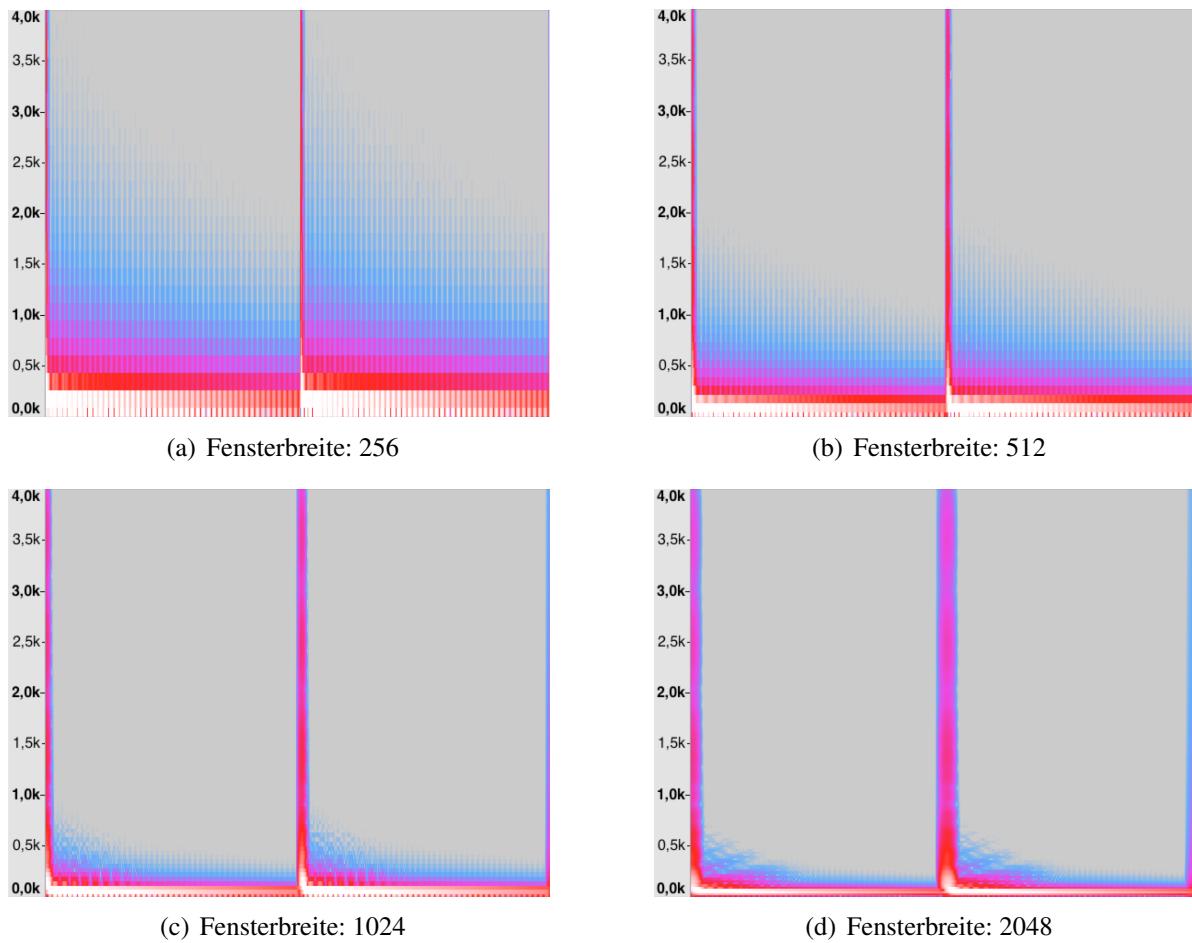


Abbildung 2.2: Spektrogramme des Audiosignals einer Bassdrum (2 Schläge) mit einer Dauer von 1s und der Verwendung unterschiedlicher Fensterbreiten

2.1.4 Fensterfunktionen

Fensterfunktionen legen die Gewichtung der Abtastwerte eines Signals fest. Sie sind bei der Kurzzeit Fourier-Transformation ein wichtiges Hilfsmittel, um den Leck-Effekt bei den einzelnen, diskreten Fourier-Transformationen zu mindern. Da das Signal für die STFT in Blöcke zerlegt wird, welche jeweils mit einer Fensterfunktion gewichtet werden, ist darauf zu achten, die einzelnen Blöcke durch Anpassung der Sprungweite R zu überlappen. Ansonsten kann es aufgrund der geringen Gewichtung im Anfangs- und Endbereich der Fensterfunktion zu einem Informationsverlust bei der Rücktransformation in den Zeitbereich kommen. Eine bekannte Fensterfunktion ist das Von-Hann-Fenster [BT58], welches wie folgt definiert ist:

$$w(n) = 0.5 - 0.5 \cdot \cos\left(\frac{2\pi n}{M-1}\right), \quad 0 \leq n \leq M-1. \quad (2.10)$$

Zudem definieren wir das in dieser Arbeit neben dem Von-Hann-Fenster verwendete Tukey-Fenster [Har78]:

$$w(n) = \begin{cases} \frac{1}{2} \left[1 + \cos\left(\pi \left(\frac{2n}{\alpha(M-1)} - 1 \right)\right) \right] & \text{für } 0 \leq n \leq \frac{\alpha(M-1)}{2} \\ 1 & \text{für } \frac{\alpha(M-1)}{2} \leq n \leq (M-1)(1 - \frac{\alpha}{2}) \\ \frac{1}{2} \left[1 + \cos\left(\pi \left(\frac{2n}{\alpha(M-1)} - \frac{2}{\alpha} + 1 \right)\right) \right] & \text{für } (M-1)(1 - \frac{\alpha}{2}) \leq n \leq (M-1), \end{cases} \quad (2.11)$$

welches als die Faltung einer auf $\frac{\alpha M}{2}$ Abtastwerte reduzierten Kosinus-Fensterfunktion mit einem Rechteckfenster der Breite $1 - \frac{\alpha}{2}$ aufgefasst werden kann. Demnach können alle Abtastwerte, welche nicht überlappt werden und nach der Rücktransformation in das Signal übernommen werden sollen, durch eine Anpassung des Parameters α mit dem Wert 1 gewichtet werden:

$$\alpha = \frac{2(L-2R)}{L}, \quad (2.12)$$

wobei L der Länge eines Blocks aus dem Signal und R der Sprungweite entspricht.

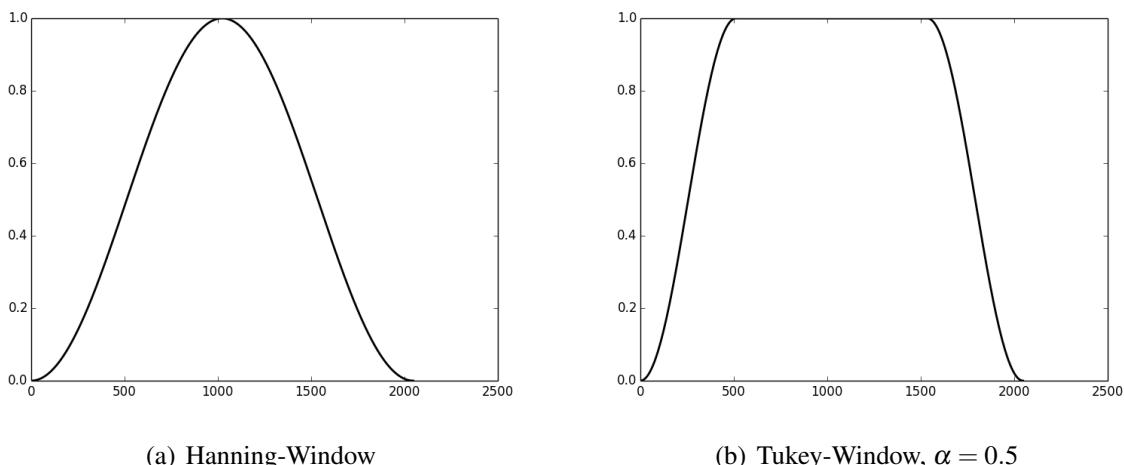


Abbildung 2.3: Bei der STFT verwendete Fensterfunktionen

2.1.5 Segmentierte Faltung (Overlap-Add)

Die segmentierte Faltung ist eine Methode, um ein Signal, welches zur Modifizierung im Frequenzraum, in sich überlappende Blöcke geteilt wurde, nach der Rücktransformation wieder zusammenzusetzen. Insbesondere wurde hier die Constant-Overlap-Add Methode (COLA) [Smi11] verwendet. Bei dieser Methode werden die Werte aus den einzelnen Gewichtungen, mit den um die Sprungweite mR verschobenen Fensterfunktionen, jeweils addiert. Der Wert an der Stelle n gewichtet mit der um m Sprünge verschobenen Fensterfunktion kann durch

$$x_m(n) \triangleq x(n)w(n - mR), \quad n \in (-\infty, +\infty) \quad (2.13)$$

berechnet werden, wobei R die Sprungweite in Abtastwerten und m den Index der Fensterfunktion bezeichnet.

Falls die Bedingung

$$\sum_{m=-\infty}^{\infty} w(n - mR) = 1, \quad \forall n \in \mathbb{Z}, \quad (2.14)$$

bei der Verwendung der COLA-Methode erfüllt ist, entspricht das Signal einer DTFT über das gesamte Eingangssignal der Aufsummierung der einzelnen DTFTs der in Blöcke geteilten Funktion. Die Gewichtung an der Stelle n , summiert über alle Fenster, entspricht damit dem Wert 1.

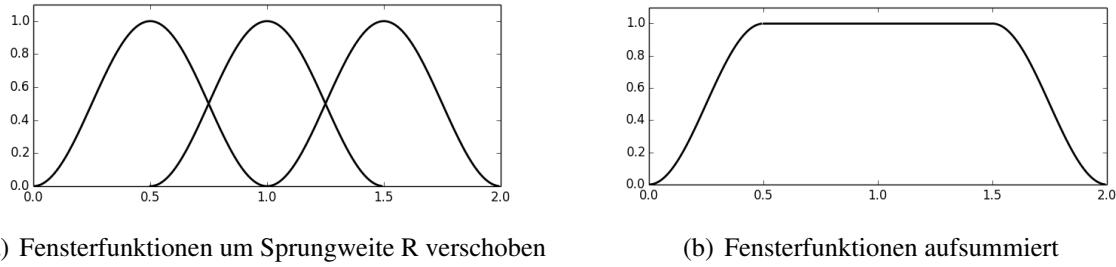


Abbildung 2.4: Für die Gewichtung eines Signals verschobene Fensterfunktionen (a), aufsummiert durch die COLA-Methode (b).

$$\sum_{m=-\infty}^{\infty} X_m(w) \triangleq DTFT_w(x) = X(w) \quad (2.15)$$

Damit die Bedingung erfüllt ist und mit der COLA-Methode und dem verwendeten Von-Hann-Fenster die segmentierte Faltung berechnet werden kann, muss für die Breite M des Von-Hann-Fensters w und die Sprungweite R gelten:

$$R = \frac{M}{2} \quad (2.16)$$

2.2 Künstliche Neuronale Netze (ANNs)

Im Allgemeinen sind künstliche, neuronale Netze (Artificial Neural Networks) Modelle zur Informationsverarbeitung und wurden ursprünglich von den biologischen Vorgängen im Gehirn inspiriert. Sie basieren auf einzelnen Einheiten, den Neuronen, welche durch gewichtete Kanten verbunden sind. Diese Verbindungen entsprechen im organischen Vorbild, den Synapsen. Eine

bereits bestehende Methode der Audiosignal-Separierung ist die Generierung binärer Masken durch ein einfaches Multilayer-Perzeptron (MLP) [SRP15]. Diese werden mit dem Spektrum eines Signalausschnitts multipliziert, um bestimmte Frequenzen auszulöschen. Dadurch soll eine Näherung an das Spektrum des Quellsignals erzielt werden. Ein Nachteil der Verwendung eines MLP besteht darin, dass nur ein lokaler Signalausschnitt für die Generierung einer binären Maske als Netzeingabe dient, und damit ein Vergangenheits- oder Zukunftskontext bei der Verarbeitung des Signals entfällt. Zudem werden in der Methode mit binären Masken ausschließlich die Amplituden im Frequenzspektrum während der Separierung verändert. Bei der Rücktransformation werden die Phasen der Frequenzen aus dem Eingangssignal verwendet. Da die Frequenzen durch die binären Masken jedoch nur ausgelöscht oder übernommen werden können, wird die Bearbeitung der Frequenzen zusätzlich stark beeinträchtigt, wodurch ein ideales Spektrum für das Ausgangssignal nicht oder nur schwer erzeugt werden kann. Aufgrund der genannten Probleme entstehen bei der Rücktransformation aus der Frequenzdomäne dadurch häufig sogenannte Artefakte. Diese äußern sich durch Störgeräusche oder Hintergrundflimmern im Ausgabesignal. Um die einhergehenden Probleme bei der Verarbeitung eines Signals durch binäre Masken und dem MLP zu beheben, wurde in dieser Arbeit eine Methode auf Basis sogenannter Long-Short-Term-Memorys (LSTMs) für die Audiosignal-Separierung erprobt. LSTM-Netze sind rekurrente Netze und können damit bei der Verarbeitung eines Signals temporale Zusammenhänge berücksichtigen. Insbesondere LSTMs besitzen die Fähigkeit, Verbindungen zwischen Ereignissen mit großer zeitlicher Distanz herzustellen. Dadurch entsteht ein deutlicher Vorteil gegenüber dem MLP. Zudem sollen die Spektren durch das Netz adaptiert werden, um genannte Einschränkungen bei der Verarbeitung mit binären Masken zu vermeiden.

In diesem Kapitel wird zunächst in Abschnitt 2.2.1 die Struktur und Berechnung eines mehrschichtigen Perzeptrons beschrieben. In Abschnitt 2.2.2 wird die Struktur des einfachen Perzeptrons um rekurrente Verbindungen zu einem sogenannten rekurrenten Netz erweitert. In Abschnitt 2.2.3 wird schließlich das komplexere, in der Methode zur Separierung, verwendete LSTM erläutert und definiert. Zur Formalisierung der ANNs wurde sich in diesem Kapitel an der Arbeit von Alex Graves [Gra12] orientiert.

2.2.1 Mehrschichtiges Perzeptron (Multilayer-Perzeptron)

Das Mehrschichtige Perzeptron (MLP) besteht aus einer Eingabeschicht, einer oder mehrerer folgender verdeckter Schichten und einer Ausgabeschicht. Ein an die Eingabeschicht angelegter Eingabevektor wird, ohne Bezug zu vorherigen Eingaben, von einer Schicht zur nächsten propagiert. Dabei werden die Ausgaben einer Schicht mit den zur Folgeschicht gerichteten Verbindungen gewichtet. Die einzelnen Neuronen besitzen eine sogenannte Aktivierungsfunktion, welche für die summierte Eingabe der einzelnen Verbindungen in das jeweilige Neuron die Ausgabe des Neurons in die nächste Schicht bestimmt. In der Lernphase des Netzes kann die Qualität der Ausgabe in Bezug auf das gewünschte Ergebnis durch eine Fehlerfunktion bestimmt werden, sowie der Fehler von der Ausgabeschicht zur Eingabeschicht zurück geführt werden kann (Backpropagation). Durch die Rückführung des Fehlers können die Gewichte des Netzes so angepasst werden, dass der Fehler minimiert wird. Da das Multilayer-Perzeptron nur vorwärtsgerichtete Verbindungen besitzt, entfällt ein Vergangenheits- oder Zukunftskontext, wodurch das Netz als eine Funktion verstanden werden kann, welche einem Eingabevektor, einen von ausschließlich diesem abhängigen Ergebnisvektor zuordnet. Demnach eignet sich

das MLP für Klassifizierungsprobleme oder einfache Abbildungen, jedoch nicht zum Erlernen von Abbildungen zwischen Eingabesequenzen.

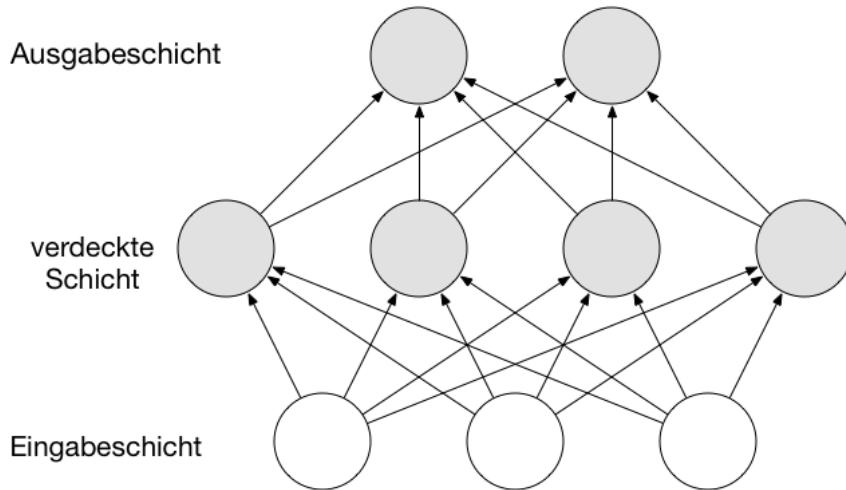


Abbildung 2.5: Multilayer Perzeptron

Ein Multilayer-Perzeptron mit einer Eingabeschicht, einer verdeckten Schicht und einer Ausgabeschicht. Graue Zellen besitzen eine Aktivierungsfunktion.

Forward Pass

Bei der Vorwärtspropagierung einer Eingabe von I Eingabewerten durch das Multilayer-Perzeptron (Forward Pass), berechnet sich die Ausgabe eines Neurons h der ersten verdeckten Schicht wie folgt:

$$a_h = \sum_{i=1}^I w_{ih} x_i \quad (2.17)$$

$$b_h = \theta_j(a_h) , \quad (2.18)$$

dabei ist a die Aktivierung des Neurons j , i der i -te Eingabewert des Eingabevektors x mit $|x| = I$, w_{ih} das Gewicht des Eingabewerts i zu Neuron h und b_i der Ausgabewert des Neurons h mit der Aktivierungsfunktion θ .

Die Ausgabe folgender verdeckter Schichten ist des Weiteren durch:

$$a_h = \sum_{h' \in H_{l-1}} w_{h'h} b_{h'} \quad (2.19)$$

$$b_h = \theta(a_h) \quad (2.20)$$

gegeben, wobei a_h ein Neuron der Schicht H_l mit der Aktivierungsfunktion θ ist und h' ein Neuron der vorherigen Schicht H_{l-1} mit der Ausgabe $o_{h'}$ und dem Gewicht $w_{h'h}$ zu Neuron h ist.

Der Ausgabevektor y mit $|y| = K$ des MLPs lässt sich zuletzt über die Ausgabe der letzten verdeckten Schicht berechnen:

$$a_k = \sum_{h \in H_L} w_{hk} b_h \quad (2.21)$$

Backward Pass

Mit einer differenzierbaren Fehlerfunktion kann das MLP durch ein Gradientenabstiegsverfahren (Gradient Descent) trainiert werden, um den Fehler zu minimieren. Dabei ist die Online-Version, bei welcher die Gewichte des MLP nach jedem angelegtem Eingabemuster adaptiert werden, von der Offline-Version, bei welcher zunächst alle Trainingsmuster angelegt und die Gradienten für die Adaption aufsummiert werden, zu unterscheiden. Das Fehlersignal einer beliebigen Einheit j des MLPs und der Fehlerfunktion L wird wie folgt definiert:

$$\delta_j = \frac{\partial L}{\partial (a_j)} \quad (2.22)$$

Für die Neuronen aus der letzten verdeckten Schicht ergeben sich die Fehlersignale zu:

$$\delta_h = \theta'(a_h) \sum_{k=1}^K \delta_{h'} w_{hh'} . \quad (2.23)$$

Für die Neuronen aus jeder vorherigen Schicht ergeben sich die Fehlersignale rekursiv zu:

$$\delta_h = \theta'(a_h) \sum_{h' \in H_{l+1}} \delta_{h'} w_{hh'} \quad (2.24)$$

Sind alle δ s berechnet, können die Gewichte zur Fehlerminimierung angepasst werden, wobei die jeweiligen Gewichtsänderungen durch

$$\frac{\partial L}{\partial w_{ij}} = \delta_j^t b_i , \quad (2.25)$$

gegeben sind. Die Gewichtsänderung wird in unterschiedlichen Lernalgorithmen durch weitere Faktoren, wie beispielsweise einer Lernrate oder einem Momentumterm, beeinflusst.

2.2.2 Rekurrente Netze

Rekurrente Netze besitzen im Gegensatz zum MLP zusätzlich rekurrente Verbindungen, durch welche die Ausgabe eines Neurons zum Zeitpunkt t in die Eingabe eines Neurons zum Zeitpunkt $t + 1$ einfließen kann. Somit entstehen im Netz Zyklen, durch welche ein zeitlicher Kontext ermöglicht wird. Dadurch kann das rekurrente Netz, gegenüber dem MLP, Abbildungen von Eingabesequenzen auf Ausgabesequenzen erlernen, wobei eine Ausgabe zu einem bestimmten Zeitschritt t von allen vorherigen Eingaben abhängen kann. Aufgrund der zeitlichen Abhängigkeit der Eingaben in rekurrente Netze, muss zur Fehlerminimierung der Fehler durch die Zeit zurück geführt werden (Backpropagation through time).

Forward Pass

Im Forward Pass des RNNs berechnet sich die Aktivierung eines Neurons durch Aufsummieren der Eingangsverbindungen (Forward Activation) aus der vorherigen Schicht und zusätzlichem

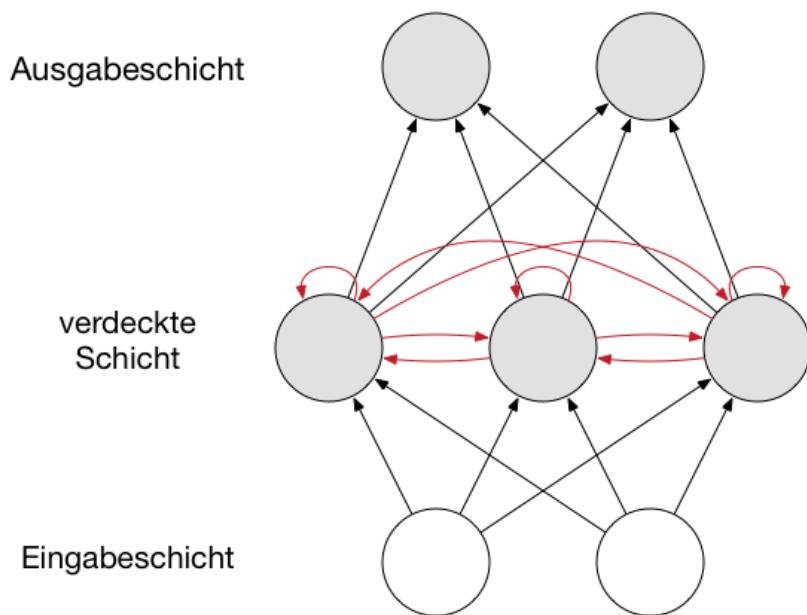


Abbildung 2.6: Rekurrentes Netz

Rekurrente Verbindungen sind rot dargestellt. In dieser Netzkonfiguration besitzt jede Zelle eine rekurrente Verbindung zu sich selbst (direkte Rückkopplung) und eine rekurrente Verbindung zu jeder anderen Zelle derselben Schicht (seitliche Rückkopplung). Weitere Verbindungen zu Zellen aus anderen Schichten (indirekte Rückkopplung) sind möglich. Graue Zellen besitzen eine Aktivierungsfunktion.

Aufsummieren der rekurrenten Verbindungen (Hidden Activation) in das jeweilige Neuron. Die Eingaben werden dabei zusätzlich um den Index t für den jeweiligen Zeitschritt erweitert. In einem einschichtigen RNN berechnet sich die Eingabe in ein Neuron h , der verdeckten Schicht, zum Zeitschritt t wie folgt:

$$a_h^t = \sum_{i=1}^I w_{ih} x_i^t + \sum_{h'=1}^H w_{h'h} b_{h'}^{t-1}, \quad (2.26)$$

wobei die Aktivierung synchron zum MLP berechnet wird:

$$b_h^t = \theta_h(a_h^t) \quad (2.27)$$

Die Ausgabe zum Zeitschritt t berechnet sich zu:

$$a_k^t = \sum_{h=1}^H w_{hk} b_h^t \quad (2.28)$$

Die Ausgabe des Neurons zum Zeitschritt t und die rekurrenten Aktivierungen für den Zeitschritt $t+1$ können dabei zur selben Zeit berechnet werden.

Backward Pass

Im Backward Pass kann das RNN durch BPTT wie folgt berechnet werden:

$$\delta_h^t = \theta'(a_h^t) \left(\sum_{k=1}^K \delta_k^t w_{hk} + \sum_{h'=1}^{t+1} \delta_{h'}^t b_{h'}^t \right), \quad (2.29)$$

mit

$$\delta_j^t = \frac{\partial L}{\partial a_j^t}. \quad (2.30)$$

Um den Fehler einer Sequenz über T Eingabemuster zu minimieren, wird bei Zeitschritt T mit $\delta_j^{T+1=0}$ gestartet und rekursiv für jeden Zeitschritt der Fehler zurückgeführt. Da durch das Auffalten des RNNs in jedem Zeitschritt dieselben Gewichte verwendet werden, ergibt sich die Gewichtsänderung durch die Addition der einzelnen Änderungen über die Zeit:

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^T \delta_j^t b_i^t \quad (2.31)$$

2.2.3 LSTM-Netze

Bei allgemeinen rekurrenten Netzen, wie im vorherigen Kapitel beschrieben, nimmt der Einfluss einer Eingabe zu einem bestimmten Zeitpunkt, auf die Ausgabe folgender Zeitschritte exponentiell ab (Vanishing Gradient Problem [BSF94], [Hoc98]). Dadurch kann der Fehler nicht beliebig weit zurückgeführt werden, wodurch Zusammenhänge zwischen Eingaben mit zunehmender zeitlicher Distanz, nur schlecht oder gar nicht, erlernt werden können. Das Long-Short-Term-Memory (LSTM) [HS97][Gra12] ist eine auf rekurrenten Netzen basierende Struktur, welche entwickelt wurde, um das Vanishing Gradient Problem zu umgehen und realisiert einen Speicher für den internen Zellzustand eines LSTM-Blocks. Über die drei verschiedene Gates (Input-Gate, Forget-Gate, Output-Gate) des LSTM-Blocks kann erlernt werden wann

und wie der Zell-Zustand durch eine Eingabe verändert wird und wann die Zelle etwas ausgibt. Dadurch kann ein Zustand beliebig lange in einem LSTM-Block gehalten werden.

Im Folgenden werden die Gleichungen für die Aktivierungen einer LSTM-Schicht im Forward- und die Deltas im Backward-Pass gelistet. Dabei wurden die Notationen aus [Gra12] verwendet, wobei i, ϕ, ω in dieser Reihenfolge das Input-, Forget-, und Output-Gate kennzeichnen und $w_{ci}, w_{c\phi}, w_{c\omega}$ die 'Peephole'-Verbindungen von Zelle c zum jeweiligen Gate kennzeichnen.

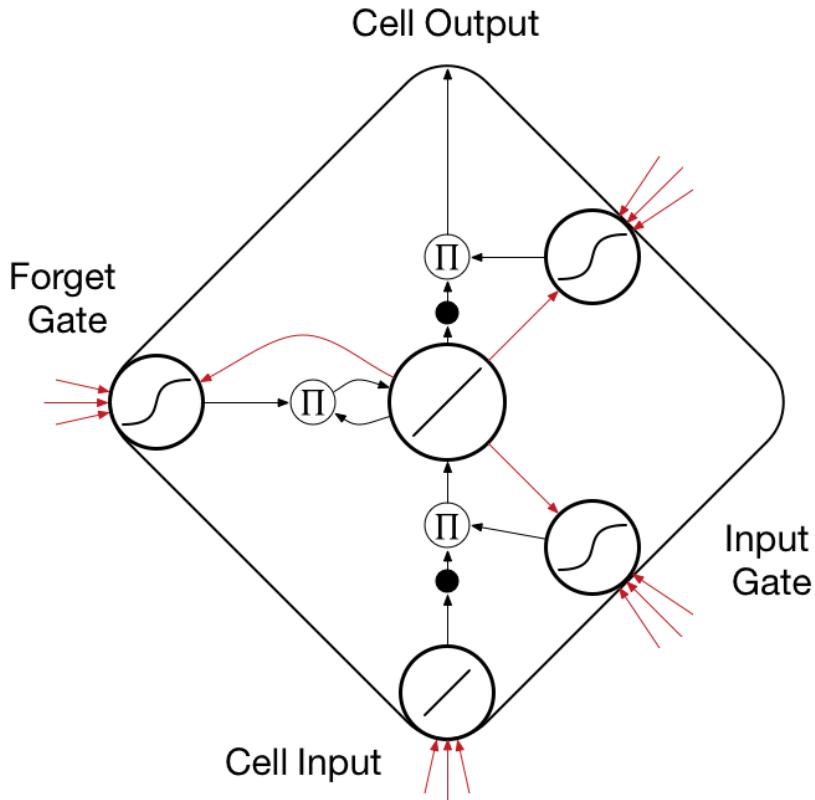


Abbildung 2.7: LSTM-Zelle

Ein LSTM-Baustein. Rote Verbindungen sind gewichtet, wobei die Verbindungen vom CEC (Constant Error Carousell) in der Mitte des Bausteins zu den Gates auch als Peepholes bezeichnet werden. Die Anzahl der Eingänge in die Zelle können variieren.

Forward Pass

Input Gates:

$$a_i^t = \sum_{i=1}^I w_{il} x_i^t + \sum_{h=1}^H w_{ht} b_h^{t-1} \sum_{c=1}^C w_{ci} s_c^{t-1} \quad (2.32)$$

$$b_i^t = f(a_i^t) \quad (2.33)$$

Forget Gates:

$$a_\phi^t = \sum_{i=1}^I I w_{i\phi} x_i^t + \sum_{h=1}^H w_{h\phi} b_h^{t-1} \sum_{c=1}^C w_{c\phi} s_c^{t-1} \quad (2.34)$$

$$b_\phi^t = f(a_\phi^t) \quad (2.35)$$

Cells

$$a_c^t = \sum_{i=1}^I w_{ic} x_i^t + \sum_{h=1}^H w_{hc} b_h^{t-1} \quad (2.36)$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_l^t g(a_c^t) \quad (2.37)$$

Output Gates

$$a_\omega^t = \sum_{i=1}^I w_{i\omega} x_i^t + \sum_{h=1}^H w_{h\omega} b_h^{t-1} + \sum c = 1^C w_{c\omega} s_c^t \quad (2.38)$$

$$b_\omega^t = f(a_\omega^t) \quad (2.39)$$

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t) \quad (2.40)$$

Backward Pass

$$\epsilon_c^t = \frac{\partial L}{\partial b_c^t} \quad (2.41)$$

$$\epsilon_s^t = \frac{\partial L}{\partial s_c^t} \quad (2.42)$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^K w_{ck} \delta_k^t + \sum_{g=1}^G w_{cg} \delta_g^{t+1} \quad (2.43)$$

Output Gates

$$\delta_w^t = f'(a_\omega^t) + \sum_{c=1}^C h(s_c^t) \epsilon_c^t \quad (2.44)$$

States

$$\epsilon_s = b_w^t h'(s_c^t) \epsilon_c^t + b_\phi^{t+1} \epsilon_s^{t+1} + w_{cl} \delta_l^{t+1} + w_{c\phi} \delta_\phi^{t+1} + w_{cw} \delta_w^t \quad (2.45)$$

Cells

$$\delta_c^t = b_l^t g'(a_c^t) \epsilon_s^t \quad (2.46)$$

Forget Gates

$$\delta_p h^t = f'(a_\phi^t) \sum_{c=1}^C s_c^{t-1} \epsilon_s^t \quad (2.47)$$

Input Gates

$$\delta_l^t = f'(a_l^t) \sum_{c=1}^C g(a_c^t) \epsilon_s^t \quad (2.48)$$

2.2.4 Gradientenabstiegsverfahren

Zur Berechnung der Gewichtsänderungen und Adaption der Gewichte in der Trainingsphase eines neuronalen Netzes, kann grundsätzlich zwischen zwei Methoden unterschieden werden. In der sogenannten Offline-Methode werden zunächst alle Trainingsmuster an das Netz angelegt und die jeweiligen Gradienten aufsummiert, um den Fehler über den gesamten Trainingsatz zu minimieren. In der sogenannten Online-Methode, dem stochastischen Gradientenabstieg (SGD), werden die Gewichtsänderungen nur über die Gradienten des zuletzt angelegten Trainingsmusters berechnet und direkt adaptiert. Dadurch wird der Gradient über den gesamten Trainingsatz nur angenähert. Um Zyklen zu vermeiden, ist es üblich, die Reihenfolge, in welcher die Muster angelegt werden, zu variieren. Da für jedes Eingabemuster die Gewichte angepasst werden, sind während dem Training Fluktuationen vorhanden. Damit das Netz trotzdem konvergiert, wird meist die Schrittweite (Lernrate) über das gesamte Training verringert. Zudem ist es auch gebräuchlich, die Gradienten über eine Gruppe von n Trainingsmustern zu berechnen (Minibatch Gradientenabstieg) und die Gewichte nur für jeden Minibatch zu adaptieren. Stochastischer Gradientenabstieg bietet bei großen Datensätzen oder langen Eingabesequenzen deutliche Speichervorzüge, da die Gradienten nicht für den gesamten Datensatz, sondern nur für das aktuelle Muster oder den aktuellen Mini-Batch gespeichert werden müssen.

Gradient Descent

Die zu trainierenden Parameter θ werden bei der Offline-Version des Gradientenabstiegs über den gesamten Trainingssatz durch eine Fehlerfunktion L wie folgt adaptiert:

$$\theta = \theta - \eta \nabla_{\theta} L(\theta), \quad (2.49)$$

wobei η die Lernrate definiert

Stochastic Gradient Descent

Der stochastische Gradientenabstieg adaptiert die Parameter für jedes angelegte Trainingsmuster $x^{(i)}$ und der zugehörigen gewünschten Ausgabe $y^{(i)}$:

$$\theta = \theta - \eta \nabla_{\theta} L(\theta; x^{(i)}; y^{(i)}). \quad (2.50)$$

Minibatch Gradient Descent

Beim Minibatch Gradientenabstieg werden die Parameter für jeweils n Trainingsmuster adaptiert:

$$\theta = \theta - \eta \nabla_{\theta} L(\theta; x^{(i:i+n)}; y^{(i:i+n)}). \quad (2.51)$$

Adam

Adaptive Moment Estimation (ADAM) [KB14] ist ein Lernalgorithmus welcher auf Basis des stochastischen Gradientenabstiegs arbeitet, die Lernrate jedoch für jeden Parameter separat anpasst. Für eine übersichtliche Schreibweise verwenden wir $g_{t,i}$ als den Gradienten der Fehlerfunktion L in Bezug zu Parameter θ_i zum Iterationsschritt t :

$$g_{t,i} = \nabla_{\theta} L(\theta_i). \quad (2.52)$$

Damit werden die Parameter durch folgende Schritte angepasst:

$$m_\tau = \beta_1 m_{\tau-1} + (1 - \beta_1) g_\tau \quad (2.53)$$

$$v_\tau = \beta_2 v_{\tau-1} + (1 - \beta_2) g_\tau^2 \quad (2.54)$$

$$\hat{m}_\tau = \frac{m_\tau}{1 - \beta_1^\tau} \quad (2.55)$$

$$\hat{v}_\tau = \frac{v_\tau}{1 - \beta_2^\tau} \quad (2.56)$$

$$\theta_{\tau+1} = \theta_\tau - \frac{\eta}{\sqrt{\hat{v}_\tau} + \epsilon} \hat{m}_\tau \quad (2.57)$$

2.3 Methode der Audiosignal-Separierung

Die verwendete Methode zur Separierung des Audiosignals einer bestimmten Quelle aus einem Musikstück, basiert auf der Separierung im Frequenzraum. Ein neuronales Netz soll dazu trainiert werden, als eine Art Filter zu fungieren, um die Frequenzen der zu isolierenden Quelle aus den Frequenzen des Audiosignals zu bestimmen. Frequenzen, welche von anderen Tonquellen stammen, sollen abgeschwächt oder bestenfalls komplett ausgelöscht werden. Der Datensatz für das Training besteht aus Musikstücken und den entsprechenden Tonspuren der zu isolierenden Quelle.

Die Separierung einer Tonspur im Frequenzbereich mittels neuronaler Netze lässt sich für eine Folge von Ausschnitten aus dem Signal durch die folgenden Schritte beschreiben:

1. Normalisierung des Eingangssignals
2. Unterteilung des Signals und Gewichtung der Abschnitte durch eine Fensterfunktion
3. Kurzzeit Fourier-Transformation
4. Verarbeitung der einzelnen Spektren durch das neuronale Netz
5. Inverse Kurzzeit Fourier-Transformation
6. Generierung des Ausgangssignal durch die einzelnen Abschnitte des rücktransformierten Signals

Wobei in der Trainingsphase die beiden Schritte

6. Bestimmen des Fehlers durch die gegebene Fehlerfunktion
7. Adaption der Parameter des Netzes durch ein Gradientenabstiegsverfahren

hinzugefügt werden.

Da die zu verarbeitenden Sequenzen (Musikstücke) in der Länge variieren, wird bei jeder Iteration über den Datensatz aus jedem Musikstück, eine zufällige Teilsequenz einer festgelegten Länge entnommen. Da Offline-Gradientenabstieg aufgrund der Speicherauslastung nicht möglich ist, wird Minibatch-Gradientenabstieg zur Fehlerminimierung verwendet. Die Trainingsdaten werden, wie üblich bei stochastischem Gradientenabstieg, nach jeder Iteration über

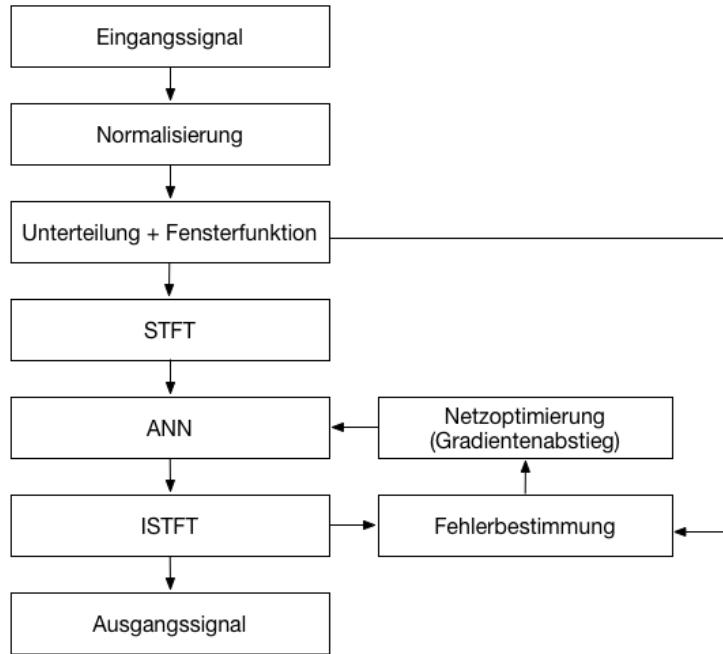


Abbildung 2.8: Die einzelnen Schritte bei der Audiosignal-Separierung. Die Fehlerbestimmung und Netzoptimierung wird nur während dem Training durchgeführt.

den gesamten Datensatz gemischt. Dadurch werden Zirkulationen während der Fehlerminimierung vermieden. Zudem wurde für die Handhabung unterschiedlich langer Sequenzen ein Parser implementiert, welcher für die Erzeugung der einzelnen Minibatches den aktuellen Index der Sequenz verwaltet und über eine Funktion die nächsten Abtastwerte der Sequenz bereitstellt. Ein Mini-batch besteht aus einer Liste von mbs (Mini-batch size) sich um R Abtastwerte überlappender Ausschnitte der Länge W . Diese Abschnitte werden sequentiell an das neuronale Netz angelegt. Der Fehler wird über den gesamten Minibatch berechnet und die Parameter für jeden Minibatch adaptiert. Da die Anzahl der in einem Minibatch enthaltenen Ausschnitte speicherbedingt begrenzt ist, werden die zusammenhängenden Minibatches durch einen Batch der Länge bs repräsentiert um lange Sequenzen verarbeiten zu können. Dabei ist zu beachten, dass während der Verarbeitung eines Batches, die variablen Parameter für jeden Minibatch adaptiert werden. Die einzelnen Trainingsabschnitte sind durch folgende Algorithmen gegeben.

2.3.1 Netzeingabe

Die aus der Kurzzeit-Fourier-Transformation gewonnenen Frequenzspektren können auf unterschiedliche Arten für die Separierung verwendet werden. Dabei können folgende Eingabemethoden unterschieden werden:

1. Netzeingabe bestehend aus dem Amplitudenspektrum
2. Netzeingabe bestehend aus dem Amplituden- und Phasenspektrum
3. Netzeingabe bestehend aus den Real- und Imaginärteilen des Frequenzspektrums

Amplitudenspektrum als Netzeingabe

Bei dieser Methode werden die Amplituden- und Phasen-Spektren aus den Frequenzspektren des Eingangssignals durch die komplexen Amplituden der Fourierkomponenten (2.2) und den Phasen der Fourierkomponenten (2.3) berechnet. Dabei werden ausschließlich die Amplitudenspektren durch das Netz verarbeitet. Um das Ausgangssignal zu bilden, werden die Werte der Netzausgabe und die Phasenspektren der Eingabe als Amplituden- und Phasen-Spektren des Ausgangssignals interpretiert und zu diesem rücktransformiert. Für die Zusammensetzung des Signals wird die COLA-Methode verwendet (Abschnitt 2.1.5). Im Ausgangssignal werden demnach die Phasen des Eingangssignals übernommen. Diese Methode ist in Kombination mit binären Masken geeignet, da lediglich Amplituden ausgelöscht, jedoch nicht verändert werden.

Amplituden- und Phasenspektrum als Netzeingabe

Eine weitere Methode ist die Verarbeitung der Amplituden- und Phasen-Spektren durch das Netz, wobei das Ausgangssignal im Gegensatz zur vorherigen Methode ausschließlich aus der Netzausgabe gebildet wird, indem die Ausgabewerte als Amplituden- und Phasen-Spektren der Ausgabe interpretiert werden. Bei der Verarbeitung der Phasen ist darauf zu achten, dass die Phasenwerte im Intervall $[-\pi, \pi]$ liegen.

Real- und Imaginärteile der Frequenzspektren als Netzeingabe

Anstelle der Amplituden- und Phasenspektren können auch direkt die Real- und Imaginärteile der komplexen Fourierkomponenten als Netzeingabe verwendet werden.

Die Eingabemethoden unter Verwendung der Amplitudenspektren und der Real- und Imaginärteile sind in Abbildung 2.3.1 illustriert.

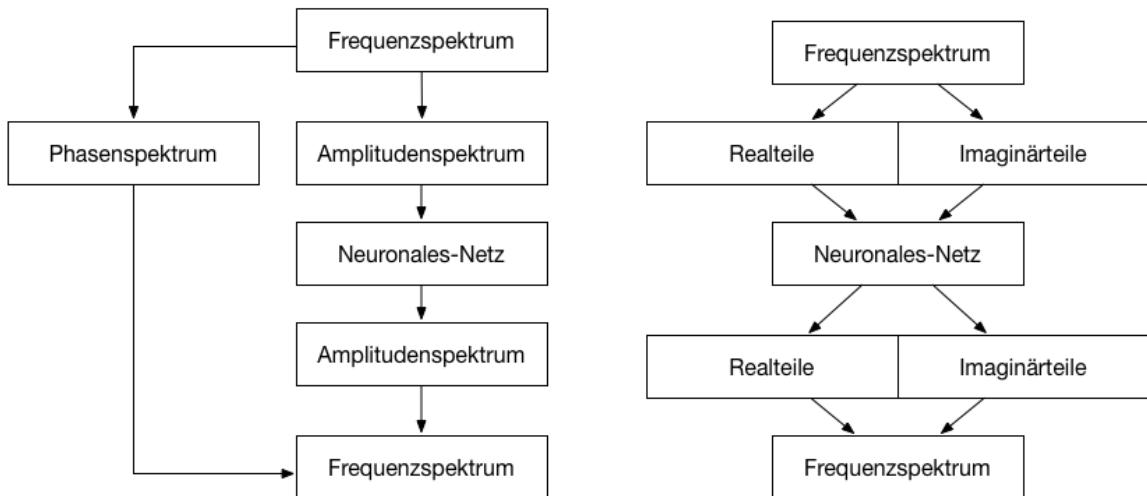


Abbildung 2.9: Netzeingabe

Links werden nur die Amplitudenspektren durch das neuronale Netz verarbeitet, rechts werden die Real- und Imaginärteile der Frequenzspektren verarbeitet.

Ein Nachteil der ersten Methode ist, dass keine Anpassungen an den Phasenspektren durch das neuronale Netz vorgenommen werden können. Dadurch muss das Zielsignal ausschließlich

lich durch Anpassung der Amplituden gebildet werden. Da bei der Rücktransformation zum Ausgangssignal die Phasen des Eingangssignals verwendet werden, können zudem Artefakte entstehen.

2.3.2 Trainingsphase

Das Training wird durch eine Epochenzahl begrenzt, wobei für jede Epoche der Trainingsdatensatz (*set*) einmal durchlaufen wird. Zu Beginn des Trainings wird das zu trainierende neuronale Netz (*net*) konfiguriert und initialisiert. In der Grundroutine des Trainings wird das Trainingsset jede Epoche gemischt, und für jedes enthaltene Musikstück die Methode *trainBatch* aufgerufen.

Algorithmus 1 Trainingsroutine

Erfordert:

epochs, Anzahl Trainingsepochen
set, Trainings-Datensatz
config, Netz-Konfiguration

Liefert: Trainiertes Netz

```

1: net  $\leftarrow$  createNet(config, initializer)
2: for i  $\leftarrow$  0 to epochs do
3:   shuffle(set)
4:   for j  $\leftarrow$  0 to size(set) do
5:     readerx  $\leftarrow$  Reader(getMix(set[j]))
6:     readery  $\leftarrow$  Reader(getSource(set[j]))
7:     trainBatch(readerx, readery, net)
8:   end for
9: end for
10: return
```

Die Methode *trainBatch* besteht aus einer Schleife, deren Anzahl an Wiederholungen der Batch-Größe (Batchsize) *bs* entspricht. In jedem Schleifendurchlauf wird ein Minibatch durch die Hilfsmethode *prepareMiniBatch* erzeugt und der Methode *trainNetwork* übergeben, welche schließlich die einzelnen Sequenzabschnitte verarbeitet, durch das Netz propagiert und die Netzparameter entsprechend adaptiert.

Die Hilfsmethode *prepareMinibatch* erzeugt die benötigte Minibatch-Liste, der sich überlappenden Signal-Ausschnitte, für die sequentielle Eingabe in das neuronale Netz. Dazu entnimmt der übergebene Reader iterativ einen Ausschnitt aus dem Signal, welcher eine der Sprungweite entsprechenden Länge besitzt. Zudem wird vom letzten Ausschnitt aus der Liste der überlappende Teil kopiert (beim Overlap-Add Verfahren besitzt dieser Teil ebenfalls die Länge der Sprungweite). Die beiden Abschnitte werden zu einem neuen Ausschnitt verknüpft und als Element in die Liste hinzugefügt, wie in 2.10 illustriert.

Das eigentliche Training besteht aus der Vorwärtspropagierung der Netzeingabe durch das neuronale Netz, der Fehlerbestimmung, der Ausgabe und der Optimierung der Netzparameter durch einen Optimierungsalgorithmus. Die Netzeingabe besteht aus den Fourier-transformierten und normalisierten Abschnitten aus dem übergebenen Minibatch. Die Vorwärtspropagierung wird durch die Methode *computeRNN* realisiert, wobei diese Methode von der Netzstruktur abhängt, welche in Abschnitt 2.3.4 beschrieben wird. Der Trainingsalgorithmus ist durch 4 gegeben.

Algorithmus 2 Training eines Batches - trainBatch

Erfordert:

$reader_x$, input signal reader
 $reader_y$, target signal reader
 W , window size
 mbs , minibatch-size (Anzahl der 'Zeitschritte' pro Parameterupdate)
 bs , batch-size (Anzahl aufeinanderfolgender Parameterupdates)

Liefert: Update der Netz-Parameter zur Fehlerminimierung

```

1:  $R \leftarrow W/2$  {Sprungweite R}
2:  $w \leftarrow windowfunction(M)$  {Fensterfunktion w der Länge M}
3:  $l \leftarrow bs * mbs * R$  {Anzahl der Abtastwerte, welche X entnommen werden}
4:  $o \leftarrow Random(0, l)$  {zufälliger Offset}
5:  $setPosition(reader_x, o)$  {setze  $reader_x$  auf Startposition}
6:  $setPosition(reader_y, o)$  {setze  $reader_y$  auf Startposition}
7:  $x \leftarrow zeros(W)$  {initialisiere x mit W Nullen}
8:  $y \leftarrow zeros(W)$  {initialisiere y mit W Nullen}
9: for  $i \leftarrow 0$  to  $bs$  do
10:    $Minibatch_x, x \leftarrow prepareMiniBatch(x, reader_x)$ 
11:    $Minibatch_y, y \leftarrow prepareMiniBatch(y, reader_x)$ 
12:    $Error \leftarrow trainNetwork(Minibatch_x, Minibatch_y, w)$ 
13: end for
14: return Error

```

Algorithmus 3 Minibatch Generierung - prepareMinibatch

Erfordert:

x , letzter dem Signal entnommener Abschnitt (wegen Überlappung benötigt)
 $reader_x$, reader eines Signals
 W , Windowsize (Abtastwerte pro Abschnitt)
 mbs , minibatch-size (Anzahl der 'Zeitschritte' pro Parameterupdate)

Liefert: Minibatch

```

1:  $Minibatch \leftarrow []$ 
2: for  $i \leftarrow 0$  to  $mbs$  do
3:    $nextFrames \leftarrow getFrames(reader, R)$ 
4:    $x \leftarrow concatenate(x[R :], nextFrames)$ 
5:    $Minibatch \leftarrow Append(Minibatch, x)$ 
6: end for
7: return  $Minibatch, x$ 

```

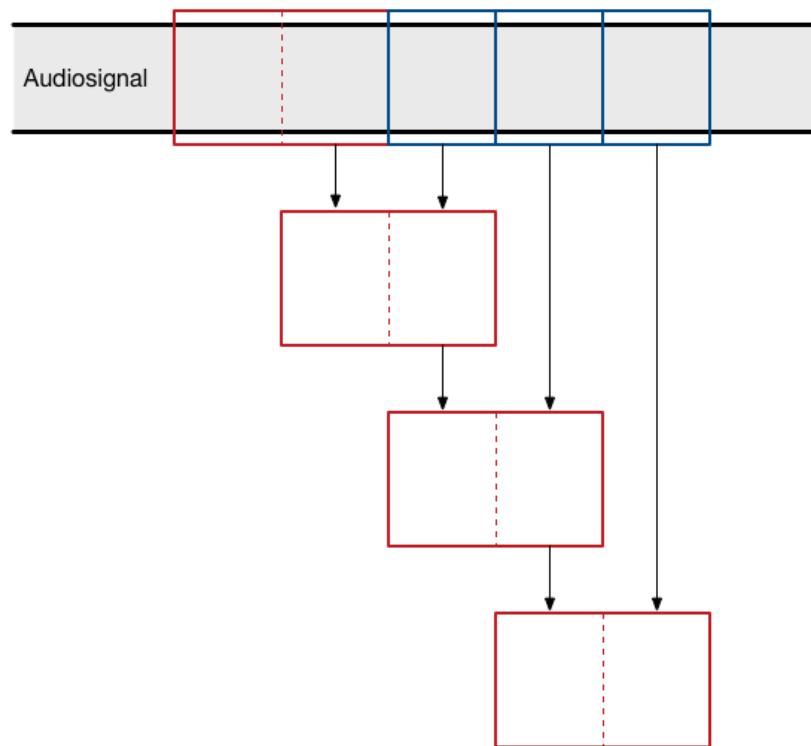


Abbildung 2.10: Minibatch Generierung

Die aus dem Signal zusammengesetzten, überlappenden Abschnitte, werden für die Eingabe in ein neuronales Netz als Minibatch zusammengefasst. Die Sequenzlänge der blau markierten Abschnitte entspricht dabei der Sprungweite

Algorithmus 4 Training - trainNetwork

Erfordert:

Minibatch_x, Minibatch Eingangssignal
Minibatch_y, Minibatch Zielsignal
w, Fensterfunktion
L, Fehlerfunktion
net, Netz
netState, Initial-State des rekurrenten Netzes
Optimizer, Algorithmus für Gradientenabstiegsverfahren

Liefert: Trainiert Netz für gegebenen Minibatch

- 1: $Minibatch_xWindowed \leftarrow Minibatch_x \otimes w$, elementweise Multiplikation mit Fensterfunktion
 - 2: $NetInputs \leftarrow RFFT(Minibatch_xWindowed)$, Schnelle-Fourier-Transformation für Reelle Eingabe
 - 3: $NetInputs \leftarrow NetInputs \oslash floor(W/2 + 1)$, elementweise dividieren
 - 4: $netOutputs, netState \leftarrow computeRNN(NetInputs, netState, net)$, *NetOutputs* generieren
 - 5: $Outputs \leftarrow netOutputs \otimes floor(W/2 + 1)$, elementweise multiplizieren
 - 6: $Outputs \leftarrow IRFFT(Outputs)$, inverse Schnelle-Fourier-Transformation
 - 7: $Error = L(Outputs, Minibatch_y)$, Fehlerberechnung
 - 8: $Net = Optimize(Optimizer, Net, Error)$
 - 9: **return** *Net, Error*
-

2.3.3 Anwendungsphase

Ist ein Netz trainiert, kann es zur Anwendung der AudioSeparierung von neuen, dem Netz unbekannten Eingaben verwendet werden. Da die Trainingsschritte entfallen und somit keine Gradienten gespeichert werden müssen, kann auf die Verarbeitung des Eingabesignals in Form von Batches bzw. Minibatches verzichtet werden. Da das Overlap-Add Verfahren in der Anwendung weiterhin verwendet wird, muss lediglich die Eingabe zum Zeitpunkt $t - 1$ gespeichert werden, um den nächsten Abschnitt des Signals zusammensetzen zu können. Die Separierung einer Quelle unter Verwendung eines trainierten neuronalen Netzes ist durch Algorithmus 5 gegeben.

2.3.4 Netzstruktur

Die verwendete Netzstruktur besteht aus einer Eingabeschicht, einer oder mehrerer aus LSTM-Zellen bestehenden verdeckten Schichten und einer Ausgabeschicht. Die Dimension der Ein- und Ausgabeschicht variiert mit der Netzeingabe. Sollen nur die Amplitudenspektren durch das Netz verarbeitet werden, entspricht die Anzahl der Eingabewerte der Anzahl der im Spektrogramm enthaltenen Frequenzen (Fourierkomponenten). Da ein Audiosignal aus reellen Abtastwerten besteht, und somit die Fouriertransformation für reelle Eingaben zum Einsatz kommt, ist die Anzahl der Frequenzen I durch $I = floor(W/2 + 1)$ festgelegt, wobei W der verwendeten Fensterbreite entspricht. Sollen Real- und Imaginärteile der Fourierkomponenten durch das Netz verarbeitet werden, bestehen Eingabe- und Ausgabeschicht aus der doppelten Anzahl an Fourierkomponenten, da für jede komplexe Komponente der Realteil und der Imaginärteil an das Netz angelegt wird. Die Dimension einer verdeckten LSTM-Schicht ist frei wählbar. Die Ausgabeschicht besitzt die gleiche Dimension wie die Eingabeschicht, da ebenso viele Frequenzen für die Rücktransformation in das Zeitsignal verwendet werden. Die Eingabe der

Algorithmus 5 Separierung - adaptNetwork

Erfordert:

file, Audiodatei
W, Fenstergröße
w, Fensterfunktion
net, Netz
netState, Netz-Status

Liefert: Audiodatei der separierten Quelle

```

1: reader  $\leftarrow$  Reader(file)
2: x  $\leftarrow$  zeros(W)
3: R = W / 2
4: y  $\leftarrow$  zeros(R)
5: outputFile = []
6: while file do
7:   nextFrames  $\leftarrow$  getFrames(reader, R)
8:   x  $\leftarrow$  concatenate(x[R :], netxFrames)
9:   xWindowed  $\leftarrow$  x  $\otimes$  w
10:  NetInputs  $\leftarrow$  RFFT(xWindowed)
11:  NetInputs  $\leftarrow$  NetInputs  $\odot$  floor(W / 2 + 1)
12:  NetOutputs, netState  $\leftarrow$  computeRNN(NetInputs, netState, net)
13:  Outputs  $\leftarrow$  NetOutputs  $\otimes$  floor(W / 2 + 1)
14:  Outputs  $\leftarrow$  IRFFT(Outputs)
15:  y  $\leftarrow$  y[R :]  $\oplus$  Outputs[: R], Overlap-Add
16:  append(outputFile, y)
17:  y  $\leftarrow$  Outputs[R :]
18: end while
19: return outputFile

```

verdeckten Schichten und der Ausgabeschicht besteht aus den Eingaben der vorherigen Schicht, addiert mit einem Biaswert für jedes Neuron.

2.3.5 Implementierung

Für die Implementierung der neuronalen Netze wurde das Framework Tensorflow [AAB⁺15] verwendet, um eine effiziente parallele Berechnung der neuronalen Netze zu ermöglichen. Unter der Verwendung einer GPU konnte das Training der neuronalen Netze beschleunigt werden.

3 Ergebnisse

Als Grundlage für die Erzeugung der Trainingsdaten wurde die MedleyDB Datenbank [BST⁺] verwendet, welche aus mehreren Multitracks besteht. Ein Multitrack bezeichnet dabei ein musikalisches Stück, für welches die einzelnen Instrumentenkomponenten als separate Audiodateien vorliegen. Diese können für eigene Zwecke gemischt oder verändert werden. Alle in der Datenbank enthaltenen Audiodateien liegen im WAV-Audioformat vor und besitzen eine Auflösung von 16-Bit und eine Abtastrate von 44100 Hz. Die einzelnen Trainingsdaten, bestehend aus einem Mix, in welchem alle vorhandenen Tonquellen im Multitrack zu einer Audiospur (Mixdown) zusammengefasst werden und der zu separierenden einzelnen Quelle, wurden aus der Multitrack-Datenbank [BST⁺] erzeugt. Dazu wurden die vorhandenen Spuren für den Mix überlagert, und die Quelle als separate Spur entnommen. Die Abtastwerte für den generierten Mix und die Quelle wurden auf Werte im Intervall $[-1 : 1]$ begrenzt, um die Eingabewerte in das neuronale Netz zu normalisieren. Dabei wurden die Abtastwerte der Quelle in Relation zum Mix normalisiert, damit sich bei einer Überlagerung der invertierten Quelle mit dem Mix, die Quelle auslöscht. In Anlehnung an die Separierung von Gesang aus Musikstücken via MLP und binärer Masken [SRP15] wurde ebenfalls zunächst dafür entschieden, den Gesang in ersten Tests als Quelltyp für die Separierung zu wählen. Entsprechende Trainingsdaten wurden erzeugt. In Anlehnung an die Separierung mittels binärer Masken [SRP15], wurde über das gesamte Training eine konstante Fenstergröße von 2048 Abtastwerten mit einer Überlappung (Sprungweite R) von 1024 Abtastwerten verwendet. Damit ergeben sich genau 1025 komplexe Fourierkomponenten nach der Transformation in den Frequenzraum durch die FFT. Aufgrund der Verwendung der COLA-Methode 2.1.5 bei der Zusammensetzung des Ausgabesignals aus den einzelnen Ausgabeabschnitten ist die Wahl der Sprungweite, durch die gewählte Fensterbreite und Bedingung $R = M/2$ (2.16), vorgegeben. Um eine Änderung der Sprungweite unter Verwendung des Tukey-Fensters und dem sich daraus ergebenden Überfluss der COLA-Methode zu vermeiden, wurde für das Tukey-Fenster $\alpha = 0.5$ gewählt. Das Training gliedert sich in 2 Phasen. In Phase 1 wurde zunächst ein kleines Trainingsset bestehend aus 3 Musikstücken (insgesamt 30s) erzeugt. Es wurden neuronale Netze mit einer unterschiedlichen Anzahl an LSTM-Blöcken und den verschiedenen Netzeingabemethoden (Real- und Imaginärteile, Amplituden) mit unterschiedlichen Fensterfunktionen trainiert, eine Abbildung der Eingabesignale auf die Zielsignale zu realisieren. Anhand der Ergebnisse aus Phase 1 wurden die Strukturen mit den geringsten Fehlern in der Abbildung, für das Training in Phase 2 übernommen. In Phase 2 wurde das Trainingsset auf den kompletten Datensatz für das Training der Separierung einer Quelle ausgeweitet.

3.1 Training - Teil 1

Im ersten Schritt der Trainingsphase wurden lediglich kurze Ausschnitte von jeweils 10 Sekunden aus einer kleinen Auswahl von 3 Musikstücken aus den Trainingsdaten entnommen. Anhand des Trainingsset wurde versucht, ein neuronales Netz, bestehend aus einer verdeckten LSTM-Schicht, darauf zu trainieren, die Eingabesignale auf die zugehörigen, gewünschten Ausgangssignale abzubilden. Dabei wurden verschiedene Kombinationen der Netzstruktur, Eingabemethode und Fensterfunktion verwendet. Als Lernalgorithmus wurde Adam (2.2.4) mit einer initialen Lernrate von 0.001 verwendet. Zudem wurde die Minibatch-Größe auf 100 Abschnitte festgelegt. Damit beträgt die Länge des Signals welches für ein Parameter-Update durch das Netz propagiert wurde $1024 \cdot 100 / 44100 \text{ s}^{-1} \cong 2.32 \text{ s}$. Es wurde eine Batch-Größe von 5 gewählt um jeden Trainingspart von 10 Sekunden abzudecken ($2.32 \text{ s} \cdot 5 = 11.6$). Überhänge wurden mit Nullen gefüllt. Als Qualitätsmerkmal der Ausgabe und Fehlerfunktion der Netzauf optimierung durch den Adam-Algorithmus, wurde die mittlere quadratische Abweichung zwischen den Abtastwerten des Ausgabe- und Zielsignals verwendet. Es wurden Netzstrukturen mit den unterschiedlichen Eingabemethoden und verdeckten LSTM-Schichten von jeweils 100, 250, 500 und 1000 LSTM-Blöcken trainiert. In Teil 1 des Trainings sollte des Weiteren festgestellt werden, ob die Qualität der Abbildung mit einer zunehmenden Anzahl an LSTM-Elementen steigt und welche Eingabemethode am effizientesten ist. Die Fehler-Entwicklung über das gesamte Training aller Konfigurationen ist in Abbildung 3.1 dargestellt. In Tabelle 3.2 ist der Gesamtfehler über das Trainingsset nach 500 Trainingsepochen für die jeweilige Konfiguration gelistet. Zudem ist die Geschwindigkeit der Netze in der Anwendungsphase auf der GPU angegeben. In Tabelle 3.2 ist der Fehler der einzelnen Abschnitte für die jeweilige Konfiguration angegeben. Die Fehlerentwicklungen in Bezug zur Eingabemethode sind zusätzlich in den Abbildungen 3.2(a), 3.2(b), 3.2(c) dargestellt.

Auswertung

Die Fehlerentwicklungen sind deutlich durch die verwendete Eingabemethode zu unterscheiden. Dabei konvergiert der Fehler für eine Netzeingabe, bestehend aus den Real- und Imaginärteilen der Fourierkomponenten, am schnellsten. Ein lokales Minimum von ~ 0.00012 konnte erreicht werden. Ausschlaggebende Verbesserungen der Ergebnisse sind nur bis zu einer Anzahl von 500 LSTM-Blöcken wahrnehmbar. Unter allen Eingabemethoden unterscheidet sich die Fehlerentwicklung für 500 LSTM-Blöcke zur Fehlerentwicklung für 1000 LSTM-Blöcke nur minimal. Die in der Anwendung des trainierten Netzes benötigte Zeit steigt allerdings bei einer Erhöhung von 500 auf 1000 LSTM-Blöcken von 5s auf 8s ($\sim 60\%$). Unter allen Konfigurationen sind die für den stochastischen Gradientenabstieg typischen Fluktuationen während der Fehlerminimierung zu erkennen. Die Ausgaben der drei trainierten Ausschnitte durch die Netze mit 1000 LSTM-Blöcken und der unterschiedlichen Netzeingabe-Methoden sind in 3.3, 3.5 und 3.7 durch den zeitlichen Verlauf des Signals und in 3.4, 3.6 und 3.8 durch die zugehörigen Spektrogramme dargestellt. Die Signale der Quelle und der Ausgaben sind visuell nur geringfügig zu unterscheiden. Im spektralen Verlauf durch die STFT ist jedoch zu erkennen, dass teilweise hohe Frequenzen, welche im Quellsignal nicht vorhanden sind, in der Ausgabe vorhanden sind. Unterzieht man die Ausgaben einem auditiven Test, kann die Separierung der Quelle bestätigt werden. Von anderen Quellen stammende Signale, welche im Eingangssignal (Mix) enthalten sind, wurden durch eine Filterung im Frequenzbereich durch das Netz größtenteils entfernt und sind in der Ausgabe nicht oder nur noch minimal vorhanden. In der Ausgabe unter ausschließlicher Verarbeitung der Amplituden durch das Netz (Konfigurationen

in Tabelle 3.2 als 'Amplitudes, Hann' und 'Amplitudes, Tukey' bezeichnet) sind zwar weniger Signalanteile aus anderen Quellen übernommen worden, dass Signal klingt jedoch unsauberer und enthält Artefakte. Bei einem Vergleich des Hanning-Fensters mit dem Tukey-Fenster, schneidet das Hanning-Fenster unter Verwendung der COLA-Methode (Abschnitt 2.1.5) deutlich besser ab. Eine Eingabe, bestehend aus Real- und Imaginärteilen der Fourierkomponenten, erzeugt über alle Testausschnitte die Ausgabe mit dem geringsten Fehler.

Methode	LSTMs	Fehler (500 Epochen)	Geschwindigkeit [s/44100 Frames]
Real+Imag, Hanning	100	0.00035	0.4
Real+Imag, Hanning	250	0.00020	0.4
Real+Imag, Hanning	500	0.00012	0.5
Real+Imag, Hanning	1000	0.00012	0.8
Amplitudes, Hanning	100	0.0025	0.4
Amplitudes, Hanning	250	0.0022	0.4
Amplitudes, Hanning	500	0.0020	0.5
Amplitudes, Hanning	1000	0.0020	0.9
Amplitudes, Tukey	100	0.0042	0.4
Amplitudes, Tukey	250	0.0037	0.4
Amplitudes, Tukey	500	0.0036	0.5
Amplitudes, Tukey	1000	0.0035	0.9

Tabelle 3.1: Gesamtfehler über alle im Trainingsdatensatz enthaltenen Musikstücke, nach einem Training von 500 Epochen.

Interpret - Titel	Eingabemethode	Fenster	Fehler
Clara Berry And Wooldog - Boys	Real+Imag	Hanning	0.00047
Clara Berry And Wooldog - Boys	Abs	Hanning	0.00100
Clara Berry And Wooldog - Boys	Abs	Tukey	0.00176
Helado Negro - MitadDelMundo	Real+Imag	Hanning	0.00037
Helado Negro - MitadDelMundo	Abs	Hanning	0.00041
Helado Negro - MitadDelMundo	Abs	Tukey	0.00071
Sweet Lights - YouLetMeDown	Real+Imag	Hanning	0.00049
Sweet Lights - YouLetMeDown	Abs	Hanning	0.00062
Sweet Lights - YouLetMeDown	Abs	Tukey	0.00103

Tabelle 3.2: Fehler der im Trainingsdatensatz enthaltenen Musikstücke, nach einem Training von 500 Epochen. Verwendet wurden die Netze mit 1000 LSTMs in der verdeckten Schicht

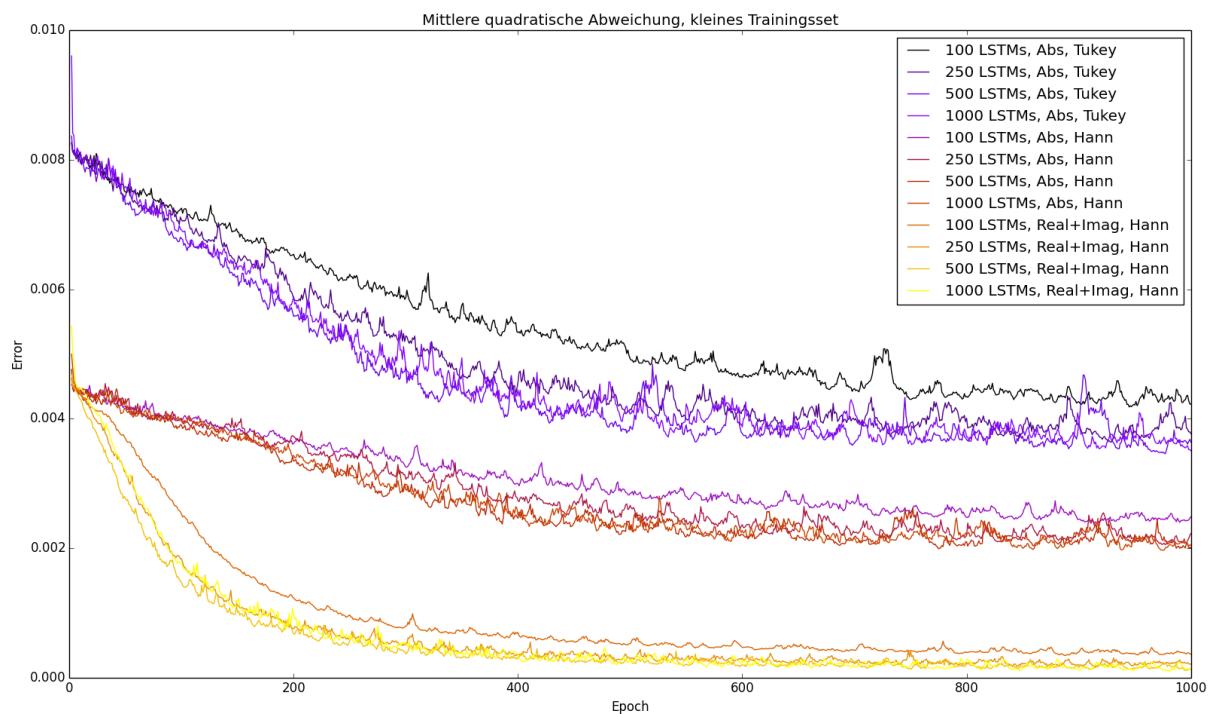
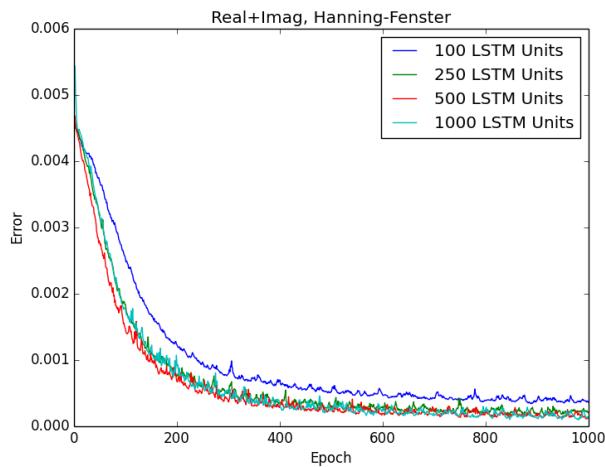
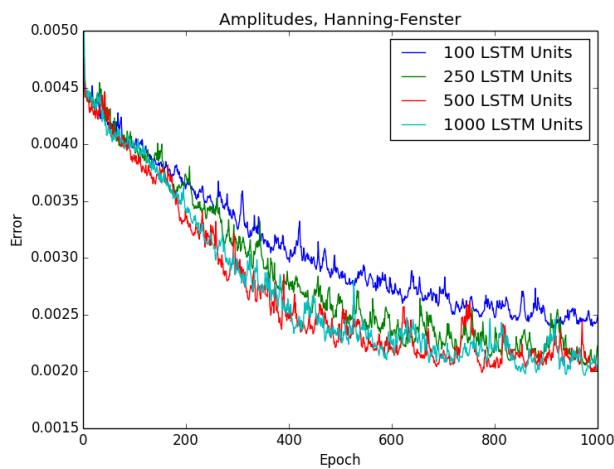


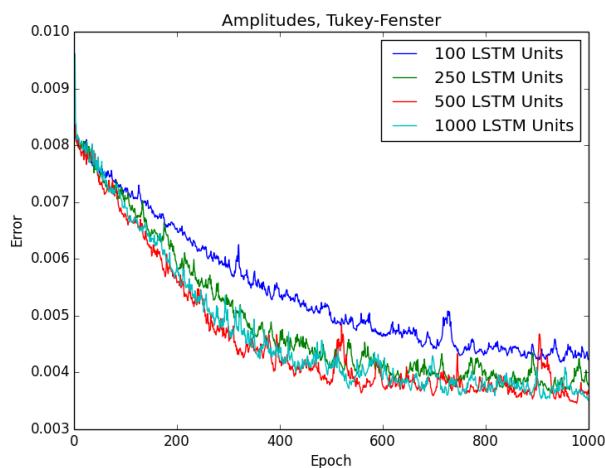
Abbildung 3.1: Fehler über das Training der unterschiedlichen Netz- und Eingabekonfigurationen.



(a) Training Teil 1, Real- und Imaginärteile als Netzeingabe,
Hanning-Fenster



(b) Training Teil 1, Amplitudenspektrum als Netzeingabe,
Hanning-Fenster



(c) Training Teil 1, Amplitudenspektrum als Netzeingabe,
Tukey-Fenster

Abbildung 3.2: Fehlerentwicklung der vLängeerschiedenen Konfigurationen über ein Training von 1000 Epochen (kleiner Test-Datensatz)

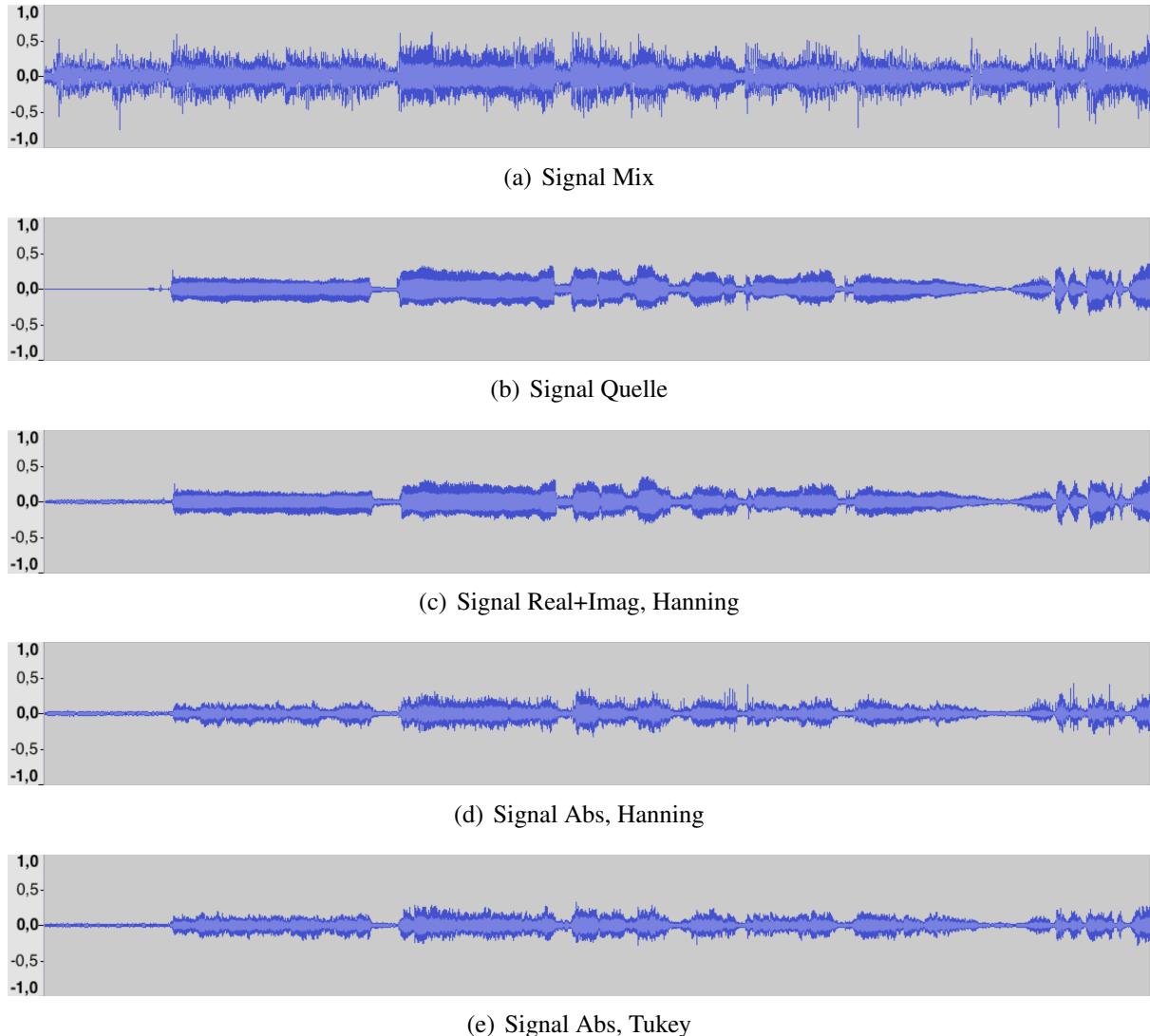


Abbildung 3.3: Audiosignale, Clara Berry And Wooldog - Boys
Audiosignale von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen.
Die Eingabe besitzt eine Länge von 10 Sekunden.

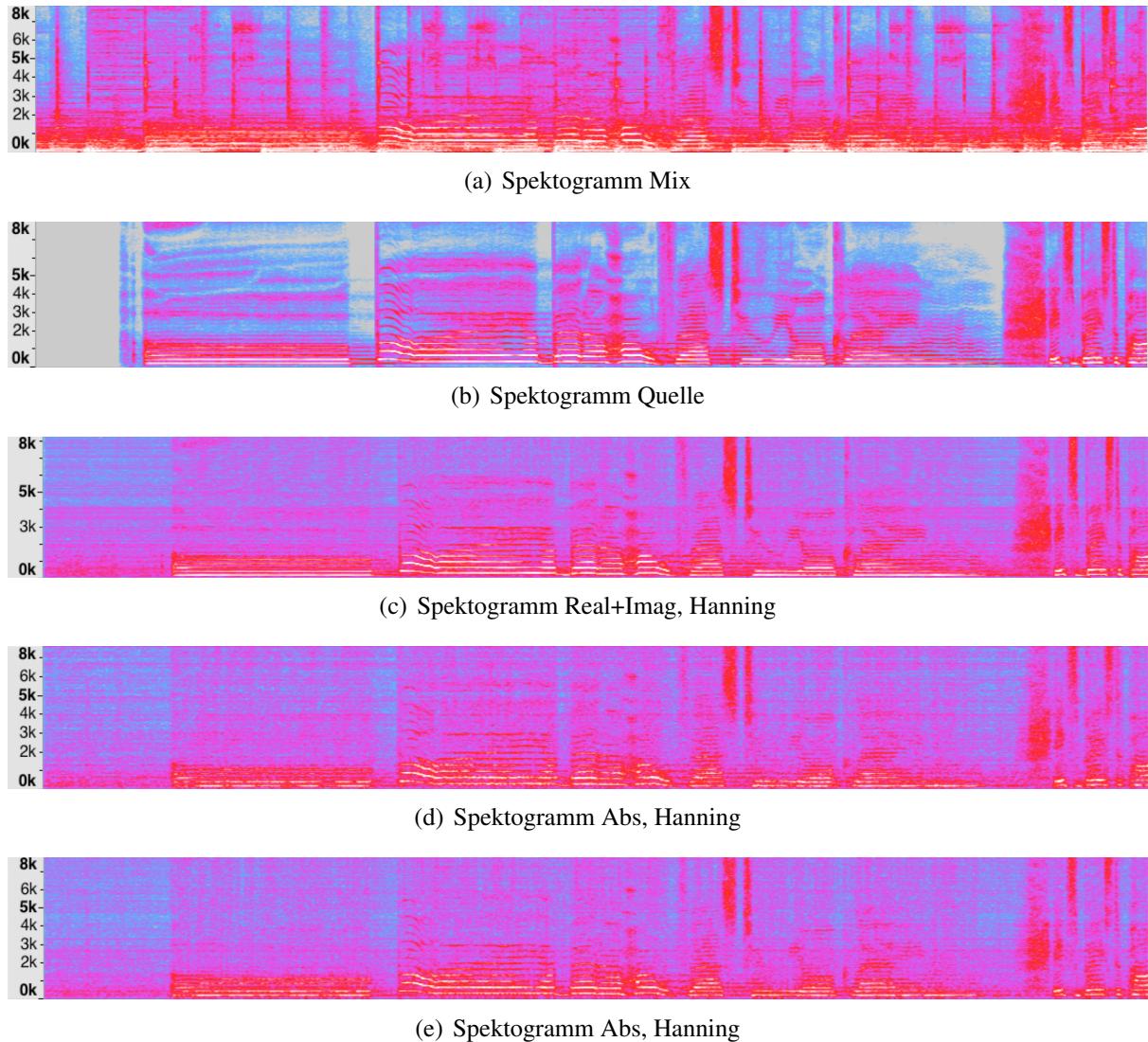


Abbildung 3.4: Spektren, Clara Berry And Wooldog - Boys

Spektren zu den Signalausschnitten aus Abbildung 3.3 (Frequenzauflösung 0-8k) von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

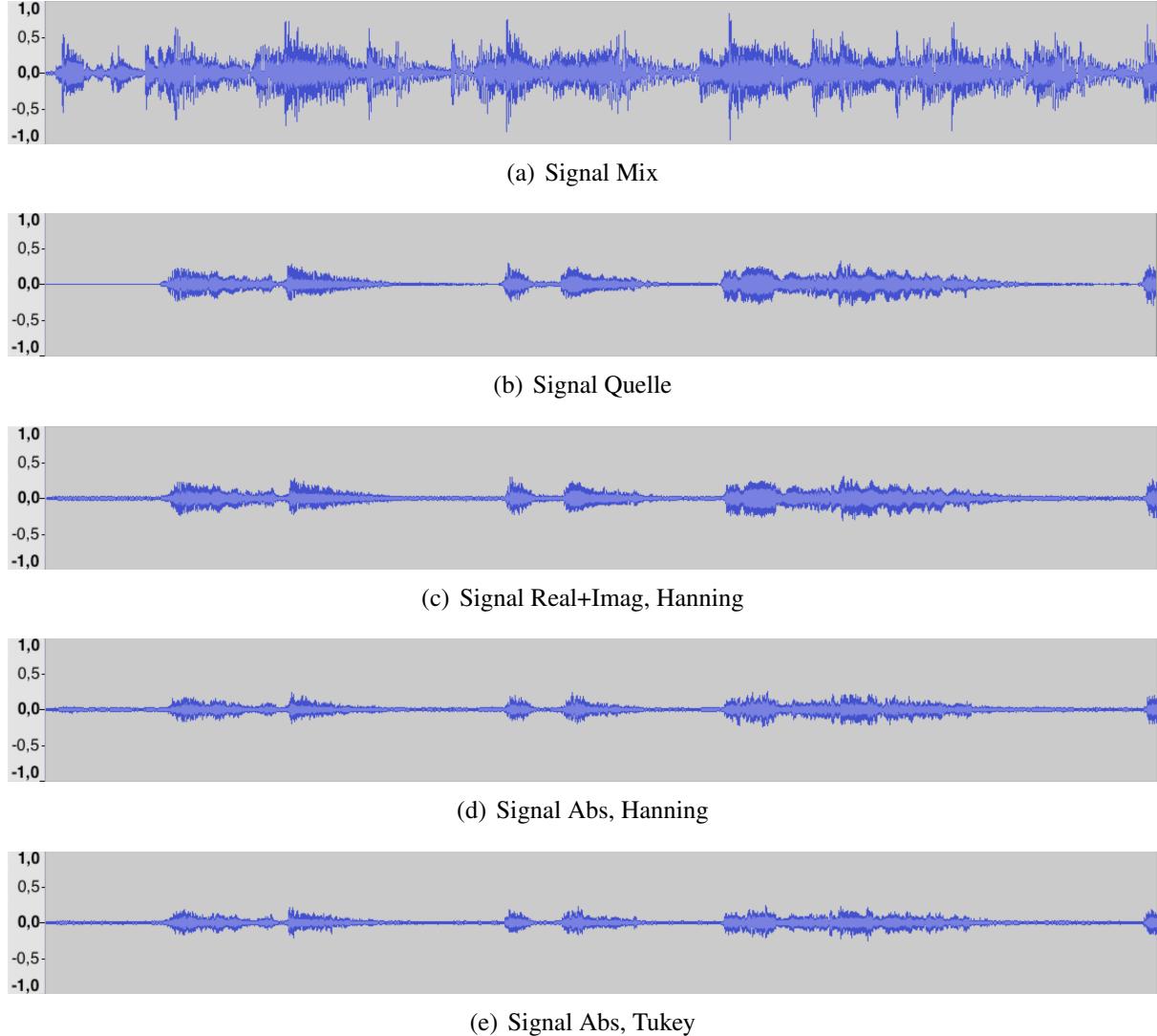


Abbildung 3.5: Audiosignale, Helado Negro - Mitad Del Mundo

Audiosignale von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

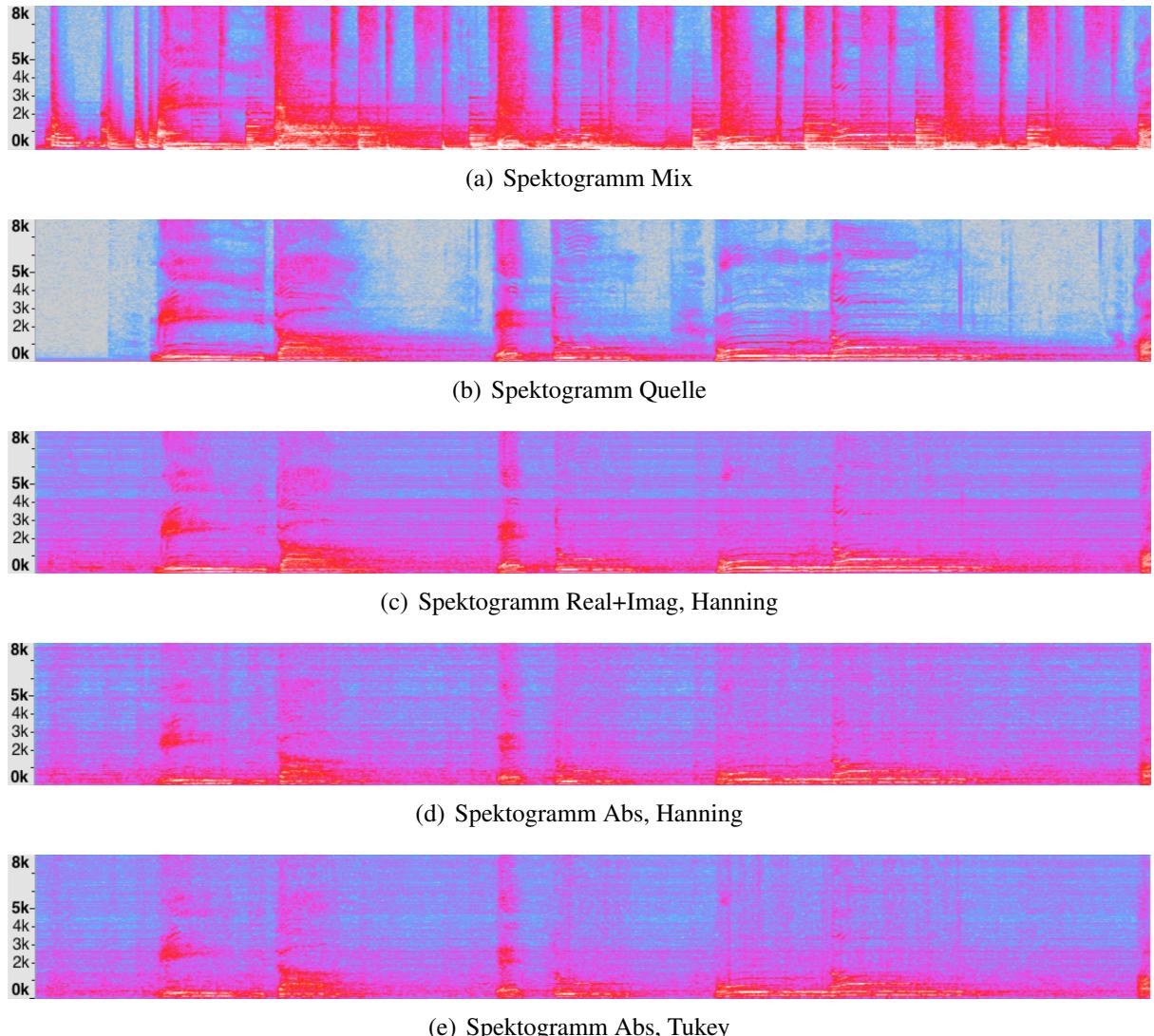


Abbildung 3.6: Spektren, Helado Negro - Mitad Del Mundo

Spektren zu den Signalausschnitten aus Abbildung 3.5 (Frequenzauflösung 0-8k) von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

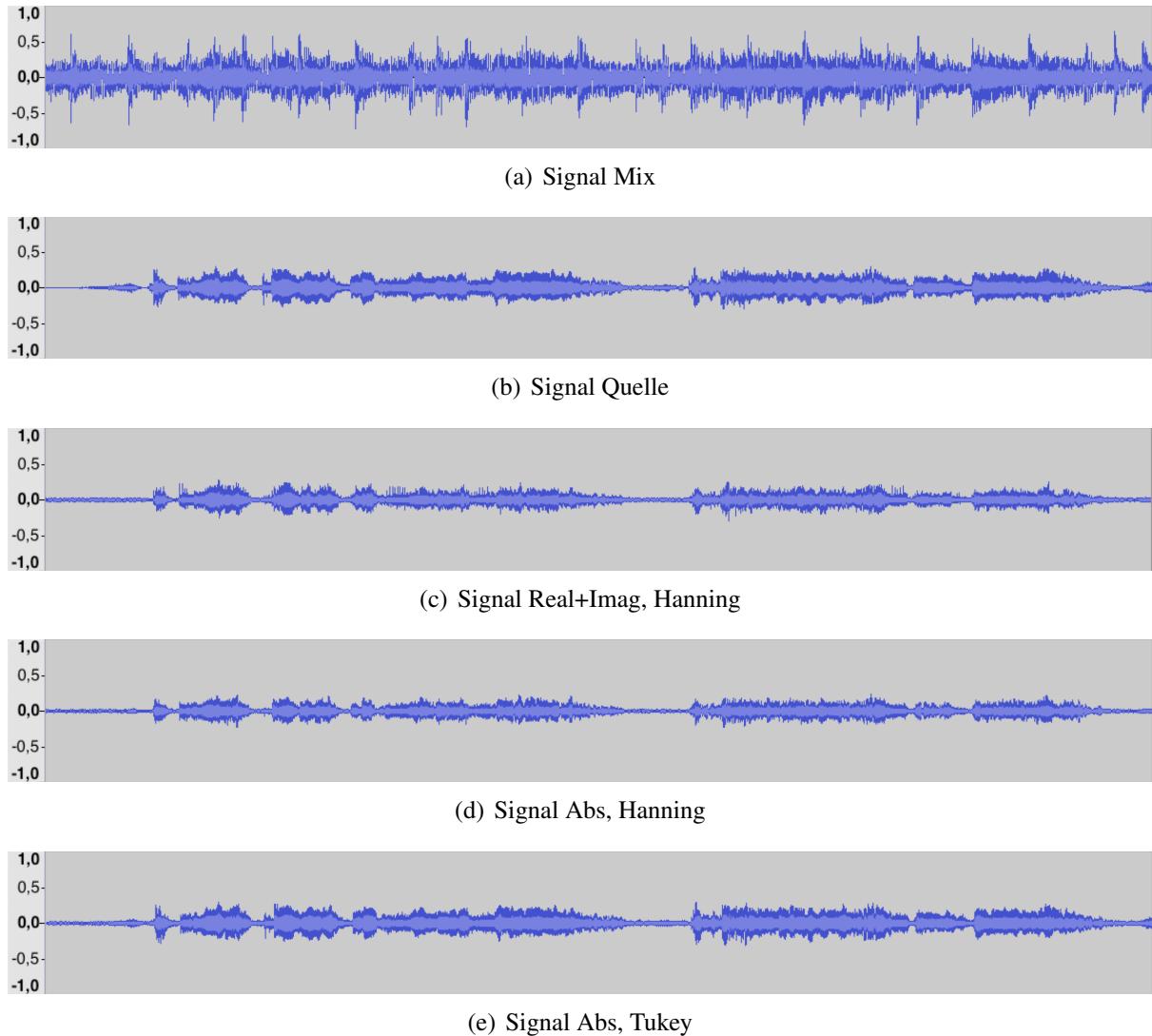


Abbildung 3.7: Audiosignale, Sweet Lights - You Let Me Down

Audiosignale von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

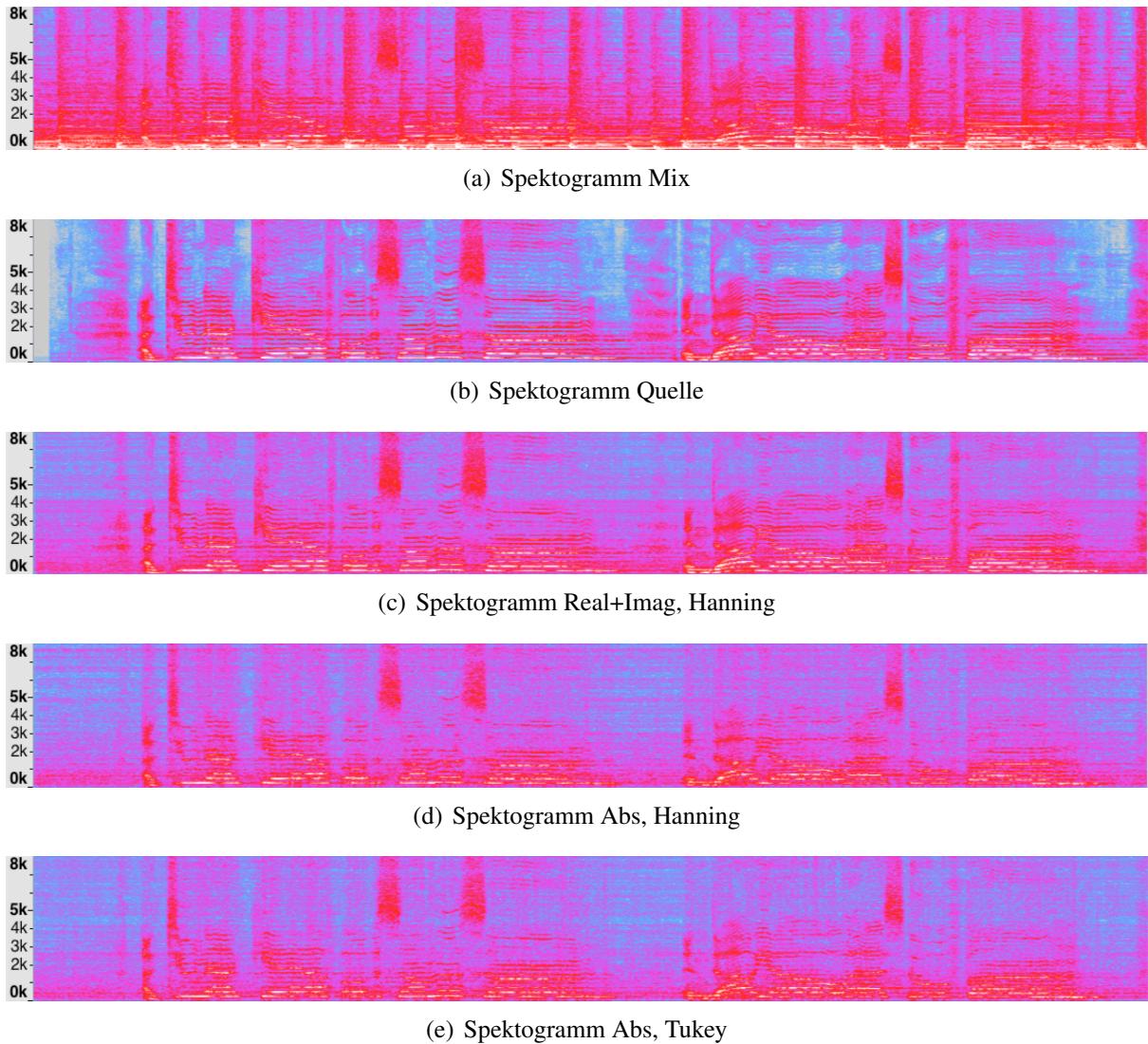
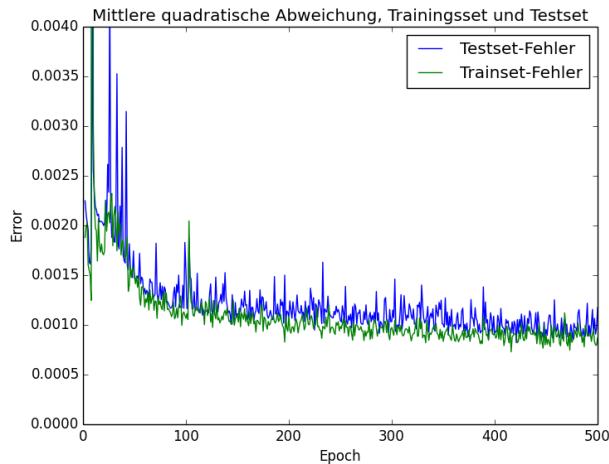


Abbildung 3.8: Spektren, Sweet Lights - You Let Me Down

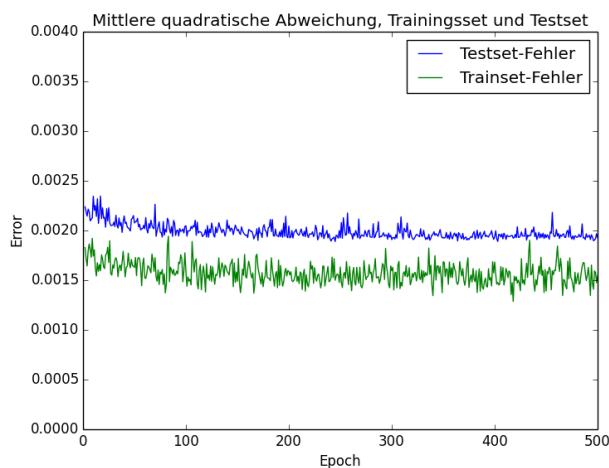
Spektren zu den Signalausschnitten aus Abbildung 3.7 (Frequenzauflösung 0-8k) von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

3.2 Training - Teil 2

Im nächsten Schritt wurde das Training auf den gesamten Datensatz bestehend aus 50 Paaren von Mix und Quelle ausgeweitet. Davon wurden 45 Musikstücke zum Training und der kleine Datensatz, aus Teil 1 des Trainings für die Validierung verwendet. Für die Minibatch- und Batchgröße wurden die Werte aus Teil 1 des Trainings übernommen ($Bs = 5, Mbs = 100$). Auf die Verwendung des Tukey-Fensters wurde aufgrund der im Vergleich zum Hanning-Fenster schlechteren Ergebnisse, in Teil 1 des Trainings verzichtet. Es wurde ein Netz mit 1000 LSTM-Blöcken für die Eingabe, bestehend aus Real- und Imaginärteilen, und ein Netz, bestehend aus 1000 LSTM-Blöcken mit der Eingabe der Amplituden trainiert. Es ist darauf hinzuweisen, dass eine Epoche einer Iteration über den gesamten Datensatz entspricht. Unter Beachtung der verwendeten Batch- und Minibatchgröße, einer Sprungweite von $R = M/2 = 1024$ und 45 Trainingstücken, entspricht dies einer Dauer von $(5 \cdot 100 \cdot 1024 \cdot 45 \text{ frames})/44100\text{s}^{-1} \approx 522.44\text{s}$ bei einer Abtastrate von 44100Hz, wobei sich diese Angabe auf die Gesamtspielzeit der in einer Epoche trainierten Eingangssignale bezieht (nicht auf die Trainingsdauer).



(a) Real+Imaginär-Teile als Eingabe



(b) Amplituden-Werte als Eingabe

Abbildung 3.9: Fehlerentwicklung über 500 Epochen (gesamter Datensatz)

Methode	LSTMs	Trainings-Fehler	Evaluierungs-Fehler
Real+Imag, Hanning	1000	0.00087	0.00117
Amplitudes, Hanning	1000	0.00154	0.00192

Tabelle 3.3: Gesamtfehler über alle im Trainingsset enthaltenen Musikstücke (Trainings-Fehler) und alle im Testset enthaltenen Musikstücke (Evaluierungs-Fehler) nach einem Training von 500 Epochen.

Interpret - Titel	Eingabemethode	Fehler
Clara Berry And Wooldog - Boys	Real+Imag	0.00117
Clara Berry And Wooldog - Boys	Abs	0.00192
Helado Negro - MitadDelMundo	Real+Imag	0.000653
Helado Negro - MitadDelMundo	Abs	0.00081
Sweet Lights - YouLetMeDown	Real+Imag	0.000865
Sweet Lights - YouLetMeDown	Abs	0.00119

Tabelle 3.4: Fehler der im Evaluierungs-Datensatz enthaltenen Musikstücke nach einem Training von 500 Epochen. Verwendet wurden die Netze mit 1000 LSTMs in der verdeckten Schicht

Auswertung

Vergleicht man die Ausgaben der einzelnen Ausschnitte, welche in Teil 1 für das Training und in Teil 2 für die Evaluierung verwendet wurden, liegt der Fehler der Evaluierungsergebnisse weit über dem Fehler der Ergebnisse aus Teil 1. Die Fehlerentwicklung des Trainings ist in Abbildung 3.9 dargestellt. Die Verwendung der Real- und Imaginärteile als Eingabe schneidet, wie schon in Teil 1 des Trainings, im Vergleich zur Eingabe bestehend aus den Amplituden besser ab. In den ersten 100 Epochen des Trainings ist eine geringe Abnahme des Fehlers sichtlich zu verzeichnen. Ab Epoche 100 sind keine oder nur minimale, weitere Verbesserungen vorzuweisen. Die Audiosignale, dargestellt in den Abbildungen 3.3, 3.5 und 3.7 aus Teil 1, für welche eine Separierung der Quelle aus dem Mix bestätigt werden konnte, gleichen den in Teil 2 erzeugten Ausgaben 3.10, 3.12 und 3.15 nur wenig. Generell ist zu erkennen, dass die Verteilungen der Abastwerte der Ausgabesignale verringert sind (schmale Audiospur). Dies entspricht einer Verringerung der Lautstärkepegel des Signals. Beim auditiven Test sind in den Ausgaben, unter Verwendung der Real- und Imaginärteile, alle im Eingangssignal enthaltenen Quellen fast vollständig erhalten und hörbar. Das Signal ist lediglich leiser und verrauschter. Bei der Verwendung der Amplituden sind andere im Eingangssignal enthaltene Quellen reduziert und nur im Geringen hörbar. Allerdings ist das Ausgangssignal stark verrauscht, undeutlich und leise. Bei einer Analyse der Spektrogramme, welche in Abbildung 3.11, 3.12 und 3.15 dargestellt sind, ist im Vergleich der Ausgabe mit der zu separierenden Quelle eine zunehmende Differenz der Amplituden, bei steigender Frequenz, zu erkennen. Markante Stellen mit hohen Amplituden (im Spektrogramm weiße Stellen) sind bis zu einer Frequenz von 1k im Quell- und Ausgabe-Spektrogramm größtenteils enthalten. Eine vollständige Isolation der Quelle aus dem Eingangssignal konnte nicht generiert werden, eine Annäherung an die Isolation wurde unter Verarbeitung der Amplituden durch das neuronale Netz erreicht.

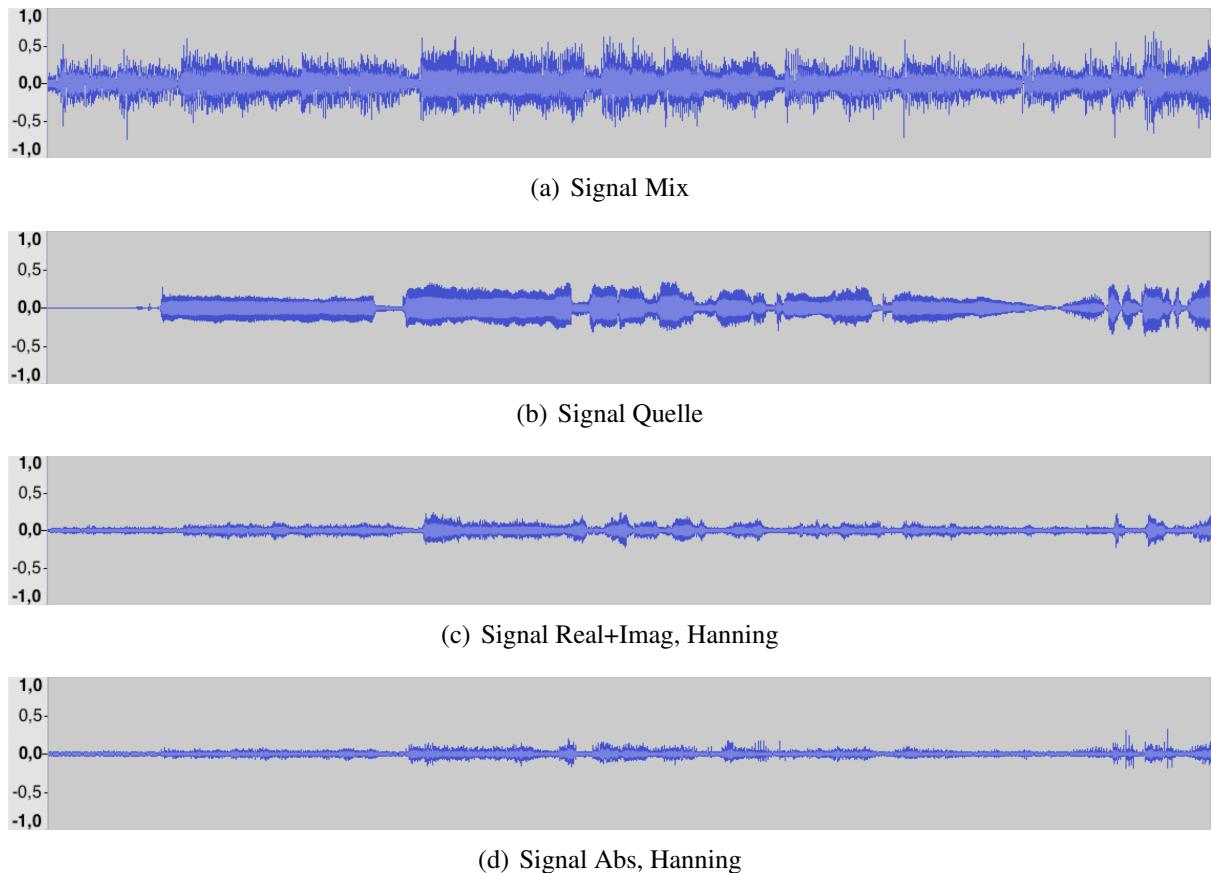


Abbildung 3.10: Audiosignale, Clara Berry And Wooldog - Boys

Audiosignale von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

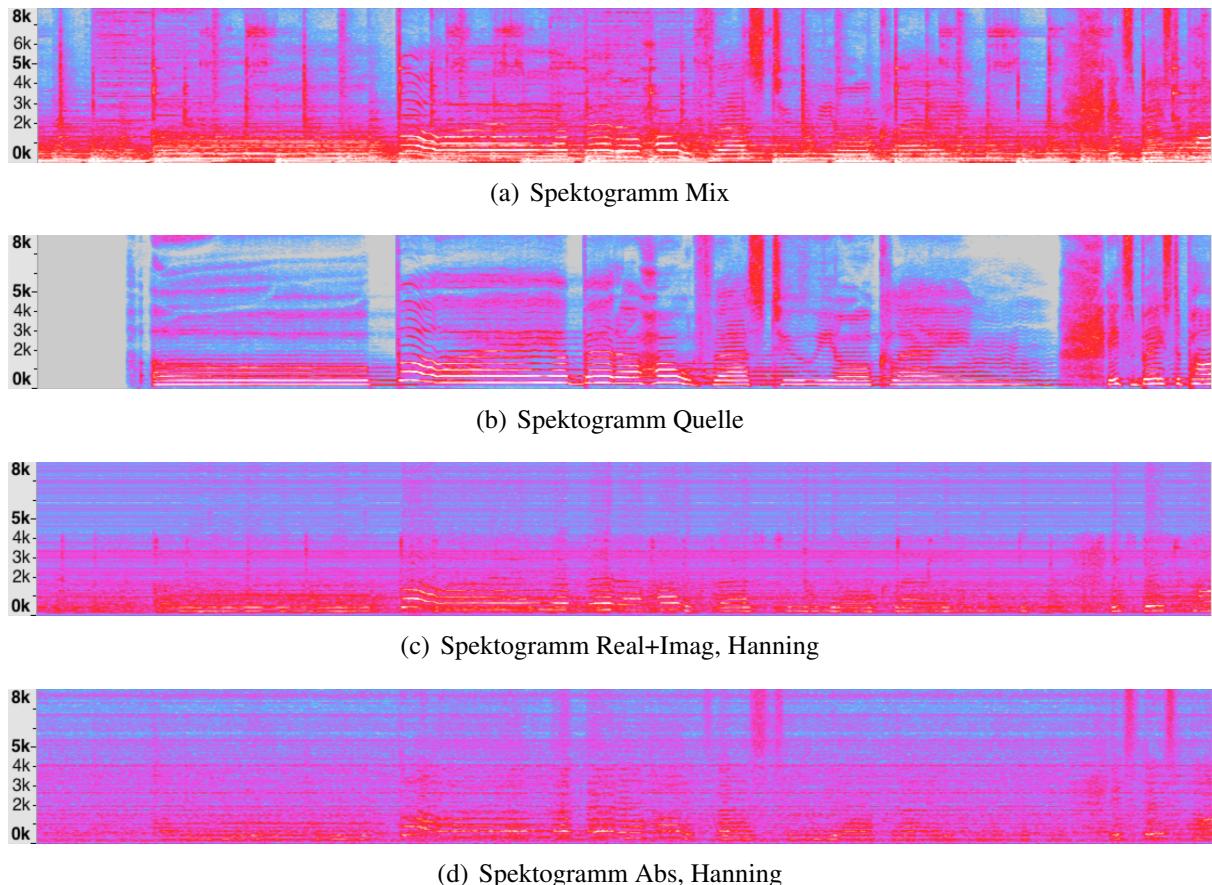


Abbildung 3.11: Spektren, Clara Berry And Wooldog - Boys

Spektren zu den Signalausschnitten aus Abbildung 3.3 (Frequenzauflösung 0-8k) von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

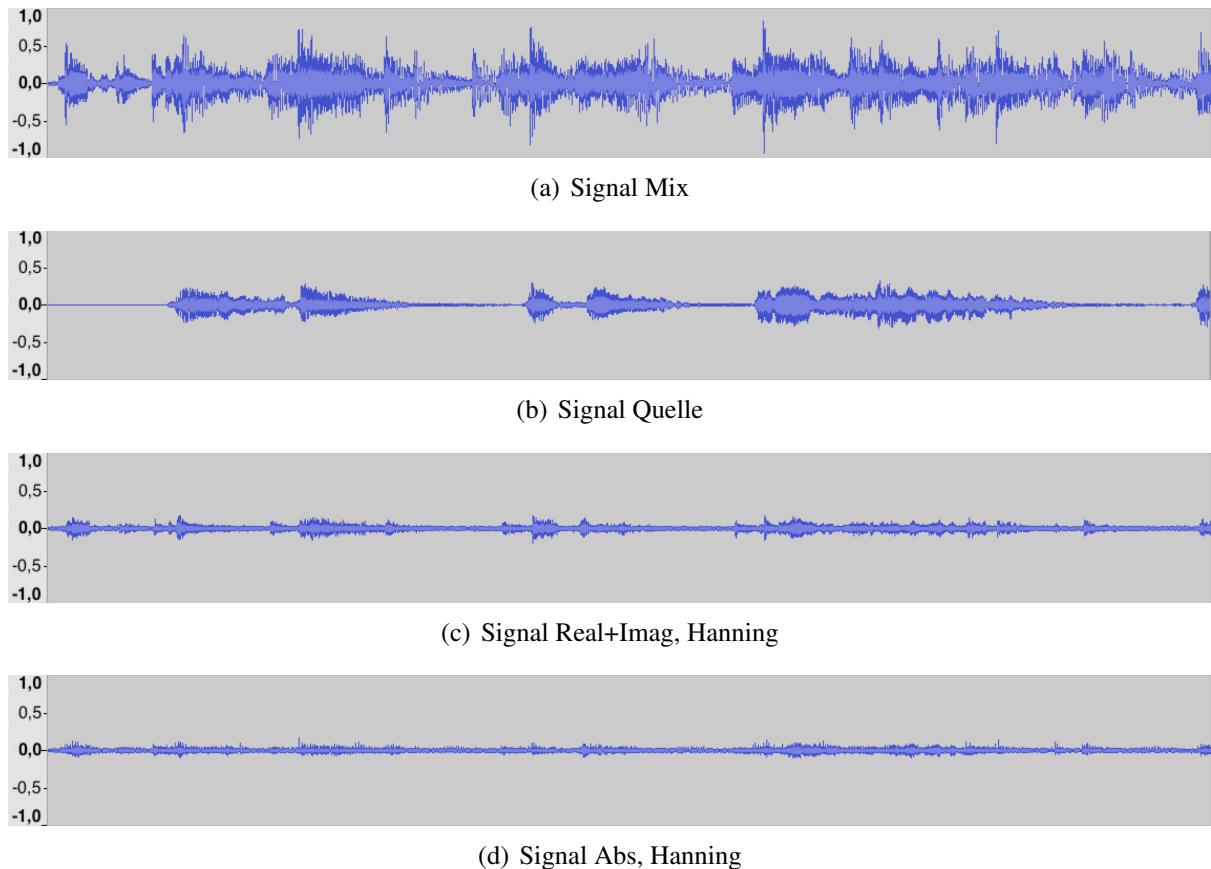


Abbildung 3.12: Audiosignale, Helado Negro - Mitad Del Mundo

Audiosignale von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

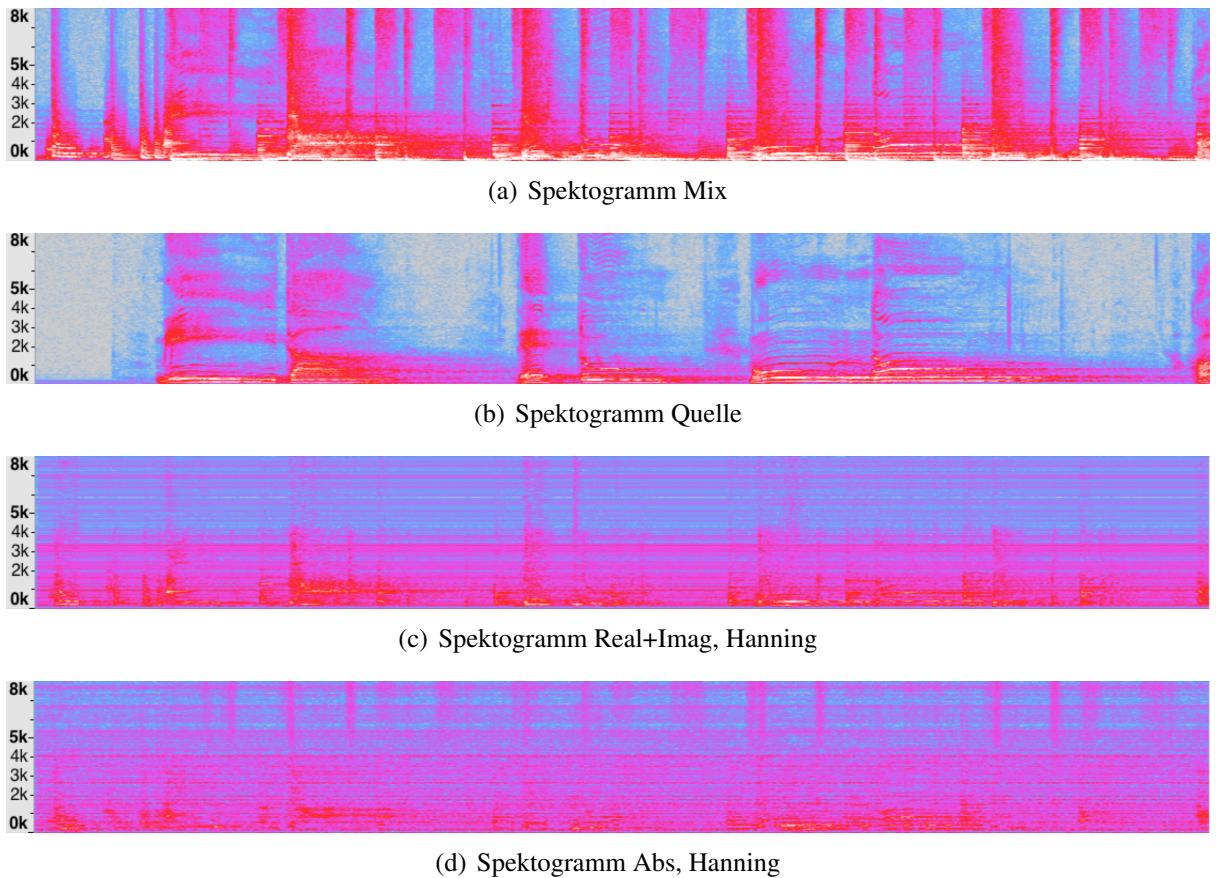


Abbildung 3.13: Spektren, Helado Negro - Mitad Del Mundo

Spektren zu den Signalausschnitten aus Abbildung 3.5 (Frequenzauflösung 0-8k) von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

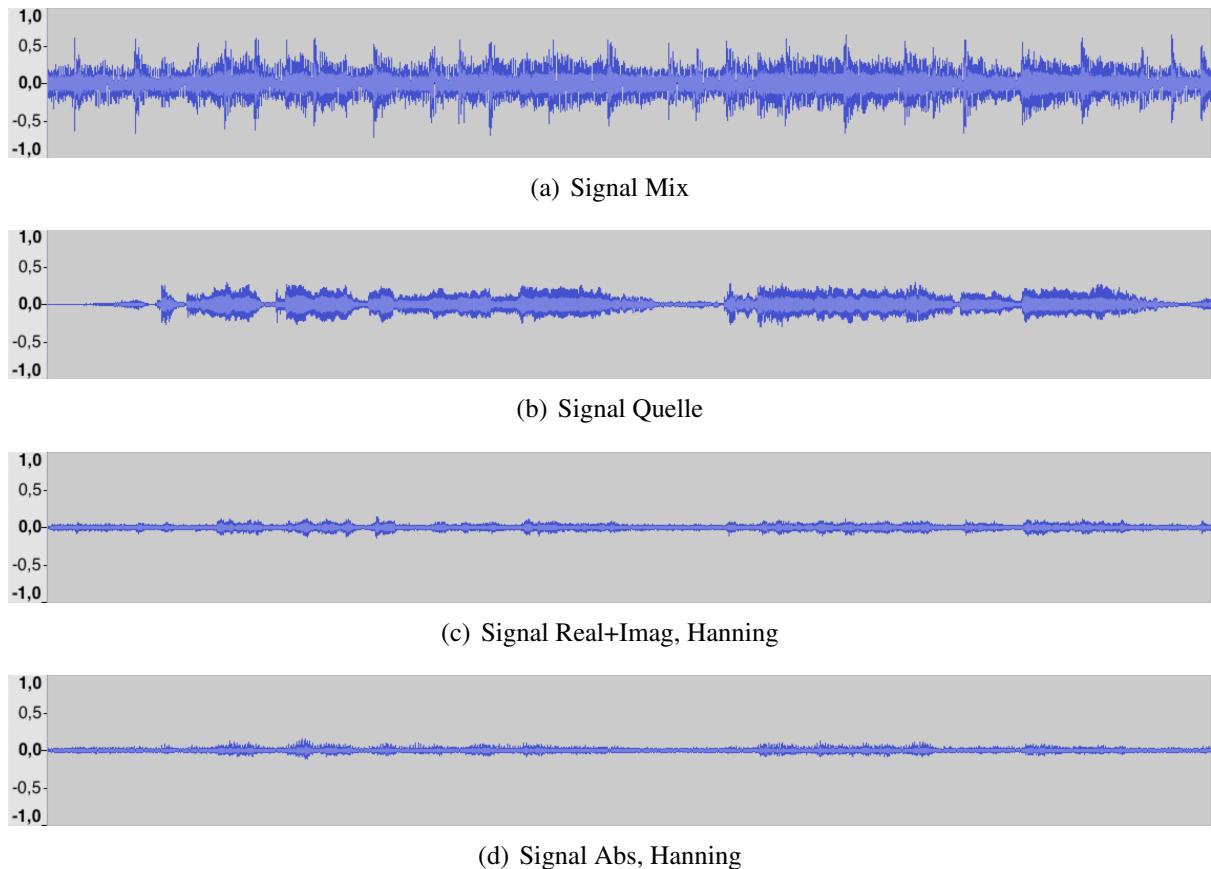


Abbildung 3.14: Audiosignale, Sweet Lights - You Let Me Down

Audiosignale von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

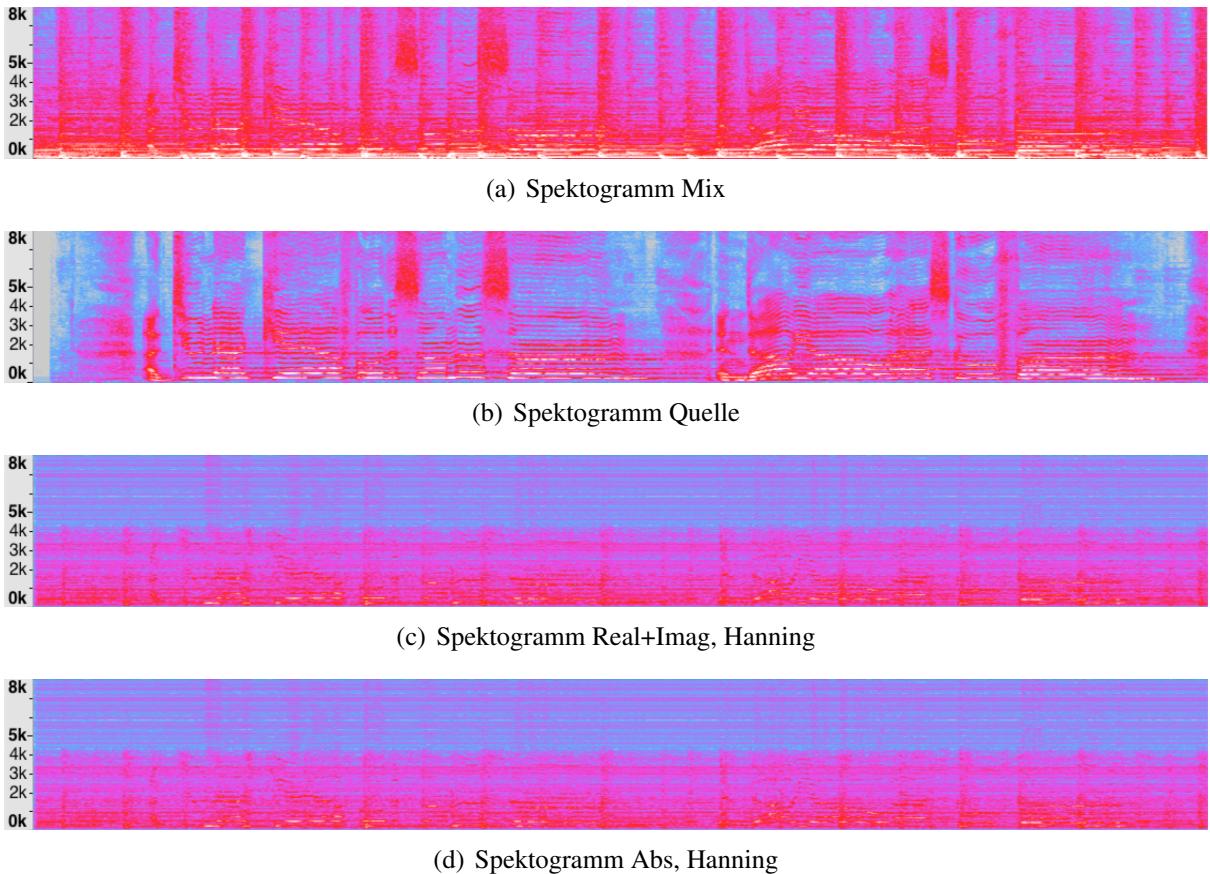


Abbildung 3.15: Spektren, Sweet Lights - You Let Me Down

Spektren zu den Signalausschnitten aus Abbildung 3.7 (Frequenzauflösung 0-8k) von Mix (Eingangssignal), Quelle (Zielsignal), und den Ausgaben der Netze (c,d,e) mit den unterschiedlichen Eingabemethoden und einer Konfiguration von 1000 LSTMs nach 500 Epochen. Die Eingabe besitzt eine Länge von 10 Sekunden.

4 Diskussion

Um eine Verbesserung gegenüber bereits vorhandener Methoden zu erzielen, wurde im Umfang dieser Arbeit versucht, die Audiosignal-Separierung mittels rekurrenter, neuronaler Netze zu erlernen. Die Audiosignal-Separierung ist definiert als die Separierung einer Tonquelle aus einem Signal, welches aus mehreren sich überlagernden Tonquellen besteht. Bisherige Lösungsansätze, eine komplexe Tonquelle aus einem Musikstück zu extrahieren, konnten keine zufriedenstellenden Ergebnisse liefern. Damit präsentiert die Audiosignal-Separierung ein aktuelles Problem der digitalen Signalverarbeitung. Für die Separierung wurde eine auf rekurrenten Netzen basierende Methode gewählt, da sich diese aufgrund ihrer zyklischen Strukturen besonders für die Verarbeitung von Sequenzen und dem Erkennen zeitlicher Zusammenhänge eignen. Als primäre, rekurrente Struktur wurden sogenannte Long-Short-Term-Memorys (LSTMs) [HS97] verwendet. Diese besitzen den Vorteil, Zusammenhänge zwischen Ereignissen mit großer zeitlicher Distanz berücksichtigen zu können. Dadurch können Charakteristika der Frequenzbereiche und Verläufe bestimmter Klangerzeuger erlernt werden. Für die Separierung eines Audiosignals, wurde eine Verarbeitung in der Frequenzdomäne gewählt. Dadurch kann, im Gegensatz zur Verarbeitung in der Zeitdomäne, eine Analyse und Anpassung der Frequenzen durchgeführt werden. Bei der Verarbeitung des Signals in der Frequenzdomäne wurde dieses zuvor in Abschnitte zerlegt und via Kurzzeit-Fourier-Transformation in den Frequenzraum transformiert. Die Frequenzen in den dadurch gewonnenen Frequenzspektren können somit durch das neuronale Netz verändert werden, um das Filtern bestimmter Signalkomponenten zu ermöglichen. Für die Eingabe und Verarbeitung der Fourierkomponenten durch das neuronale Netz wurden verschiedene Möglichkeiten getestet. Eine Verarbeitung in der Frequenzdomäne kann durch die Veränderung der komplexen Fourier-Komponenten (Real- und Imaginärteile) oder durch die Anpassung der Amplituden- und Phasenspektren erfolgen. Um die effizienteste Methode für die Audiosignal-Separierung zu bestimmen, wurden für die unterschiedlichen Eingabemethoden jeweils Netze mit 100 bis 1000 LSTMs trainiert. Die dafür benötigten Trainingsdaten, bestehend aus Musikstücken und den Audiospuren der zu isolierenden Tonquelle, wurden auf Basis der MedleyDB [BST⁺] erzeugt.

Für einen ersten Vergleich der Methoden wurde eine Stichprobe von drei Musikstücken aus den Trainingsdaten der Netze verwendet. Alle für die Stichprobe trainierten Netze konnten die Separierung mit zunehmender Qualität bei einer steigenden Anzahl an LSTMs erlernen. Zudem konnten für eine Eingabe, welche aus Real- und Imaginärteilen besteht, bessere Ergebnisse erzeugt werden. Durch die Verwendung von Amplituden entstehen bei der Verarbeitung sogenannte Artefakte. Diese machen sich durch ein ausgedünntes Klangspektrum hörbar und kamen bei der Verwendung der Real- und Imaginärteile im Ausgabesignal nicht zustande. Damit konnte ein deutlicher Vorteil durch die Verarbeitung der Real- und Imaginärteile der Frequenz-Komponenten bei der Separierung ermittelt werden.

Des Weiteren wurde das Training auf den gesamten Datensatz von 45 Musikstücken ausgeweitet, um ein neuronales Netz auf die Separierung einer bestimmten Quelle aus einem beliebigen Musikstück zu trainieren. Als Netzstruktur wurde die Netztopologie mit 1000 LSTMs verwendet, mit welcher in Teil 1 des Trainings die besten Ergebnisse erzielt werden konnten.

Für die beiden zuvor verglichenen Eingabe-Methoden wurden jeweils ein Netz trainiert.

Die Separierung über den gesamten Datensatz konnte nur schlecht erlernt werden. In der Ausgabe unter Verwendung der Real- und Imaginärteile waren die einzelnen, im Eingangssignal enthaltenen, unterschiedlichen Quell-Signale zwar verändert, allerdings noch deutlich hörbar und mit zusätzlichem Rauschen überlagert. Unter Verwendung der Amplituden zur Separierung konnten andere Quellen zwar reduziert werden, jedoch wurden eine Vielzahl an Artefakten durch das Netz hinzugefügt und somit nur eine qualitativ minderwertige Ausgabe erzeugt.

Für die unzureichende Lernfähigkeit der Audiosignal-Separierung bei einem Training über den gesamten Datensatz können verschiedene Ursachen ausschlaggebend sein. Eine ungünstige Wahl der initialen Parameter, kann den gesamten Trainingsverlauf negativ beeinflussen. Es wurde eine initiale Lernrate von 0.001 verwendet, welche möglicherweise zu hoch für eine Konvergenz des Netzes gewählt wurde. Ein weiterer Grund kann eine zu geringe Anzahl der verwendeten LSTMs sein, welche die Lernfähigkeit des Netzes strukturell begrenzt. Zudem nimmt die gewählte Fenstergröße bei der Kurzzeit-Fourier-Transformation aufgrund der Kupfmüllerschen Unbestimmtheitsrelation einen maßgeblichen Einfluss auf die Auflösung des Signals im Zeit- und Frequenzbereich. Mit einer verwendeten Fenstergröße von 2048 Abtastwerten wurde eine hohe Auflösung im Frequenzbereich gewählt, wobei eine unscharfe Auflösung im Zeitbereich einhergeht und damit die Qualität der Ausgabe vermutlich weiter beschränkt wurde. Im Weiteren wurden die Trainingsdaten durch einfaches Abmischen einzelner, in einem Musikstück enthaltener Quellen generiert, ohne diese dabei zu verändern.

Für eine Verbesserung der Ergebnisse werden folgende Anpassungen der Methode vorgeschlagen: Weitere Trainingsdaten können durch die Variierung, beispielsweise in der Lautstärke, einzelner Spuren generiert werden. Durch umfangreichere Trainingsdaten können somit möglicherweise bessere Resultate bei der Separierung erreicht werden. Durch zusätzliche Verarbeitung des Signals in der Zeitdomäne können, Artefakte eventuell verminder werden und damit ein weiter positiver Einfluss auf das Ergebnis erzielt werden. Eine mehrfache Verarbeitung eines Signalausschnitts durch unterschiedliche Fenstergrößen, kann eventuell weitere Verbesserungen ermöglichen.

Fazit

Ein Netz für die Separierung einer bestimmten Musikquelle konnte nicht trainiert werden. Allerdings konnten die trainierten Netze eine Abbildung eines Eingangssignals auf die zu separierende Quelle im Umfang eines kleinen Trainings-Datensatzes erlernen. Dabei konnte anhand der Verwendung verschiedener Netzstrukturen und Eingabemethoden gezeigt werden, dass die Verarbeitung der komplexen Fourier-Komponenten in Form der Real- und Imaginärteilen deutlich bessere Ergebnisse als bei der Verwendung der Amplituden erzielen konnten.

Literaturverzeichnis

- [AAB⁺15] Abadi, Martin, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin et al.: *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org, 1, 2015.
- [AR77] Allen, Jont B und Lawrence R Rabiner: *A unified approach to short-time Fourier analysis and synthesis*. Proceedings of the IEEE, 65(11):1558–1564, 1977.
- [BSF94] Bengio, Yoshua, Patrice Simard und Paolo Frasconi: *Learning long-term dependencies with gradient descent is difficult*. IEEE transactions on neural networks, 5(2):157–166, 1994.
- [BST⁺] Bittner, Rachel M, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam und Juan Pablo Bello: *MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research*.
- [BT58] Blackman, Ralph Beebe und John Wilder Tukey: *The measurement of power spectra*. 1958.
- [CT65] Cooley, James W und John W Tukey: *An algorithm for the machine calculation of complex Fourier series*. Mathematics of computation, 19(90):297–301, 1965.
- [Gra12] Graves, Alex: *Neural Networks*. In: *Supervised Sequence Labelling with Recurrent Neural Networks*, Seiten 15–35. Springer, 2012.
- [Har78] Harris, Fredric J: *On the use of windows for harmonic analysis with the discrete Fourier transform*. Proceedings of the IEEE, 66(1):51–83, 1978.
- [Hoc98] Hochreiter, Sepp: *The vanishing gradient problem during learning recurrent neural nets and problem solutions*. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02):107–116, 1998.
- [HS97] Hochreiter, Sepp und Jürgen Schmidhuber: *Long short-term memory*. Neural computation, 9(8):1735–1780, 1997.
- [KB14] Kingma, Diederik und Jimmy Ba: *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.
- [Smi11] Smith, Julius O: *Spectral audio signal processing*. W3K, 2011.
- [SRP15] Simpson, Andrew JR, Gerard Roma und Mark D Plumley: *Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network*. In: *International Conference on Latent Variable Analysis and Signal Separation*, Seiten 429–436. Springer, 2015.

- [Tea08] Team, Audacity Developer: *Audacity (Version 2.1.2)* [Computer software]. Available: audacity. sourceforge. net/download, 2008.