DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2017

Audio Effects Emulation with Neural Networks

OMAR DEL TEJO CATALÁ

LUIS MASÍA FUSTER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

KTH Computer Science
and Communication

Audio Effects Emulation with Neural Networks

Omar del Tejo Catalá
Luis Masiá Fuster

Degree Project in Computer Science, DD142X
Supervisor: Pawel Herman
Examinator: Örjan Ekeberg

CSC, KTH. Stockholm, Sweden. June 4, 2017.

Abstract— This paper discusses if using Neural Networks we can develop model which emulates audio effects and also if it can stand up to traditional audio effect emulators. This report includes the comparison of the performance between Recurrent Neural Networks such as Long Short Term Memory and Gated Recurrent Unit, and also Convolutional Neural Networks. This paper also checks if the best performing network, dealing with a online stream of inputs, can produce its outputs without a significant delay, as the ones of traditional audio effect emulators.

The networks were trained to emulate an EQ effect. The results compared the audio produced by the network with the audio we want the network to produce, which is the audio modified by the EQ. These results were compared quantitatively, calculating the absolute difference between the two audio and comparing the frequency spectrum; and qualitatively, checking if people could hear both audios as the same one.

Long Short Term Memory turned out to be the ones which achieved the best results. However, they could not produce a stream of outputs without a significant delay nor an acceptable error.

Index

Terminology

NN: Neural Network.

LSTM: Long Short Term Memory Recurrent Neural Network.

GRU: Gated Recurrent Unit.

CNN: Convolutional Neural Network.

BPPT: Backpropagation through time.

Frequency spectrum: A representation of an audio using its frequency components instead of its amplitude.

Dilation: Separation between two contiguous samples in a subsequence of the input to a convolution. If no dilation is specified, it is 1.

Stride: Separation between two contiguous subsequences of the input to a convolution. If no stride is specified, it is 1.

Sample: We will refer as sample to each individual point in the audio. One second of audio rendered with 44100 Hz would give us 44100 samples.

Target audio: The audio that we want our network to output. However, most of the time the network will not output the target audio (that is why the network is continuously learning). We call this output “output audio”.

Output audio: The output audio of the network while training.

Timestep: Is the position in the audio. That is, if we have an audio with 44100 samples, it will have a range from 1 to 44100 different timesteps. The first sample in the audio will be at timestep 1 and the last at timestep 44100.

1. Introduction

When radio broadcasting became popular, it used effects to change and improve the characteristics of audio. Those effects used vacuum tubes and other electronics. After the invention of the transistor, vacuum tubes were replaced because transistors were cheaper to produce and maintain. However, it is popular belief that the audio quality of transistors is not as good as the vacuum tubes. They were cheaper and became a popular choice for consumer audio. Professional audio is one of the fields that still uses vacuum tubes. When computers were fast enough, digital audio processing became a reality and the first equivalents of physical effects appeared. Developers used a hard-coded model of the schematic. Although the emulations are good enough, some aspects of the electronics are difficult to mimic, for instance, the subtle sound changes to the output audio when you move a microphone in front of the speaker of a guitar amplifier.

Professional audio equipment can be expensive. Emulating it using software is an interesting and cheaper alternative. It also has the benefit of not being limited to owning several copies of the the same physical device to process many audio tracks. However, we propose a different approach than coding the inner structure of the physical effect. We propose Neural Networks (NN).

1.1 Problem statement

During training, the network will learn how to generate this effect, so trying to generate the effect later will be trivial. The input sound will just need to go one-way through the network to get the input audio modified.

If such an algorithm exists, it might have a high temporal cost, thus facing some questions. Will it be fast enough to work along with a continuous stream of input signal or will it just be able to modify pre-recorded chunks of audio? In the latter case, a lot of the use potential will dwindle, incapable of dealing with online sound modification such as needed, for example, in radio broadcasting.

Learning how to modify audio keeping the integrity of the features of the audio is quite an interesting challenge because audio processing is a rather error sensitive problem. A difference between the target audio and the output audio would led it to be heard completely different.

1.2 Purpose

There is much research studying the behaviour of networks such as Convolutional Neural Networks and Recurrent Neural Networks for different problems. This problems can be divided into two types: image processing and sequence processing (i.e. audio and text because they need that their causal dependencies within their data are taken into account). We are interested in the latter.

Research in sequence processing is focused on addressing problems such as language translators (f.i. Text To Speech) or natural language processing [10]. However, not much can be found about audio modification. Nonetheless, sound generation has been researched. In [10], they took some sounds (music, for example) and made the system learn the dependencies within the sample (for instance, chord progression). However, unlike the sound generation, we have a base audio which we want to modify, not create a new one. So we need to preserve the features of the input audio, because the modification of these would change completely how we hear the audio.

Thus our motivations in this paper can be summarized as:

> Study the effectiveness of the best performing sequence processing algorithms to work up audio effect emulation and comparatively study the results of each algorithm for this problem.

1.3 Thesis outline

The following Background section gives an brief insight to some concepts used in this paper, such as Neural Networks, Recurrent Neural Networks and some submodels of them, how RNNs learn and Convolutional Neural Networks. Method section gathers all the steps and decisions taken in this paper in order to generate its following section, Results, which includes how well the networks proposed perform on the task of effect emulation. In Discussion and in Conclusion sections we will synthesise research made.

2. Background

Sequence processing has been researched for many practical reasons. Data may come in forms of causal sequential data, there are some dependencies between the value in a sequence and its followings. This happens, for instance, in stock market, quality loss over time measurement or signal processing.

In rough outlines, sequence processing studies methods that can predict the output over time given an input sequence. Some models proposed for this include Hidden Markov Models (HMM), Deep Neural Networks (DNN) and Recurrent Neural Networks (RNN). Although HMMs were the state-of-the-art once, now they have been outperformed by other models such as Long-Short Time Memory (LSTM) or Gated Recurrent Unit (GRU) RNNs, and Convolutional Neural Networks (CNN).

Although CNNs are not as good as RNNs handling long term dependencies [6], it has been proved that they perform quite better than RNNs and other models in Text To Speech [10]. They are naturally good dealing with grids, that is why they are the state of the art in image processing (f.i. Image classification). However, RNNs were designed to work with sequences, that is why we are going to focus on them. However, due to the similarities with some of the problems addressed in [10], we will also try CNNs.

GRU and LSTM, gated-networks, both have a similar performance [5]. These have some subtle differences between them which make them slightly outperform one another on different tasks. However, GRUs are slightly faster than LSTMs. But first, we will give a brief summary of some important aspects to the work.

2.1 A brief insight to how digital audio works

Audio signals are a continuous stream of pressure waves. However, representing digital audio a continuous stream is not possible without having some to sacrifice some of its characteristics. This is achieved by first reducing the continuous signal to a discrete one (sampling) by selecting one value every T seconds. T is calculated using the inverse of the sampling rate f_s. Typically the sampling rate is set to 44100 Hz because it is the minimum frequency required to represent the highest frequency humans can hear (20000 Hz). After being sampled the value needs to be quantized to bits in order be processed or stored. The standard resolution is 16 bits. [18]

2.2 A brief insight to Neural Networks

Neural Networks (NN) are computational models which map a set of input values to a set of output ones. They are formed by several neurons connected with each other. This connections have a value, called weight, which is multiplied by the values in the input set. Usually after this multiplication, a nonlinear function is applied to the result. This enables the network to approximate more complex functions. The weights are designed to activate or inhibit the connection between two neurons. It is based on Hebbian theory, that is, if two neurons fire together, then the weights will adapt to increase the likelihood that if one of the two neurons fire, so will the other. Neurons wire together if they fire together [8].

Usually the weights of the NN are trained by means of backpropagation. This algorithm computes the error at the output layer (usually the last layer of the network). This information is used to adapt the weights to make the output of the network closer to the target value.

2.3 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a class of NN whose connections between units form a directed graph. RNNs do not use the exact templates of the training data to make its prediction, they perform linear interpolations between the samples [7]; namely, the sample on the actual time step and the previous output value of the network. Unlike many other neural network, the weights in RNNs are shared by all the different neurons for every time step. The fact that they perform well over sequence predictions makes them be used mainly, for instance, on handwriting recognition and speech recognition.

We should imagine RNN predicting sequences as a multilayer neural network with just one neuron per layer (it is easier to imagine), and this network grows, in parallel, with the dimension of time (as shown in Fig. 1). The weights used by each neuron are shared, that means that each time a neuron needs to compute its new state, they will use a globally shared weight variable. Therefore, on the learning process while backpropagating, each gradient generated will modify the same global weight.

A basic update formula to calculate the output for each time step could be the following:

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

Where the W 's are the two weights of the RNN cell, x is the input at that time step, h_t is the output at time t and b is the bias of the RNN cell.

As RNNs deal with time sequences, they need a modification of the traditional backpropagation algorithm, called Backpropagation Through Time (BPTT). BPTT not only computes the gradients for a single time step, it propagates them to the previous ones, until the first time step is reached.

Nonetheless, RNNs are just the beginning. So many different models have been proposed to improve RNNs performance such as Echo State, Hopfield, Elman, Liquid State Machines, Long Short Term Memory or Gated Recurrent Unit networks [17].

Theoretically RNNs are able to work up any sequence of any complexity. In practice, the temporal cost is not feasible. This is because this model cannot handle long term dependencies good enough due to the use of algorithms that compute the whole gradient (f.i. BPTT), which usually tends to vanish (training time becomes not feasible) or blow out (the weights start to oscillate) [9]. Consequently, LSTMs were proposed to overcome this problem.

As is shown in Fig. 1, the connections between the input layer and the hidden layer, and the connections between hidden layer and output layer are skip connections, they reduce the number of steps from input to output, reducing the problem of the vanishing gradient [7].

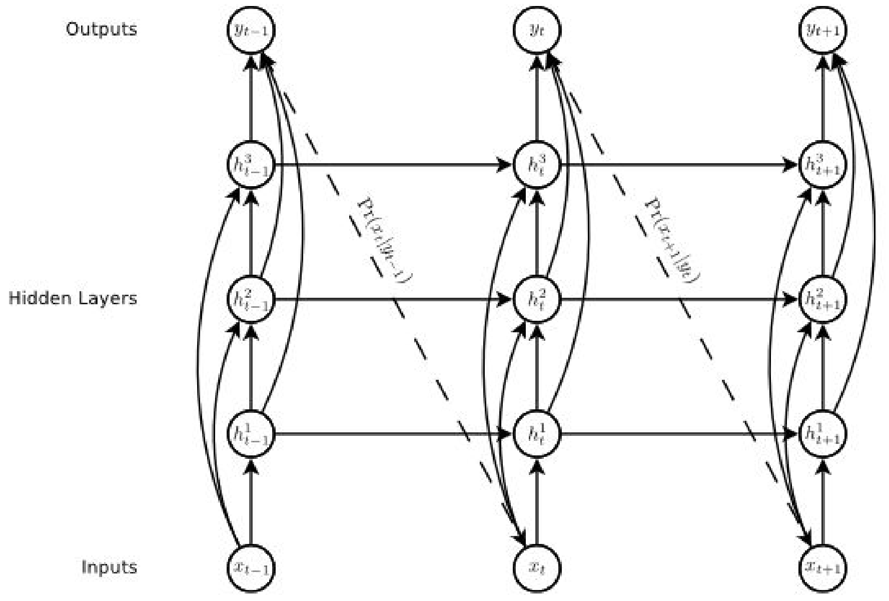


Fig. 1 RNN over time [7].

A major difference between LSTMs (and GRUs) and RNNs is that the former carries dependencies along time better than traditional RNNs. This is because LSTMs keep an internal memory cell (unlike standard RNNs [15]) that enables a better performance with long sequences. Furthermore, LSTM structure creates shortcuts within the stream of inputs, bypassing multiple temporal steps, reducing the problem of the vanishing gradients of going through several time-steps.

2.4 Long Short Term Memory Recurrent Network

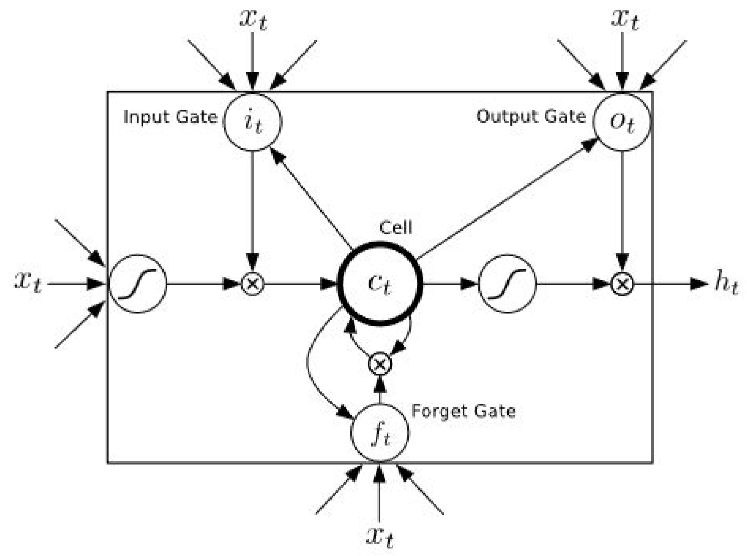


Fig. 2 Architecture of an LSTM cell [7].

Long short term memory (LSTM) was proposed initially by Hochreiter and Schmidhuber in [3]. This expansion of the RNN structure has proven to perform better than RNN finding and exploiting long term dependencies within data. LSTMs have built-in memory cells to store information that helps preserving those long term dependencies in a data sequence. However, LSTMs have several gates that control the flow of the information throughout its structure. Such gates are: the input gate (i), the forget gate (f) and the output gate (o). There are many implementations for LSTMs but all share a basic pattern. One such implementation is Graves' [7]:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)\\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)\\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o)\\
c_t &= f_tc_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)\\
h_t &= o_t tanh(c_t)
\end{aligned}
$$

In this implementation:

- i_t, f_t and o_t are the gates of the LSTM and c_t and h_t are the value for the memory cell and the output of the LSTM cell respectively.
- x is the input at a time step.
- The subindexes i and j in W_{ij} specify that weight matrix is the one that links i to j (i.e. W_{xi} is the weight matrix for that connects x with the input gate).
- σ is the sigmoid function.

As can be seen from Graves' implementation, gates i_t, o_t and f_t are all calculated in the same way, they are linear interpolation between the new input and the previous output plus the bias of the gate.

The output gate controls the amount of information of the memory cell that will flow to other LSTMs. In the beginning, the forget gate was not included. However, it was added to address a problem of the LSTM models preventing them from processing continuous input stream which are not segmented into subsequences [6], because it is able to reset the internal state of the cell. The original LSTM (the one proposed in Hochreiter and Schmidhuber) only had two gates, input and output. The internal memory cell of the LSTM could store the information along time quite well. Nonetheless, this caused the internal cell state to go grow in an unbounded fashion, thus saturating its squashing nonlinear function [12]:

$$h(x) = \frac{2}{1+e^{-x}} - 1$$

Due to the cell output function being $y^c = y^{out} * h(s_c)$ (being s_c the state of the cell and y^{out} the output gate activation) this caused that if s_c is too large, the output gate function is the same as the cell output thus removing LSTMs feature of keeping an internal memory cell; and also if it is too large the derivative of $h(x)$ is too small, disabling the ability of the LSTM to learn from incoming errors [12]. The reason why Hochreiter and Schmidhuber didn't have such a problem is because they manually set for different sequences the cell state to 0.

Notice that if the f gate is set all to 0 and the input and output gates are set to 1 we have a classical RNN with an update function:

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

2.5 Gated Recurrent Unit Recurrent Network

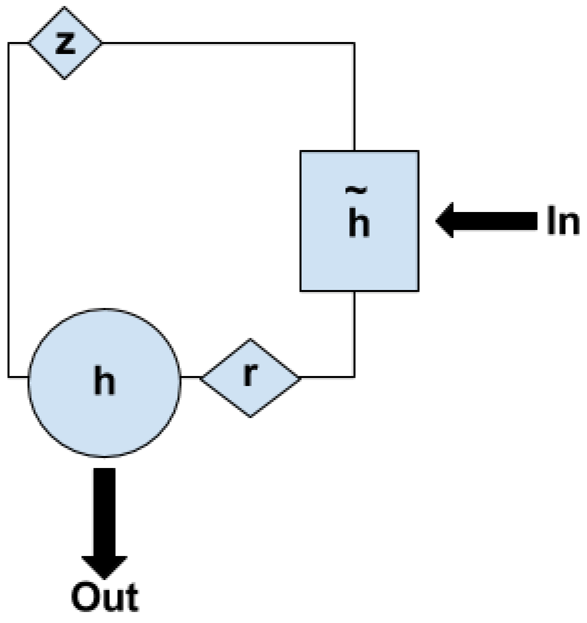


Fig. 3 Architecture of a GRU cell.

Gated Recurrent Unit (GRU) architecture is similar to LSTMs, simpler because it has one gate less, so it has less parameters to learn; hence quicker. Therefore the update function will change, being this the linear interpolation of the previous value and the input one as shown in the implementation used in [5]:

$$
\begin{aligned}
h_t^j &= (1 - z_t^j)h_{t-1}^j + z_t\bar{h}_t^j,\\
z_t^j &= \sigma(W_z x_t \ + U \ \ (r_t \odot h_{t-1}))^j.\\
\bar{h}_t^j &= tanh(W x_t + U(r_t \odot h_{t-1}))^j\ ,\\
r_t^j &= \sigma(W_r x_t + U_r h_{t-1})^j.
\end{aligned}
$$

Here:

- z is the update gate, controls how much of the input will flow to the memory cell.
- r is the reset gate. When it is close to 0, forgets the previous values of the memory cell and reads the next values of the sequence.
- $\bar{h}$ is the candidate activation, while h is the activation value of the GRU.
- W and U are the weight matrices.

There are not many differences between LSTMs and GRUs, that is why they perform similar in most of the problems [5]. However, one of the major differences between GRUs and LSTMs is that GRUs cannot control the amount of memory cell that is sent as an output, unlike LSTMs which have the output gate to handle it.

2.6 Backpropagation through time

Backpropagation through time (BPTT) is a modification of the classical backpropagation of feed forward networks to adapt to RNNs, which process information over time. Therefore, we should unfold the RNN over the time dimension. We will just consider one RNN cell with one weight matrix, but more cells and weights (LSTMs) can be added to the algorithm. The weight of the network is duplicated for each time step, each one calculating its own output and so its error. This error is calculated from the output, the input value to de RNN cell and the weight matrix at that time step, but it is also propagated to previous time steps, to be added to its error. This is done for each one of the time steps until the first one. Then, as all the weights matrices are actually the same one but unfolded over time, we should add up all the gradients computed for each time step for that weight matrix.

Although it seems rather effective, as any network (f.i. Deep Neural Networks) who goes through several nonlinear functions, it suffers a lot from vanishing gradients. Each time step in which the gradient is propagated information is lost due to the diminishing of the gradients, caused by the derivative of the intermediate functions; so that when it reaches the beginning of the sequence not much has arrived. The usage of cell memories and several gates in LSTMs or GRUs shrink this problem, because they reduce the computations needed for the gradients to reach the beginning of the sequence. That is why LSTMs are able to better handle long term dependencies in data.

There are some techniques to improve the performance of BPTT, for example, creating skip connections or using techniques to reduce the length of the sequence. This latter modification is used in this paper and will be further described in the following section.

2.7 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a Neural Network model in which an input goes through several mathematical operations (convolutions), each of which increased the level of abstraction of the input, recognising some particular important features of it. A convolution can be seen as a dot product between the input of size I and a weight matrix, called filter, of size F (kernel size) for each F-gram of the input [14]. For example, if the input is a vector such as [1, 2, 3, 4, 5, 6] and we have a filter with kernel size 3 that is [10, 20, 30], then the output vector would be:

$$\begin{gathered}[1 \star 10 + 2 \star 20 + 3 \star 30, \\ 2 \star 10 + 3 \star 20 + 4 \star 30, \\ 3 \star 10 + 4 \star 20 + 5 \star 30, \\ 4 \star 10 + 5 \star 20 + 6 \star 30]\end{gathered}$$

We could also apply some strides to the convolution. As can be seen, each time we multiply a subsequence of the input vector and the filter we move the subsequence one step to the right. This is a stride of 1. If we had a stride of 3, the result would be:

$$\begin{gathered}[1 \star 10 + 2 \star 20 + 3 \star 30, \\ 4 \star 10 + 5 \star 20 + 6 \star 30]\end{gathered}$$

Each filter will be trained to learn the most important features of the input given.

We followed the idea behind [10], where they created a CNN for Text-To-Speech translation and music generation achieving good results. The results of the report are interesting for the research done in this paper taking into account that they work with sequences of audio. They used dilated convolutions to increase the receptive field of the network in order to handle long term dependencies within data without increasing much the training time. These are a modification of the traditional convolutions where they process input values skipping a certain number of values between them, thus increases the receptive field. A traditional convolution would be a dilated convolution with a dilatation value of 1, each input value is 1 step from the previous. Therefore, we can stack several convolutions where the dilatation value increases by a factor of 2, leaving what it is shown in Fig. 4.

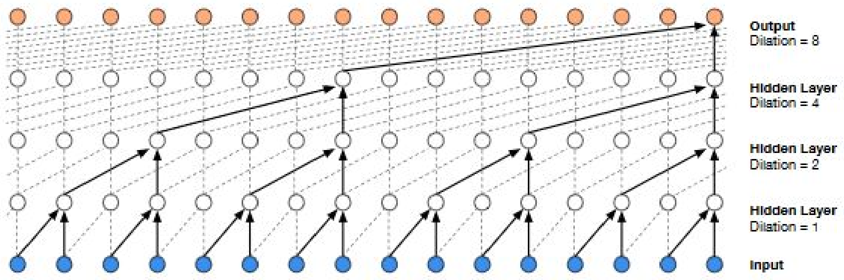


Fig. 4 Dilated convolutions [10].

Do not confuse dilation with strides. Stride is the separation between two subsequences in different multiplications within a convolution, and dilation is the separation between two samples within a subsequence.

3. Method

As we described in section 1, we trained three networks (LSTM, GRU and CNN) in order to check how well could audio effects be emulated by these. This section described how these models and all the required resources are created.

3.1 Description

We can then summarize the goal as follows. Given two digital signals, one being the input that was sent to an audio effect and one being the output of the same, we want to produce an algorithm that given the same input returns a signal as close as the original output as possible. As shown in Fig. 5, our goal is to generate the audio effect emulator which creates an output as similar as possible to the target audio.

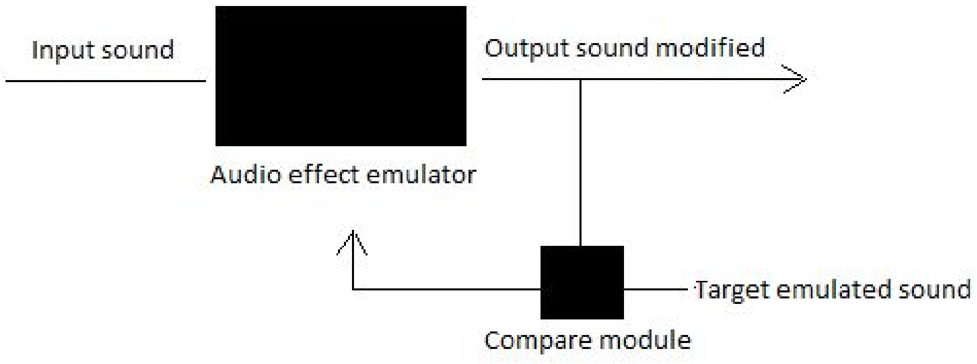


Fig. 5 Structure of the model.

In the network proposed by Graves at [7] the loss function depends on $P(X(t+1) \mid Y(t))$. That is because this network tries to generate a stream of data, so the previous outputs are important to know what value needs to come next. However, the aim of a network which tries to modify each sample of a audio sequence needs to allow for the previous input samples rather than the outputs. Therefore, the probability that we are trying to maximize is $P(Y(t) \mid X(1), X(2) \ldots X(t))$. Were we using a parameter h, which is the level of the impact the effect makes on the input and should be included in further research, the probability would be modified to include it, namely $P(Y(t) \mid X(1), X(2), \ldots, X(t), h)$.

Fortunately, we do have the target values, because they are the modification by conventional software effects of the original samples. How we get those target values will be described later in section 3.3.

3.1.1 LSTM implementation

In this paper we will use Tensorflow's implementation of LSTMs. It is a similar implementation as the one explained in section 2.3, but this one does not include the memory cell in the previous time step in the linear interpolation to calculate the gates. Thus, leaving a formula such as:

$$g_t = W * x_t + U * h_{t-1} + b_g$$

Being g_t the gate value at time t, W the weights for the input, U the weights for the previous output and b_g the bias of the gate.

3.1.2 Backpropagation through time

As mentioned in section 2.5, BPTT has problems with vanishing gradients. A solution for this problem is to reduce the length of the sequence, reducing the number of time steps which the gradients need to flow back through, preserving the integrity of the gradient. This is called Truncated Backpropagation Through Time, although it is called "epochwise truncated backpropagation" in [4], and is the algorithm implemented in Tensorflow [13]. It divides the sequence into subsequences of length K1, and it applies BPTT for that subsequence and its target values. Fig. 6 shows Tensorflow's BPTT for a sequence length of 6 and K1 = 3.

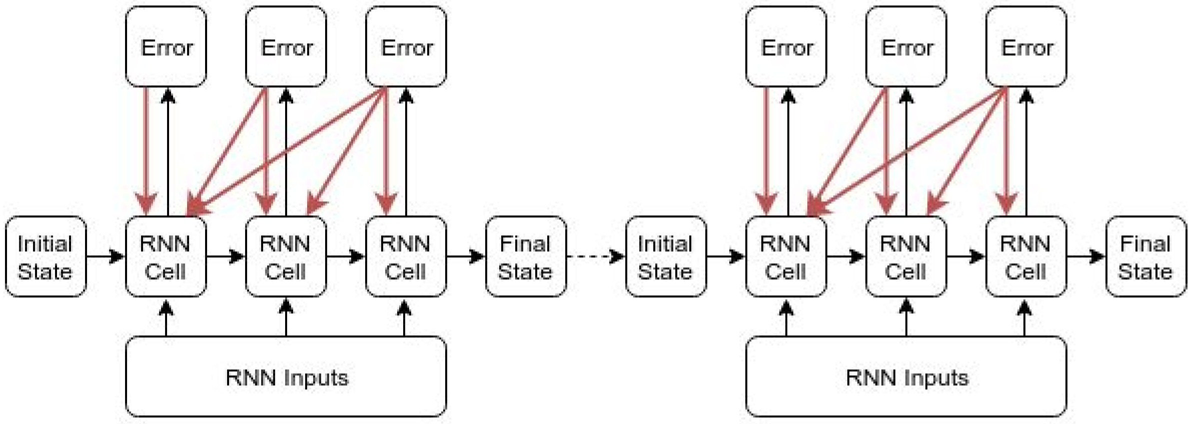


Fig. 6 Tensorflow's BPTT [13].

3.1.3 CNNs implementation

As in [10], we included dilated convolutions in our implementation of the CNN. We used a CNN with 4 stacked groups of 2 convolutions and 1 pooling layers, each convolution used a Tanh as an output function. The first convolution of the pair used strides and the second one used dilation. The first convolution values were set to 128 filters, a kernel size of 128, 1 stride and 1 dilation and a pool size of 1 and 1 stride for the pooling layer. The number of filters and the kernel size halved their size on each group and the other parameters doubled their value. At the output we have 4 fully connected layers, each one halving the output size starting at 8 (so in the end is 1). This stack of layers outputs one value that is added to the last of the input values.

3.2 Parameter settings

The parameters of a network is one of the most determining things to allow for when designing a network in order to achieve the utmost performance from it. Therefore, we tried several configurations of the parameters of the network and tracked if they improved the performance, and which were the optimal settings. We did not focus much on learning rate because we are using optimizers included in Tensorflow, and they are the ones responsible of the learning rate annealing. However, although the optimizer modifies the learning rate throughout the training phase, we gave a good initial value to it. This paper uses Adam optimizer.

For other parameters, such as the number of hidden nodes in RNNs and the number of convolutions in CNNs, we performed a coarse-to-fine search to find good values for them.

3.3 Data creation

The dataset was created using Reaper and Audacity, which are Digital Audio Workstations (i.e. a piece of software used for recording, editing and producing audio), and some Virtual Studio Technologies included in it (which are the software effects programs used in this paper).

The data was formed using the following functions:

- White noise: provides uniform intensity for all frequency intervals.
- Pink: provides the same intensity for all octaves (double or half frequency).
- Gaussian: all samples follow a normal distribution.
- Brown noise: a random offset is applied to the previous sample to calculate the next one.
- Sine function which increased its frequency over time
- Sine function which decreased its frequency over time

Copies of the functions were also used in which the amplitude increased, decreased or both over time. We also created a reduced version of this training set, "miniaudio", where we reduced the length of the each of the audios and combined them into a 20 seconds audio. We mostly trained with the latter because the first one was too big to be processed by our computation capability.

For validation purposes, we created an audio that combined new chunks of the samples listed above and also a recorded voice audio. We divided this into two: Validation 1, which includes this new combined audio; and Validation 2, which includes the voice audio.

Taking into account that we are using one second per audio, each sample rendered with 44100Hz, we are dealing with a good amount of training samples: 7813000 in the big set and 882000 for the small one.

We also applied some preprocessing to the data: we used mini-batch training, dividing the whole training set into several batches which were picked sequentially. Fig. 7 shows how the dataset is prepared to later be used by the network.

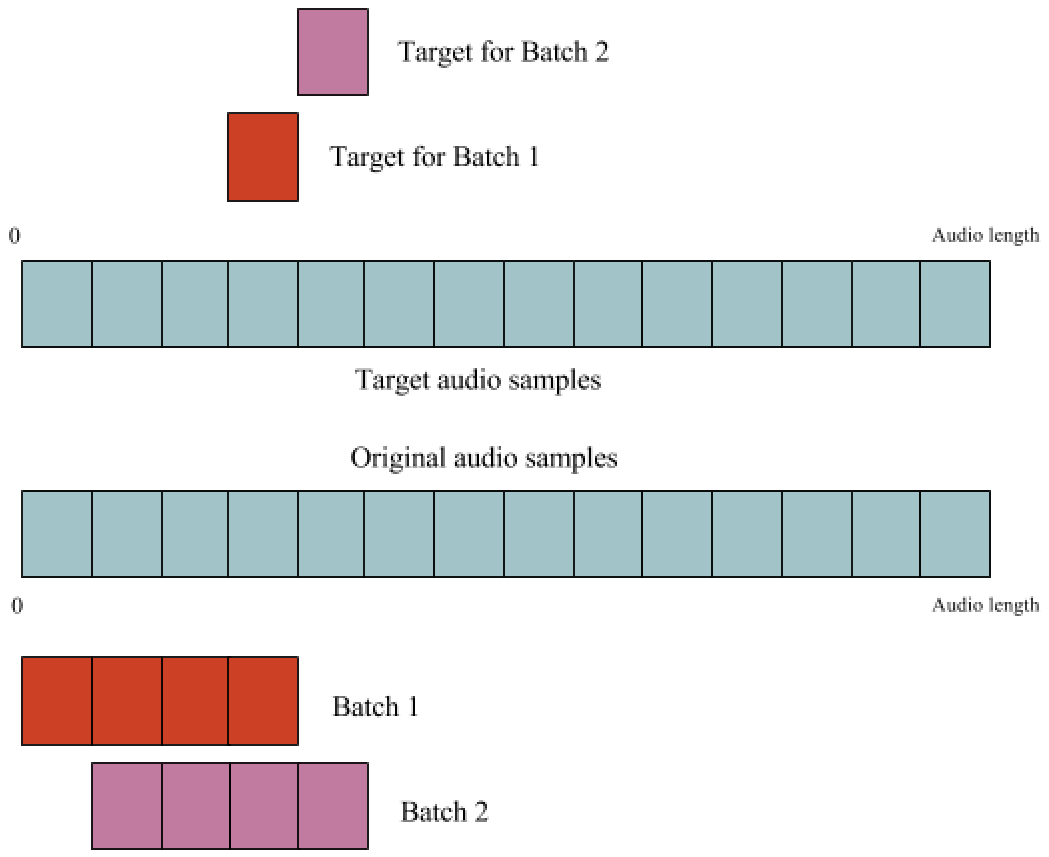


Fig. 7 Structure of the data used by the network.

3.4 Training

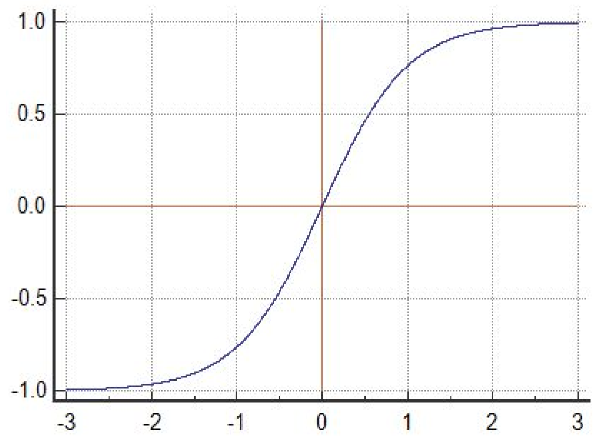
In this paper we used minibatch training, using several subsets of the training dataset instead of using all at once. Due to memory limitation, we could not work with the whole training dataset when using the big dataset, so we needed to divide this big dataset into two and swap between them along the training time. This increases the temporal cost due to the added cost of loading the dataset into memory. Also, this was done each time the validation error during training met some conditions. This problem did not happen when using the small dataset. We also used early stopping, that is, when the validation error increased during the training 100 epochs in a row, the training is stopped.

Because we are working with audio, we need to keep the frequencies of each subsequence of the audio the same. Also we need to allow for all the available frequencies. Therefore, we were in need to change the batch size to fit in even the lowest frequencies (20Hz). This means that the optimal value we need to set for the input size of the network is 2205. However, this increased too much the memory consumption, making it unfeasible. So we set our input size as high as possible (later in section 4 the values will be specified).

We tried to emulate what we reckon is a rather difficult effect, and therefore the one we expect to give a good insight of what can be achieved facing effect emulation, the EQ. The most difficult part of the EQ is that the network needs to learn that it is aiming to remove some frequencies from the data, not just to modify the samples. In particular, we applied a high pass filter at 440 Hz and so we created the modification of the training dataset for this effect. Also some fade in and fade out in the audios was included to increase the difficulty of the problem and the size of the dataset.

In the training phase, as described in Fig. 7, the network learnt how to map the values in the input batch to the value modified by the effect at the last time step of the batch. That is, is we have a batch which includes the first 1024 values of a sequence, the network will try to predict the modified value at timestep 1024 (if the sequence starts at time step 1).

As we are dealing with sound, we are using tanh as the nonlinear output function instead of sigmoid because the values of the target sound are values between -1 and 1, and so are the ones returned by tanh.



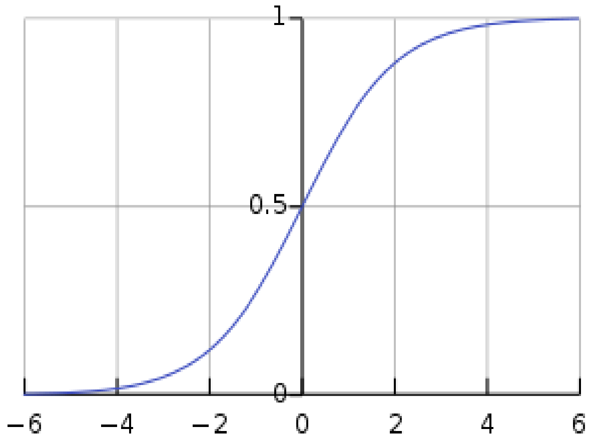


Fig. 8 and 9. The image on the left depicts the tanh function and the one on the right, the sigmoid function.

3.5 Testing

This paper will evaluate the results using two different methods: Qualitative and quantitative.

3.5.1 Quantitative testing

We tested the absolute difference between the samples of each time step for the target sample and the output one (Mean Square Error). However, in CNNs we also used the accuracy of the output in order to know how close was the target sample to the output. It is considered accurate enough if the difference between the output value and the target was less than the minimum difference between two values of a 12 bit audio, 0.000244 ($\frac{1}{2^{12\,bits}}$).

We also compared the spectrogram of the predicted audio and the target audio. A spectrogram is the representation of the frequencies of a sound which may vary over time [11]. So we were in need to compare if the predicted output had the same frequency scope as the target one, because this variation would make a huge impact on the predicted sound.

3.5.2 Qualitative testing

We used human raters that marked how similar was the output audio to the target one. Human raters are also used in [10] to rate how well WaveNet could change text to speech. Due to the subtle differences between two audios we use a small rating scope; 1 being not similar at all; 2 they have some similarities; 3 almost the same; and 4 the same audio.

3.6 System

We used libraries such as Google's Tensorflow along with Numpy to create our models. We leverage the multiple cores in the GPUs to perform matrix multiplication, increasing the speed performance. Tensorflow includes this advantage for Linux OS.

We worked with a NVIDIA GeForce GTX 960M GPU with 2GB of main memory to obtain the results. Furthermore, we also used the platform FloydHub, which is a cloud computing service. In FloydHub we have available a GPU with 10 GB capacity.

4. Results

This section shows how well LSTMs, GRUs and CNNs perform trying to learn the modification made by an audio effect. We will also compare LSTM performance with GRU's in order to deliberate which one achieves the best results. First we begin with the RNNs, in particular, LSTM.

4.1 LSTMs

This section includes the performance of LSTMs in learning the EQ effect. The first section shows some graphs to numerically check how well it performed. The second shows how similar human raters reckon the output audio is to the target one. The third one shows a reduced version of the LSTM which is trained to be later compared with the results of the GRU. The reason behind this is that the GRUs needed to be small due to a smaller computation capability when training them, and both GRUs and LSTMs need to have the same parameters to be effectively compared.

4.1.1 Quantitative testing

After some tuning of the LSTM network, we found that the best results when comparing the validation set (the voice sample) were achieved using 384 hidden layers and 1024 input size.

We trained the network with the big dataset for the high pass EQ at 440 Hz. The results achieved are shown in Fig. 10. As can be seen, we got a good error for the validation set in the last iteration (7.9e-5). You may expect this error to be pretty low, however the trained network does not emulate quite good the effect. Fig. 11 shows the difference between the output audio and the expected one. Even more, one of the problems we expected, the frequency spectrum, is not preserved from the output audio and the target one, as shown in Fig. 12 and Fig. 13. As can be seen, in Fig. 12 it learns better because the validation set is closer in its content to the training one, but Fig. 13 is the voice validation audio, which is completely different. The one to focus on is the latter.

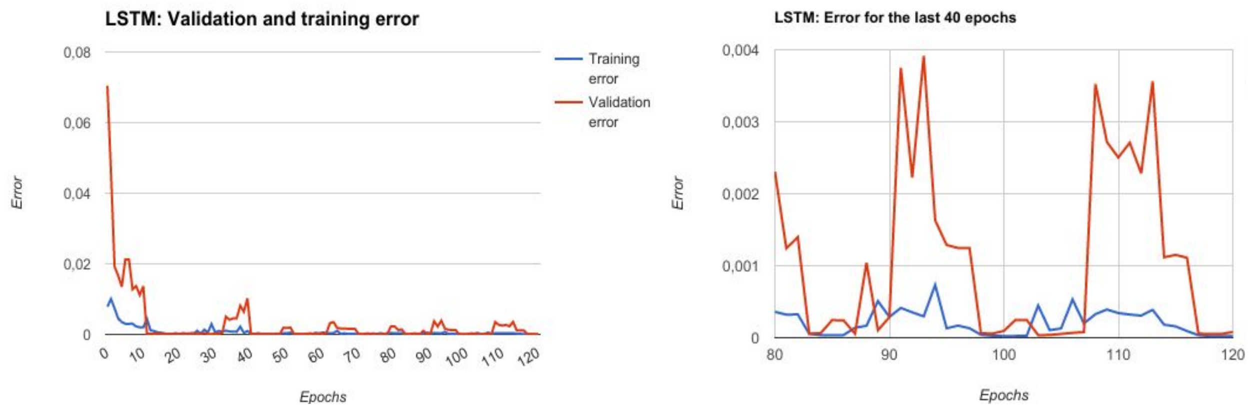


Fig. 10 Validation and training error over time. Red line is for validation error and blue one for training error.

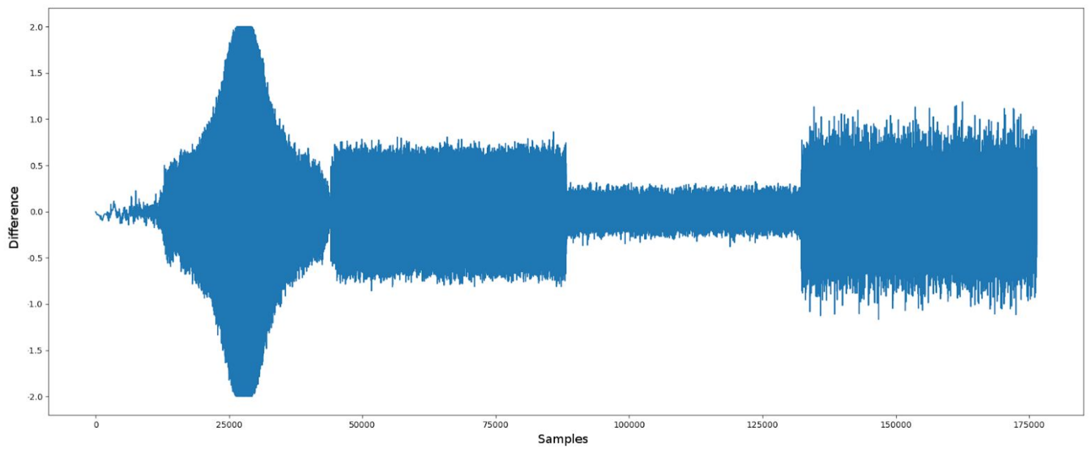
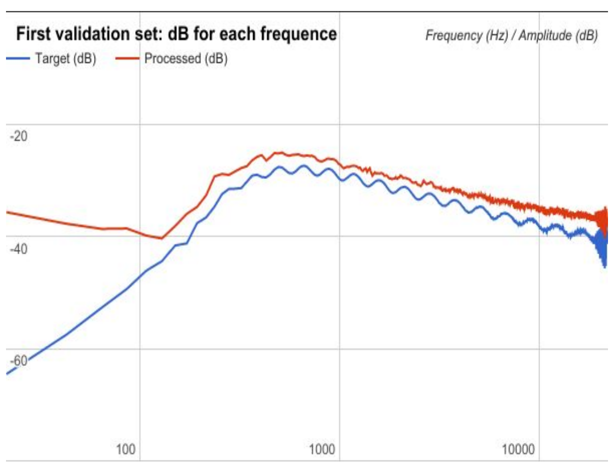


Fig. 11 Difference between the target validation audio and the output one. The x axis are the samples and the y axis the difference between the output audio and the target.



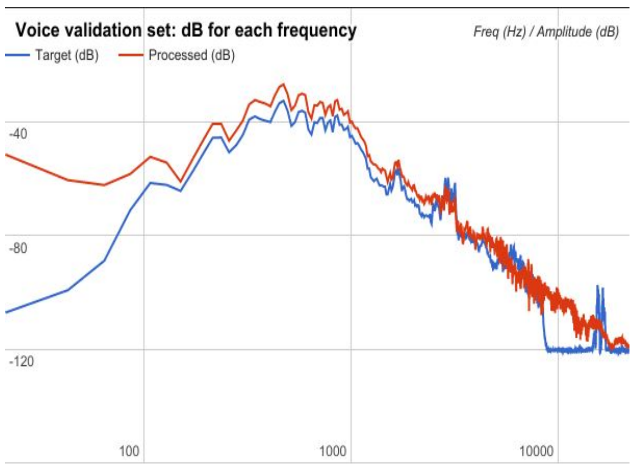


Fig. 12 and Fig. 13 Difference in the frequency spectrum.

We tried to figure out if the LSTM could deal with a stream of input samples to generate an output stream. We calculated that it takes 0.8 milliseconds to do one forward pass of the LSTM for one input. Therefore, taking into account that the sampling rate is 44100 Hz, it will take 35 seconds to process a 1 second audio.

4.1.2 Qualitative testing

Doing qualitative testing, most of the people answered that the voice validation set target and the voice validation set output were similar but not quite the same; namely, the mean of the answers was 2’25. We asked 9 people and their answers were:

- Not similar at all (3).
- Some similarities (4).
- Almost the same (2).
- The same (0).

4.1.3 Reduced LSTM

This section shows the results of a LSTM but with smaller number of hidden layers and input size, therefore it should perform worse than the LSTM created in section 4.1.1. This network is trained with the small ("miniaudio") dataset. The parameters for this network are:

- Input size of 512, rather than 1024 as the network in 4.1.1.
- Hidden layers of 160, rather than 384 as the network in 4.1.1.

As it can be seen in Fig. 14 we trained for 450 epochs achieving at the end a validation error of 3.03e-3. This result will be later compared with GRUs and CNNs in section 5. In Fig. 15, we combined the all validation audios into one. The first one is Validation 2 (the voice validation audio) and then Validation 1 (the combination of several audio modifications). There, it is shown the output audio, the target one and the difference between both.

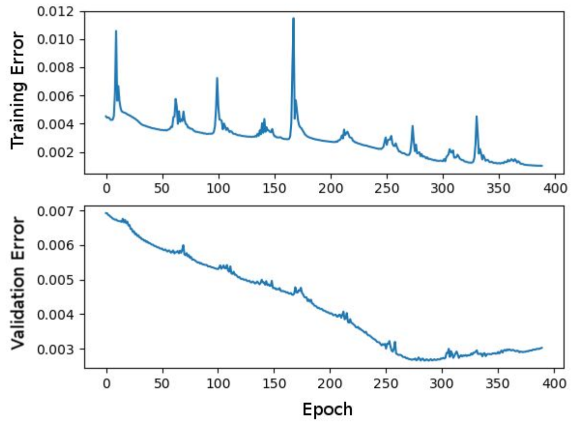


Fig. 14 Training and validation error for the reduced LSTM. The first box is the training error and the second the validation error. The x axis is the number of epochs and the y is the error.

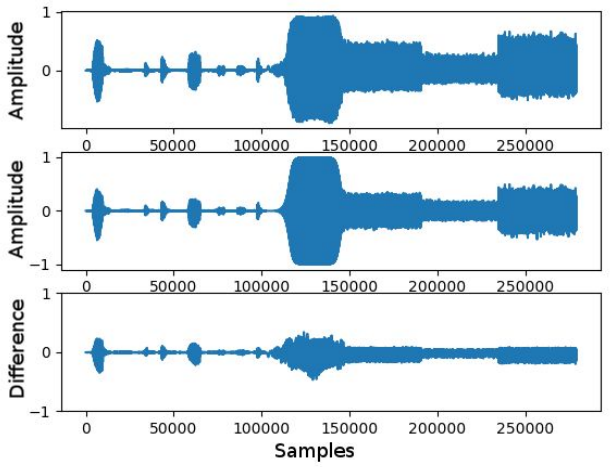


Fig. 15 Comparison between the assembled target validation audio (the first box), the output one (the second) and the difference between them (the third one). The x axis are the samples.

4.2 GRUs

For this network we expected more or less similar results as the LSTM section, as they have quite a similar performance.

The training error and the validation error for the last 3000 epochs are shown in Fig. 16. As can be seen, not much of a change is happening so late in the training phase. The final values for the validation error is 7.823e-3. Fig. 17 shows the difference between the target validation audio and the output validation audio.

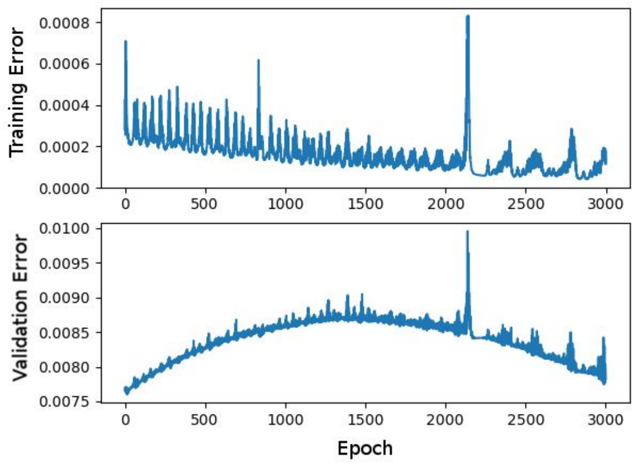


Fig. 16 Training and validation error for GRUs. The first box is the training error and the second the validation error. The x axis is the number of epochs and the y is the error.

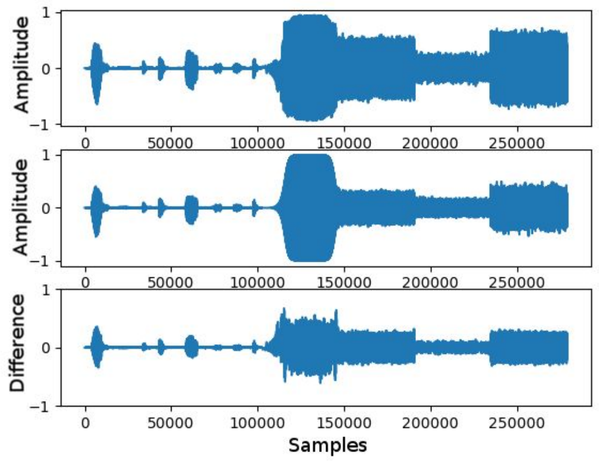


Fig. 17 Comparison between the assembled target validation audio (the first box), the output one (the second) and the difference between them (the third one). The x axis are the samples.

4.3 CNNs

We trained the CNN for over 18 hours (3 epochs). As Fig. 18 shows, the error does not change significatively while the accuracy decreases drastically, giving a final validation error of 1.905e-2. Fig. 19 shows that although it trained for the same time, it achieved worse results than the others when we compare the output audio and the target one.

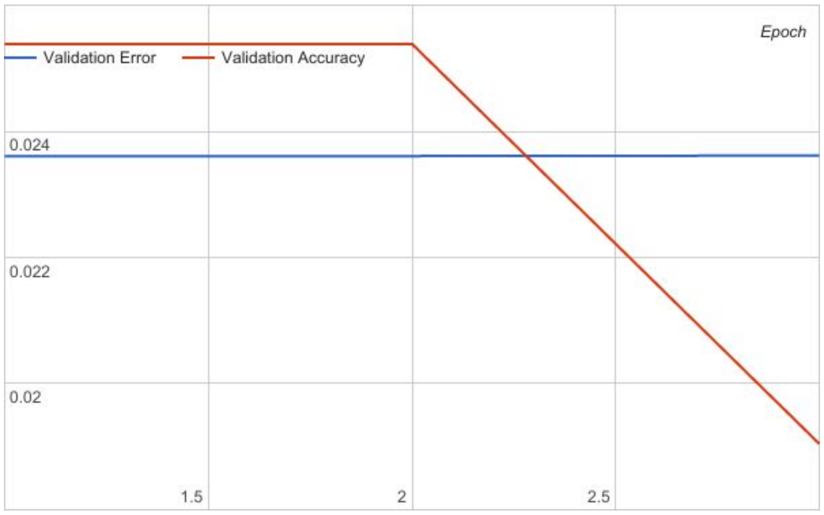


Fig. 18 Training error and validation error over time.

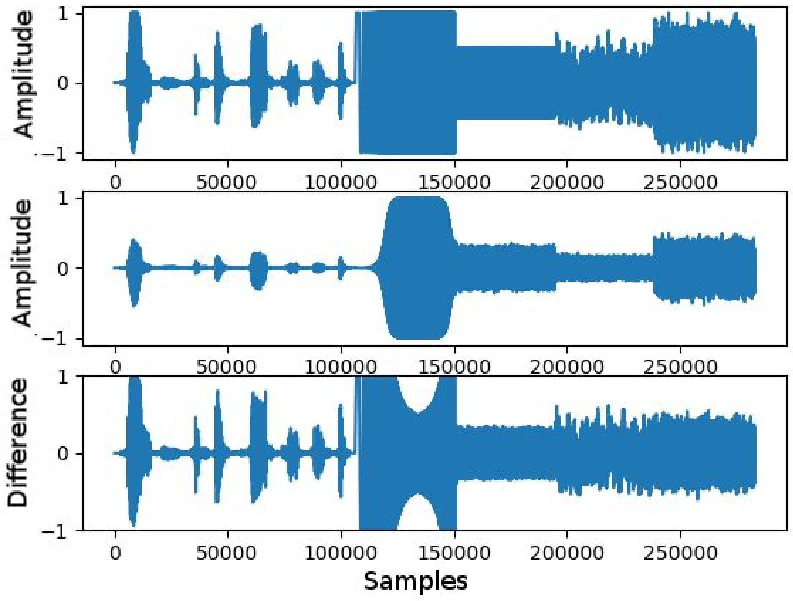


Fig. 19 Comparison between the assembled target validation audio (the first box), the output one (the second) and the difference between them (the third one). The x axis are the samples.

5. Discussion

From the beginning we assumed that LSTMs were going to be the ones which would perform better and so have the results shown. LSTMs are the ones which gave us the lowest error for the validation test, even with the reduced version of it in 4.1.3, outperforming GRUs with a 61.3% less error, and CNNs with a 84.09% less error. So the LSTMs were the ones which really stood a chance to emulate effects effectively. However, they were not able to perform good enough to be compared with the traditional effect emulators. One of the major problem is that they do not preserve the frequency spectrum of the audios.

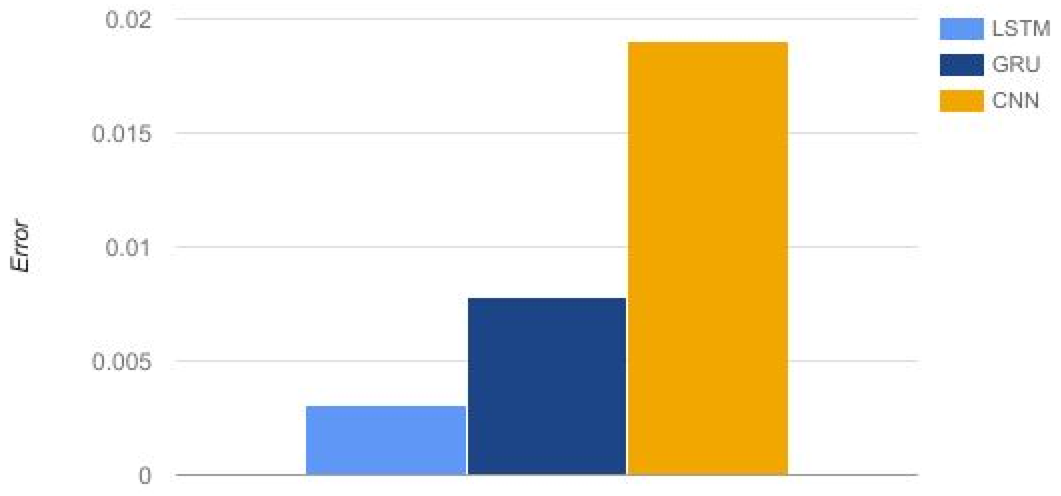


Fig. 20 Comparison of the different networks.

As shown in qualitative studies for LSTM, nobody reckon the output audio to be the same as the target one. Furthermore, the mean was 2'25, which indicates that the output just had some similarities with the target audio. This is not acceptable if our goal is to develop a model that can stand up to traditional effect emulators.

As found measuring the time cost of a forward pass in the LSTM network, it would take 35 milliseconds to process 1 millisecond of audio. As it is shown in [16], the optimal latency value for, for instance, a guitar is 13 ms and for voice is 3 ms. 35 milliseconds is far from an optimal latency. Therefore, online processing for this network would not be good enough.

Due to the hardware limitations, if someone wanted to emulate their own effects, they would need access to a computer with more computing power than the one we used, defeating the purpose of making an emulation cheaper than the traditional one.

5.1 Constraints

Despite that we modified some parameters of the network, due to the limited amount of time we could not fit the model entirely to our problem. We would have liked to check how techniques such as momentum, batch normalization and others to increase performance are used, change some inner parameters and check if the modification of them would increase the performance of our network.

Furthermore, the computational cost of the networks used is rather big, hence the time needed for training was not feasible for bigger networks than the one used. Therefore, the results are not as optimal as they could have been but they give a good insight of what to expect when approaching this problem. The biggest limiting factors were the immense amount of data required to faithfully represent the audio data and the level of precision the output samples had to have in order to match the output from the neural networks and the target output.

6. Conclusion

LSTMs were the ones to perform better than the other networks in audio effects emulation. Therefore, they were the ones which could be able to stand up to traditional audio effect emulators the most. However, not even them could effectively achieve this. The differences between the frequency spectrum of the output audio and the target audio were noticeable. Hence, the network could not preserve the integrity of the frequency spectrum of the audios nor learn that its goal was to modify the input samples to remove more abstract features of the audio, the frequencies. The modification of the samples was just a mean, the goal was to remove some underlying frequencies.

Furthermore, LSTMs could not produce a stream of outputs without a significant delay. So, the algorithm has a too high temporal cost to deal with online effect emulation.

6.1 Future research

This paper leaves some future research in this field. We could not try to emulate more complicated effects, such as a distortion from a guitar amplifier, because we did not have good results trying to emulate a simpler effect. Also, we could have developed a network approach tailored to this problem rather than using the generic algorithm implemented in Tensorflow. This network could have included that the network also learns from the difference in the frequency spectrums of the output and the target audio, backpropagating the error to all the layers. Such a thing would penalize the modification of the frequency spectrum, keeping the integrity of it when applying the effect.

We also could not include parameters into our network. It would have been preferable to be able to modify some features of the effects applied, for instance, the frequence from where the high pass filter is applied. This problem may be solved by including the parameter along with the input data, increasing the each input's size by 1. Maybe if the input size is too big this is not the way to approach this problem because the parameter input will not have much impact through the flow of the network. This should be researched.

Furthermore, some techniques can be applied to the networks which increase the performance, such as ensemble of models. This could not be done with the computational capability we possessed, but it would be interesting to see if this approach could stand up to traditional audio effect emulators.

References

[1] Back, A. D., & Tsoi, A. C. (1991). FIR and IIR synapses, a new neural network architecture for time series modeling. *Neural Computation*, *3*(3), 375-385.

[2] Williams, R. and Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. Neural Computation, 1(2), pp.270-280.

[3] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), pp.1735-1780.

[4] Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, *1*, 433-486.

[5] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[6] Sak, H., Senior, A. W., & Beaufays, F. (2014, September). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In Interspeech (pp. 338-342).

[7] Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

[8] Lowel, S., & Singer, W. (1992). Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. Science, 255(5041), 209.

[9] Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[10] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. CoRR abs/1609.03499.

[11] (2017, February 8). Retrieved from https://en.wikipedia.org/wiki/Spectrogram

[12] Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural computation*, *12*(10), 2451-2471.

[13] Styles of Truncated Backpropagation - R2RT. (n.d.). Retrieved May 07, 2017, from http://r2rt.com/styles-of-truncated-backpropagation.html

[14] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

[15] Antoine, J. P. T. (2017). Introduction to CNNs and LSTMs for NLP.

[16] When does audio latency matter and not matter? (n.d.). Retrieved May 11, 2017, from https://music.stackexchange.com/questions/30323/when-does-audio-latency-matter-and-not-matter

[17] Recurrent neural network. (2017, May 19). Retrieved May 26, 2017, from https://en.wikipedia.org/wiki/Recurrent_neural_network

[18] Lavry, D. (2004). Sampling Theory For Digital Audio. *Lavry Engineering, Inc. Available online: http://www.lavryengineering. com/documents/Sampling_Theory. pdf (checked 24.5. 2010)*.