FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Acoustic Noise Cancellation by Machine Learning
Student:	Artem Yutukov
Supervisor:	Ing. Stanislav Kuznetsov
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

The Thesis aims to design and implement a prototype Acoustic Noise Cancellation system with a machine-learning approach.
The application represents several methods for reducing unwanted sound waves by adding a second sound specifically designed to cancel the first. The model will be tested in a real environment using real-life noise surround.

1. Introduce the Acoustic Noise Cancellation systems.
2. Survey state-of-the-art methods in domain of Noise Cancellation.
3. Record audio data for noise analysis in different real environments.
4. Describe the method of sound preprocessing.
5. Preprocess sound data and describe the metadata and specifications.
6. Develop a proprietary model using the selected ML approach.
7. Find appropriate metrics to evaluate the model based on previous surveys.
8. Perform experiments with appropriate metrics and present the result.
9. Test the model in a real-life environment.

Reference:
Zhang, H., Wang, D. (2020) A Deep Learning Approach to Active Noise Control. Proc. Interspeech 2020,
doi: 10.21437/Interspeech.2020-1768.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 16 February 2023 in Prague.

Bachelor's thesis

ACOUSTIC NOISE CANCELLATION BY MACHINE LEARNING

Artem Yutukov

Faculty of Information Technology
Katedra aplikované matematiky
Supervisor: Ing. Stanislav Kuznetsov
May 12, 2023

Contents

Acknowledgments vi

Declaration vii

Abstract viii

Abbreviations x

Introduction 1

1 ANC systems 3

1.1 First prototypes . . . 3
1.2 ANC Headphones . . . 3
1.3 Adaptive filters . . . 4
1.3.1 Least Mean Squares . . . 4
1.3.2 Further modifications of the LMS . . . 5

2 Machine Learning background 7

2.1 Neural Networks . . . 7
2.1.1 Single-layer Perceptron . . . 8
2.1.2 Multi-layer perceptron . . . 8
2.1.3 Neural Network Training . . . 9
2.1.4 RNN . . . 12
2.1.5 CNN . . . 14

3 Sound miscellaneous 17

3.1 Waveform . . . 17
3.2 Fourier Transforms . . . 18
3.3 Fast Fourier Transofrms . . . 19
3.4 Mel scale spectrogram . . . 20
3.5 Sound Metrics . . . 20

4 State-of-the art 23

4.1 GRU and music reproduction . . . 23
4.2 Convolutional Recurrent Networks for ANC . . . 24

5 Experiments 25

5.1 Selected model details . . . 25
5.2 Collecting the data . . . 25
5.3 Sound pre-processing details . . . 26
5.4 Training strategy . . . 26
5.5 Staging the experiment . . . 27
5.6 Model evaluation . . . 27
5.7 Results discussion and further works . . . 28

Contents of the attached media 33

List of Figures

1 Active Noise Cancellation system[2] 2

1.1 LMS filter block scheme 4
1.2 FxLMS block scheme[10]. 6

2.1 Schematic diagram of a simple neural network. Green color indicates input neurons, blue – hidden neurons, yellow – output neuron[13] 7
2.2 RNN structure (Left) and its unfolded representation (Right)[17] 13
2.3 LSTM and GRU with their gates side by side 13
2.4 Example of 2D convolution with 3x3 kernel size 15

3.1 Waveform example 17
3.2 Wavenet causal convolution 18
3.3 Example of STFT power-scaled spectrogram 19

4.1 Outline of GRU-based Baranov's model 23
4.2 CRN architecture 24

List of Tables

5.1 Gruoups and their summary time durations 26
5.2 Models and their parameters number 27
5.3 Models and their parameters number 27

List of code listings

I would like to thank my supervisor Stanislav Kuznetsov, my family and friends, who supported me throughout my studies.

Declaration

In Prague on May 12, 2023 ....

Abstract

This bachelor thesis contains general information about active noise cancellation systems, the history of the field, and a discussion of how we can represent sound for artificial intelligence processes. In particular, it discusses the use of machine learning in this area, and demonstrates the success of such systems, taking as a basis the convolutional recurrent neural network (CRN) architecture introduced by Tan K. and Wang D.[1]. Demonstration model is designed to predict future Short Time Fourier Transform (STFT) windows, using previous and zero-padded STFT windows as inputs. The proposed system was trained using a supervised learning approach and evaluated based on its ability to accurately predict several future STFT windows. The performance of the system was measured using the normalized mean square error (NMSE) between the predicted and actual STFT windows. The results show that the CRN-based system achieved low MSE values, indicating a high level of accuracy in predicting future STFT windows. The paper also discusses the limitations and possible improvements of the proposed system as well as its potential application in real scenarios. Overall, the results obtained in this work provide valuable insights into audio pre-processing and processing using neural networks and offer a promising foundation for future research in this area.

Keywords Active Noise Cancellation, Short-Time Fourier Transform, Machine Learning, Neural Sound Processing

Abstrakt

Tato bakalářská práce obsahuje obecné informace o systémech aktivního potlačování hluku, historii oboru a diskusi o tom, jak můžeme reprezentovat zvuk pro procesy umělé inteligence. Zejména pojednává o využití strojového učení v této oblasti a demonstruje úspěšnost takových systémů, přičemž za základ bere architekturu konvoluční rekurentní neuronové sítě (CRN), kterou představili Tan K. a Wang D.[1]. Demonstrační model je navržen tak, aby předpovídal budoucí okna Krátkodobé Fourierove Transformace (STFT), přičemž jako vstupy používá předchozí a nulou doplněná okna STFT. Navržený systém byl vycvičen pomocí přístupu supervizovaného učení a vyhodnocen na základě své schopnosti přesně předpovídat nekolik budoucích oken STFT. Výkonnost systému byla měřena pomocí normalizované střední kvadratické chyby (NMSE) mezi předpovězenými a skutečnými okny STFT. Výsledky ukazují, že systém založený na CRN dosáhuje nízké hodnoty MSE, což svědčí o vysoké úrovni přesnosti při předpovídání budoucích oken STFT. Práce se rovněž zabývá omezeními a možnými vylepšeními navrhovaného systému, jakož i jeho potenciálním použitím v reálných scénářích. Celkově lze říci, že výsledky získané v této práci poskytují cenné poznatky o předzpracování a zpracování zvuku pomocí neuronových sítí a nabízejí slibný základ pro budoucí výzkum v této oblasti.

Klíčová slova Aktivní potlačení šumu, Krátkodobá Fourierova transformace, Strojové učení, Neuronové zpracování zvuku

Abbreviations

ANC	Active Noise Cancellation
CNN	Convolutional Neural Network
CRN	Convolutional Recurrent (neural) Network
FFT	Fast Fourier Transoform
GRU	Gated Recurrent Unit
LMS	Least Mean Squares
LSTM	Long Short-Term Memory
MSE	Mean Squared Error
NMSE	Normalized Mean Squared Error
NLMS	Normalized Least Mean Squares
RNN	Recurrent Neural Network
FxLMS	x-Filtering Least Mean Squares
SGD	Stochastic Gradient Descent
SLP	Single-layer perceptron
MLP	Multi-layer perceptron
STFT	Short-Time Fourier Transform

Introduction

We want to begin our narrative with why we chose this particular topic. We will also talk about the goals we have set for ourselves to cover it more broadly.

Motivation

Today it is increasingly difficult to find a quiet place to work or relax. With the growth of cities, the number of cars and population density is increasing, leading to a steady increase in noise levels. If you like to sleep in on weekends and live in an apartment building, you've woken up at least once to the fact that your neighbor has decided to drill a hole in the wall in the morning. In factories and construction sites, the problem of noise also affects people's health. You can fight this problem, for example, with earplugs, which are not very effective. However, what if you don't try to muffle the sound, but just absorb it by using backward sound waves for that noise? It sounds like a fairy tale, but these properties of sound have been known for a long time.

Seeing the success of neural networks in solving different problems and having a personal interest in working with sound, we decided to try to create an Acoustic Noise Cancellation System using Machine Learning. Such a system could be easily integrated, for example, into a smart home or to be used as units on construction sites when there is a place for noisy work. The system will have information about real noises and could automatically control the noise level.

Problem Statement

Thus, noise can be a big problem for people, reducing their quality of life and affecting their health. On the other hand, noise isolation methods are not so effective, especially when we are dealing with low-frequency noise or there is close proximity to the source of the noise. With this approach, I decided to consider the problem of active noise cancellation as a more effective method to escape from noise.

The principle of operation of ANC, from the point of view of physics, is very simple and based on wave interference. Sound is a pressure wave, which consists of two stages: compression and rarefaction. The noise-canceling speaker emits a sound wave with the same amplitude but with an inverted phase (antiphase) of the original sound. Waves in the process of interference mix into a new wave and suppress each other.

However, due to the presence of delays in the system and the loudspeakers[3], inverting the phase of the audible noise and the output of such anti-noise is not a simple task.

There is also a number of problems and open questions related to the use of ML for acoustic noise suppression, such as the learning strategy, selection of suitable data representations,

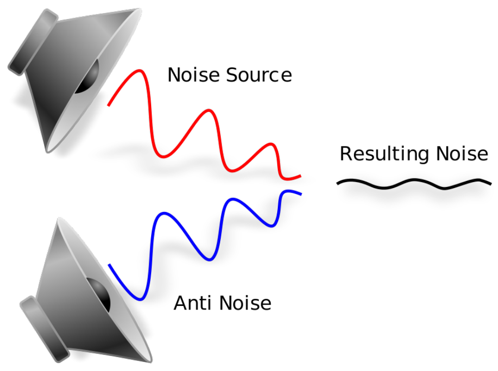

Figure 1 Active Noise Cancellation system[2]

selection and tuning of ML algorithms, integration with existing signal processing systems, and evaluation of performance indicators. These are the aspects we will try to handle in our work.

Goals

To address the problems presented above, we set ourselves the following goals:

1. Introduce the Acoustic Noise Cancellation systems.
2. Survey State-of-the-art methods in the domain of Noise Cancellation.
3. Get audio data for noise analysis in different real environments.
4. Describe the method of sound pre-processing.
5. Pre-process sound data and describe the metadata and specifications.
6. Develop a proprietary model using the selected ML approach.
7. Survey appropriate metrics and evaluate the model.
8. Perform experiments with appropriate metrics and present the result.
9. Test the model under real conditions.

Reference

The idea of ANC using ML is not inherently new. With the development of ML in the field of sound, there have been many attempts to create such a system. One of the successful studies in my opinion was the work of Hao Zhang and DeLiang Wang[3]

Taking this work as a basis for my own models, I will try to review their approach to investigate the possibilities of Active Noise Cancellation using Machine Learning. In building my own model, I will also focus on possible improvements of the model, the possibility of simplifying some parts to reduce model parameters, and the effect of the form of data representations on the speed and performance.

Chapter 1

ANC systems

In this part, we will take a look at different Active Noise Cancellation systems and approaches, from the emergence of this area to more advanced approaches that were at the origins of machine learning and use some of its approaches.

1.1 First prototypes

The first stamp on the noise management system was issued to inventor Paul Lueg[4]. In his patent, explained a method for canceling sinusoidal tones in ducts through the phase advancement of the wave. Additionally, it detailed a technique for canceling arbitrary sounds in the vicinity of a loudspeaker by reversing the polarity. He describes a method of Active Noise Cancellation in a space where sound can travel far away, such as in a pipe. His main idea was simple: since sound travels slower down the pipe than an electrical signal, you can listen to it at one end of the pipe and send the signal from that microphone in reverse phase to a speaker further down the pipe, thereby absorbing the initial noise. Paul Lueg's invention was seen as a significant improvement over existing methods of noise cancellation, as it provided a more effective and efficient way of reducing noise levels. His patent was widely recognized and became the basis for many subsequent developments in the field of sound insulation and noise reduction.

The next step in the ANC was taken by American scientist and evolutionary computing pioneer Lawrence J. Fogel. In the 50s of the 20th century, he was working on a noise suppression system for helicopter and airplane cockpits. An object of one of J. Fogel's inventions[5] was a method for improving speech intelligibility by suppressing the dominant vowel sounds and accentuating the consonant sounds. Interestingly, this method relied on the psycho-acoustical ability of the mind to fill in gaps in sound that are omitted from a familiar pattern of speech. J. Fogel also outlines various circuit configurations, including a peak-detecting circuit, gating circuit, and low-level squelching circuit to implement the method of the electronic transmission of sound.

1.2 ANC Headphones

With the development of technology, headphones with active noise canceling began to appear.

In 1957 Willard Meeker proposed the first ANC system for headphones. His model proved to be capable of cutting off the noise in the audio range from 50 to 500 Hz, with a maximum attenuation of approximately 20 dB[6].

However, ANC headphones appeared on the market much later. The first mass-produced ANC headphone was introduced by Bose in the late 1980s.

1.3 Adaptive filters

Subsequent improvements in ANC were the emergence and further start of the use of adaptive filtering algorithms in this area.

Adaptive filtering algorithms are a class of digital signal processing algorithms that adjust their parameters in response to changes in the input signal, allowing them to adapt to changing conditions and extract the desired signal from noisy or unpredictable input data. The basic idea behind adaptive filtering is to adjust the filter coefficients in real time based on the current input data and the desired output.

1.3.1 Least Mean Squares

The LMS algorithm is a popular adaptive filtering algorithm used in digital signal processing applications. It was invented in 1960 by Stanford University professor Bernard Widrow and his first Ph.D. student, Ted Hoff[7], and further described in many works.

The main idea behind the LMS algorithm is to iteratively minimize the MSE-cost function, which is expressed by the difference between the output of the adaptive filter and the desired output signal. In order to converge to the optimal values, the LMS updates the filter weights.

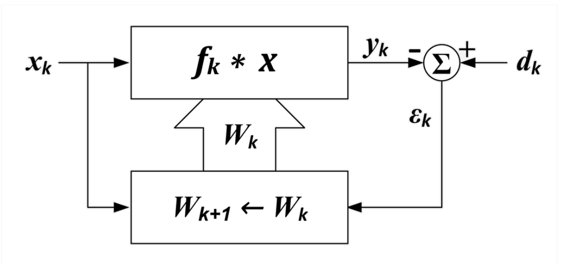

Figure 1.1 LMS filter block scheme

Consider Figure 1.1 let's take a closer look at how this algorithm works. Therefore, the signal $x(k)$ is the input signal representing the known noise source. Signal $d(k)$ is the primary signal received at the error microphone. Estimation of $d(k)$, that is, $\hat{y}(k)$ is an output of the adaptive filter, by which we mean inverse noise signal, which calculates with the conjugate or, also known as Hermitian, transpose of the estimated filter's weights matrix $\hat{w}^H(k) \in \mathbb{C}^P$.

Thus, consider the equations described by Widrow and Hoff[7]. We want to represent the reference signal with it estimation as follows:

$$\hat{y}(k) = \hat{w}^H(k) \cdot x(k). \tag{1.1}$$

Deduction $d(k)$ and $\hat{y}(k)$ is an error signal $e(k)$:

$$e(k) = d(k) - \hat{y}(k). \tag{1.2}$$

Then, we need to calculate the *cost function* $C(k)$:

$$C(k) = \mathbb{E}(e(k)\, e^H(k)). \tag{1.3}$$

This is exactly where the algorithm takes its name from, since the cost function, in fact, is a mean square error, which the algorithm then tries to reduce. Then, we calculate the so-called *gradient* of the cost function with respect to each of its individual weight entries. We will discuss this action in more detail later, but for now let's just look at this computation in the LMS algorithm in equations:

$$\nabla_{\hat{w}^H(k)} C(k) = \nabla_{\hat{w}^H(k)} \mathbb{E}(e(k)\, e^H(k)) = 2\,\mathbb{E}(\nabla_{\hat{w}^H(k)}(e(k)) \cdot e^H(k)). \tag{1.4}$$

$$\nabla_{\hat{w}^H(k)}(e(k)) = \nabla_{\hat{w}^H(k)}(d(k) - y(k)) = \nabla_{\hat{w}^H(k)}(d(k) - w^H(k)\, x(k)) = -x(k). \tag{1.5}$$

$$\nabla_{\hat{w}^H(k)} C(k) = -2\,\mathbb{E}(x(k)\, e^H(k)). \tag{1.6}$$

This is then used to calculate the next step's weights $\hat{w}(k+1)$ of the adaptive filter:

$$\hat{w}(k+1) = \hat{w}(k) - \frac{\mu}{2} \nabla C(k) = \hat{w}(k) + \mu\,\mathbb{E}(x(k)\, e^H(k)), \tag{1.7}$$

where μ represents the step size parameter, which controls the rate of convergence of the filter coefficients. A larger step size can lead to faster convergence, but it can also cause instability and divergence of the algorithm. It usually takes a value from 0.0001 to 0.3[8].

For counting $\mathbb{E}(x(k)\, e^H(k))$ we can use statistical estimation :

$$\hat{\mathbb{E}}(x(k)\, e^H(k)) = \sum_{i=0}^{S-1} x(k-i)\, e^H(k-i). \tag{1.8}$$

In this equation, S indicates the number of samples, which in the simplest example equals 1. This way, the loss function is simplified to this form:

$$\hat{\mathbb{E}}(x(k)\, e^H(k)) = x(k) \cdot e^H(k), \tag{1.9}$$

and next weights calculate as follows:

$$\hat{w}(k+1) = \hat{w}(k) + \mu x(k) e^H(k) \tag{1.10}$$

In headphones, such a system can be applied as follows. The ANC system contains two microphones connected to an adaptive filter that is controlled by the LMS algorithm. The first microphone is outside the headphones and picks up only noise, while the second microphone is inside the headphones and picks up a mixture of noise, the desired signal (music or speech) and anti-noise. The LMS algorithm adjusts the filter coefficients to match as closely as possible the noise signals from both microphones and then outputs a phase-inverted noise signal. This signal is then overlaid on the original mixture of the desired signal and noise, resulting in an approximation of a pure music or speech signal. This approximation is then used as an error signal to update the filter coefficients in the next iteration, and the process repeats.

While the LMS algorithm is simple and computationally efficient, it does have some shortcomings. One of the main problems is its slow convergence rate, which can be especially pronounced in applications with highly correlated input data. This way, the algorithm can struggle to handle non-stationary input signals or rapidly changing system parameters.[9]

1.3.2 Further modifications of the LMS

To combat some shortcomings of the LMS, several improvements have been proposed by its author. We'll take a brief look at two of the most popular ones: NLMS and FxLMS.

1.3.2.1 Normalised LMS

Let's look one more time at equations 1.7 and 1.10. There we can see, that the loss function in this in the LMS filter is directly proportional to the signal $x(k)$. Hence the idea of normalization of the input signal, also reviewed by Widrow[8]. So, the equation for updating the weights of the NLMS algorithm is as follows:

$$\hat{w}(k+1) = \hat{w}(k) + \frac{\mu\, x(k)\, e^H(k)}{x(k)\, x^H(k)} \tag{1.11}$$

Widrow also found and proved the optimal value of learning rate μ for the NLMS algorithm. It is:

$$\mu_{opt} = 1, \tag{1.12}$$

when there is no additional interference $\nu(k)$, and:

$$\mu_{opt} = \frac{\mathbb{E}(|d(k) - \hat{y}(k)|^2)}{\mathbb{E}(|e(k)|^2)}, \tag{1.13}$$

in the general case.

1.3.2.2 Filtered-x LMS

Because the LMS algorithm is applied directly without filtering the reference signal, it cannot guarantee convergence because of the delay caused by the power amplifier and loudspeaker.

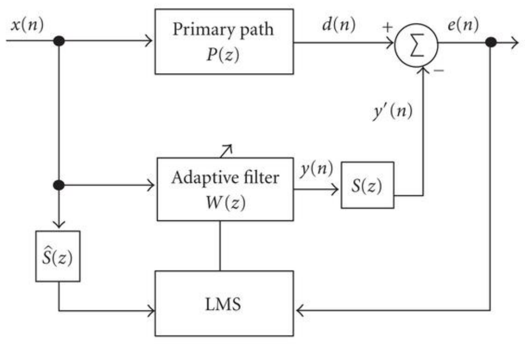

Figure 1.2 FxLMS block scheme[10].

The x-LMS Filtering algorithm, i.e. the FxLMS [11] algorithm, is an improvement of the LMS algorithm, taking into account the influence of this problem. We can see the principle of its work in Figure 1.2. The idea is in adding an additional block, that allows to compensate for the reference signal delay in the electroacoustic tract.[12]. For that proposes, the transfer function of secondary path $S(z)$ and it's estimation $\hat{S}(z)$ are used.

Summary

In this chapter, we got acquainted with the first ANC systems and also looked at the main advanced systems of its implementation – adaptive filters, in particular, on the LMS family of filters. We have seen that the principle of these filters is largely inspired by Machine Learning approaches. The successes of these approaches tell us that the idea of creating a more complex model using ML to better suppress noise may make sense.

Chapter 2

Machine Learning background

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that aims to enable computer systems to learn from digital data and improve their performance on a specific task without being explicitly programmed. ML emerged from the fusion of several scientific disciplines, including mathematics, statistics, computer science, and artificial intelligence. These fields have worked together to develop methods and algorithms that allow computers to learn from data and perform a task without explicit program instructions. In last years Machine learning has become a key tool for a variety of tasks, such as pattern recognition, text classification, data analysis, and more.

Before we continue our discussion and move on to more specific approaches created with machine learning for ANC, let's talk about based concepts in this area, without which it is impossible to understand the further technical context of our work.

2.1 Neural Networks

As we have already learned, machine learning is a method that allows a computer to learn from data and experience without requiring explicit programming. One of the most popular approaches in ML is neural networks.

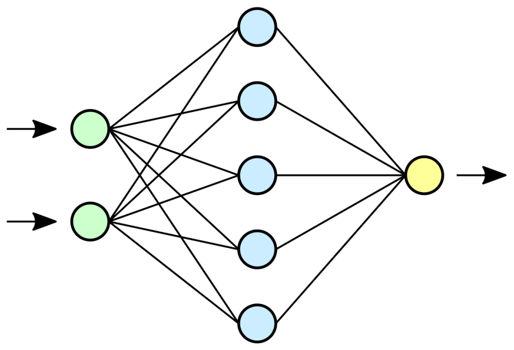

Figure 2.1 Schematic diagram of a simple neural network. Green color indicates input neurons, blue – hidden neurons, yellow – output neuron[13]

Neural networks are mathematical models consisting of neurons connected by synaptic connections. The idea of neural networks was inspired by the structure of interwoven neurons in biological systems. Each neuron may process information received from several inputs neurons and generates one output. An example of such a network is shown in Figure 2.1 above.

2.1.1 Single-layer Perceptron

The *single-layer perceptron* (SLP) is one of the simplest and most common units of neural networks. This neural network model is used to solve the problem of binary classification, i.e. separation of objects into two classes. The main component of the single-layer perceptron is an artificial neuron, which can receive several inputs and provide one output. The weights in the neuron determine which input data are the most important to determine the class.

Let us translate this process more deeply. In mathematical terms, the resulting output of a neuron is achieved through the application of an activation function, denoted by f, to the internal potential value ξ. This potential value is calculated by taking the sum of the inputs $x_1, \ldots, x_n$, multiplied by their respective weights $w_1, \ldots, w_n$, and the bias w_0[14].

In this way, the internal potential ξ is described in the following way:

$$\xi = w_0 + \sum_{i=1}^{n} w_i\, x_i = \boldsymbol{w}^T\, \boldsymbol{x} + w_0, \tag{2.1}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ and $\boldsymbol{w} = (w_1, ..., w_n)^T$. Note, that $(.)^T$ is the matrix transposition operation.

Further, the output of a neuron is found like this:

$$\hat{Y} = f(\xi) = f(\boldsymbol{w}^T\, \boldsymbol{x} + w_0). \tag{2.2}$$

In equation 2.2, f is the *activation function*. For the *single-layer perceptron* model used the so-called *step function*:

$$f(\xi) = \begin{cases} 1, & \text{when } \xi \geq 0, \\ 0, & \text{when } \xi < 0. \end{cases} \tag{2.3}$$

This in turn corresponds to the following:

$$f(\xi) = \begin{cases} 1, & when\ \boldsymbol{w}^T\, \boldsymbol{x} \geq -w_0, \\ 0, & when\ \boldsymbol{w}^T\, \boldsymbol{x} < -w_0. \end{cases} \tag{2.4}$$

Overall, this process of obtaining the estimation $\hat{Y}$ we described for SLP in equations 2.1 and 2.2, as well as in all neural networks models, is called *forward pass*. Note also that, the potential value function ξ, generally, represents a straight line dividing the input space into two parts, in one of which the activation function $f(\xi)$ is 0 and in the other 1.[14]

2.1.2 Multi-layer perceptron

In the previous section, we discussed the single-layer perceptron and its activation function, which allows it to make binary classifications by learning a linear decision boundary. However, many real-world problems require more complex decision boundaries that cannot be represented by a single straight line. This is where the *multi-layer perceptron* (MLP) comes in.

MLP is a standard SLP extension for an artificial neural network consisting of multiple layers of neurons, each of which uses a nonlinear activation function to transform the input signal into a more complex representation. By combining multiple non-linear transformations, MLP models can learn to represent very complex decision boundaries and solve a wider range of problems than *single layer perceptron*. An example of just such a model is shown in Figure 2.1 above. This neural network conventionally consists of 3 layers: input, hidden and output.

Let's take a look at the *forward pass* of such networks.

Since each neuron receives information from all neurons of the previous layer, the function of the i-th neuron on the l-th layer is:

$$g_i^{(l)} : R^{n-1} \to R. \tag{2.5}$$

In general, the forward pass for MLP networks then takes the following form of layers functions compositions:

$$g = g^{(l)} \circ g^{(l-1)} \circ \ldots \circ g^{(2)} \circ g^{(1)}. \tag{2.6}$$

As activation functions in MLP models non-linear functions and, so-called, piecewise linear functions are used, enabling it to model complex relationships between inputs and outputs. One of the most used of these functions is *Rectified Linear Unit (ReLU)*, which has the following form:

$$f(\xi) = \begin{cases} \xi, & \text{when } \xi \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

Another activation function – ELU stands for *Exponential Linear Unit* – is a popular modern activation function used for neural networks. Like the ReLU function, the ELU activation function is also piecewise linear. However, ELU has a non-zero negative output for negative input values. Authors of ELU have shown that the use of ELUs can result in improved classification accuracy compared to the use of ReLUs. They describe[15], the formula for the ELU activation function in the following form:

$$f(\xi) = \begin{cases} \xi & \xi \geq 0, \\ \alpha\left(e^x - 1\right) & \xi < 0, \end{cases} \tag{2.8}$$

where α is the hyperparameter to be tuned with the condition that $\alpha > 0$. The main advantage of the ELU function is that it can alleviate the "dying ReLU" problem, which occurs when the gradients become zero for a large fraction of the input domain during training, causing the corresponding neurons to effectively "die" and not contribute to the learning process. The non-zero negative output of ELU ensures that the gradients never completely vanish for negative input values.

However, in regression problems, we do not need to partition the space, but only to get a numerical estimate of the value of some function at a certain point. To obtain such an estimate, we can do the usual linear transformation on the output corresponding to SLP, i.e. leave its output identical to ξ, without applying the activation function, i.e:

$$f(\xi) = \xi. \tag{2.9}$$

The output of this linear transformation is then used as the final output of the network for the regression task.

2.1.3 Neural Network Training

We've already recognized how you can get value from a neural network but haven't quite figured out how they learn their weights.

The training of neural models is not a trivial task, since we will need to find a way to update the weights correctly for all layers. In the 20th century, this problem caused much skepticism about the further development of Machine Learning. One of the first comparisons of the training of such networks was published in the journal "Avtomatika" by the Ukrainian and Soviet mathematician Oleksiy Grigorovich Ivakhnenko. His work was called "Group Method of

Data Handling - a rival of the method of stochastic approximation". The basic principle of GMDH is to construct a model based on multiple regressions, where each regression involves a subset of input variables. Ivakhnenko's method is the earliest method to train Deep networks(networks with a number of layers of 3 or more).

The main breakthrough in the development of Machine Learning, however, came a little later, with the advent of the *backpropagation* method. The earliest descriptions of this method date back to the early 70s of the 20th century and include the works of S. I. Linnainmaa, A. I. Galushkin and P. J. Verbos. This method was later significantly developed and is now used as one of the main ones for training neural networks.

In this process, the network's output is compared to the desired output, and then this evaluation is backward propagated through the layers to update the weights and biases of the neurons. This iterative process continues until the network reaches a satisfactory level of accuracy on the training data.

To get this performance evaluation of the neural networks, a so-called *loss function* is used. In this case, network training is presented as an optimization problem.

For the regression problem can be used, for example, *Mean Squared Error* (MSE), which we have already mentioned several times in our work, but haven't provided its formula for neural networks. Thus, it is calculated as follows:

$$L(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2.10}$$

Here, Y is the vector of true values, $\hat{Y}$ is the vector of predicted values, and n is the number of samples in the dataset.

Cross-entropy is a loss function commonly used in classification tasks. It measures the difference between the predicted probability distribution and the actual probability distribution of the target variable.

Consider $\hat{p} = \hat{P}(Y = 1|\boldsymbol{X} = \boldsymbol{x})$, the cross-entropy loss for a binary classification problem can be expressed as follows:

$$L(Y, \hat{p}) = -Y \log \hat{p} - (1 - Y) \log (1 - \hat{p}). \tag{2.11}$$

For multi-class classification problems with c number of classes, we denoting the probability estimation of $\hat{p_i} = \hat{P}(Y = i|\boldsymbol{X} = \boldsymbol{x})$, i.e. $\hat{\boldsymbol{p}} = (\hat{p_1}, \ldots, \hat{p_c})$.. Thus, the cross entropy loss can be expanded to categorical:

$$L(Y, \hat{p_i}) = -\sum_{j=1}^{c} \mathbb{1}_{Y=j} \log \hat{p_j} = -\log \hat{p}_Y \tag{2.12}$$

where $\mathbb{1}_{Y=j}$ is the indicator variable, which means $\mathbb{1}_{Y=j} = 1$ if the sample belongs to class j, and 0 otherwise. In both cases, the cross entropy loss is minimized when the predicted probabilities match the actual probabilities (i.e. when the predicted distribution is identical to the target distribution).

After obtaining the result of the loss function, we can start to propagate it, using so-called *gradient descent*.

2.1.3.1 Gradient descent

The goal of training a neural network is to find the optimal set of model parameters that minimize the loss function, thus producing the most accurate predictions on unseen data. These parameters correspond to dimensions in the space of possible models, and the gradient of the loss function points in the direction of the steepest ascent of the loss in this space.

Thus, firstly, we need to find this minimum. From courses in mathematical analysis, we know that finding the minimum of a function with one variable requires finding its derivative. For functions with many variables, we need to find the derivatives of all its variables. This is where the partial derivatives come to the aid. Thus, to use the gradient descent, the activation function of neurons must be differentiable or at least have a nonzero derivative in its positive range. This requirement is necessary as we need to compute the gradient ∇L, with respect to all the trained model parameters. In general, ∇L is denoted as follows:

$$\nabla L = \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \dots, \frac{\partial L}{\partial w_n} \right). \tag{2.13}$$

where n is the number of model parameters $w_1, ..., w_n$, which are generally the weights and biases corresponding to layers of the trained network.

To actually calculate the gradient ∇L, we can use the method of finding derivations of compound multivalued functions, also called *chain rule*.

$$\frac{\partial f \circ g}{\partial w}(w) = \sum_{i=1}^{n} \frac{\partial f}{\partial w_i}(g(w)) \frac{\partial g_i}{\partial w}(w_i), \tag{2.14}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}^n$, $g = (g_1, \dots, g_n)^T$.

Once we have computed the gradient, we can use it to update the model parameters and descent to minima. The simplest form of this descent involves updating the parameters in the opposite direction of the gradient, scaled by a learning rate α:

$$w_{t+1,i} = w_{t,i} - \mu \frac{\partial L}{\partial w_{t,i}} \tag{2.15}$$

where $w_{t,i}$ is the value of i–th parameter on t-th step, $w_{t+1,i}$ is the value of the same parameter on $t + 1$-th step and μ is a learning rate.

2.1.3.2 SGD

However, this simple gradient descent method has some limitations. One issue is that it can be slow to converge, especially for large datasets or complex models with many parameters. Another issue is that it can get stuck in local minima, which are points in the parameter space where the loss function is lower than its neighbors but not necessarily the global minimum.

To address these limitations, stochastic gradient descent (SGD) was introduced. In SGD, instead of computing the gradient over the entire dataset, we use a randomly selected subset (or mini-batch) of the dataset to compute an approximate gradient. This not only reduces the computational burden but also adds a level of noise that can help the algorithm escape local minima. Additionally, SGD allows us to take more frequent steps in the parameter space, which can speed up convergence.

2.1.3.3 Adaptive optimizers

The following attempts to optimize gradient descent were mainly concerned with the problem of fitting the learning rate, or more precisely, its adapative change to different model parameters during training. Thus, one of the first SGD improvements is Adagrad[16], which adapts the learning rate for each parameter based on the historical gradients. This means that it assigns a higher learning rate to the parameters that have a smaller historical gradient and a lower learning rate to the parameters that have a larger historical gradient. This helps in faster convergence and better generalization.

The Adagrad algorithm updates the parameters based on the following formula:

$$w_{t+1,i} = w_{t,i} - \frac{\mu}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i}, \tag{2.16}$$

where $w_{t,i}$ is the value of i–th parameter on t-th step, $w_{t+1,i}$ is the value of the same parameter on $t+1$-th step and μ is a learning rate.

where $w_{t,i}$ is the i-th parameter at time t, η is the learning rate, $g_{t,i}$ is the gradient of the loss function with respect to $w_{t,i}$ at time t, and G_t is the diagonal matrix of the sum of squares of past gradients up to time t:

$$G_{t,i} = \sum_{\tau=1}^{t} g_{\tau,i}^2. \tag{2.17}$$

Adam (Adaptive Moment Estimation) is another optimization algorithm that adapts the learning rate for each parameter based on first and second moment gradients. Adam combines the advantages of Adagrad and RMSprop optimizers by tracking historical gradients and their moving averages.

The Adam algorithm updates parameters based on the following order. Firstly, it uses two exponentially decaying moving averages to estimate the first and second moments of the gradients. The first moment estimation m_t is the exponentially weighted average of the gradient g_t, and the second moment estimation v_t is the exponentially weighted average of the squared gradient g_t^2. The formulas are as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t; \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2. \end{aligned} \tag{2.18}$$

Here, β_1 and β_2 are hyperparameters that control the decay rates of the moving averages.

Because the first and second moment estimations are initialized at zero and biased towards zero, Adam uses bias correction to compute the corrected moment estimations $\hat{m}_t$ and $\hat{v}_t$:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}; \ \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{2.19}$$

where t is the current timestep.

The final step in the Adam optimization algorithm is to update the parameters using the bias-corrected moment estimations. Thus, the formula for the update is as follows:

$$w_{t+1} = w_t - \frac{\mu}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t. \tag{2.20}$$

Here, w_t is the parameter vector at time t, μ is the learning rate, ϵ is a small constant added for numerical stability while dividing.

Once we have learned how neural networks are trained, we can move on to consider more advanced architectures that have proven themselves in certain types of tasks. This is what we will be talking about next.

2.1.4 RNN

Recurrent Neural Networks (RNNs) are a type of neural network architecture that has been widely used in natural language processing, speech recognition, and other applications that deal with sequential data. The key feature of RNNs is their ability to capture temporal dependencies in sequential data by maintaining an internal state or memory. This allows them to process variable-length sequences of inputs, making them well-suited for tasks like language modeling, machine translation, and speech recognition.

In an RNN, the input at each time step is processed along with the state from the previous time step to produce an output and a new state. The output at each time step can be used for

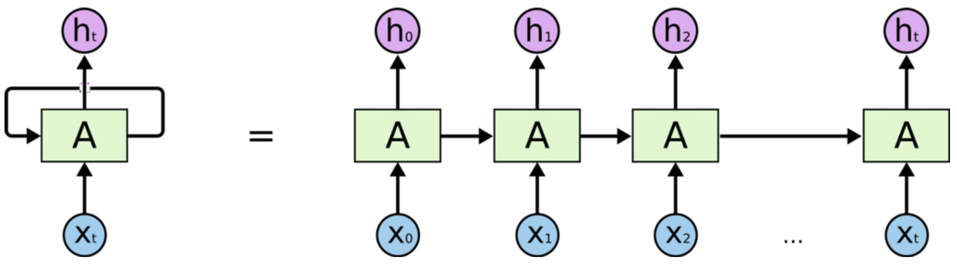

Figure 2.2 RNN structure (Left) and its unfolded representation (Right)[17]

prediction or fed back into the network as input for the next time step. This feedback loop allows the network to maintain a ”memory” of previous inputs and use it to inform future predictions. We can see it’s structure in Figure 2.2.

However, RNNs suffer from the ”vanishing gradient” problem, where the gradients become too small to effectively update the network parameters during backpropagation through time. To address this issue, several variants of RNNs have been developed, such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), which use specialized memory cells and gating mechanisms to better preserve and update the internal state over time.

2.1.4.1 LSTM

Long Short-Term Memory (LSTM) is a type of RNN architecture that was specifically designed to address the vanishing gradient problem and improve the network’s ability to capture long-term dependencies in sequential data. //add historical// LSTMs use a special memory cell that can selectively ”forget” or ”remember” information from previous time steps, allowing the network to maintain long-term dependencies and avoid the vanishing gradient problem.

The LSTM cell contains three main components: the input gate, the forget gate, and the output gate. We can see its structure in the figure 2.3 below.

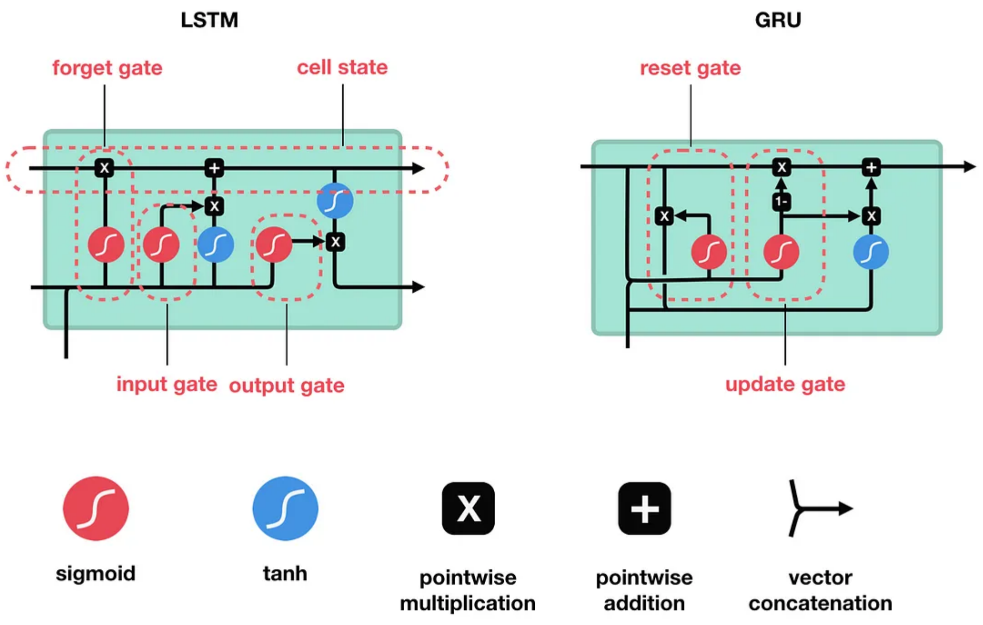

Figure 2.3 LSTM and GRU with their gates side by side

The input gate controls the flow of information into the cell by regulating how much of the input should be added to the cell state. The forget gate determines how much of the previous cell state should be retained, allowing the network to selectively ”forget” information that is no longer relevant. Finally, the output gate controls the flow of information out of the cell by regulating how much of the cell state should be output as the new hidden state.

We can see that, when the cell state context is passed to the next layer, only simple and straightforward mathematical operations are performed on it: multiplication and addition. In this way, the cell's states form a kind of highway along which the gradient can travel much farther than in the usual RNN architecture.[18] The LSTM architecture has been widely used in natural language processing, speech recognition, and other applications that deal with sequential data. It has been shown to be effective at capturing long-term dependencies and outperforms traditional RNNs on a wide range of tasks, such as language modeling, machine translation, and speech recognition.

2.1.4.2 GRU

Gated Recurrent Units (GRUs) are another type of RNN architecture that was also designed to address the vanishing gradient problem and improve the network's ability to capture long-term dependencies in sequential data. Like LSTMs, GRUs use gating mechanisms to selectively ”remember” or ”forget” information from previous time steps. However, GRUs have a simpler architecture compared to LSTMs, with only two gates: the reset gate and the update gate. These differences, as well as the structure of this unit, can be seen in Figure 2.3 below. Thus, the reset gate determines how much of the previous hidden state should be ignored, allowing the network to selectively ”reset” the hidden state if it determines that the previous state is no longer useful. The update gate determines how much of the new hidden state should be added to the previous hidden state, allowing the network to selectively ”update” the hidden state if it determines that the new state contains useful information.

GRUs have been shown to be competitive with LSTMs on a wide range of tasks, and are often used as a simpler and more efficient alternative to LSTMs, especially in sound processing.

2.1.5 CNN

A Convolutional Neural Network (CNN) is a type of neural network generally proposed for computer vision and also has shown success in other tasks, some of which are natural language and financial time series processing. The basic building block of a CNN is the convolutional layer, which applies a set of filters to the input image to produce a set of feature maps.

The filters, also known as kernels, are small matrices that are convolved with the input image. During this convolution operation, the kernel ”slides” over the input image, and the dot product of the kernel and the corresponding input patch are computed at each position. The resulting values are then summed and used to produce a single output value in the feature map.

The size of the kernel is a hyperparameter that is usually set manually, and it determines the size of the receptive field of the feature map. The receptive field is the region of the input image that contributes to the computation of a single output value. A larger kernel size will result in a larger receptive field, allowing the network to capture larger patterns in the input. In Figure 2.4 below, provided by Joseph Nelson[19], we can see the application of such a filter.

In convolutions, also defined the stride, which determines the step size of the kernel as it ”slides” over the input. A larger stride will result in a smaller output feature map, while a smaller stride will produce a larger output feature map.

Another it's parameter – padding, is used to control the size of the input feature map, especially when the used stride size is greater than 1. In this situation, padding can help ensure that the convolutional filter is applied to the edge ”pixels” of the input, which can be important for detecting features near the edges of an input map.

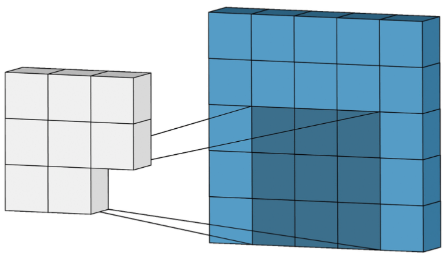

Figure 2.4 Example of 2D convolution with 3x3 kernel size

In addition to kernel size, stride, and padding, another important parameter in convolutional layers is dilation. Dilation is a technique used to increase the receptive field of a convolutional layer without increasing the number of parameters or the amount of computation required.

The basic idea behind dilation is to add gaps between the values in the kernel, effectively making it "sparser." For example, consider a 3x3 kernel with a dilation rate of 1 (no dilation):

$$K_{dil=1} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{2.21}$$

When this kernel is convolved with an input image, it computes a single output value by taking the dot product of the kernel with a 3x3 patch of the input image centered on the current position of the kernel. The receptive field of this kernel is 3x3.

Now, let's consider the same 3x3 kernel with a dilation rate of 2:

$$K_{dil=2} = \begin{bmatrix} 1 & 0 & 2 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 8 & 0 & 9 \end{bmatrix} \tag{2.22}$$

Notice that we have added gaps (zeros) between the values in the kernel. When this kernel is convolved with an input image, it still computes a single output value by taking the dot product of the kernel with a 3x3 patch of the input image centered on the current position of the kernel. However, because of the gaps, the receptive field of this kernel is now 5x5, even though the actual size of the kernel is still 3x3.

This increased receptive field can be useful for capturing larger patterns in the input, while still keeping the number of parameters and the amount of computation required relatively small. Dilation rates can be set manually as a hyperparameter, and different dilation rates can be used in different layers of a CNN to capture patterns at different scales.

Summary

In this chapter, we provided an introduction to the Machine Learning industry, focusing on the role of neural networks. We discussed the various techniques used to train neural networks and explored the more advanced types of these networks, which are capable of analyzing sequential and multidimensional data. Thus, we emphasize the potential neural networks in sound processing tasks, as they have successfully analyzed such data. By studying their success in this area, we can gain valuable insights into the capabilities and limitations of these advanced neural networks.

Chapter 3

Sound miscellaneous

There are many issues associated with the preparation and processing of sound. In order to be able to work with sound, and to be able to use it as an input for the neural network, we have to represent it in some way. Also, we have to understand what is a good metric to evaluate the performance of our system, because the standard loss functions, like MSE, may not represent the real performance of the system.

3.1 Waveform

Waveform is the most basic representation for ML sound processing tasks. It represents the sound as a series of amplitude values as a function of time. Although simple, it can still be a powerful representation of a task, as it can capture subtle changes in sound over time.

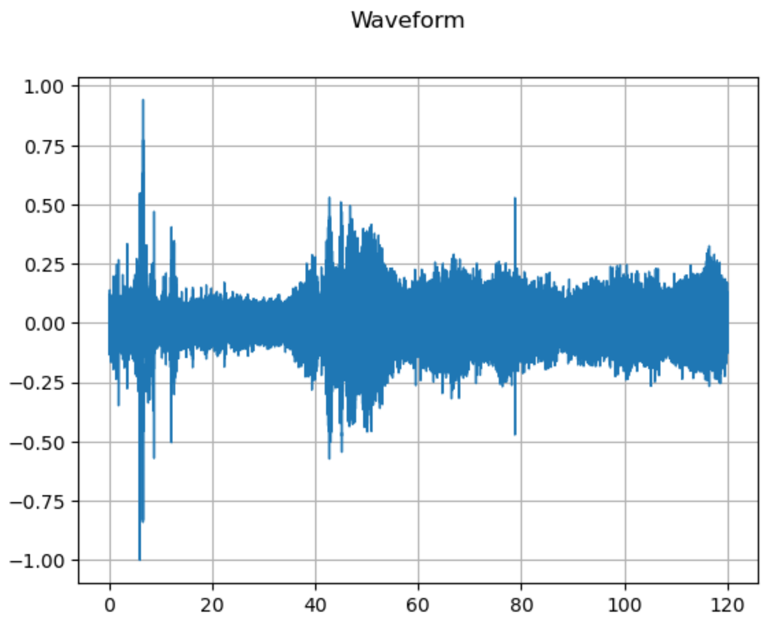

Figure 3.1 Waveform example

One interesting application of waveform representation in sound processing is in sound generation using neural networks. One such example is WaveNet, developed by DeepMind[20]. WaveNet is a deep neural network architecture that generates high-fidelity audio waveforms directly, without relying on intermediate representations such as spectrograms or mel-spectrograms. WaveNet is based on autoregressive modeling, where the model is trained to predict the next sample in the waveform given the previous samples. The model also uses dilated causal convolutions with exponentially increasing dilation rates to capture long-term dependencies in the audio signal, allowing the model to generate highly realistic and natural-sounding audio. This causal convolution technique is illustrated in the figure below:

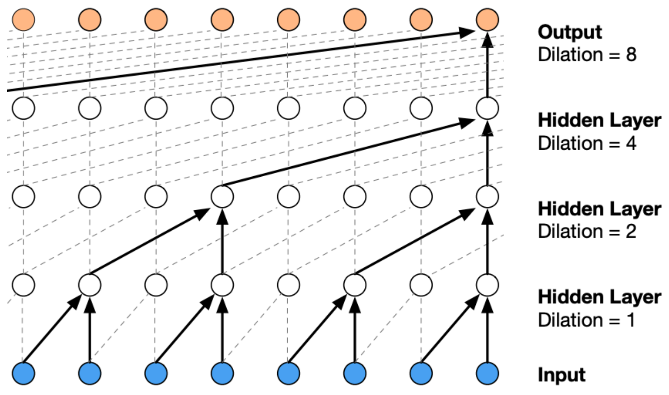

Figure 3.2 Wavenet causal convolution

Additionally, the model uses skip connections to bypass several layers and help propagate low-level details to higher layers, which is critical for generating high-quality audio. This way, it has been used in a variety of applications, such as generating music, speech synthesis, and text-to-speech.

3.2 Fourier Transforms

The representation of sound signals in the time domain is achieved through waveforms, which illustrate the variation of the signal's amplitude with respect to time. However, waveforms alone do not provide any information about the frequency content of the signal. The Fourier transform is a mathematical technique that can decompose a complex signal into its individual frequency components, thereby allowing us to determine the frequencies present in a waveform.

The Fourier transform $X(f)$ of a continuous-time signal $x(t)$ is defined as:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-2\pi ift}dt \tag{3.1}$$

where $X(f)$ is the frequency domain representation of the signal, and f is the frequency in Hertz.

The limitation of the Fourier transform is that it is applied to the entire signal, thereby not providing any information about how the frequency content changes over time. The short-time Fourier transform (STFT) is a modification of the Fourier transform that enables the analysis of the frequency content of a signal over small, overlapping time intervals, referred to as windows.

The STFT of a continuous-time signal $x(t)$ is defined as:

$$STFT_x(\tau, f) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-2\pi ift}dt \tag{3.2}$$

where $STFT_x(\tau, f)$ is the frequency content of the signal $x(t)$ at time τ and frequency f, and $w(t)$ is a window function that is typically chosen to have finite support and be symmetric about its midpoint.

The magnitude of the STFT coefficients represents the strength of the frequency content at each time and frequency bin and can be illustrated using a color map, where different colors correspond to different magnitudes.

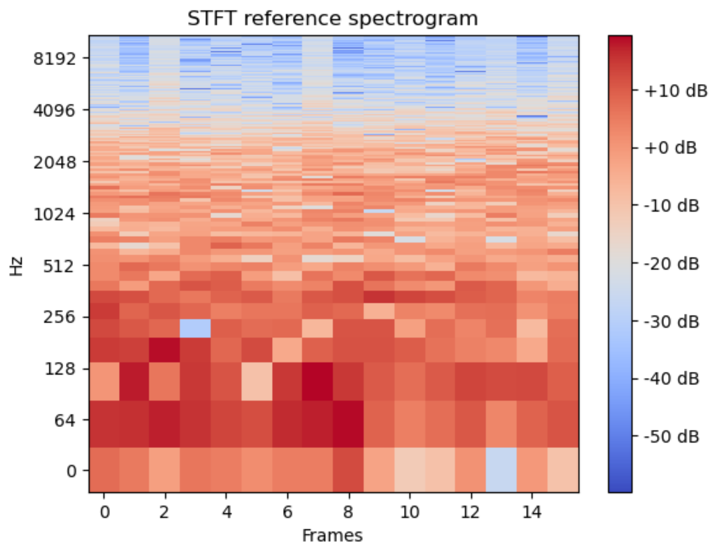

Figure 3.3 Example of STFT power-scaled spectrogram

Applications of spectrograms include speech recognition, music analysis, and acoustic signal processing. Spectrograms are particularly useful in these applications because they reveal details about the sound signal, that are not immediately apparent from the waveform representation.

3.3 Fast Fourier Transofrms

The computation of Fourier transform and STFT can be computationally expensive for large signals, especially for real-time applications. Fast Fourier Transform (FFT) is an algorithm that computes the Fourier transform in $O(n \log n)$ time, where n is the length of the input signal. Similarly, a fast implementation of the STFT can be computed using the FFT algorithm by applying the windowed FFT to the overlapping segments of the signal. This approach is computationally efficient and can be implemented in real-time applications.

The FFT of a discrete-time signal x_n is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi ikn/N} \tag{3.3}$$

where X_k is the frequency domain representation of the signal, and k is the frequency bin.

Similarly, the fast STFT can be computed using the FFT algorithm as follows:

$$X(\omega, m) = \sum_{n=0}^{N-1} \omega(n)\, x(n + m\, H)\, e^{-j2\pi\omega n/N}. \tag{3.4}$$

In the equation, $x(n)$ is the input signal, $\omega(n)$ is a window function that is applied to the signal before taking the Fourier Transform, N is the length of the window, H is the hop size, and m is an index that indicates the time position of the window. A larger window size provides better frequency resolution but poorer time resolution, while a smaller window size has the opposite effect. Similarly, a smaller hop size provides better time resolution but results in more spectral leakage and larger data size.

3.4 Mel scale spectrogram

In many sound processing tasks, it is not enough to use a standard spectrogram that only represents the energy of each frequency component at each time frame. Human perception of sound is not linear, meaning that we do not perceive equal steps in frequency as equally spaced. This non-linear perception is called the Mel scale, and it is commonly used in sound processing to represent better how humans perceive sound.

In the Mel spectrogram, the frequency axis is transformed to the Mel scale. This is done by applying a transformation function that converts the frequency f (in Hertz) to the Mel scale m:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{3.5}$$

The inverse of this function is used to convert Mel scale back to Hertz:

$$f = 700\left(10^{m/2595} - 1\right) \tag{3.6}$$

The Mel scale is divided into equally spaced intervals, each associated with a triangular filter. The outputs of each filter are then summed over frequency to obtain the energy in that Mel frequency band. This process is called Mel filtering. With these properties, MEL spectrograms found wide use in speech recognition and music analysis tasks, as they provide a more perceptually accurate representation of sound compared to Fourier spectrograms in these tasks.

3.5 Sound Metrics

Sound metric functions are critical in training models for sound processing tasks. They provide a measure of how well the model is able to reconstruct the original signal from the processed one. Due to the wide dynamic range of many signals, they are often expressed using the logarithmic decibel (dB) scale. This allows for a more convenient and compact representation of the signal's power, especially in cases where the signal spans several orders of magnitude.

Normalized Mean Squared Error (NMSE) is a commonly used metric to evaluate the accuracy of a denoised signal relative to the original signal. It provides a measure of the amount of noise remaining in the denoised signal compared to the original signal. A lower value of NMSE indicates better denoising performance. The NMSE is calculated as the ratio of the sum of squared errors between the original and denoised signal, and the sum of squared values of the original signal:

$$\text{NMSE} = 10\log_{10}\left(\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} {y_i}^2}.\right) \tag{3.7}$$

In this equation, y_i is the original signal, $\hat{y}_i$ is the output signal, and n is the number of samples. The denominator in the equation is the energy of the original signal, while the numerator is the energy of the difference between the original and the predicted signal.

Signal-to-Noise Ratio (SNR) is another commonly used metric in signal processing to quantify the ratio of the power of a meaningful input signal to the power of unwanted background noise. This ratio provides a measure of how much the meaningful signal stands out from the noise. A higher SNR value generally indicates a better quality of the signal.

Its equation is very simple, corresponding to our review above:

$$\text{SNR} = 10\log_{10}\left(\frac{P_{signal}}{P_{noise}},\right) \tag{3.8}$$

Summary

In this section, we described powerful ways of representing sound and two important sound logarithmic decibel scale metric functions: Normalized Mean Squared Error (NMSE) and Signal-to-Noise Ratio (SNR). They are used to represent sound model performance in the dynamic range of many signals. An understanding of these metrics will be useful in the upcoming section, which will cover specific ML approaches for active noise cancellation.

Chapter 4

State-of-the art

Referring to the elements and methods of data processing described in the previous parts, let's take a look at two interesting approaches, one of which was subsequently chosen as the basis to create the architecture of our own model.

4.1 GRU and music reproduction

First, let's consider the model, in Figure 4.1. This model was described by Mikhail Baranov in the article "Recurrent Neural Active Noise Cancellation", in which he talked about the problems of audio processing with RNNs and demonstrated the opportunity to predict structured noise to suppress it in a complex acoustic environment on music reproduction.[21]

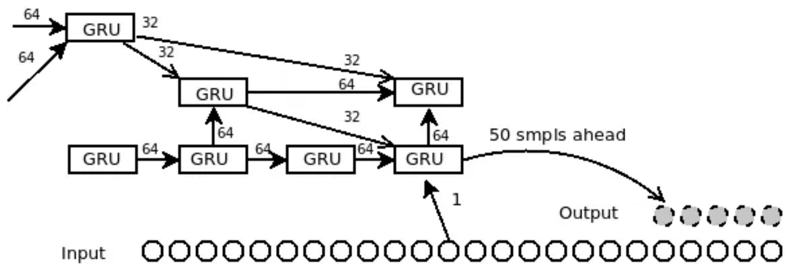

Figure 4.1 Outline of GRU-based Baranov's model

The primary approach used in the solution proposed by the author involves multispeed signal processing of sound waveform in the RNN domain. This approach is necessitated by the complexity of learning due to the unwinding of the history of input samples and states in truncated backpropagation. Consequently, the forward path is calculated hundreds of times faster than the feedback path. To address this challenge, the author developed a three-layer deep neural network of GRU cells that were custom-built to enable seamless integration with the system. The system also operates at a lower audio sample rate of 8kHz to speed up calculations. Baranov tested the system's ability to reproduce a fragment of Ievan polka[22] and obtained a positive result. The target metric was MSE. The first epoch had an error of 0.0176689, and the last 800 epoch of 0.000453576.

Mikhail also noted, that he could not find any patents on his topic, which may indicate that acoustic active noise cancellation is an open question in the field of ML.

4.2 Convolutional Recurrent Networks for ANC

Another approach to ML ANC was offered by Hao Zhang and DeLiang Wang from The Ohio State University[3]. In their work, they formulate ANC as a supervised learning problem, with the main task to solve the problem of noise absorption and cleaning the speech signal. Particular attention in their work is also paid to nonlinear ANC problems. Thus, they use a convolutional recurrent network(CRN) to estimate the real and imaginary spectrograms of a canceling signal from the reference signal. Let's look closer at this architecture:

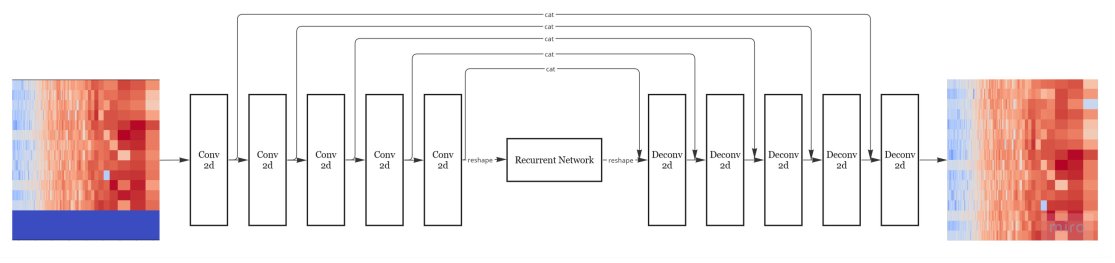

Figure 4.2 CRN architecture

The architecture includes several convolutional encoder layers, a 2-layer (encoder and decoder) recurrent, especially, LSTM network with group policy, and several decoder deconvolution layers to restore signal. Direct connections were also added to the network. Thus, during deconvolution, the result from the corresponding convolutional encoder layer is concatenated with the related same-size input for the decoder layer. In this way, the model can better track long-run dependencies. An important part of the CRN architecture is also that each convolutional and sweeping layer is causal, which means that no future information is used in generating the context. This approach has also been used in the Wavenet model.

Zhang and Wang use an STFT with a window length of 320 and a hop size of 160 to represent a 16 kHz sound waveform in their model. The main idea of their approach is that they try to solve the delay problem in the system by predicting the suppressing signal by several STFT windows in advance. So they take an initial input of length N, add to it M zero-padded windows at the beginning, and use now N windows in, together with these null windows, removing M windows from the end of the initial input. They tried different sizes of time M shifts: without it (0), 1 and 2 windows. You can see an example with padded windows at the bottom of input in Figure 4.2, where STFT is represented as *freq* x *time*. After training their system, they got good results, For example, they got NMSE for engine noises: -11.07 dB NMSE without shifting, -9.60 dB for 1, and -7.93 dB for 2 predicted windows.

Chapter 4 Summary

In this chapter we broke down 2 working examples of models for active noise reduction using Machine Learning. We have seen what types of neural networks are used to solve this kind of problem and their specifics. Also we received information about the possible problems in creating such a system. Now, with this knowledge in hand, we can move on to creating and training our own model for active noise reduction.

Chapter 5

Experiments

This part will focus on the practical part of our study. We will talk about the approach we chose, the changes we made to the architecture of the model, where and how we obtained and prepared the sound data, trained the model, and what results we obtained. Also, we will not leave aside the real application of such a model and will provide a general scheme of work of this system in real-time.

5.1 Selected model details

Looking at the success and relative complexity of Zhang and Wang's solution, we decided to choose their approach to create our trial active noise cancellation model. We take the CRN model, presented in the previous section, as the basis of the system. The solutions described below, however, do not replicate it completely. Thus, we want to check if it is possible to use simplifications in the system, for example, using GRU as a recurrent network with fewer parameters, compared to LSTM, without the implementation of group policy. To create causal convolutions and deployments, we first add padding of k zeros to both sides of the input, and after receiving an output, we cut off the last k elements to exclude the context of future windows in each step. While deconvolution, on the contrary, we delete the first k, since we need only the data to generate future windows. We are also trying to use so-called, depthwise convolutions, which apply a single filter to each channel of the input, independently. This means that the same filter is applied to all spatial locations of each channel. The result is a set of output channels that are linear combinations of the input channels, with each output channel corresponding to one filter. Depthwise convolutions are computationally more efficient than traditional convolutions because they have fewer parameters, and can extract features more efficiently, which is particularly useful in applications with limited computational resources, such as mobile or embedded devices, which is very useful in the future possible incorporation of the created system.

5.2 Collecting the data

For the better adaptation of our model to different noises, we decided to use groups of recordings, recorded in various environments to train our model. Based on urban noises, we chose three large groups:

- Construction noises
- Traffic noises

- The noise of people on a busy street

We collected this data from the NoiseFX sound library[23], using a tag search, converted it to lossless `.wav` format, and resampled it to 16kHz. The time lengths of all recordings for specific groups are presented below:

Construction noises	01:56:06
Traffic noises	02:15:06
Street noises	01:28:04

Table 5.1 Gruoups and their summary time durations

Since there is a skew in size in this dataset, the model will probably be a little better adapted to traffic noises. This allows us to customize the performance of the system, and tweak it to work in certain conditions. The training strategy we choose, which we will talk about in the future, should solve the problem of too much overfitting to a certain group.

5.3 Sound pre-processing details

As in Zhang and Wang's approach, we will use STFT to represent sound. It is worth saying that increasing the frequency resolution of the transform will probably require increasing the layers of parameters, which will slow down its speed. In this case, however, the model will predict longer windows, which gives hope that this increase in resolution will not affect the performance of the model. In our work, we have tried to increase the window size slightly to 400 by also increasing the number of model parameters. In this way, we try to reduce computational latency when using not very powerful hardware by increasing the length of the predicted windows and proportionally decreasing the parameters of the model, while using GRU instead of LSTM. We also increased the number of input windows to 16 by taking data from the first 13 current windows and adding 3 zero-padded windows at the beginning. Thus, we prepared our data set to predict the 16 current STFT windows by shifting the input to the front, by 3 time windows, from the beginning of the current windows, filling the incoming past ones on the left with zeros, when the last 3 windows are dropped out. Essentially, these null windows indicate the delay – the time when the model calculates the output and further delay in the loudspeaker. In this way, when setting the problem as $n \to n + 1$, the model predicts in general, more than 2 future windows and should better deal with the problem of delays.

5.4 Training strategy

In general, we have already described, on which pairs we train the model, but it is worth saying that we do not use STFT transformations for the reference signal, since we measure the error on the output after converting it to waveform. So, to create one train pair, we take 3000 samples of the reference single signal, which corresponds to 16 STFT windows with a length of 400 and a hop size of 200 for a 16 kHz sample. We perform the transformation only on the first 2400 samples and then add zeros at the beginning.

We assemble training pairs into batches of 8, keeping the order for each recording, and storing bathes of one sample together as one training entry. After preparing the training data, we randomly select 25% of the records from each group for the valid set, and then, in the same way, 25% of the valid set for the test after training.

Next, in the training itself, we use what is called cross-validation. It can be seen that this technique is used mainly for classification tasks. We chose it because we wanted to prevent overfitting of the model on one of the groups, as well as for general stabilization of training. So in each epoch, we take all combinations of 3 by 2 groups, concatenating its group's training data and shuffling their order. After that, we validate the model on the remaining set for each combination. At the end of the epoch, we use the overall mixed validation group, to evaluate the combined performance of the model on a given epoch. When the model Is trained, we check our model on all test set and for each separate group. separately, to not adjust the error to the validation data. Thus, our learning strategy combines the concepts of normal data partitioning and cross-validation.

5.5 Staging the experiment

To test the CRN architecture for noise reduction, we trained several models. In particular, as we said above, we try to use GRU networks instead of LSTM, and depthwise convolutions. Thus, we want to test in total 4 models: 2 of them with regular causal convolutions, with GRU and LSTM, 2 with depthwise causal convolutions, and also with these two different recurrent models. After training, we want to compare the performance of a given architecture with different components. We trained our models for max 12 epochs, using the strategy described above, so each group participates in training about 24 times, which corresponds to 24 epochs in a normal training strategy. We use the Adam optimizer because Adagrad, by virtue of its implementation, suffers from the problem of a decreasing learning speed, after several epochs. We also make dumps of the model at each training epoch to be able to choose the best model. As a loss function, we use MSE to train the model and NMSE in dB scale to check the performance. Some of the samples in our recording are very quiet and contain almost no sound, so we added a small constant ϵ equal to $2.2 \cdot 10^{-16}$, to the NMSE measure in the denominator, accordingly regularizing it. To some extent, this allows the model to be trained to behave correctly in a quiet environment, without creating unnecessary noise.

The tested models and the number of their parameters are shown in the table below:

Depthwise with GRU CRN	19727760
Normal with GRU CRN	20502624
Depthwise with LSTM CRN	26286480
Normal with LSTM CRN	27061344

Table 5.2 Models and their parameters number

As it should be, the lightest model is the one with depthwise convolutions and GRU as a recurrent network and the largest – with standard convolutions and LSTM.

5.6 Model evaluation

For testing, based on the results and learning metrics, we chose the best models from each group. We present their results on the testing set:

model / NMSE in dB	all groups	construction	city	traffic
Depthwise with GRU	−3.66	−3.27	−4.21	−3.66
Standard with GRU	−6.17	−5.7	−6.56	−6.36
Depthwise with LSTM	−3.88	−3.33	−4.52	−4.57
Standard with LSTM	**−6.77**	**−6.41**	**−6.90**	**−6.82**

Table 5.3 Models and their parameters number

5.7 Results discussion and further works

Expectedly, the model with the largest number of parameters, namely the CRN architecture with standard causal convolutions and LSTM, showed the best result. Its superiority, in general, is not so great, but a big difference in results on the construction noises set, in comparison with the corresponding GRU model. It gives us to understand that LSTM copes better with more complex and repetitive noises. However, the improvement obtained using the LSTM model is not proportional to the increased number of parameters, which means that using GRU is not a bad idea. As for the results of the lightweight models, they showed much worse results and are probably not the best solution for the ANC task.

In general, these results are more or less justified compared to the results of Wang and Zhang. We suspect several factors that may affect the obtained metrics:

- we predict 3 windows instead of 2;
- we use wider STFT windows, which decreases temporal and increases the frequency resolution of transformation;
- we do not implement group policy for recurrent networks;
- we have only about 6 hours of audio data.

All of these factors may ultimately influence the result and can be considered improvements for further work.

The fact that we used open-source data recorded from different devices deserves special attention. So, the recordings, in general, were slightly different, because different microphones have their own frequency sensitivity. Using a greater variety and quantity of data, as well as their better quality, can seriously affect the model in a positive way.

We also see an opportunity to improve the model in the possibility of using the newfangled Transformer networks with their Attention mechanism[24], instead of GRU and LSTM. Transformers are brilliant when working with serial data, for which they became State of the art in this area.

Unfortunately, we were also unable to test the system in a real environment, as this requires tests in a specially isolated sound lab and professional instrumentation to take measurements. However, the model is ready to work and requires no additional configuration, but only the correct connection to the input and output devices, Thus, we can continuously process samples from input in size of 2400 in a loop.

Conclusion

During our undergraduate studies, we delved into the problem of active noise suppression and explored the potential of applying machine learning techniques to address it. We began by studying the fundamental algorithms of LMS and FxLMS, which we discovered are based on machine learning principles. As we delved deeper, we learned about the basic concepts of machine learning and how neural networks operate, including the various types of architectures that are best suited for processing sound. We also examined the different types of sound representations and the metrics used to evaluate the quality of sound in machine learning models. To tackle the problem of active noise suppression, we utilized the approach developed by Zhang and Wang and created our own model based on CRN architecture. Thus, we trained four variants of the model, using different types of recurrent networks and depthwise convolutions. Our results were promising, with the best model achieving a minimum NMSE metric of -6.77 dB. These results provide a strong basis for further research in this area. We have also had great enjoyment dealing with this problem. We hope that this research will contribute to further studies in sound processing with ML and that our work will find its place and be useful to people. We are grateful for the opportunity to conduct this research and gain valuable experience in this field.

Bibliography

1. TAN, Ke; WANG, DeLiang. Complex Spectral Mapping with a Convolutional Recurrent Network for Monaural Speech Enhancement. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 6865–6869. Available from DOI: 10.1109/ICASSP.2019.8682834.
2. MAREKICH. *Active Noise Cancelling* [online]. 2023. [visited on 2023-05-11]. Available from: https://commons.wikimedia.org/w/index.php?curid=21697674. [File: Active_Noise_Reduction.svg, CC BY-SA 3.0].
3. ZHANG, Hao; WANG, DeLiang. A Deep Learning Approach to Active Noise Control. In: *Proc. Interspeech*. 2020, pp. 1141–1145. Available from DOI: 10.21437/Interspeech.2020-1768.
4. PAUL, Lueg. *US2043416A - Process of silencing sound oscillations - United States Patent and Trademark Office*. 1933. Available also from: https://image-ppubs.uspto.gov/dirsearch-public/print/downloadPdf/2043416.
5. FOGEL, Lawrence J. *US2866848A - Method of improving intelligence under random noise interference - United States Patent and Trademark Office*. 1954. Available also from: https://image-ppubs.uspto.gov/dirsearch-public/print/downloadPdf/2866848.
6. RYAN, L. Urquhart. *The Effects of Noise on Speech Intelligibility and Complex Cognitive Performance*. 2002. Available also from: https://hdl.handle.net/10919%5C%2F27111.
7. WIDROW, B.; HOFF, M. E. Jr. Adaptive Switching Circuits. *IRE WESCON Convention Record*. 1960, vol. 4, pp. 96–104. Available also from: https://www-isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf.
8. WIDROW, B; STEARNS, Samuel D.; BURGESS, John C. Adaptive Signal Processing edited by Bernard Widrow and Samuel D. Stearns. *The Journal of the Acoustical Society of America*. 1986, vol. 80, no. 3, pp. 991–992. ISSN 0001-4966. Available from DOI: 10.1121/1.393898.
9. HAYKIN, Simon. *Adaptive Filter Theory*. Prentice Hall, 1996. Prentice-Hall information and system sciences series. ISBN 9780133227604. Available also from: https://books.google.cz/books?id=l78QAQAAMAAJ.
10. LIU, Lichuan; GUJJULA, Shruthi; THANIGAI, Priya; KUO, Sen. Still in Womb: Intrauterine Acoustic Embedded Active Noise Control for Infant Incubators. *Advances in Acoustics and Vibration*. 2008, vol. 2008. Available from DOI: 10.1155/2008/495317.
11. WIDROW, B; SHUR, D; SHAFFER, S. On Adaptive Inverse Control, Record of the Fifteenth Asilomar Conference on Circuits, Systems and Computers. In: 1981, pp. 185–189.

12. MIRONOVA, T. *Aktivnoe gahsenie shuma v truboprovode.* Samara National Research University named after S.P. Korolev (Samara University), Samara University Press, 2017. Available also from: `http://repo.ssau.ru/handle/Metodicheskie-materialy/Aktivnoe-gashenie-shuma-v-truboprovode-Elektronnyi-resurs-metod-ukazaniya-71954`.

13. DAKE; MYSID [online]. 2023. [visited on 2023-05-11]. Available from: `https://commons.wikimedia.org/w/index.php?curid=1412126`. [File: Neural_network.svg, Vectorized by Mysid in CorelDraw on an image by Dake, CC BY-SA 3.0].

14. KLOUDA, Karel; VAŠATA, Daniel. *Data Mining: Neural Networks* [Online]. 2023. [visited on 2023-05-01]. Available from: `https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-11-cs-handout.pdf`.

15. DJORK-ARNÉ, Clevert; UNTERTHINER, Thomas; HOCHREITER, Sepp. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).* 2016. Available from arXiv: `1511.07289 [cs.LG]`.

16. DUCHI, John; HAZAN, Elad; SINGER, Yoram. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research.* 2011, vol. 12, pp. 2121–2159. ISSN 1532-4435. Available also from: `https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf`.

17. OLAH, Christopher. *Understanding LSTM networks* [online]. 2015-08. [visited on 2023-05-01]. Available from: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`.

18. KOZLOV, Semyon. *Deep learning on your fingers!* [Online]. 2019. [visited on 2023-05-01]. Available from: `https://dlcourse.ai/`.

19. NELSON, Joseph. *You might be resizing your images incorrectly.* Roboflow Blog, 2020. Available also from: `https://blog.roboflow.com/you-might-be-resizing-your-images-incorrectly/`.

20. OORD, Aaron van den; DIELEMAN, Sander; ZEN, Heiga; SIMONYAN, Karen; VINYALS, Oriol; GRAVES, Alex; KALCHBRENNER, Nal; SENIOR, Andrew; KAVUKCUOGLU, Koray. WaveNet: A Generative Model for Raw Audio. 2016. Available from arXiv: `1609.03499 [cs.SD]`.

21. BARANOV, Mikhail. Recurrent Neural Active Noise Cancellation - Towards Data Science. 2021. Available also from: `https://towardsdatascience.com/deep-active-noise-cancellation-e364ce4562d4`.

22. KETTUNEN, Eino. *Eino Kettusen savo-karjalaisia y.m. humoristisia lauluja: 9:s vihko.* Lieksa: Eino Kettunen, 1928.

23. NOISEFX, team. *Noise Library, an online collection of sounds.* [Online]. 2023. [visited on 2023-03-26]. Available from: `https://noisefx.ru`.

24. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. *Attention Is All You Need.* 2017. Available from arXiv: `1706.03762 [cs.CL]`.

Contents of the attached media

- `readme.txt` .... a brief description of the content of the medium
- `src`
  - `thesis` .... source form of the work in format LaTeX
  - `thesis.pdf` .... text of the thesis in PDF format
  - `impl` .... implementation source codes
    - `notebooks` .. jupyter notebooks with work done as well as their drafts with draft data
    - `audio_data` .... audio data source used for work
    - `tained_models` .... best trained models