Machbarkeitsuntersuchung: Komprimierung von Audiodateien durch Anwendung eines Autoencoders

Damian Lippert 563804
Denis Hirndorf 564704

14.01.2020

Einleitung

Im folgenden legen wir unsere Projektarbeit im Rahmen des Kurses ausgewählte Kapitel sozialer Web Technologien dar.
Als erstes geben wir einen Einblick in die Motivation dieses Thema zu behandeln. Danach werden das Ziel des Projektes und die benötigten Grundlagen dargestellt.
Anschließend folgt eine Erläuterung des von uns verfolgten Konzeptes und dessen Umsetzung.
Abschließend werden die durch die Projektarbeit erreichten Ergebnisse und erlangten Erkenntnisse zusammenfassend dargestellt.

Motivation

Wir leben in einer Epoche die durch Themen wie Big Data und Künstliche Intelligenz geprägt ist, und in der das Sammeln, Verarbeiten und Speichern von Daten eine immer wichtigere Rolle spielt.
Bei den exorbitanten Datenmengen die entstehen, wird die Verarbeitung immer weiter automatisiert, da die dafür erforderliche Leistung schon lange nicht mehr allein von Menschen erbracht werden kann.
In den letzten Jahren hat das schon länger bekannte Konzept der neuronalen Netze einen neuen Aufschwung erlebt, da der dafür benötigte und als langsam geltende backpropagation Algorithmus durch die enorme Leistungsverbesserung von Grafikkarten nun anwendbar ist.

Laut Statista.com sind im Jahr 2017 alleine durch private Haushalte, Universitäten und Internet Cafés monatlich durchschnittlich 100 Exabyte Datenverkehr über das Internet Protokoll (IP) entstanden.
Bei einer angenommenen jährlichen Wachstumsrate von 27% wird sich dieser Wert bis ende dieses Jahes (2020) mehr als verdoppelt haben (Statista.com, 2019).

Beim Austausch und der Verarbeitung dieser großen und stetig wachsenden Datenmengen ist die Komprimierung von Daten schon längst nicht mehr weg zu denken.
Maschinelles Lernen ermöglicht vieles was durch herkömmlichen Algorithmen nicht umsetzbar ist**,** wird jedoch im Bereich der Kompression momentan kaum angewendet.
Dieses Feld wird von Menschen geschriebenen Algorithmen dominiert. Perfekt sind diese allerdings dennoch nicht.
Google beispielsweise, erzielte bei der Komprimierung von Bildern mit Hilfe eines neuronalen Netzwerkes ein Ergebnis, dass im Vergleich zum verbreiteten JPEG Format eine vierfach so effektive Komprimierung erreichte und dabei weniger Verluste bei der Qualität zu verzeichnen hatte (Johnston and Minnen 2016).
Wenn es um Musik geht ist MP3 für viele das Standardformat, ähnlich wie JPEG bei Bildern.

Projektziel

Ziel unseres Projektes ist es ein deep neural network zur Komprimierung von Daten, um genau zu sein Audiodateien zu entwickeln. Neuronale Netzwerke zur Komprimierung beziehungsweise Kodierung von Daten werden allgemein als Autoencoder bezeichnet. Dabei soll die Datei nach dem Komprimierungs Vorgang durch die menschliche Wahrnehmung so wenig wie möglich von der Ausgangsdatei unterschieden werden können.

Jedoch werden wir die Machbarkeit einer solchen Komprimierung anhand von nur einer Sounddatei überprüfen, wodurch ein sehr Spezialisierter Kompressionsalgorithmus entsteht

Durch die Projektarbeit wollen einen tieferen Einblick in die Art und Weise wie neuronale Netzwerke Eigenschaften und Merkmale aus Daten extrahieren gewinnen und ein Verständnis dafür entwickeln dies zu kontrollieren und dahingehend zu beeinflussen, dass für die menschliche Wahrnehmung signifikante Merkmale erkannt werden und bei der Komprimierung erhalten bleiben.
Ein weiterer wichtiger Aspekt der Arbeit ist das Verständnis über die Größe des erstellten Netzwerkes und die Anzahl der lernbaren Parameter.
Der für die Parameter benötigte Speicher soll gering wie möglich sein um eine bessere effektive Komprimierung zu erzielen.
Während die Anzahl der Parameter gering gehalten werden soll, sollen die durch das Netzwerk zusammengefassten Eigenschaften so detailliert wie möglich zusammengefasst werden um eine möglichst verlustfreie Komprimierung zu erzielen.
Eine komplett verlustfreie Komprimierung mit einem Autoencoder zu erreichen ist kaum möglich und auch nicht notwendig. Viele der gängigen Komprimierungsverfahren sind auch nicht komplett verlustfrei, von Menschen allerdings bleibt dies oft unerkannt.
Unser Ziel ist es also eine Datei, genauer ein Lied derart zu komprimieren, dass eine nennenswerte Speicher Einsparung entsteht und das Ausgangs Artefakt durch Menschen nicht oder zumindest nicht unmittelbar von der Originaldatei zu unterscheiden ist.

Grundlagen

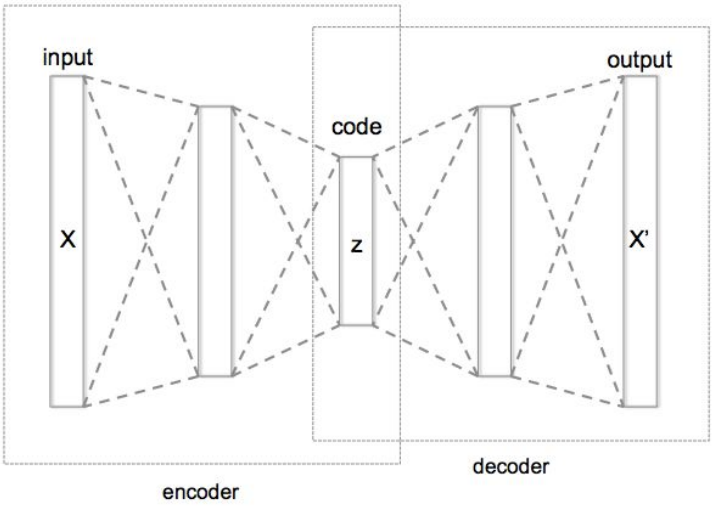
In diesem Kapitel werden werden von uns verwendete Begriffe und Verfahren erläutert. Als erstes werden wir genauer auf die Form der von uns verwendeten Daten eingehen. Darauf folgt eine allgemeine Erläuterung des Begriffes Autoencoder und der Techniken die von uns angewendet wurden um einen solchen zu erstellen.

Sounddateien

Es gibt unzählige Dateiformate, welche Sound darstellen. In dieser Arbeit wird eine Sounddatei folgend auch als Trainings- und Tondateien bezeichnet jedoch zu Folgen von Zahlen abstrahiert, welche jeweils den momentanen Ausschlag der Tonwellen repräsentieren. Für jeden der unterschiedlichen Tonkanäle, welche für Stereo oder Surround benötigt werden, gibt es eine eigene Zahlenfolge.

Autoencoder

Autoencoder sind eine Form von feed forward Netzwerken. Da Autoencoder keine beschrifteten Trainingsdaten benötigen fallen sie in die Kategorie unsupervised learning. Im allgemeinen unterscheidet man zwischen rauschunterdrückenden (= denoising) und komprimierenden Autoencodern. Während rauschunterdrückende Autoencoder die Qualität einer Aufnahme verbessern befassen sich komprimierende Autoencoder mit der Komprimierung von Dateien (Patterson and Gibson, 2017, S.106 ff). Im folgenden werden wir uns ausschließlich mit komprimierenden Autoencodern befassen.
Die Alleinstellungsmerkmale eines komprimierenden Autoencoders sind einerseits, dass die Dimension der Eingabe Schicht gleich der Dimension der Ausgabe Schicht ist. Dies ist zwingend erforderlich, da es das Ziel ist die Originaldatei nach der Komprimierung so verlustfrei wie möglich wieder herzustellen. Andererseits ist es, um eine Komprimierung zu erreichen notwendig die Dimension in den verborgenen Schichten (*hidden layern*) zu reduzieren, dadurch entsteht eine für komprimierende Autoencoder typische Sanduhr Form. Die Reduktion der Dimensionalität kann statisch durch das Modell bestimmt sein, was einem Undercomplete Autoencoder entspricht. Eine andere Variante ist der Sparse Autoencoder auf diesen wird in dieser Arbeit aber nicht weiter eingegangen.



(Abb. 1 Aufbau eines Autoencoders, Quelle: Wikipedia)

Der Aufbau eines Autoencoders lässt sich in zwei Bestandteile aufteilen, in der linken Hälfte des in der Abbildung zu sehenden Graphen ist der Encoder.
Als Eingabe X bekommt der Encoder die zu komprimierende Originaldatei. Die in der Originaldatei enthaltenen Informationen werden bis zu der Stelle des Netzwerkes mit der geringsten Dimension durchgereicht und verarbeitet. Diese Stelle wird in der Abbildung als code z bezeichnet.
Hier entsteht die komprimierte Repräsentation, die Kodierung der Eingabedatei.
Die Komprimierungsrate entspricht dem Verhältnis der Dimension der Eingabe X und der Kodierung z. Die Kodierte Darstellung z bildet die Grenze zwischen dem ersten Bestandteil, dem Encoder und dem zweiten Bestandteil, dem Decoder. Wobei z die letzte Schicht, also die Ausgabe des Encoders und die erste Schicht, also die Eingabe des Decoders bildet.
Der Decoder versucht nun alle in z gespeicherten Informationen wieder korrekt zu extrahieren und eine der Originaldatei möglichst ähnliche Rekonstruktion X’ auszugeben.
Die Güte des Modells wird durch den Rekonstruktionsfehler, also den Unterschied zwischen der Eingabedatei X und der Ausgabedatei X’ gemessen. Da es sich bei Autoencodern um eine Verlustbehaftete Komprimierung handelt ist ein Rekonstruktionsfehler von null nur in Sonderfällen möglich. Außerdem soll erwähnt werden, dass geringe Rekonstruktionsfehler, also Komprimierungen mit geringem Verlust nur bei Daten, die den Trainingsdaten ausreichend ähnlich sind möglich sind. Aufgrund dieser auf den ersten Blick nachteilhaft klingenden Eigenschaft werden Autoencoder in der Praxis oft eingesetzt um Anomalien in Datensätzen zu finden.

Convolution und Deconvolution Layer

Um die Ausarbeitung unseres Projektes zu verstehen ist es wichtig mit der Funktionsweise von Convolution- und Deconvolution Layern vertraut zu sein, da diese den Hauptbestandteil unseres Netzwerkes ausmachen.
In einem Convolution Layer wird ein Bereich, auch Filter genannt der einen Schicht mit einem Punkt der dahinter liegenden Schicht verbunden indem der Filter die einzelnen Werte des betrachteten Bereichs gewichtet und zusammen fasst. Da ein Filter auf die Gesamte Datei angewendet wird, wird der betrachtete Bereich schrittweise um einen vorher festgelegten Wert, dem Stride verschoben (Hope et al., 2017).
Auf diese Weise werden Muster erkannt und Informationen zusammengefasst, da der betrachtete Bereich eine größere Dimension als der mit ihm verbundene Punkt der nachfolgenden Schicht hat.
In einem Deconvolution Layer wird dieses Verfahren auf eine ähnliche Weise angewendet. Hier wird die Eingangsdatei mit nullen ergänzt um die Dimension zu erhöhen. Die Ergänzung erfolgt durch eine Umrandung oder durch Erzeugung eines Schachbrett Musters indem die Nullen zwischen den einzelnen Werten eingesetzt werden (Dumoulin and Visin, 2016).
Durch die Erhöhung der Dimension kann der Filter öfter auf die Datei angewandt werden und damit mehr Aktivierungen erzeugen. Auf diese Weise wird die Dimension der Ausgabe erhöht.

Umsetzung

Konzept

Im Folgende Abschnitt werden wir den Theoretischen Aufbau unseres Projektes erörtern.
Wie bereits erwähnt trainieren wir den Autoencoder mit nur einem Trainingsdatum.
Das hat zur Folge, dass das trainierte Modell nur auf Dateien, die dem Trainigsdatum ähnlich sind gut funktioniert, da im Grunde eine komprimierte Darstellung dieser Datei auswendig gelernt wird.
Außerdem muss der Speicherbedarf der für die zur Dekodierung benötigten lernbaren Parameter bei der Bestimmung der Komprimierungsrate berücksichtigt werden.
Um eine neue Datei - ohne ausreichender Ähnlichkeit - möglichst verlustfrei zu komprimieren müsste das Modell also neu trainiert werden.

Um das Modell zu Trainieren wird die Sounddatei in einem zweidimensionalen Array gespeichert, welches entlang der X-Achse die Zeit repräsentiert, entlang der Y-Achse die verschiedenen Kanäle speichert. Die Werte in diesem Array repräsentieren den Ausschlag des Tonsignals zu einem bestimmten Zeitpunkt auf dem entsprechendem Kanal. Danach wird dieses Array entlang der X-Achse in mehrere Abschnitte geteilt. Performance ist der Hauptgrund für die Segmentierung, da das Netzwerk bei großen Dateien sehr

speicherintensiv wird. Außerdem ergibt sich hieraus der Vorteil flexibler bei der verwendeten Dateigröße zu sein.
Diese Segmente sollten dabei deutlich größer sein, als das Receptive Field des letzten Deconvolution Layers, da die Aussagekraft des Modells in den Grenzbereichen zu Beginn und am Ende der Segmente nachlässt da der angewandte Filter hier durch das zero padding beeinflusst wird.

Zum Komprimieren setzen wir im Encoder des Netzwerkes ausschließlich auf zweidimensionale *convolution-layer*.
Die Verkettung mehrerer convolution-layer ermöglicht das Erkennen immer komplexerer Eigenschaften, da die Filter immer auf einen Ausschnitt der vorherigen Schicht angewandt werden und damit der betrachtete Ausschnitt der Originaldatei, das *receptive field*, größer wird desto mehr Schichten man aneinander reiht.
Eine Komprimierung wird dadurch erzielt, dass die Schrittweite mit der die Filter über die einzelnen Segmente bewegt werden größer als eins gewählt wird.
Bei der Dekomprimierung im Decoder wird das ganze Verfahren umgedreht. Hier werden sogenannte Deconvolution Layer verwendet um Eigenschaften zu extrahieren.
Nachdem die Datei Segmentweise dekodiert wurde werden die Segmente wieder zusammengeführt und die Überlappung entfernt.

Implementation

Zur Umsetzung des Projektes nutzten wir mehrere Bibliotheken. Um den Autoencoder an sich zu erstellen haben wir die Keras Bibliothek mit einem Tensorflow Backend benutzt.
Zum einlesen der Audiodateien in dem von uns verwendeten flac Format(= free lossless audio codec) kam die soundfile Bibliothek zum Einsatz.
Wie der Name bereits sagt handelt es sich hierbei um eine frei verfügbare und verlustfreie Kodierung.
Die Verarbeitung der eingelesenen Daten wurde mit Numpy durchgeführt.
Bei der Architektur des Netzwerkes haben wir uns was die Anzahl der Schichten im Encoder und Decoder angeht für einen fast Symmetrischen Aufbau entschieden.
Die Güte des Modells wurde durch die mean-absolut-error Kostenfunktion bestimmt.
Lediglich bei den verwendeten Hyper Parametern und der Art der Schichten gibt es Unterschiede zwischen dem Encoder und Decoder.
Im folgenden werden wir uns zuerst die Vor- und Nachbearbeitungs Pipeline der Daten und anschließend die detaillierte Umsetzung des Encoders und Decoders im einzelnen ansehen.

Pre- und Postprocessing der Daten

Zu Beginn der Vorverarbeitung wird das Trainingsdatum als Ganzes normalisiert.
Dieser Schritt ist nicht zwingend erforderlich aber sehr empfehlenswert, da Grafikkarten, die die Berechnungen durchführen besser mit Gleitkommazahlen arbeiten.

Danach wird die Datei mit nullen am Ende aufgefüllt, dass die Dateigröße einem vielfachen der Kompressionsstärke entspricht. Dadurch bleibt bei der Kompression die Dimensionalität von Input und Output gleich.

Anschließend wird die Datei in sich überlappende Segmente aufgeteilt
Dadurch vergrößert sich die kodierte Datei um rund 25%.
Die Überlappung ist erforderlich um die Segmente am Ende wieder fehlerfrei zusammenführen zu können. Dabei ist zu beachten, dass die benötigte Überlappung von dem durch die Filtergröße und Anzahl der Schichten entstehendem receptive field abhängt.
Erst nachdem die einzelnen Segmente durch den Encoder komprimiert und durch den Decoder wieder dekomprimiert wurden werden die Segmente wieder zusammengeführt und die durch die Überlappung entstandenen redundanten Informationen entfernt.
Nach dem Zusammenführen wird die anfänglich stattgefundene Normalisierung wieder rückgängig gemacht.

Encoder

Der Encoder ist in der Lage eine Datei mit einem Faktor, der einer Zweierpotenz entspricht zu komprimieren. Dieser Faktor bildet sich aus der Formel 2^n.
Die erste Schicht des Encoders besitzt keine Aktivierungsfunktion und dient lediglich dazu die eingegebenen Werte auf verschiedene Kanäle zu verteilen.
Darauf folgen n / 2 Schichten aus Convolutional Layern in denen der Filter mit einem Stride von vier über die Zeitachse bewegt wird. Ist n ungerade wird dieser Wert abgerundet.
Abschließend kommt noch ein weiterer Convolutional Layer mit einem Stride von vier falls n gerade und von acht falls n ungerade ist und ein Layer ohne Aktivierungsfunktion um die Ausgabe auf vier Kanäle zu bringen.
Die hier entstehende dreidimensionale Datei wird auf der X-Achse auf $1 / 2^{(n+2)}$ der ursprünglichen Länge reduziert, die Dimension der Y-Achse bleibt unverändert bei zwei und die Dimension der Z-Achse beträgt wie bereits erwähnt nach Anwendung der letzten Schicht vier.
Als Lernrate beziehungsweise als Optimizer haben wir hier adam und als Aktivierungsfunktion selu gewählt, da hiermit die besten Ergebnisse erzielt wurden.

Decoder

Die Eingabe Schicht des Decoders entspricht der letzten Schicht des Encoders.
Der Decoder unterscheidet sich durch die Art der Schichten und die verwendeten Hyperparameter vom Encoder.
Der Aufbau des Netzwerkes ist nur fast Symmetrisch, da die letzte Schicht des Encoders die verwendet wurde um die Dimension der Ausgabe zu ändern hier nicht gespiegelt wird.
Außerdem wird statt den Convolutional Layern, die im Encoder genutzt wurden um Muster zu erkennen und Informationen zusammen zu fassen hier das Gegenstück, so genannte Deconvolution Layer eingesetzt.
Die Anzahl der lernbaren Parameter des Decoders ist wie bereits erwähnt für die Berechnung der effektiven komprimierung erforderlich

Bei einem Komprimierungsfaktor von vier kommt der Autoencoder mit seinen acht Schichten bei jeweils 16 Kanälen mit einer Filtergröße von 64x2 kommt der Autoencoder auf insgesamt 118.869 lernbare Parameter. Davon 43.041 im Decoder, wodurch der Decoder eine theoretische Speichergröße von 43.041 * 4 Byte = 172.164 Byte ≙ 172kB erreicht. Diese größe ist für längere Sound Dateien, wie Musik jedoch vernachlässigbar.

Training

Beim Training entstand das Problem, dass zu große Batch Sizes sehr viel Arbeitsspeicher brauchen. Darüber wie die Batchsize die Generalisierung beeinflusst müssen wie uns keine Gedanken machen, da wir nur ein Trainingsdatum benutzen und dem Netzwerk beibringen wollen eine komprimierte Darstellung von diesem so gut wie möglich auswendig zu lernen.
Insgesamt haben wir das Netzwerk über 64 Epochen trainiert, da ab hier kaum noch Verbesserungen zu verzeichnen sind. 16 Epochen davon werden auf recht kleine Segmente ausgeführt, da das Netzwerk hier schneller lernt. Danach wird auf größere Segmente trainiert um das Trainieren der Grenzbereiche der Segmente zu verhindern.
Da wir Generalisierung aber wie bereits erwähnt vernachlässigen können sind auch mehr Epochen möglich, das Ergebnis verbessert sich dadurch allerdings nur unwesentlich.
Insgesamt haben wir das Lied ungefähr 256 mal komplett durch das Netzwerk laufen lassen um es zu trainieren.

Evaluation

Das von uns erzielte Ergebnis aussagekräftig zu beziffern ist keine triviale Aufgabe. Den numerischen Unterschied zwischen der Originaldatei und dem Ausgangsartefakt zu bestimmen ist kein Problem, allerdings stimmt der auf diese Weise berechnete Rekonstruktionsfehler oft nicht mit der menschliche Wahrnehmung überein.
Unser Modell erreichte einen durchschnittlichen numerischen Unterschied von ungefähr 0.0300 gegenüber der Originaldatei.
Der für die menschliche Wahrnehmung entstehende Fehler ist schwer zu beziffern aber derart beschreibbar, dass die Hauptbestandteile der Datei, beispielsweise Melodien korrekt wiederhergestellt werden und es nur zu Abweichungen in den Details kommt. Um Musik zu komprimieren ist unser Ansatz deswegen ungeeignet, da bereits geringe Abweichungen die Tonlagen verändern, Stimmen unkenntlich machen und Störgeräusche erzeugen können.
Die Ursache dafür liegt an den komplexen Eigenschaften von Sounddateien. Der von uns verfolgte Ansatz diese mit einem hauptsächlich in der Bildverarbeitung vorkommenden Convolutional Neural Network zu erfassen gelang nicht ausreichend.

Quellen

Statistica, Feb. 2019, Global data volume of consumer IP traffic 2022, https://www.statista.com/statistics/267202/global-data-volume-of-consumer-ip-traffic/ (letzter Aufruf 10.01.2020)

Nick Johnston, David Minnen, 29.09.2016, Image Compression with Neural Networks, Google AI Blog, https://ai.googleblog.com/2016/09/image-compression-with-neural-networks.html (letzter Aufruf 10.01.2020)

Tom Hope, Yehezkel S. Resheff, Itay Lieder, 2017, Learning TensorFlow First Edition, O'Reilly Media Inc

Josh Patterson, Adam Gibson, 2017, Deep Learning A Practitioner's Approach First Edition, O'Reilly Media Inc

Vincent Dumoulin, Francesco Visin, 2016, A guide to convolution arithmetic for deeplearning, https://arxiv.org/pdf/1603.07285v1.pdf (letzter Aufruf 10.01.2020)

Abbildung1: https://en.wikipedia.org/wiki/File:Autoencoder_structure.png