Output-Target-Interpolation für Spektrale Fehlerfunktionen

Jens Johannsmeier[1], Sebastian Stober[1]

[1] *Otto-von-Guericke Universität, 39106 Magdeburg, Deutschland, Email:{jens.johannsmeier,stober}@ovgu.de*

Einleitung

Tonhöhenerkennung stellt eine zentrale Aufgabe in der Audioanalyse dar, vor allem in Kontext musikalischer Daten. Hier wird angenommen, dass ein Signal eine definierte Fundamentalfrequenz aufweist, welche extrahiert werden soll. Dies ist sowohl für sich genommen relevant, beispielsweise zur Klassifizierung von Tönen anhand bestimmter Merkmale, als auch als Komponente in generativen Modellen wie Differentiable Digital Signal Processing (DDSP) [1]. Häufig kommen dabei künstliche neuronale Netze [2] zum Einsatz.

Untersuchungen haben gezeigt [3], dass diese Aufgabe überraschend schwierig ist. Naive Tonhöhenerkennung als Teil anderer Lernaufgaben scheitert häufig. Bestehende Ansätze umgehen das Problem daher in der Regel mit verschiedenen Methoden. Zum Beispiel verwendet [1] ein vortrainiertes Netz für diesen Zweck.

Wir stellen in dieser Arbeit eine simple Methode vor, um Tonhöhenerkennung mit gängigen Fehlerfunktionen zu verbessern. Diese basiert auf der Interpolation zwischen Modellausgaben und Zieldaten. Wir zeigen in Beispielen mit synthetischen Daten, dass unsere Methode das Trainieren von Modellen erlaubt, die mit herkömmlichen Methoden scheitern. Allerdings stellen wir fest, dass unsere Methode starke Annahmen macht und nur in einer begrenzten Zahl von Umgebungen anwendbar ist. Ferner kontrastieren wir Tonhöhenerkennung als *Regression* und als *Klassifizierung*, und zeigen, dass letztere deutlich einfacher zu Ergebnissen führt.

Tonhöhenerkennung

Gegeben sei ein digitales (Audio-)Signal y, welches in Form von diskreten Samples vorliegt. Wir nehmen an, dass y ein *kontinuierliches* Signal $\tilde{y}$ unterliegt, welches aus periodischen Komponenten besteht, denen somit eine Frequenz f zugewiesen werden kann. Bei realistischen Signalen, z.B. aus der Musik, exisitiert häufig eine *Grundfrequenz* F_0, wobei andere im Signal enthaltene Frequenzen Vielfache der Grundfrequenz sind. Für die Betrachtung in dieser Arbeit reichen Sinussignale, die nur eine einzige Frequenz enthalten. Ein Beispiel ist in Abbildung 1 gegeben. Die Aufgabe ist nun, anhand des diskreten Signals y die darunterliegende F_0 zu bestimmen.

Um das Problem praktisch zu untersuchen, können wir eine Frequenz F_0 wählen und eine geeignete Anzahl Samples anhand der Funktion $y = \sin(F_0 \cdot 2\pi t)$ generieren, wobei t die Zeitdimension beschreibt. Anschließend definieren wir ein Modell $\hat{y} = \sin(f \cdot 2\pi t)$, wobei f ein lernbarer Parameter ist. Das Ziel ist nun, anhand der generierten Daten den korrekten Modellparameter zu lernen, d.h. $f = F_0$. Hierfür stehen verschiedene Methoden zur Auswahl; wir verwenden Gradientenabstieg, der auch in neuronalen Netzen Anwendung findet

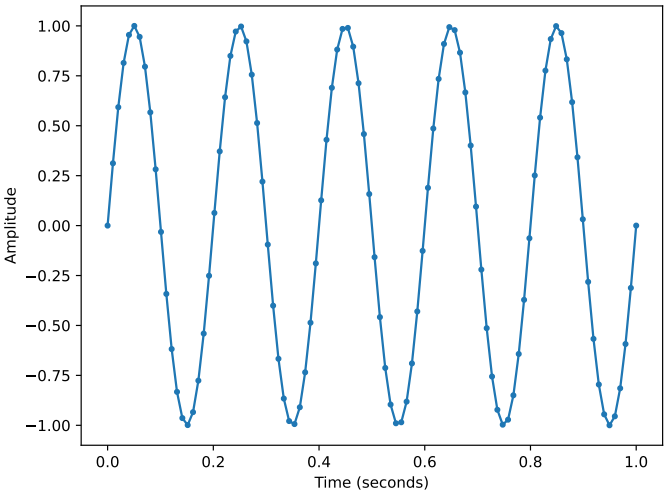

Abbildung 1: Einfaches digitales Audiosignal (Punkte) mit darunterliegendem analogen Signal (Linie). Die Frequenz bzw. Tonhöhe beträgt 5 Hz.

[4]. Dafür müssen wir eine *Fehlerfunktion* definieren, welche den Abstand zwischen Modell- und Zielausgaben quantifiziert. Eine häufig verwendete Funktion für Audiodaten ist der Multi-Scale Spectral Loss (MSSL) [1]; dieser basiert darauf, die Audiosignale zuerst via Short-Time Fourier Transformation (STFT) in Spektrogramme zu transformieren und diese anschließend zu vergleichen. Da es bei der STFT einige Hyperparameter wie Fenstergröße und Schrittweite gibt, für die generell keine optimalen Werte existieren, werden mehrere Spektrogramme mit verschiedenen Parametersätzen berechnet und verglichen, und die Fehler aufsummiert.

Abbildung 2 zeigt das Ergebnis eines Trainingsprozesses. Dafür haben wir aus Gründen der Reproduzierbarkeit die Zielfrequenz $F_0 = 500$ Hz und den Initialwert für $f = 1000$ Hz gewählt. Wir generieren 16000 Samples gleichmäßig im Zeitraum einer Sekunde. Die Lernrate wird über den Trainingsprozess linear von 1 zu 0 interpoliert, außerdem verwenden wir ein Momentum von 0.9. Es ist zu sehen, dass der Fehler über die Trainingsdauer kaum verringert wird. Ferner bewegt sich der Parameter f kaum vom Startwert – tatsächlich findet die Bewegung gar in die falsche Richtung, weg von 500 Hz, statt.

Der Grund für dieses Verhalten ist in Abbildung 3 aufgezeigt: Die Fehlerfunktion ist fast im gesamten Parameterraum sehr flach, was wiederum zu sehr kleinen Gradienten führt. Nur wenn der Parameter bereits sehr nah an der Lösung liegt, ist der Fehler informativ und ein ausreichender Gradient vorhanden. Da der Gradient aber benötigt wird, um überhaupt in die Nähe der Lösung zu kommen, ist der MSSL scheinbar ungeeignet. Dieses Problem wird in [3] näher beleuchtet.

An dieser Stelle möchten wir auf die Relevanz solch einer ein-

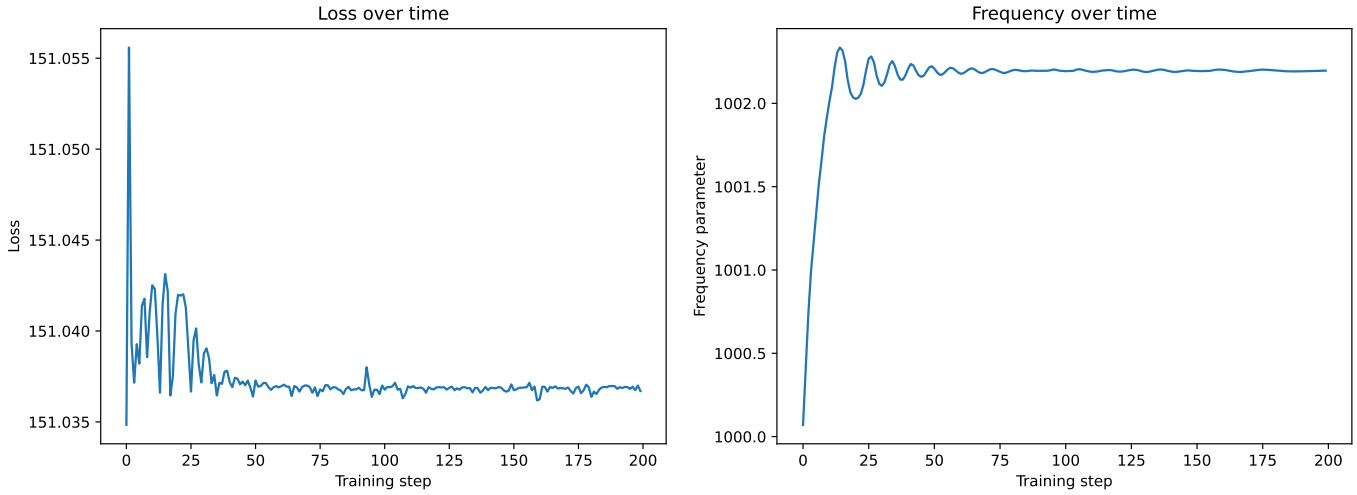

Abbildung 2: Ergebnis des naiven Trainings mit MSSL. Links: der Fehler zeigt kaum Verringerung über das Training. Rechts: Der Parameter f bewegt sich vom Startwert von 1000 Hz nach oben, anstatt in Richtung des Zielwerts von 500 Hz.

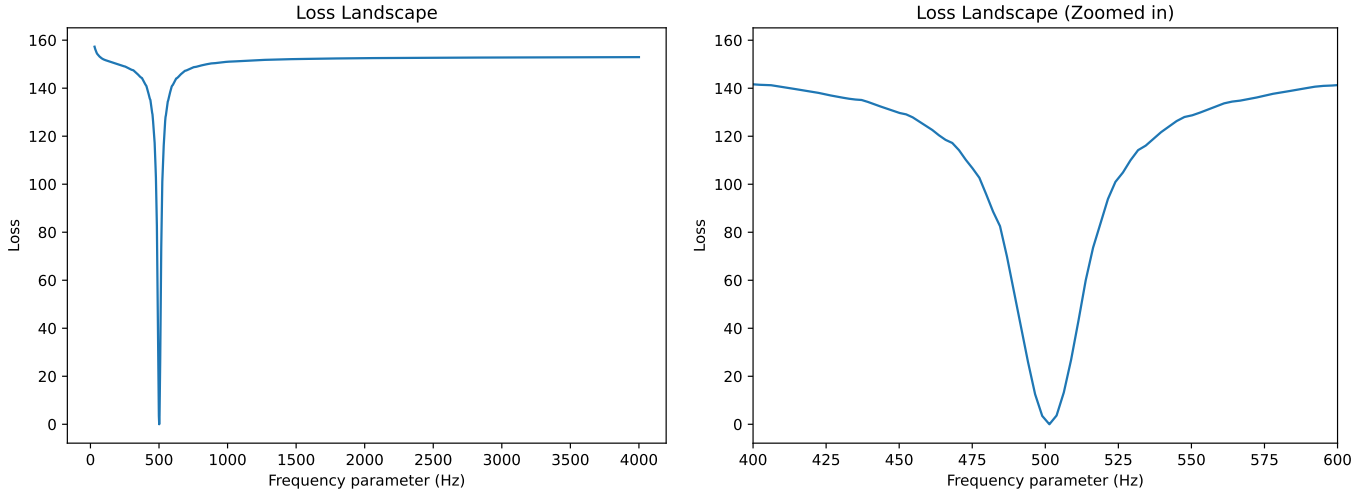

Abbildung 3: Fehlergebirge des MSSL. Links zeigt einen größeren Parameterraum, rechts einen kleineren Ausschnitt um die Zielfrequenz herum.

fachen synthetischen Lernaufgabe eingehen. [3] argumentiert, dass das Scheitern einer Fehlerfunktion bzw. eines Lernalgorithmus auf einfachen Aufgaben ein Indiz für Probleme bei komplexeren Problemen sein kann. Aufgrund der Komplexität der verwendeten Architekturen sind diese allerdings oft schwer zu analysieren und zu verstehen. Eine synthetische Aufgabe kann somit eine Art notwendige Bedingung darstellen, die eine Lernmethode bewältigen muss, um verlässlich zu sein. In unserem Fall könnte Audiogenerierung auf komplizierteren Datensätzen ein analoges komplexes Problem sein. Beispielsweise können Variational Autoencoder (VAEs) [5] auf großen Musikdatensätzen trainiert werden. Hier kommt ebenfalls eine Fehlerfunktion zwischen Modellausgaben und Zieldaten zum Einsatz. Da musikalische Daten in der Regel wohldefinierte Tonhöhen aufweisen, muss ein Modell, um korrekte Daten zu generieren, diese Tonhöhen intern repräsentieren und damit auch lernen. Modelle wie DDSP stellen die Tonhöhe gar explizit dar. Es ist daher davon auszugehen, dass das Lernen von Tonhöhen als Komponente solcher Modelle notwendig ist; und wie wir sehen, ist dies mit herkömmlichen Fehlerfunktionen sehr schwierig. Aus diesem Grund benutzt beispielsweise das DDSP-Paper ein *vortrainiertes* Netz, um Tonhöhen zu extrahieren und in den Generator einzuspeisen. Dies bedeutet, dass Raum ist für *simple* Methoden, die das Erkennen von Tonhöhen als *integrierte* Komponente eines Modells ermöglichen, ohne auf externe vortrainierte Modelle zurückgreifen zu müssen. Im Folgenden stellen wir einen Ansatz für eine solche Methode dar.

Interpolation für Spektrale Fehlerfunktionen

Unsere Methode beruht auf der einfachen Beobachtung, dass beim MSSL der Gradient besser wird, je näher wir am Ziel sind. Das Problem ist dabei natürlich, überhaupt nahe ans Ziel zu kommen. Da wir ja aber das Ziel kennen, können wir uns von jedem Punkt beliebig nahe daran bewegen, nämlich via *Interpolation*. Sei y das Zielsignal und $\hat{y}$ die aktuelle Modellausgabe. Ferner sei $\alpha \in [0,1]$ der Interpolationsparameter. Dann ist $\alpha y + (1-\alpha)\hat{y}$ die interpolierte Ausgabe. Für $\alpha = 1$ ist dies einfach das Ziel, für $\alpha = 0$ die eigentliche Modellausgabe, und mit $\alpha \to 1$ können wir uns beliebig nahe an das Ziel heranbewegen. Damit können wir, ausgehend von jeder beliebigen Ausgabe, von besseren Gradienten für die Modellausgabe profitieren. Dies ist schematisch in Abbildung 4 dargestellt.

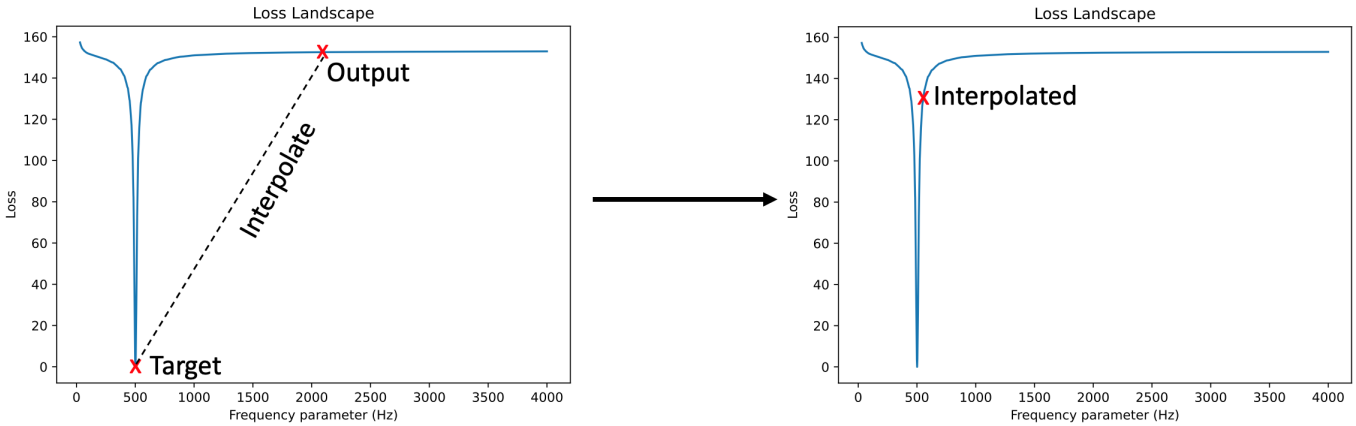

Abbildung 4: Schematische Darstellung der Interpolation zweier Signale. Links: Zielsignal und potentiell weit entferntes Modellsignal, mit schlechtem MSSL-Gradienten. Rechts: Interpoliertes Signal mit besserem Gradienten.

Es gibt es nun ein großes Problem: Es ist naheliegend, entweder die Audiosignale direkt, oder die Spektrogramme (während der Berechnung des MSSL) zu interpolieren. Lineare Interpolation ist letztendlich nichts anderes als eine gewichtete Summe. Eine Summe von zwei Sinussignalen ist selbst aber *kein* Sinussignal mit einer dazwischenliegenden Frequenz, sondern ein komplexeres Signal, in dem beide Ursprungsfrequenzen vorhanden sind. Dies bringt keine Verbesserung, da die Zielfrequenz keine Information liefert bzw. wir das Zielsignal nicht ändern können, und andererseits die Ursprungsfrequenz sich weiterhin im problematischen Bereich der Fehlerfunktion befindet, wo kaum Gradient vorhanden ist.

Stattdessen ist es möglich, direkt die *Frequenzen* zu interpolieren, was den gewünschten Effekt herbeiführt. Ein Trainingsprozess ist in Abbildung 5 aufgeführt. Hierbei starten wir mit $\alpha \approx 1$ und reduzieren den Wert linear Richtung 0, ähnlich wie die Lernrate. Somit ist das interpolierte Signal anfangs automatisch nahe am Ziel, und wird schrittweise Richtung Modellausgabe bewegt, welche sich wiederum mit fortschreitendem Training an das Zielsignal annähern sollte. Es ist aber auch möglich, einen fixen Wert zu wählen, beispielsweise $\alpha = 0.99$. Tatsächlich ändert die Interpolation nichts an der optimalen Lösung des Modells; da das Zielsignal herabgewichtet wird, muss das Modellsignal die korrekte Frequenz erreichen, um dies auszugleichen und einen minimalen Fehler zu erzielen. Es lässt sich auch zeigen, dass die Gradienten des MSSL mit interpolierten Ausgaben (mit hinreichend großem α) in einem wesentlich größeren Bereich nützliche Information liefern, auch weit entfernt von der Zielfrequenz. Aus Platzgründen können wir dies hier nicht darstellen.

Auch wenn die Methode so funktioniert, so ist es eine starke Einschränkung, dass die Zielfrequenz bekannt sein muss. Häufig sind nämlich nur *digitale Signale* gegeben, nicht aber die Frequenz selbst. Ferner muss das Modell die Frequenz explizit darstellen, damit interpoliert werden kann. Somit ist unsere Methode nur begrenzt nutzbar. Wir hoffen aber, in Zukunft realistische Anwendungen zu finden, die in diesem eingeschränkten Kontext funktionieren, oder aber Möglichkeiten, diese Begrenzung abzuschwächen.

Tonhöhenerkennung als Klassifizierung

Bisher haben wir Tonhöhenerkennung im Wesentlichen als Regressionsproblem betrachtet. Es ist allerdings auch eine Beschreibung als *Klassifizierungsproblem* möglich: Hier wählen wir im Voraus eine Menge an zulässigen Frequenzen, und das Modell muss für ein gegebenes Signal die korrekte Frequenz auswählen. Dies unterscheidet sich vom Regressionsproblem, wo der Parameter beliebig eingestellt werden kann. Tatsächlich benutzen existierende Tonhöhenerkenner auch diesen Ansatz [6].

Für unsere synthetische Aufgabe können wir das Modell wie folgt beschreiben: $\hat{y} = \sum_i c_i \sin(f_i \cdot 2\pi t)$, wobei i die gewählten Frequenzen indiziert. c_i sind lernbare Gewichte, die die Stärke der jeweiligen Frequenz im Signal bestimmten. Unser Modell ist somit eine gewichtete Summe vieler Sinussignale. Frequenzen einfacher Sinussignale können damit bestimmt werden, dass genau ein Gewicht den Wert 1 erreichen sollte, und alle anderen Gewichte den Wert 0. Aus Platzgründen stellen wir nur die Lernkurve eines solchen Modelles in Abbildung 6 dar. Tatsächlich erreicht das Modell die gewünschte Gewichtsverteilung mit dem Standard-MSSL, ohne Tricks wie Interpolation. Ferner zeigt Abbildung 7, dass der Fehler über den Parameterraum hinweg gleichmäßig abfällt, was bessere Gradienten bedeutet.

Dies wirft die Frage auf, wie Modelle zur Audiogenerierung Frequenzen intern repräsentieren. Muss die Frequenz wie mit einem Drehregler „eingestellt“ werden? Dies entspräche eher dem Regressionsmodell. Oder können verschiedene Frequenzen unabhängig voneinander herauf- und herabgesetzt werden? Dies entspräche eher dem Klassifizierungsmodell. Wie wir sehen, ist letztere Aufgabe wesentlich leichter mit gängigen Fehlerfunktionen wie MSSL zu lösen. Eine genauere Untersuchung des Lernprozesses und der Repräsentationen tiefer neuronaler Netze zur Audiogenerierung könnte somit nützlich sein.

Fazit

Wir haben eine einfache Methode vorgestellt, um den Gradientenfluss bei Fehlerfunktionen wie MSSL für Tonhöhenerkennung zu verbessern. Auch wenn unsere Methode sehr effektiv, und ferner einfach zu implementieren

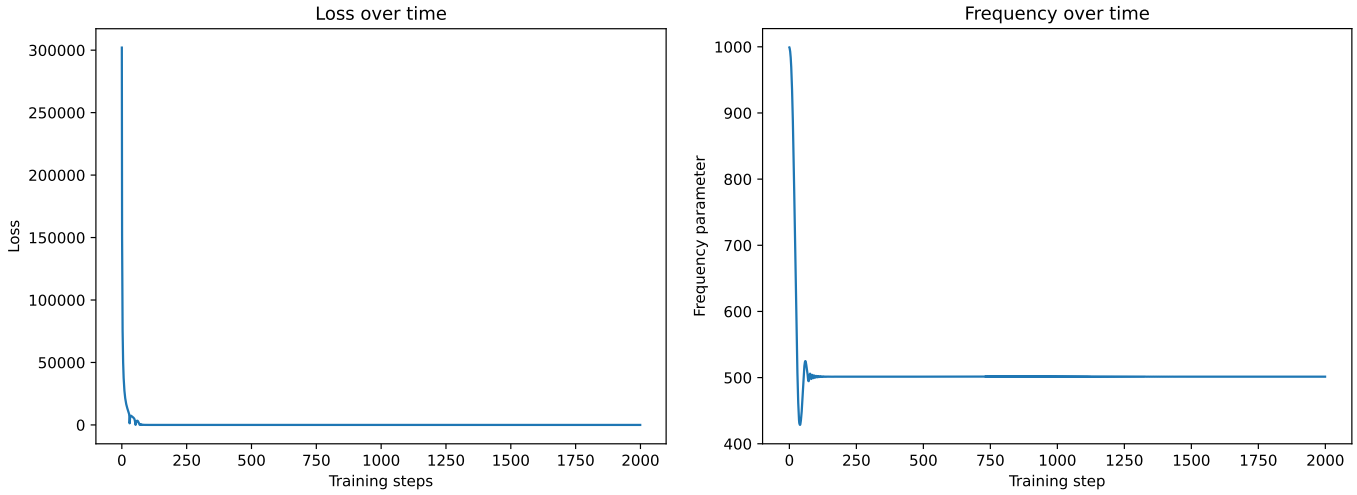

Abbildung 5: Ergebnis des Trainings mit MSSL und Interpolation. Links: Der Fehler verringert sich schnell Richtung 0. Es ist zu beachten, dass der Fehler mit $\frac{1}{1-\alpha}$ multipliziert wird, um die Herabgewichtung der Modellausgabe (Multiplikation mit $1-\alpha$) auszugleichen, weswegen die absoluten Werte anfangs recht hoch sind. Rechts: Der Modellparameter erreicht die korrekten 500 Hz.

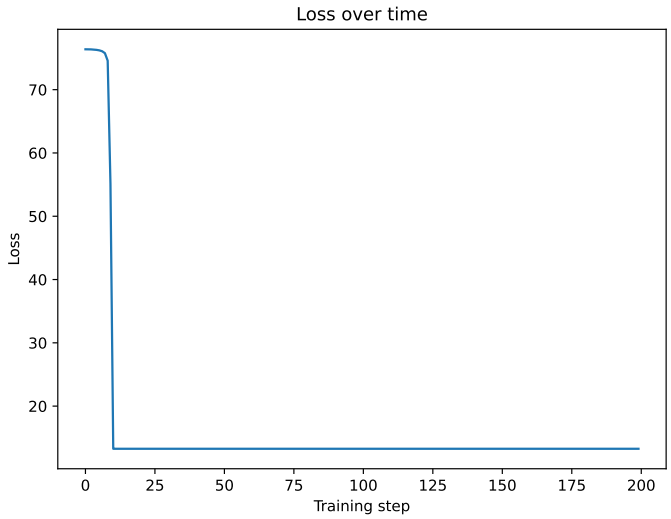

Abbildung 6: Ergebnis des Trainings mit dem alternativen Klassifizierungsmodell. Der Fehler verringert sich schnell auf einen niedrigen Wert.

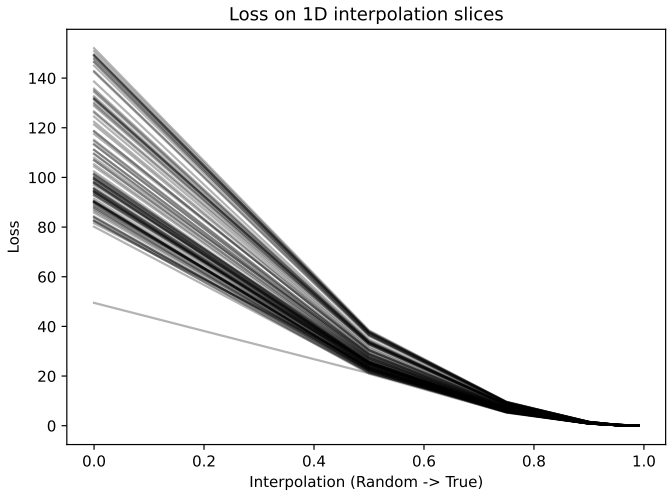

Abbildung 7: Fehler auf eindimensionalen Schnitten des Parameterraums für das Klassifizierungsmodell. Für jede Linie wählen wir eine zufällige Gewichtsverteilung und interpolieren Richtung der Zielverteilung. Somit sehen wir die Entwicklung des Fehlers auf direktem Weg zum Ziel. Für alle dargestellten Startpunkte verringert sich der Fehler gleichmäßig über die gesamte Strecke, was gute Gradienten bedeutet.

ist, so ist sie in ihrer Anwendbarkeit stark eingeschränkt. Wir hoffen, dies in Zukunft korrigieren zu können, oder aber komplexere Anwendungen zu finden, in denen diese Einschränkungen tolerierbar sind. Ferner haben wir dargestellt, dass bekannte Probleme mit der Tonhöhenerkennung lediglich auf das Regressionsmodell zutreffen, nicht aber auf Klassifizierung. Die Relevanz dieser Darstellung sollte untersucht werden, indem mehr Verständnis über die internen Vorgänge tiefer neuronaler Netze erlangt wird, in denen Tonhöhenerkennung eine Rolle spielt.

Literatur

[1] Engel, J., Gu, C., und Roberts, A.: DDSP: Differentiable Digital Signal Processing. International Conference on Learning Representations (2019).

[2] LeCun, Y., Bengio, Y., und Hinton, G.: Deep learning. nature, 521(7553) (2015), 436-444.

[3] Turian, J., und Henry, M.: I'm sorry for your loss: Spectrally-based audio distances are bad at pitch. arXiv preprint arXiv:2012.04572 (2020).

[4] Rumelhart, D. E., Hinton, G. E., und Williams, R. J.: Learning representations by back-propagating errors. nature, 323(6088) (1986), 533-536.

[5] Kingma, D. P., und Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013).

[6] Kim, J. W., Salamon, J., Li, P., and Bello, J. P.: Crepe: A convolutional representation for pitch estimation. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2018), 161-165.