Babes-Bolyiai University Cluj-Napoca
Faculty of Mathematics and Computer Science
Specialization Computer Science

Diploma Thesis

Audio Separation using Deep Learning

Titus-Traian Trifon

supervised by
Tudor Mihoc

June 23, 2019

Contents

Chapter 1

Introduction

1.1 Background and Motivation

For the last two decades, machine learning and deep learning in particular has greatly improved and developed. It became relevant in almost all fields of study, including audio recognition. As deep learning evolved and new concepts such as Long Short-Term Memory (LSTM) were being theorized, researchers began to move the audio source/signal separation problem from a signal processing problem to a deep learning one.

There are a lot of reasons for implementing a software that is capable of separating audio sources. Needs such as post production of raw recording, automatic karaoke song maker, automatic soloist, and other come to mind. Music signals are mostly a mixture of several sources active simultaneously (voice, musical instrument or synthetic sounds) thus the previously mentioned needs could be solved by unmixing these sources and labeling them. We will next try to come up with a convenient solution using state of the art deep learning and signal processing techniques.

1.2 Problem Statement

Music is a complex thing. Trying to distinguish the sound of an instrument in a song is a task that only humans can do it successfully. Finding patterns of the sound the instrument makes doesn't seem possible using basic signal processing. Using AI to find such patterns might however work, if a good model and enough training data is provided.

My main goal is to design a software that is able to extract audio coming from a specified musical source (instrument or voice). Such software has been previously developed. Unfortunately, these either have poor accuracy or are only capable of extracting vocals. By taking advantage of the latest discoveries in deep learning, we will try to design and train a deep network that can achieve prefessional separation of music sources from a (monaural) recording. The model will be described below, along with other techniques for processing and filtering the audio.

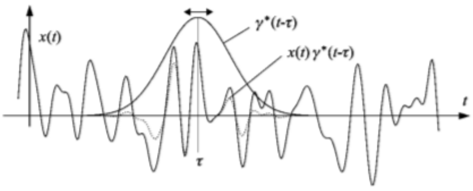
Chapter 2

Theory

2.1 Signal Processing

2.1.1 Short-Time Fourier Transform

The Short-Time Fourier Transform is the most used method of time-frequency analysis. The main concept relies on multiplying $x(t)$ with an analysis window $y * (t - r)$ and then compute the Fourier Transform of this windowed signal.



STFT formula:

$$F_x^\gamma(\tau,\omega) = \int_{-\infty}^{\infty} x(t)\gamma * (t-\tau)e^{-j\omega t}dt.$$

In a digital environment we will use the discrete formula instead:

$$F_x^\gamma(m,e^{j\omega}) = \sum_n x(n)\gamma * (n - mNe^{-j\omega n}).$$

The window $y * (t - r)$ suppresses $x(t)$ outside a certain region and the FT outputs a local spectrum. The FT is complex valued, so we use a spectrogram to display it. This also helps us for further processing the signal. In our application we will use the amplitude spectrum as input for our neural network. The spectrogram is created by computing the squared magnitude of the STFT:

$$S_x(\tau,\omega) = |F_x^\gamma(\tau,\omega)|^2 = |\int_{-\infty}^{\infty} x(t)\gamma * (t-\tau)e^{-j\omega t}dt|^2.$$

After we are done filtering the spectrum, we will need to reconstruct the signal to get an audio back. A reconstruction of x(t) is possible and is done so easily by the Inverse Short-Time Fourier Transform (ISTFT) formula:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F_x^{\gamma}(\tau, \omega) g(t - \tau) e^{j\omega t} d\tau d\omega.$$

We will implement the equivalent discrete formula to reconstruct our signal.

2.1.2 Spectrogram

A spectrogram is the visual representation of the frequency spectrum for a given signal. Spectrograms show energy or dB (by color) as a function of time and frequency. To form a spectrogram the sampled data must be broken into overlapping chunks and transformed by the Fourier transform into the magnitude of the frequency spectrum. Each chunk is then arranged one after another to better see the evolution in time of the signal. This process can be replaced by computing the squared magnitude of the STFT.

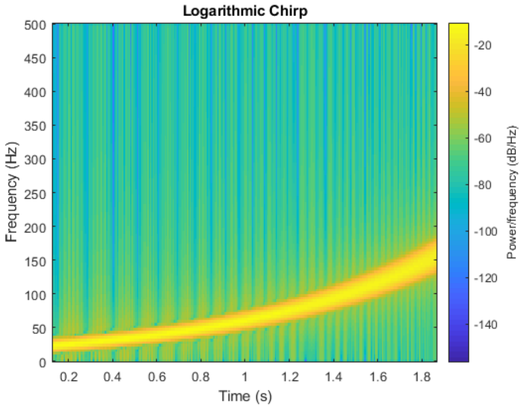


Figure 2.1: Spectrogram

Working with frequencies proves to be easier than working with the waveform directly. A spectrogram can be edited and turned back into the corresponding waveform by a reverse process using ISTFT.

2.2 Deep Learning

2.2.1 Generalization

In the early days of AI, the field solved mostly problems that are difficult for humans but relatively easy for computers, problems that can be described easily using mathematical rules. However, problems that we solve intuitively, such as recognizing objects or spoken words, were the real challenge of AI.

Deep learning is an answer to problems such as these, by making the computer learn from experience and creating a hierarchy of concepts in increasing difficulty of understanding. To understand a concept, the computer must understand a lot of simpler concepts first. The number of layers create a “deep” graph, hence one of the reasons for the name. Deep learning can scan large data sets and discover intricate structures by using the backpropagation algorithm. This method uses a so called “Artificial Neural Network” about which we will talk below. Mostly used is the convolutional neural network architecture. It is best known for the results in computer vision. Although convolutional neural network can be used to solve our problem, we chose the more suitable recurrent neural network as the architecture for our deep learning algorithm, seeing as our data can be viewed as sequential.

Deep learning greatly improved the state-of-the-art in object detection and speech processing. Deep recurrent networks brought breakthroughs in processing sequential data, while deep convolutional networks greatly improved the processing of images, video and even audio.

2.2.2 Supervised Learning

Supervised learning is the most common form of machine learning. The goal of supervised learning is to find a mapping from x to y, given a set of pairs (xi, yi) named training set. Here, x will be the input(example) and y the output (label or target). Next, we will consider a simple problem that will help understand supervised learning better. Imagine we want to classify images based on what object they contain: a car, a dog or a house. A training data consisting of pairs of pictures and the correct label must be given to our supervised learning algorithm. Given a picture as input, the machine will give output a vector of scores, corresponding to the probability of the image belonging to each category. Because we actually know the correct answer, we can check how good the algorithm is by computing the error or “loss”. This measures how far the algorithm was from the right answer (label). The machine modifies some internal adjustable parameters to reduce this error. This is often referred to as backpropagation and the adjustable parameters as weights. Stochastic gradient descent is the most common procedure for minimizing the loss. For a few inputs, the weights and the error are computed, along with the average gradient. The weights are adjusted according to this average gradient. The process is repeated multiple times for many small sets until a minimum average loss is obtained. This method can lead to finding a local minimum loss and staying there, without ever reaching the global minimum we desire.

We are interested in the learning algorithms based on Neural Networks (Multiplayer Perceptron).

2.2.3 Artificial Neural Network

An Artificial Neural Network (ANN) can be described as a weighted directed graph in which the nodes are artificial neurons and the edges are the connections between the neuron outputs and the neuron inputs. They have been inspired by the biological central nervous system, even if the correspondence between the two is fairly weak.

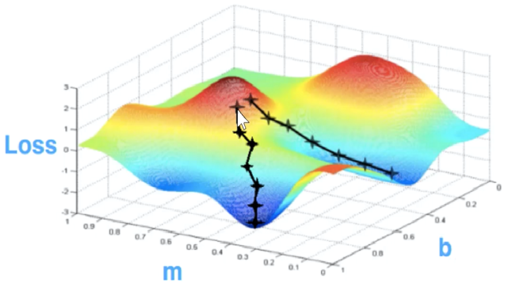


Figure 2.2: Representation of the case when a local minimum is obtained using SGD

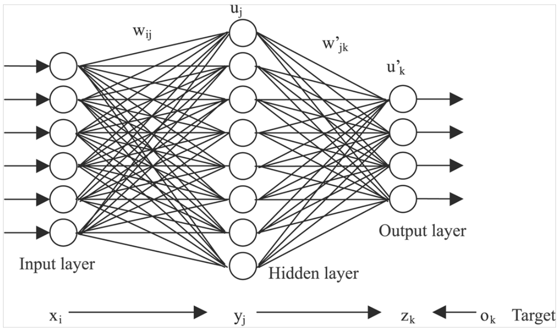


Figure 2.3: Abstract representaion of a feed-forward ANN

The first model of an artificial neuron was proposed by Walter Pitts and MuCulloch in 1943. This neuron computes a weighted sum of its inputs and generates an output of 1 if the sum is above a threshold, or 0 otherwise.

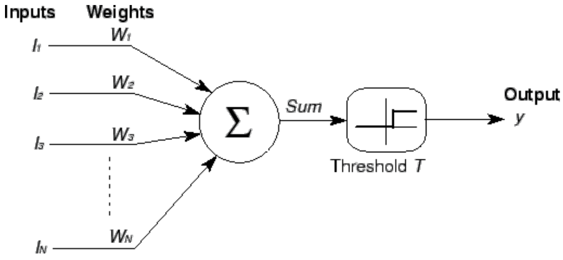


Figure 2.4: The MuCulloch-Pitts neuron

This kind of neuron has been further generalized. Instead of a threshold, an activation function is used, such as Gaussian, sigmoid and piecewise linear. We chose the rectifier as our activation function for its simplicity and its better gradient propagation: $F(x) = x^+ = max(0, x)$

Based on the architecture, we can group ANNs into two: feed-forward networks, where the information flows in one direction, and recurrent networks, loops occur so we can better process sequential data. We will be using the latter to implement our algorithm.

2.2.4 Recurrent Neural Network

As mentioned before, Recurrent Neural Networks (RNNs) are a class of dynamic models, used when a sequence of data needs be processed and generated. They were first proposed in the 80's for modelling time series. They compute a sequence one element at a time, while maintaining a "history" of all past elements. We could also view the structure of the network similar to that of a deep multi-layer feed-forward network, with the outputs of the hidden units at previous steps as inputs of different neurons at the current step. This visualization makes it easy to see how backpropagation can be implemented in a regular RNN.

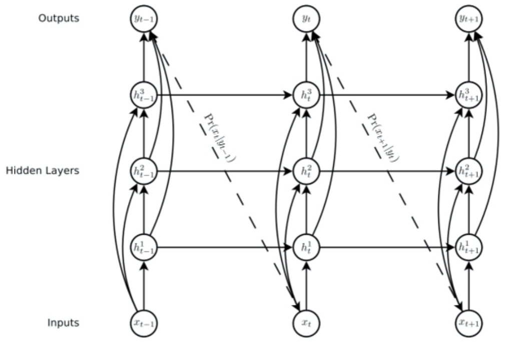


Figure 2.5: Deep RNN (prediction) architecture

Training RNNs proved to be a difficult task, as the backpropagated gradients tend to get either too big, or too small from a step to another, making some data be irrelevant or quite to opposite. These problems are referred to as the vanishing gradient and the exploding gradient, the first one being the more encountered of the two.

RNNs are being used to generate sequences in various domains such as music, text processing and even motion capture data. This dynamic model looks like a promising approach to our audio separation task, giving that we take care of the vanishing gradient.

2.2.5 Vanishing gradient

RNNs need to learn which past inputs have to be stored and how important they are to compute the current desired output. Because of this, backpropagation suffers from a too long learning time, as the time lag between inputs and relevant stored signals are extended. Error signals propagating back in time tend to vanish, making long-term dependencies hard to achieve or almost impossible to train. We will look over a few methods to prevent the problem of vanishing gradients.

Introducing time constants could help with long time lags, by modifying the unit activations. Related to this, a time-delay network called NARX networks were designed and proposed to take care of the vanishing gradient problem. The solution lies mostly in implementing "shortcuts" in the error backpropagation. This does not solve the root problem, but is an improvement to the regular design of a RNN.

A naive solution is searching without gradients. Instead of a continuous optimization that guides the weight search, various other search methods are used. The simplest example is by initializing randomly, until acceptable outputs to the available inputs are found. Weight guessing is not a good algorithm, but with certain optimizations it can lead to a realistic solution to certain easy tasks.

Currently, the most efficient solution to the vanishing gradient, is “Long Short-Term Memory” (LSTM), a gradient based method that truncates the gradient when it doesn’t harm. Further, we will explain this method in detail, as it will be used in the architecture of our RNN.

2.2.6 Long Short-Term Memory

Long Short-Term Memory has proved to be a more efficient, easier to train architecture than conventional RNNs. First theorized in 1997 by Hochreiter and Schmidhuber, it drew a lot of attention in the deep learning community, and obtained good results in tasks such as music composition or speech recognition. LSTM derives from the simple RNN architecture, the core of the first being the LSTM cell, a memory cell that is better at finding and exploiting long term dependencies. An LSTM cell is a structure made of different gates that control how the information flows to hidden neurons.

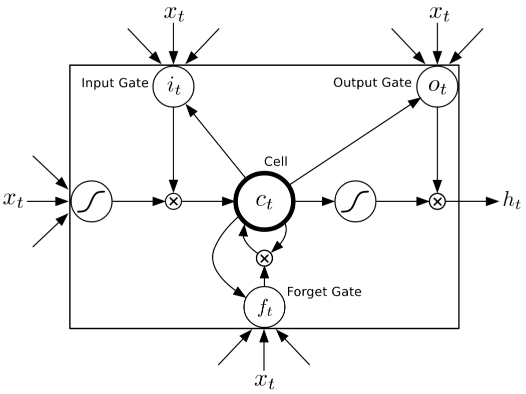


Figure 2.6: LSTM Cell

Figure 2.5 portrays a basic LSTM Cell, where i is the input gate, o is the output gate, f is the forget gate and c is the cell input activation vector. All these are the same size as the vector h. The below functions describe hoe information flows through these gates/actiavtion vectors.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$

$$f_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$f_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)$$

$$h_t = o_t tanh(c_t)$$

2.2.7 Overfitting

Overfitting is a problem that occurs in our model, when the performance of the network improves for the training data but the network keeps performing worse for “unseen”, evaluation data. This may happen from multiple reasons, but most frequently by including more variable parameters than necessary, or by simply not feeding the network with enough training data. Basic feedforward neural network regularizations do not work with RNNs.

We are going to talk about 2 techniques that will help reduce overfitting of our data. The more intuitive solution is to multiply our training data. This is actually possible through data augmentation. To augment our dataset, we need to find a way to modify our input while keeping the expected output the same. This helps the neural network to better find the important (relevant) features. Thinking back to our problem, one way to augment our data is to add some background noise to each song and keep the same output. The training dataset will double in size and overfitting will most certainly reduce.

In regular deep networks, dropout is the most common technique for reducing overfitting, where random network neurons are masked during training. Unfortunately, we cannot apply this to our LSTM RNN. A similar weight-dropping technique might work. This requires setting random set of weights to 0 from the hidden to hidden weight matrices. This solution would not impact the training speed and it should work on our chosen neural network architecture. A more technical review and experiments using this technique can be found in [13] .

2.2.8 Optimization

The success of the Neural Network resides mostly in optimizing and fine tuning the model for your specific task. Let’s further talk about some factors that influence our model’s performance.

First, just having a training dataset is not enough. The data must be correct and diverse so that the model can distinguish important features. A correlation between the quantity of data and the size of the neural network is also important to prevent overfitting and underfitting. To help the model find features in our data, we can use different preprocess techniques. In our case, instead of feeding the waveform into the network, we will first transform it using STFT and only give the magnitude spectra as input.

Second, fine tuning parameters is a process that takes a lot of time but is still as important. Things like learning rate, layer size or batch size have a huge impact on the output. For example, increasing the size of the training batch helps decrease the time for the gradient to converge. These parameters are usually tuned by successive experiments and evaluations.

Last, we need to make sure we are using the optimal functions. Using a good loss function for computing our error and backpropagating it can help the learning curve significantly. Activation functions need to be chosen carefully according to our desired behavior for that specific layer. After the model is created, optimizing should be an ongoing process.

Chapter 3

Related Work

3.1 Source Separation

Source separation in a digital audio is the isolation or extraction of a signal component from a mixture of signal components. Musical recordings are being recorded with several microphones, one for each source. These sources are then mixed together to form the song (final mixture). We can consider this final mixture as a sum of all sources, with the mixing filters applied to them. To represent this in a mathematical way:

$$x_i(t) = \sum_{j=1}^{J} \sum_{\tau=-\infty}^{\infty} a_{ij}(t-\tau,\tau)s_j(t-\tau)$$

$x_i(t)$ – final mixture,
s_j - source,
a_{ij} – filter,
J – number of sources.

If the filters are linear, the final mix will also be linear. If, however, we add other mixing effects to the song, it will no longer be a linear sum of those sources. Keeping this in mind, we can categorize the mixture by the way it was recorded as follows:

a) Under-determined/Over-determined based on the microphones that were used. Using a microphone to catch all sound sources will result in an under-determined mixture, while using a microphone for each source will get you an over-determined mixture.

b) Time-varying/Time-invariant depending on the static aspect of the song.

c) Convoluted/Instantaneous depends on the post-production effects

Source separation algorithms can also be categorized into two:

a) Informed source separation takes advantage of additional data regarding the signals. An example is a MIDI representation of the music.

b) Blind source separation is about extracting components with no extra information about them. It can be done by taking advantage of the statistical independence of each source signal. In this category the deep learning algorithms are explored and improved. We can include here the work of Lopez P.: Blind

3.2 Vocal Isolation

Vocal isolation is a useful tool to get a clean vocal signal, which can furthermore be used for tasks such as singer identification or lyrics transcription. Various methods and algorithms have been created to try and develop a vocal isolation software. Successful results have been achieved with Bayesian methods, and non-negative matrix factorization. Recently, more powerful alternatives have emerged from the field of machine learning, more specifically deep learning.

According to [11] the decomposition of a (music) audio signal into its vocal and background track is analogous to image-to image translation, where a mixed spectrogram is transformed into its constituent sources. Convolutional encoders and decoders have been researched because of this. They rely on spectrograms to be compressed, analyzed, filtered and decompressed. This technique has a major weakness regarding the amount of training data required. A sufficiently large dataset is not publicly available.

Some breakthroughs were made when U-net architecture was used. Because in the visual field, a few pixels don't carry as important information as a pixel in a spectrogram does, it is crucial that the reproduction preserves a high level of detail. More details on the algorithms can be found in [11].

Chapter 4

Methodology

4.1 Approach

4.1.1 Considered Approaches

A first approach to blind signal source separation is using what is known as Independent Component Analysis (ICA). This assumes each source signal is statistically independent, also known as the nonGaussianity of the source. ICA exploits this property and uses different finite impulse response (FIR) filters to separate the signals. This method seems ideal for separating noise, but proved to be impractical for separating sound recorded in a real acoustic room environment.

A Deep Learning approach using the latest advances in Convolutional Neural Networks (CNNs) is also possible. The main idea is modelling a CNN to compute a time-frequency soft mask which we apply to the initial signal. Such a CNN is theorized in [5] . The network has 2 phases: an encoding or convolutional one and a decoding or deconvolution one. The encoding consists of 3 convolution layers: a vertical one (that can recognize things such as timbre), a horizontal one (to learn temporal pattern and evolutions) and a fully connected one that can learn to classify different features. The decoding stage is all about reversing the previous convolutions. Such a algorithm was implemented and submitted to the MIREX2016 where it achieved one of the best results. Although a solid approach to our blind audio separation problem, the seeming difficulty of implementing the algorithm makes it our second choice.

4.1.2 Selected Approach

The chosen approach is similar to the previously discussed, but instead of using the CNN model, we went with the LSTM RNN architecture, which we already found to be superior for processing sequential data and keep long time-lag dependencies. The network will be given as input the frequency spectrum, instead of the raw waveform. The RNN will ideally find patterns for recognizing our desired source using the LSTM cells. It will next filter the signal through a time-frequency masking layer and give as output two waveforms that combined should give the initial signal (audio).

To create an interface for an everyday user, two Graphical User Interfaces will be created: first for the mobile environment using Flutter and the other as a web application using React. Our algorithm will be stationed on a server machine. A RESTful service will be created using Flask so that the GUIs can interact with the server.

4.2 Technology

4.2.1 Python

Python is currently the most used programming language in machine learning. Python has an expressive syntax, that makes code easy to understand and maintain. Considering it is a high-level language, Python can still compare in speed with some of low-level programming languages. However, what really sells it to be used as a machine learning language is the great number of open source libraries and frameworks available. Libraries like NumPy can speed up calculations, and frameworks such as TensorFlow can help develop and train machine learning models, while keeping the clean syntax and structure of the program.

I personally chose Python because I was already familiar with it, and knew there is a lot of support for it online. For this project it seemed like the perfect fit, with existing libraries for audio files processing, modelling a neural network and learning it, and even developing a RESTful service.

4.2.2 Tensorflow

TensorFlow is an open source framework, used specifically for designing ML models and doing numerical computations inside them. TensorFlow received a lot of attention ever since its release, mostly because of its extensibility and hardware utilization. Its performance was not competitive with other ML libraries, but it has improved since considerably. The parts of TensorFlow that we are interested in are the TensorFlow library for Python and the TensorBoard toolkit for data visualization.

For our software, TensorFlow can help us create the model of the network, by simply specifying each layer type, activation function, size and every other parameter we need. We can add multiple layers of LSTMs in just two lines of code. After we have created our model, we can use this library to help us with training the network. We can easily arrange the data into batches, compute the loss and backpropagate it. TensorFlow will help us develop a well-structured program, with improved performance because of the efficient functions it provides.

TensorBoard is a toolkit for visualizing our data. During the execution of the program, information on the model, dataset, training and evaluation are logged into files. TensorBoard can next interpret these files and provide us with an elegant web interface to visualize this data. For example, we can see if the model was created as intended, or if the training is going as planned.

4.2.3 Flask

Flask is a small framework for Python that does not require any additional library or tool. The framework contains development server and debugger and RESTful request dispatching. We are mainly interested in these two features as our main goal is to create a simple RESTful service that can communicate with our neural network. Flask is well known for its simplicity and lightweight design. This and the fact that it is a Python framework persuade me to use it for the server communication.

4.2.4 UI Frameworks

Flutter

Flutter is a cross-platform framework released by Google in 2016. The framework was created to help developers make high-performance mobile applications that can run both on Android and iOS. Fluter applications are written in Dart, a programming language also implemented by Google, which has some similarities to JavaScript. Flutter has a rendering engine that renders every view component, and can compile an application with the same high-performance as that of a native application.

Flutter was chosen for implementing our mobile GUI because of its consistency and tidiness in syntax along with the fact that the application can run on both Android and iOS. The design of the mobile application will be simple consisting of a few views and buttons. The library http form dart will help us communicate with the server through http calls.

React

React is a JavaScript Library that makes creating interactive UIs a lot easier. Features like virtual DOMs and stateful component make sure that only the components we change get rendered again. The only real drawback of using React is the high memory (RAM) requirement. In our case, this will be practically inexistent, as our application will be a simple, low resource one.

React is perfect for building a single page web application, which is what we are going to do. The design of our web application will be similar to the mobile design. A few views so we can choose what service we require and a upload screen along with a audio player or a download button will probably get implemented.

4.2.5 Other Third-party Libraries

NumPy comes from Numerical Python. It is considered the fundamental Python package for scientific computing. NumPy provides lots of tools for multidimensional arrays and functions to perform complex calculation. The speed drawback of python being a high-level language is mostly resolved by this package. NumPy will prove its usefulness when we are working with arrays such as the waveforms or spectrograms. Other mathematical functions will be used during the implementation. The speed difference gained by using this library will be visible especially during the training of the neural network. Training such a model is usually a process that takes a lot of time so any performance gain is very helpful.

Librosa is another python package, this one specializing in audio analysis. It is usually used to develop Music Information Retrieval (MIR) systems. Librosa will provide us with simple methods for reading and writing audio files in formats such as wav and mp3. This package also comes with a great implementation of the short-time Fourier transform. We will use it to obtain the spectrograms of the input signals. Functions to reverse these processes are also available in this package. Overall, Librosa is a great tool that will reduce the implementation time and our application performance significantly.

4.3 Data

4.3.1 Selection

Multiple datasets related to audio source separation are published and available on the world, though few of them fit the requirements for our specific need. Finding professional recordings of songs, along with independent recordings for each musical instrument proved to be harder than expected task.

The dataset chosen to work with is INRA:DSD100 a collection of 100 songs. For each song the mixture audio is provided, and the composing sources audio labeled as: vocals, bass, drums, other. The songs are professionally recorded with each source being totally isolated. The only problem that this dataset presents is the low number of samples. To work around this, we can augment the data like we mentioned previously. This can be done by adding some noise to each mixture, or by combining two songs together, resulting in a totally different mixture. For example we could use the waveform of the vocals in one song and combine it with the drums and bass from another song.

4.3.2 Preprocessing

An experiment where we will only use mixtures of two sources will eb conducted. This requires forming another mixture for each song using only the two recording (for example only combining the vocals and the drums). This will require reading the two wav files and summing the waveforms. The result will be saved in another wav file. Using the methods provided by Librosa this will be a relatively easy task.

Preprocessing of the training and evaluation data consists of retrieving the waveform from the wav file, and using the STFT formula to obtain the phase and the magnitude spectra. We will only use the magnitude spectra for further computations. The phase spectra will remain in memory as we need it to reconstruct the altered waveform.

After our neural network is trained, our software will receive as input an mp3 format audio. This will need to be converted to the corresponding waveform which will pass through the filters mentioned above. In the evaluation phase, some additional processing might occur in order to calculate the neural networks performance.

Chapter 5

Implementation

5.1 Preprocessing

5.1.1 Audio Decoding

..

5.1.2 Data Preparation

The DSD100 dataset must be acquired and added to the project. The dataset consists of 100 full lengths music tracks, and their isolated bass, vocals, drums and accompaniment, all saved under the `wav` file format. 50 songs are found in the `dev` folder, meaning there are meant for training, and another 50 are in the `test` folder, which is meant to use for evaluating. We are actually going to use both for training our model. This will improve it, but will make the tuning phase a little more difficult. To ensure the mixture is a perfect combination of the 2 sources, the mixture will be computed in our code, after the two different wavs are opened. A simple arithmetic mean of the two waveforms will do the trick.

The wavs need to be further translated to the equivalent spectrograms. This will be done using the `stft` method provided by Librosa. We will be more interested in the magnitude spectra. The phase spectra are not needed for the training phase. They will be used to reconstruct the audios after they are separated, but have no impact on the training of the network. Just another method `spec_to_batch` is required in order to transform the spectra to a batch that can be passed to our model. This method just reshapes and adds padding where needed.

5.2 Model

5.2.1 Architecture

Here we have an overview of the overall architecture for our neural network. As mentioned before, we have decided to use the LSTM variation of the RNN. The network will consist of an input layer, 3 hidden layers with LSTM cells, 2 fully connected (dense) layers and another 2 layers representing the time-frequency masking layer. We will discuss these layers in detail below. The loss function defined by: $loss = mean((y_{1p}-y_1)^2 + (y_{2p}-y_2)^2)$, where: y_{1p} – predicted source 1 magnitude tensor, y_1 – source 1 magnitude tensor, y_{2p} – predicted source 2 magnitude tensor, y_2 – source 2 magnitude tensor

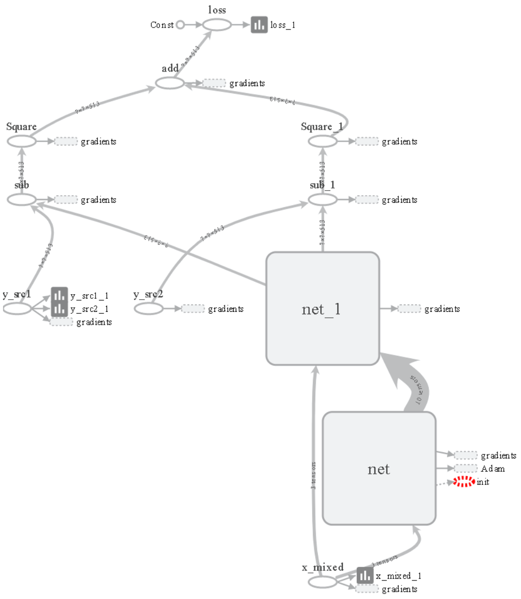


Figure 5.1: visualization of the data flow in TensorBoard

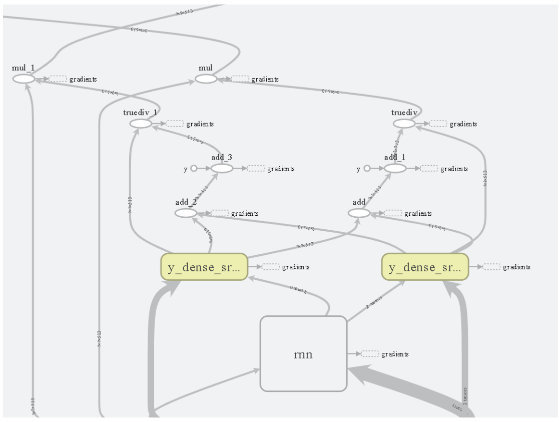


Figure 5.2: LSTM, dense and masking layer viewed in TensorBoard

5.2.2 Configuration

Here we will have an overview over the configurable part, and what it represents. Looking at the more important hyperparameters we have: sample rate for the audio files, frame length of the spectrograms, learning rate, checkpoint step and number of seconds of the training audios (batch size). The sample rate is

44100, as the songs are also encoded at this value of Hertz. Frame length of the spectrograms is 1024, the default one using the Librosa package. Learning rate is not fixed. We might start with a 0.001 learning rate and decrease it at about 80% training, so the model can better tune the features found. Checkpoints should be done as frequently as possible to make sure we aren't losing progress. This parameter is dependent on the number of training iterations. The batch size is 8.192 seconds. I chose this exact number so we can draw a Mel spectrogram of size 512:512 from the batch if we ever want to visualize the data.

Other objects like session configuration or data path are available in a configuration file. We will only train the model to recognize drums and vocals from a mixture of the two. This should be a good start giving the limited time and data available.

5.2.3 Layers

Input Layer

Input layer is composed by a tensor, containing floats, and representing a frame from the magnitude spectrum of the mixture. We create a placeholder for the tensor, using `tf.placeholder` and call it `x_mixed`. We specify one of the dimensions' size to be equal to half the length of a frame from our spectrogram. This layer is connected to the LSTM layer through edges with trainable weights.

LSTM Cells

A configurable number (default 3) of LSTM layers defined. To do this, an LSTM cell must be created for each layer. The size of those cells is hardcoded to 256.To create the LSTM cells we call `rnn_cell.LSTM` and provide the mentioned size and to define the layers we use the method `rnn_cell.MultiRNNCell` The RNN is created by calling `tf.nn.dynamic_rnn`. The input layer will act as input and the LSTM layers defined will act as the core of this network. The output of this network will pass on to two fully connected layers.

Dense Layer

Two fully connected layers (one for each source) are formed right after the RNN. These layers will hopefully find patterns that identify each source. It's important that these two layers are located right after the RNN in order to get information on past data as well. The number of units is the same as the dimension from the input tensor. A ReLU activation function is associated with these layers, which is perfect for the type of filtering that is done. The method for generating them is `tf.layers.dense`.

Time-Freqeuncy Masking Layer

Two simple time-frequency masking layers will compose the predicted magnitude spectra from the previous dense layers. To obtain those, we divide the dense layer tensor to the sum of the dense layers and multiply it with the input tensor.

Output

The output is the two predicted magnitude spectra for our two sources. In the training session, this output is compared to the original sources, in order to compute the loss. In the evaluation session or when we want to obtain the corresponding audio files, these spectra need to be further processed. We define a method to get frequency masks from these magnitudes. These masks are applied to the original mixed magnitude, and are transformed to a waveform. This transformation is done by obtaining the STFT matrix using the stored phase spectra, and further applying the ISTFT to the matrix. We will use the ISTFT implementation provided by Librosa. The resulting waveforms can be written to file.

5.2.4 Model Training

To train our model, we load the model, with the latest state available. Next, we specify the loss method, and the optimizer. After some investigation on this matter, the Adam optimizer was chosen, an implementation being available on TensorFlow. After the initialization, we can start the training session. Drums and vocals audio are chosen from a random song. From this, only a portion of configurable seconds will be processed. Using the preprocess techniques, we get the corresponding batches for the mixture and the two sources. We pass the loss function, optimizer and the batches to the method run of our current session. This method gets us the loss value, and backpropagates it through our model. In order to save the progress, we will use the checkpoint system implemented in TensorFlow. This lets us save the Variables and parameters of our model and restore them when we restart the software.

Currently 1000 training iterations are done in about 6 hours. Judging by how the loss value decreases, a satisfiable number of iterations would be around 15.000, which takes over 3 days on this hardware configuration. In order to tune the hyperparameters, multiple individual training session should run. Because of time issues we will research what values these parameters should have and personally observe how it affects our model. No special evaluations are to be done.

Current hardware configuration:

Processor	i7-7700HQ, 2.80Ghz
Physical Memory (RAM)	8.00 GB
Graphic Card	GTX 1050 4 GB

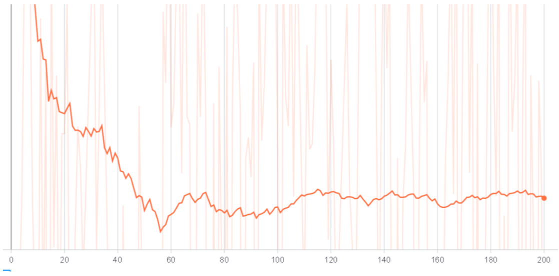


Figure 5.3: Loss value decreasing in time as the model is learning

5.3 Rest API

..

5.4 Mobile Interface

..

5.5 Web Interface

..

Chapter 6

Conclusion

6.1 Results

The aim of this thesis was to develop a reasonable program that can separate instruments from a song, and also provide a platform and an interface so that regular users easily utilize this software. For the most part, we achieved what we have proposed. The application can separate from a mixture of drums and vocals the two different sources. Due to time constraints, the model and hyperparameters were not perfectly tuned. This resulted in difficult training of the network and suboptimal results. Even if the separation seems lousy in some instances, the statistics assure us that the model is learning and improving consistently.

As for the mobile and web user interfaces, they are working as intended. The users can easily access the separation software. Both the mobile and the web applications have a modern, easy to use design. In the same time, their structure is scalable. The REST API is simple but efficient. It can successfully store the provided audio file, give it as input to the model, and after that return the output audio to the user.

6.2 Discussion

This thesis provided a subjective and very specific look at the audio blind separation problem. There is currently no implementation of a software that can separate a musical source without noticing the error. Our attempt was therefore a valiant effort in solving this problem.

Even so, our model did not perform as well as expected, presumably because of insufficient training. Our software and network were designed in their simplest form. Of course, a better model, specific for our task, could be conceptualized. Other systems such as stacked hourglass networks could be explored. This specific architecture presented at last year's ISMIR Conference showed promising results in this field.

6.3 Future Work

Our software, obviously has not yet reached its potential. There is a lot of room for improvement, mainly by optimizing the model and training it with more data. A more substantial dataset needs to be found or composed. The architecture of the LSTM RNN can also be updated to gain better results. Even so, our initial goal has not been fully achieved. We are still to train our model for recognizing and separating more instruments. This is a deficient that came mostly from the lack of data available.

Assuming the software produces professional results, a lot of doors open for future developments. This software can be a gateway to other applications such as converting the audio to a midi format, and further transpose it to a music sheet. Another similar network can be taught to recognize chords or beats from the audio. One more software I would personally be interested in is an automatic piano tutorial of the song. After gaining the isolated audio of the piano, we could separate the individual notes by analyzing the magnitude spectra and recognize which notes are being played at a given time.

If further improved, this software will pave the way for numerous MIR algorithms and applications. Assuming that it is optimized to give faultless results, it would be a true landmark for audio signal processing and deep learning in general.

Bibliography

[1] Dsd100, 2012. data obtained from AIST MIR.

[2] Emmanuel Vincent. Aditya Arie Nugraha, Antoine Liutkus. Multichannel audio source separation with deep neural networks. 2015.

[3] Soheil Bahrampour. Comparative study of deep learning software frameworks.

[4] Leon Bottou. Large-scale machine learning with stochastic gradient descent.

[5] Pritish Chandna. Monoaural audio source separation using deep convolutional neural.

[6] Alex Graves. Generating sequences with recurrent neural networks.

[7] Sepp Hochreiter. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[8] Sepp Hochreiter. Long short-term memory.

[9] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions.

[10] Anil K. Jain. Artificial neural networks.

[11] Humphrey E. Jansson A. Singing voice separation with deep u-net convolutional networks. *18th ISMIR Conference*, 2017.

[12] Hinton Geoffrey LeCun Yann, Bengio Yoshua. Deep learning.

[13] Stephen Merity. Regularizing and optimizing lstm language models.

[14] Alfred Mertins. Signal analysis: Wavelets, filter banks, time-frequency transforms and applications.

[15] Carabias J. J. Miron M. and Janer J. Improving score-informed source separation for classical music through note refinement. *ISMIR Conference*, 2015.

[16] Gautham J. Mysore Prit. Non-negative hidden markov modeling of audio with application to source.

[17] Emmanuel Vincent. Musical source separation using time-frequency source priors. ieee transactions on audio, speech and language processing. (14):91–98, 2006.

[18] Ian Goodfellow Yoshua Bengio. Deep learning.

[19] Wojciech Zaremba. Recurrent neural network regularization.

[20] Wojciech Zaremba. Spectrogram track detection an active contour algorithm.

[21] Udo Zölzer. Digital audio signal processing-2nd edition. 2008.