SAPIENZA
UNIVERSITÀ DI ROMA

# Advancements in EEG Denoising and Artifact Removal using Transformers


Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Applied Computer Science and Artificial Intelligence

**Martina Doku**
ID number 1938629

Supervisor
Prof. Danilo Avola

Co-Supervisor
Prof. Daniele Pannone


Academic Year 2022/23

**Advancements in EEG Denoising and Artifact Removal using Transformers**
Bachelor's degree internship report. Sapienza University of Rome



This internship report has been typeset by LaTeX and the Sapthesis class.

Author's email: doku.1938629@studenti.uniroma1.it

*To the things I have left behind to get here,*
*because this milestone has given a meaning*
*to every sacrifice.*

## Abstract

This thesis delves into the application of machine learning techniques, specifically Transformers, for denoising EEG signals.
The first section introduces the problem, outlines the research questions, and provides an overview of fundamental concepts. The second section explores the historical context and current state of signal analysis, with a focus on EEG signal analysis and denoising methods. The third section delves into machine learning techniques employed in this study. The fourth section presents the experimental results, highlighting the significant achievements attained. Finally, the last section concludes the thesis, discussing the implications of the findings and suggesting avenues for future research.

# Acknowledgments

*This thesis would not have come to fruition without the unwavering support and assistance of numerous individuals, to whom I owe my deepest gratitude. Among them, I would like to express my heartfelt appreciation to the following:*

*First and foremost, I am profoundly grateful to my supervisor, Prof.Danilo Avola, for his invaluable guidance, constant support, and expertise throughout the entire development of this thesis. His mentorship has been instrumental in shaping the direction and quality of my research.*
*I would also like to extend my sincere thanks to Romeo Lanzino for his invaluable advice and the generous amount of time he dedicated to providing insights into my work. His input has greatly enriched the overall quality of this thesis.*

*To my parents, who have devoted their entire lives to working hard and providing me with the best education and opportunities, I am forever grateful. Your sacrifices and belief in me have been a constant source of motivation.*
*I would like to express my deepest gratitude to Virginia and Francesco for being my second family, offering the most genuine support despite the physical distance between us.*
*I am at a loss for words to express my gratitude to Lucrezia, who has been a constant presence in my life, standing by my side through every moment, both good and bad. My heartfelt appreciation also goes to my colleagues, whose dedication and achievements have served as a strong source of inspiration throughout this challenging journey.*

*Finally, I want to acknowledge the two most significant individuals that this university has bestowed upon me.*
*Giusy, you have been an exceptional friend, a precious treasure that fate has graciously granted me. I am forever indebted to you for your precious friendship and the immense light you have brought into my life.*
*Lastly, I want to express my deepest gratitude to my boyfriend, Dennis. You have been my rock during the most challenging moments, always believing in me, much more than I do, and providing genuine support. Thank you for being the most incredible partner I could ever ask for.*

*To all those mentioned above, and to those whose support may not be specifically acknowledged, I am deeply grateful for the role you have played in this journey. Your contributions have been immeasurable, and I am fortunate to have had your support throughout this years.*

# Contents

# Chapter 1

# Introduction

The chapter is built as follows: in the first section there is a brief introduction of the problem, the reasons that led to the choice of the topic and the outline of the thesis. In the second section there is the statement of the research questions. In the third section there is a brief description of the basic concepts.

## 1.1 Problem statement

Electroencephalography (EEG) is a non-invasive technique used to measure the electrical activity of the brain. EEG signals are still among the less explored types in the field of signal processing, despite their widespread use in clinical practice and research. In recent years, there has been a growing interest in developing denoising methods for EEG data to improve their quality and reliability.

The main objective of this thesis is to investigate and compare different EEG denoising methods, and to evaluate their performance in terms of signal quality, artifact removal, and preservation of underlying brain activity. Specifically, we will focus on the most recent machine learning techniques, advanced models like Transformers. We will also investigate the potential of combining different denoising methods to improve the performance of EEG denoising.

Overall, this thesis aims to provide a better understanding of the strengths and limitations of different EEG denoising methods, and to help researchers and clinicians make informed decisions when selecting the most appropriate denoising method for their EEG data analysis. By improving the quality of EEG signals, we can enhance our understanding of brain function and ultimately contribute to the development of more effective diagnostic and therapeutic tools for neurological disorders.

## 1.2 Research questions

The research questions are the following:

- How can we remove artifacts from EEG signals?

- How can we exploit the most recent machine learning techniques for EEG denoising?
- What are the strengths and limitations of these new methods?
- What are the performance of these new methods?

## 1.3 Basic concepts

In this section we will introduce the basic concepts that will be used in the thesis. The first part is dedicated to the EEG signal, the second part is dedicated to the EEG denoising.

### 1.3.1 EEG

When talking about EEG,in this thesis, we are referring to the electroencephalogram, in particular we are interested in the EEG waves. Electroencephalogram waves are the patterns of electrical activity that are recorded by EEG measurements. These waves have different frequencies and amplitudes, and they reveal different states of brain activity. We divide the EEG waves in 5 main categories depending on their frequency: Alpha, Beta, Theta, Delta and Gamma waves.[1]

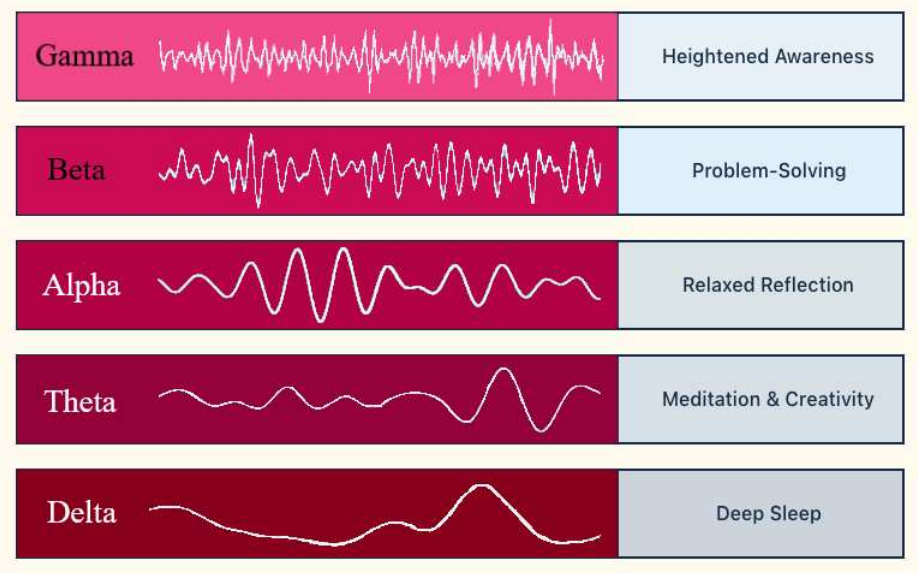


**Figure 1.1.** Types of EEG waves

Delta waves are typically observed during deep sleep and have a frequency of less than 4 Hz. Theta waves are associated to light sleep or drowsiness and have a frequency of $4-7$ Hz. Alpha waves correspond to the relaxed and awake state, with a frequency of $8-13$ Hz. Beta waves, on the other hand, are associated with active cognitive processing and have a higher frequency of $14-30$ Hz. Gamma waves have a frequency of $30-100$ Hz and are associated with higher cognitive functions such

as attention and memory.

It's important to remember that EEG waves are not distinct entities but represent a continuous spectrum of activity that can be influenced by various factors such as task demands, attention, and emotion. Interpreting EEG waves requires expertise and context since different patterns of EEG activity may reflect different states of brain activity depending on the individual and the experimental conditions. Furthermore, research has shown that EEG waves can be useful in clinical diagnosis and prognosis, as well as in the assessment of cognitive function and brain injury. Therefore, understanding the various EEG waves and their characteristics can provide valuable insights into brain function and activity.

### 1.3.2 EEG denoising

EEG signals can be influenced by various factor that alter the real waves originated from neural activities, those factors are defined as artifacts. The artifacts can be classified as[2]:

- **intrinsic artifacts:** artifacts that depend on physiological sources, such as ocular artifacts (EOG) that come from eye movement and blinking, muscle artifacts (EMG) and cardiac artifacts (ECG)
- **extrinsic artifacts:** artifacts generated from external electromagnetic such as power line noise sources.

Denoising of EEG data is an essential task to be able to work on data and to extract meaningful information from it. The denoising process is complex and it does lead to different level of quality of the data depending on the method used, the quality of the data and the type of artifact.

There are several challenges[3] related both to single methods characteristic and general artifact removal. Some methods are computationally expensive and require a lot of time to be applied, some other methods are not able to remove all kind of artifacts and some require a lot of data to be applied. On a general level, there is the problem of the lack of a standard method to evaluate the quality of the denoised data and the EEG applications are not yet fully commercial, so there hasn't been a sufficient investment in hardware and software to make the denoising process easier.

However the main goal of latest studies is to find a method that can denoise from all kind of artifacts and that can be used in a flexible and fast way, to accomodate the needs of all the different EEG applications.

# Chapter 2

# Literature review

In this chapter we will present the main methods used for the denoising of EEG data. In the first part we will focus on the methods used for the removal of artifacts from signals in general, in the second part we will present the ones specifically deveelopped for the denoising of EEG data and in the third part we will explore the latest, machine learning related, methods.

## 2.1 Signal analysis

In this section we will present the basic techniques used for the analysis of signals that constituted a base for the ones developed for the analysis of EEG data.

### 2.1.1 Fourier transform

The first method used for the removal of artifacts from signals is the Fourier transform[4]. The Fourier transform is a mathematical tool used used to decompose a signal into sine and cosine functions, that are used as basis functions for the original signal. It is defined as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{2.1}$$

where $f(t)$ is the signal and $\omega$ is the frequency. The Fourier transform of a signal is a complex number, which can be decomposed into its real and imaginary parts.

### 2.1.2 Fast Fourier transform

The Fourier transform is a very useful tool for the analysis of signals, but it is computationally expensive. The Fast Fourier transform (FFT)[5] is an algorithm that is used to calculate the Fourier transform of a signal in a shorter time: it reduces the time nedded to Nlog (N), obtaining a speed-up of a factor of N/log (N).

### 2.1.3 Short Time Fourier Transform

The Short Time Fourier Transform (STFT)[6][7] is a method used to calculate the Fourier transform of a signal. It is used with non stationary signals to find the

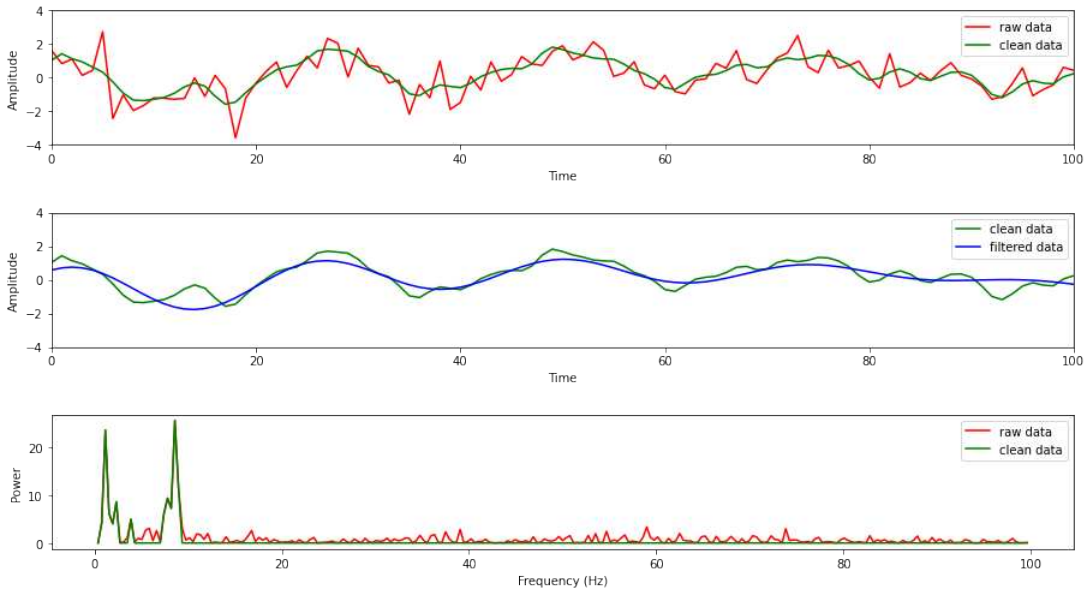


**Figure 2.1.** Result of the FFT of a signal

frequency components of a signal over time performing a Fourier transform on each time window. The STFT equation is defined as:

$$S(\tau) = s(t) \cdot h(t - \tau) \tag{2.2}$$

where $s(t)$ is the signal, $h(t)$ is the window function and $\tau$ is the time.
The STFT can be used to find the frequency components of a signal:

$$S(\omega) = \frac{1}{2\pi} \cdot \int_{-\infty}^{\infty} s(\tau) \cdot h(t - \tau) e^{-i\omega t} d\tau \tag{2.3}$$

where $s(\tau)$ is the signal, $h(t)$ is the window function and $\omega$ is the frequency.

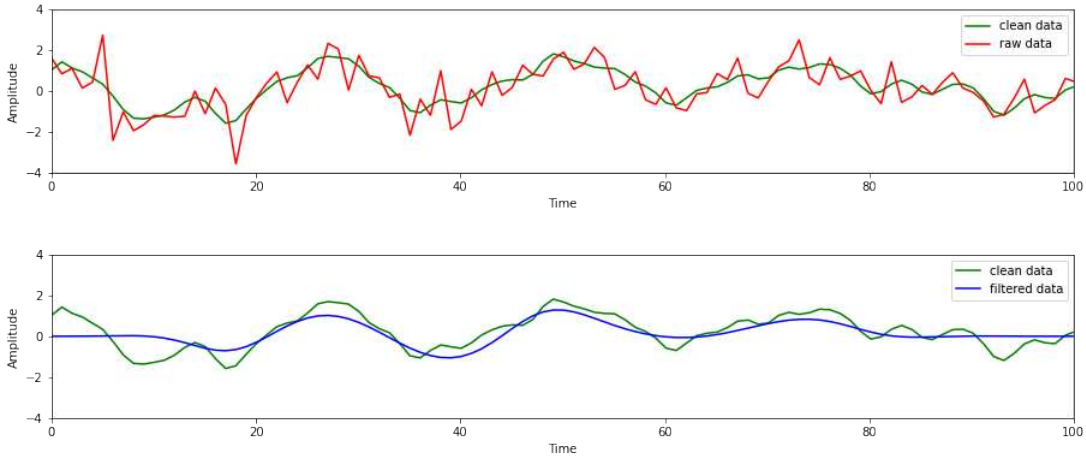


**Figure 2.2.** Result of the STFT of a signal

### 2.1.4 Wigner-Ville distribution

The Wigner-Ville distribution (WVD)[8] is a method used to provide the description of a signal in the time-frequency domain with higher resolution. It is described by the following equation:

$$W(\tau,\omega) = \int_{-\infty}^{\infty} s(t-\tau)\cdot s^*(t-\tau)e^{-2\pi\omega\tau}d\tau \tag{2.4}$$

where $s(t)$ is the signal, $\tau$ is the time and $\omega$ is the frequency. The main problem of the WVD is that it introduces the so called cross terms. In fact, for a signal

$$x(t) = x_1(t) + x_2(t) \tag{2.5}$$

the corresponding WVD is:

$$WVD(t,f) = WVDx_1(t,f) + WVDx_2(t,f) + 2Re[WVDx_1x_2(t,f)] \tag{2.6}$$

where $2Re[WVDx_1x_2(t,f)]$ is the cross term which is removable at the expense of a loss of resolution.

### 2.1.5 Wavelet transform

The wavelet transform[9] is a method used to find the frequency components of a signal over time. It is used with non stationary signals and it provides a better time-frequency resolution than the STFT and the WVD since it gives a better simultaneous time-frequency localization.
It expresses the signal as a linear combination of mother wavelets. The wavelet transform is defined as:

$$W(\tau,\omega) = \int_{-\infty}^{\infty} s(t-\tau)\cdot \psi(t-\tau)e^{-2\pi\omega\tau}d\tau \tag{2.7}$$

where $s(t)$ is the signal, $\tau$ is the time, $\omega$ is the frequency and $\psi(t)$ is the wavelet function.
A possible choice for the wavelet function is the Morlet wavelet:

$$\psi(t) = \frac{1}{\sqrt{2c\pi}}\cdot e^{-\frac{t^2}{2c^2}}\cdot e^{i\omega_0 t} \tag{2.8}$$

where $\omega_0$ is the central frequency of the wavelet and $c$ is the width of the wavelet.

### 2.1.6 Matching Pursuit

The Matching Pursuit (MP)[11] is a greedy algorithm used to approximate a signal with a linear combination of basis functions called time-frequency atoms, selected from a dictionary of functions.
A general family of time-frequency atoms can be generated by scaling, translating and modulating a single window function $g(t)$ as follows:

$$g_\gamma(t) = \frac{1}{\sqrt{s}}\cdot g(\frac{t-u}{s})\cdot e^{i\xi t} \tag{2.9}$$

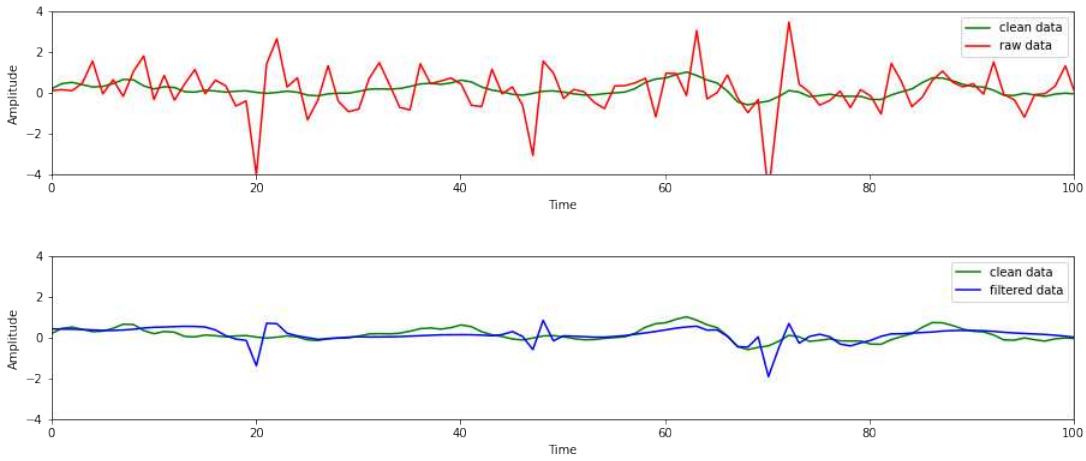


**Figure 2.3.** Result of the wavelet transform decomposition of a signal

where $\gamma$ is the tuple (s, u, $\xi$) $s$ is the scale, $u$ is the translation and $\xi$ is the frequency modulation.
The algorithm aims to find a set of atoms $\gamma_1, \gamma_2, \ldots, \gamma_N$ that best approximate the signal $f(t)$, i.e.:

$$f(t) \approx \sum_{i=1}^{N} a_i \cdot g_\gamma(t) \tag{2.10}$$

where $a_i$ are the expansion coefficients.
The vector f can be decomposed into

$$f(t) =< f, g_\gamma > g_\gamma + Rf(t) \tag{2.11}$$

where $R_f$ is the residual vector after approximating f in the direction of $g_\gamma$.
MP iteratively decomposes the residue $R_f$ by projecting it on a vector of D that matches $R_f$ at best.

## 2.2 EEG denoising

The denoising of EEG signals is performed using techniques developed from the field of signal processing, but more suited to the specific characteristics of EEG signals and artifacts. Here we present the most common methods.

### 2.2.1 Regression

The regression method[12], considers the EEG signal to be a linear combination of the artifacts and the clean EEG signal. In particular, the EEG signal is modeled as:

$$EEG = \alpha \cdot EMG + \beta \cdot EOG + \gamma \cdot ECG + \delta \cdot CEEG \tag{2.12}$$

where $EEG$ is the registered EEG signal, $EMG$ is the muscle artifact , $EOG$ is the ocular artifact, $ECG$ is the cardiac artifact and $CEEG$ is the clean EEG signal.

The regression coefficients $\alpha, \beta, \gamma, \delta$ are estimated using the least squares method. The regression method is simple and fast, but it doesn't take into account the temporal correlation between the artifacts and the EEG signal, the so called cross terms. Furthermore it requires the knowledge of the artifacts signals.

### 2.2.2 Blind source separation

Blind source separation (BSS)[13] is a class of methods used to separate a mixture of signals into their individual components. There are several BSS algorithms, all based on an unsupervised component separation. The most common BSS algorithms are the Principal Component Analysis (PCA), the Independent Component Analysis (ICA) and the Canonical Correlation Analysis (CCA).

#### Principal Component Analysis

The PCA[14] is a method used to find a linear transformation of the mixture signals that maximizes the variance of the transformed signals. First, data are standardized by subtracting the mean and dividing by the standard deviation.

$$\mathbf{Z} = (\mathbf{X} - \mathbf{m})/\mathbf{s} \tag{2.13}$$

Then, the covariance matrix is computed.

$$\mathbf{C} = \frac{1}{n-1}\mathbf{Z}^T\mathbf{Z} \tag{2.14}$$

The eigenvectors of the covariance matrix are the principal components of the data while the eigenvalues are the variances of the principal components.

$$\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{2.15}$$

The principal components found by projecting x onto those perpendicular basis vectors are uncorrelated, and their directions orthogonal. This scheme is very efficient in redundancy reduction, and can be said to maximize the amount of information spanned by a subset of dimensions of the initial vector.

#### Independent Component Analysis

The ICA[15] is a method used to find a linear transformation of the mixture signals that maximizes the non-Gaussianity of the transformed signals. It consists in decomposing the signal in Independent components and discarding the ones containing artifacts. In blind source separation, the original independent sources are assumed to be unknown, and we only have access to their weighted sum.[16]
The signal is modeled as:

$$x(t) = \sum_{i=1}^{N} a_i \cdot s_i(t) \tag{2.16}$$

where $x(t)$ is the mixture signal, $s_i(t)$ is the $i$-th source signal and $a_i$ is the mixing coefficient.
The ICA algorithm aims to find the source signals $s_i(t)$ and the mixing coefficients $a_i$ that maximize the non-Gaussianity of the mixture signal $x(t)$. It is based on the assumption that the sources are statistically independent.

#### Canonical Correlation Analysis

The CCA[17] is a method used to find a linear transformation of the mixture signals that maximizes the correlation between the transformed signals. More specifically, CCA identifies two sets of variables, X and Y, and finds linear combinations of X and Y that are maximally correlated. These linear combinations are called canonical variates. The first canonical variate is the linear combination of X and Y that has the highest correlation, and each subsequent canonical variate is the linear combination that has the highest correlation subject to being orthogonal to the previous canonical variates.

### 2.2.3 Empirical Mode Decomposition

The EMD[18] is a method used to decompose a signal into a set of intrinsic mode functions (IMFs). The EMD is a data-driven method that does not require any prior knowledge of the signal.
It decomposes a signal into a finite number of intrinsic mode functions (IMFs), which are functions that capture the local behavior of the signal. EMD is based on the concept of sifting, which involves iteratively extracting the local maxima and minima of the signal to generate IMFs. The basic steps of EMD are as follows:

- Given a signal $x(t)$, find all of its local maxima and minima.
- Interpolate between the local maxima and minima to create an upper and lower envelope for the signal. This step effectively eliminates the high-frequency components of the signal and captures its slowly varying behavior.
- Calculate the mean of the upper and lower envelopes to obtain a first IMF, $c_1(t)$.
- Subtract $c_1(t)$ from the original signal to obtain a new signal, $r_1(t) = x(t) - c_1(t)$.
- Repeat steps $1-4$ on the residual signal $r_1(t)$ to obtain the second IMF, $c_2(t)$.
- Continue this process until a stopping criterion is met, such as the number of IMFs or the amplitude of the residual signal falling below a certain threshold.

The resulting IMFs are functions that oscillate around zero with a characteristic scale and capture the local behavior of the signal at different scales. The final residual signal is a monotonic function that represents the long-term trend of the signal.

### 2.2.4 Filtering techniques

#### Adaptive filtering

Adaptive filtering[19] is a method used to estimate the unknown input signal from the noisy output signal. It is based on the assumption that the input signal is a linear combination of the unknown input signal and the noise.
The basic steps of adaptive filtering are as follows:

- Given a noisy signal $y(t)$, estimate the unknown input signal $x(t)$.
- Initialize the filter coefficients $w_0$.
- For each sample $y(t)$, compute the error signal $e(t)$.
- Update the filter coefficients $w(t)$.
- Repeat steps 2−4 until a stopping criterion is met.

The error signal is defined as the difference between the noisy signal and the estimated input signal. The filter coefficients are updated according to the following equation:

$$w(t+1) = w(t) + \mu \cdot e(t) \cdot x(t) \tag{2.17}$$

where $\mu$ is the step size.

#### Wiener filtering

Wiener filtering[20] is a method used to estimate the unknown input signal from the noisy output signal. It is based on the assumption that the input signal is a linear combination of the unknown input signal and the noise.
The basic steps of Wiener filtering are as follows:

- Given a noisy signal $y(t)$, estimate the unknown input signal $x(t)$.
- Compute the power spectral density (PSD) of the noise signal $n(t)$.
- Compute the PSD of the input signal $x(t)$.
- Compute the PSD of the output signal $y(t)$.
- Compute the Wiener filter coefficients $w(t)$.
- Apply the Wiener filter to the noisy signal $y(t)$ to obtain the estimated input signal $x(t)$.

The Wiener filter coefficients are computed according to the following equation:

$$w(t) = \frac{S_x(t)}{S_x(t) + S_n(t)} \tag{2.18}$$

where $S_x(t)$ is the PSD of the input signal $x(t)$ and $S_n(t)$ is the PSD of the noise signal $n(t)$.

## 2.3 Machine learning

The latest approaches involve the use of machine learning techniques to denoise EEG signals. Machine learning is a subfield of artificial intelligence that focuses on the development of computer programs that can learn from data. In the following chapter we will discuss the different architectures of neural networks that have been used in previous works and that will be exploited in the proposed approach.

### 2.3.1 Fully connected neural network

Fully Connected Neural Networks, also known as Feedforward Neural Networks or Multilayer Perceptrons, are the the most basic type of neural networks. They consist of neurons[6.1.1], organized in layers and each of the neurons in a layer is connected to all the ones in the previous and subsequent layers. These connections are weighted, and each neuron computes a weighted sum of its inputs, passes this sum through an activation function[6.1.3], and then forwards the result to the next layer.

The architecture of a Fully Connected Neural Network consists of one or more layers of neurons, including an input layer, one or more hidden layers, and an output layer. The input layer receives the data to be processed, and each neuron in this layer represents a feature of the input. The hidden layers perform complex computations on the input data, and the output layer produces the final output of the network. The number of neurons in the input and output layers is determined by the size of the input and output data, while the number of neurons in the hidden layers is a hyperparameter that can be tuned to improve the performance of the network.

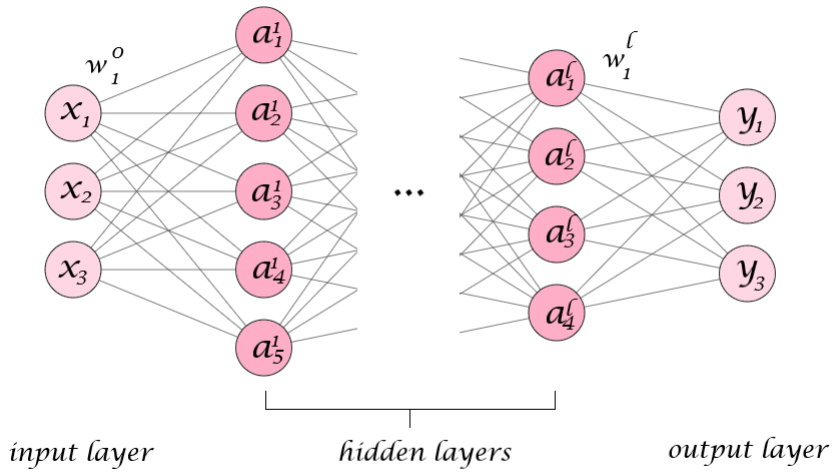


**Figure 2.4.** Fully connected neural network

The operation of a fully connected neural network can be described mathemati-

cally using the following equations.

#### Linear transformation

Each neuron in the hidden layers and output layer performs a linear transformation of its inputs, which is a weighted sum of the outputs of the neurons in the previous layer. The linear transformation of neuron i in layer l can be expressed as:

$$z_i^l = \sum_{j=1}^{n^{l-1}} w_{ij}^l a_j^{l-1} + b_i^l \tag{2.19}$$

where $n^{l-1}$ is the number of neurons in the previous layer, $w_{ij}^l$ is the weight of the connection between neuron j in layer l-1 and neuron i in layer l, $a_j^{l-1}$ is the output of neuron j in layer l-1, $b_i^l$ is the bias term of neuron i in layer l, and $z_i^l$ is the pre-activation value of neuron i in layer l.

#### Activation function

The pre-activation values of each neuron are passed through an activation function to introduce non-linearity into the network. Commonly used activation functions include sigmoid, tanh, ReLU, and softmax. The activation function of neuron i in layer l can be expressed as:

$$a_i^l = \sigma(z_i^l) \tag{2.20}$$

where $\sigma$ is the activation function.

#### Output computation

The output of the network is produced by the output layer, which applies a final activation function to the pre-activation values of its neurons. The activation function of the output layer depends on the task the network is designed to perform. The output of neuron i in the output layer can be expressed as:

$$\hat{y}_i = \sigma(z_i^L) \tag{2.21}$$

where L is the index of the output layer, $\sigma$ is the output activation function, and $\hat{y}_i$ is the predicted value of the i-th output.

The network is trained by adjusting the weights and biases using backpropagation [6.1.4] with respect to a loss function [6.1.2].

### 2.3.2 Convolutional neural network

Convolutional Neural Networks (CNNs) are a type of artificial neural network architecture commonly used in deep learning for multidimensional data analysis. Unlike fully connected neural networks, CNNs take into account the spatial structure of input data, such as images, by using convolutional layers that apply filters to local regions of the input.

The architecture of a CNN consists of one or more convolutional layers paired with their activation functions, followed by one or more fully connected layers. The convolutional layers perform feature extraction by applying filters to local regions of the input data, producing feature maps that highlight different aspects of the input data. The fully connected layers perform classification or regression on the feature maps produced by the convolutional layers.

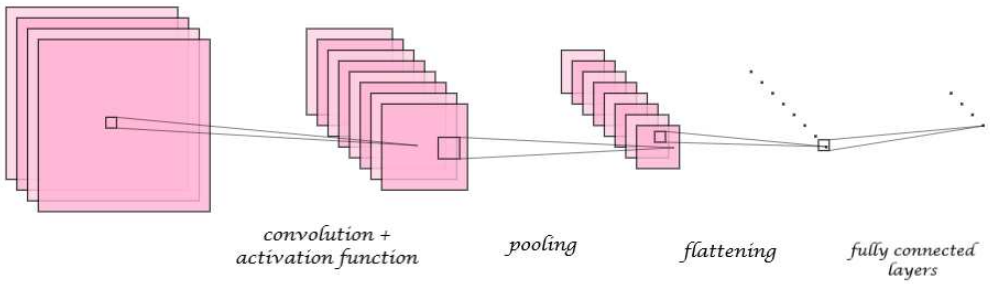


**Figure 2.5.** Convolutional neural network

The operation of a CNN can be described mathematically using the following equations:

### Convolutional layer

In the convolutional layers, a set of filters or kernels is applied to local regions of the input data to produce feature maps. The convolution operation can be expressed as:

$$z_{i,j,k} = \sum_{l=1}^{d_{in}} \sum_{m=1}^{f} \sum_{n=1}^{f} x_{i+m-1,j+n-1,l} w_{m,n,l,k} + b_k \tag{2.22}$$

where $x$ is the input data, $w$ is the filter, $b$ is the bias, $d_{in}$ is the number of input channels, $f$ is the size of the filter, $z$ is the output feature map, and $i, j, k$ are the indices of the feature map.

### Pooling layer

The pooling layer downsamples the output of the convolutional layer, reducing the size of the feature maps and introducing translation invariance. Commonly used pooling operations include max pooling and average pooling that perform a max or average operation on local regions of the input data, respectively.

### Flattening

The output of the pooling layer is flattened into a single vector, with dimension equal to the number of neurons of the input layer of the fully connected part of the

network, to which it is passed.

### Fully connected layer

The fully connected layers perform classification or regression on the feature maps produced by the convolutional and pooling layers. The operation of a fully connected layer is similar to that of a fully connected neural network.

## 2.3.3 Long short-term memory

Long Short-Term Memory (LSTM) Networks are a type of recurrent neural network (RNN) specifically designed to address the vanishing gradient problem[6.1.5] in traditional RNNs. LSTMs have a memory cell that is able to maintain information over long periods of time, and they use three gates − the input gate, the forget gate, and the output gate − to regulate the flow of information into and out of the memory cell. Here are the key equations that govern the behavior of an LSTM:

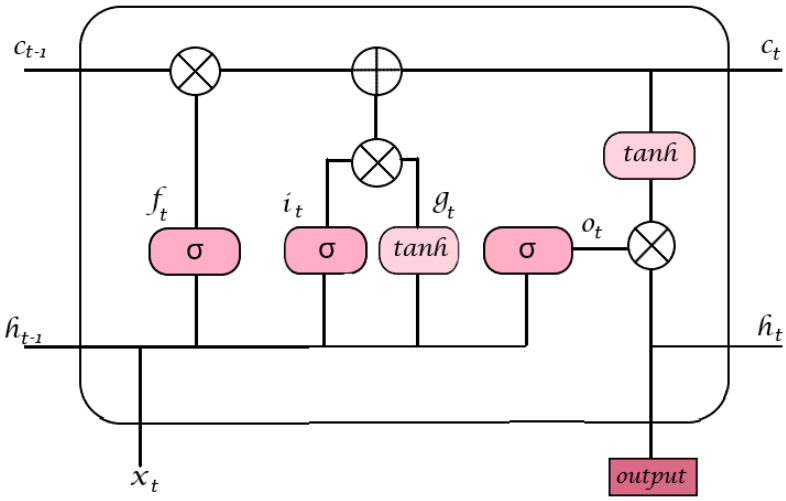


**Figure 2.6.** Long Short Term Memory Architecture

### Input gate

The input gate determines how much of the new input should be added to the memory cell. The equation for the input gate is:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.23}$$

where $i_t$ is the input gate activation vector, $x_t$ is the input vector at time t, $h_{t-1}$ is the hidden state vector from the previous time step, $W_i$ is the weight matrix for the input gate, and $b_i$ is the bias vector for the input gate.

**Forget gate**

The forget gate determines how much of the previous memory cell state should be retained. The equation for the forget gate is:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.24}$$

where $f_t$ is the forget gate activation vector, $x_t$ is the input vector at time t, $h_{t-1}$ is the hidden state vector from the previous time step, $W_f$ is the weight matrix for the forget gate, and $b_f$ is the bias vector for the forget gate.

**Candidate memory cell state**

The memory update combines the input with the previous memory cell state to create a new memory cell state. The equation for the memory update is:

$$g_t = \tanh(W_g[h_{t-1}, x_t] + b_g) \tag{2.25}$$

where $g_t$ is the candidate memory cell state, $x_t$ is the input vector at time t, $h_{t-1}$ is the hidden state vector from the previous time step, $W_g$ is the weight matrix for the memory update, and $b_g$ is the bias vector for the memory update.

**Output gate**

The output gate determines how much of the new memory cell state should be output. The equation for the output gate is:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.26}$$

where $o_t$ is the output gate activation vector, $x_t$ is the input vector at time t, $h_{t-1}$ is the hidden state vector from the previous time step, $W_o$ is the weight matrix for the output gate, and $b_o$ is the bias vector for the output gate.

**Hidden state update**

The new memory cell state is combined with the output gate to create the new hidden state. The equation for the hidden state update is:

$$h_t = o_t \tanh(c_t) \tag{2.27}$$

where $h_t$ is the new hidden state, $o_t$ is the output gate activation vector, $c_t$ is the new memory cell state, and tanh is the hyperbolic tangent function.

These equations allow an LSTM network to selectively forget or remember information over time, making it well-suited for modeling sequences of data where long-term dependencies are important

### 2.3.4 Gated recurrent unit

The GRU is a type of recurrent neural network architecture that aims to address the problem of vanishing gradients in traditional RNNs. It uses gating mechanisms to control the flow of information between the hidden states of the network, allowing it to selectively retain or discard information over time.
The GRU has two gates: the reset gate and the update gate. The operations performed by these gates are described below.

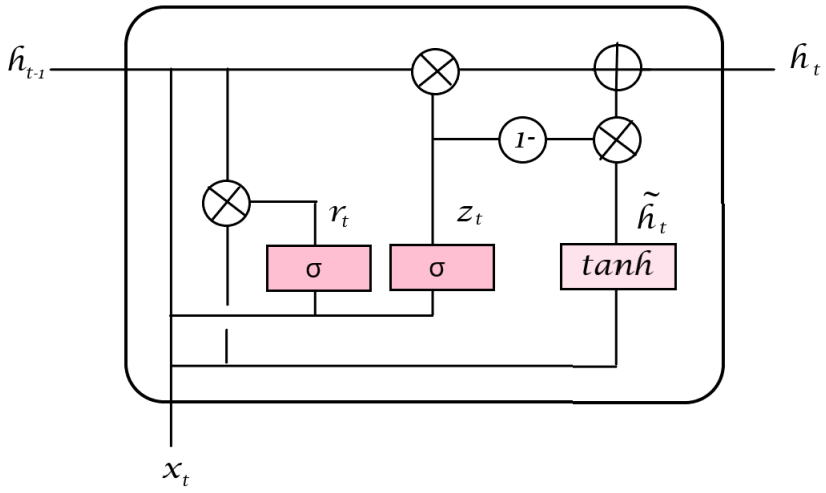


**Figure 2.7.** Gated Recurrent Unit

#### Reset gate

The reset gate controls how much of the previous hidden state should be forgotten. The equation for the reset gate is:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.28}$$

where $x_t$ is the current input at time $t$, $h_{t-1}$ is the previous hidden state, $W_r$, $U_r$ are learned weight matrices, and $b_r$ is a learned bias vector. $\sigma$ is the sigmoid activation function.

#### Update gate

The update gate controls how much of the current input should be incorporated into the new hidden state. The equation for the update gate is:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.29}$$

where $x_t$ is the current input at time $t$, $h_{t-1}$ is the previous hidden state, $W_z$, $U_z$ are learned weight matrices, and $b_z$ is a learned bias vector. $\sigma$ is the sigmoid activation function.

**Candidate activation**

The candidate activation is a new candidate hidden state that can be incorporated into the new hidden state based on the reset gate. The equation for the candidate activation is:

$$\tilde{h_t} = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{2.30}$$

where $x_t$ is the current input at time $t$, $h_{t-1}$ is the previous hidden state, $W_h$, $U_h$ are learned weight matrices, and $b_h$ is a learned bias vector. tanh is the hyperbolic tangent activation function and where $\odot$ is the element-wise multiplication (Hadamard product).

**Hidden state update**

The new hidden state is a combination of the previous hidden state and the candidate activation, based on the update gate. The equation for the hidden state update is:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h_t} \tag{2.31}$$

where $\odot$ is the element-wise multiplication (Hadamard product) and tanh is the hyperbolic tangent activation function.

### 2.3.5 Bidirectional GRU

A bidirectional GRU (Gated Recurrent Unit) is a type of recurrent neural network (RNN) that processes input sequences in both forward and backward directions simultaneously. It consists of two GRU subnetworks: one processes the sequence from the beginning to the end, and the other processes it in reverse. This bidirectional nature allows the GRU to capture information from both past and future contexts of each element in the sequence.
The forward GRU computes hidden states in the forward direction, while the backward GRU computes hidden states in the reverse direction. Each GRU subnetwork has its own set of weights and biases.
The calculations for the forward and backward GRU can be described as follows:
Forward GRU:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.32}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.33}$$

$$\tilde{h}t = \tanh(W_h x_t + U_h(r_t \odot ht - 1) + b_h) \tag{2.34}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.35}$$

Backward GRU:

$$\tilde{z}t = \sigma(W_z x_t + U_z \hat{h}t + 1 + b_z) \tag{2.36}$$

$$\tilde{r}t = \sigma(W_r x_t + U_r \hat{h}t + 1 + b_r) \tag{2.37}$$

$$\tilde{\tilde{h}}_t = \tanh(W_h x_t + U_h(\tilde{r}t \odot \hat{h}t + 1) + b_h) \tag{2.38}$$

$$\hat{h}_t = (1 - \tilde{z}t) \odot \hat{h}t + 1 + \tilde{z}_t \odot \tilde{\tilde{h}}_t \tag{2.39}$$

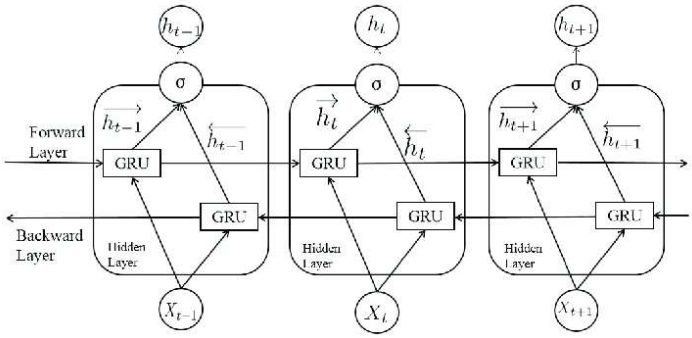


**Figure 2.8.** Bidirectional GRU Architecture

Here, $x_t$ represents the input at time step $t$, $h_t$ denotes the hidden state at time step $t$ in the forward direction, $\hat{h}_t$ represents the hidden state at time step $t$ in the backward direction, $z_t$ and $\tilde{z}_t$ are the update gate values, $r_t$ and $\tilde{r}_t$ are the reset gate values, and $\tilde{h}_t$ and $\tilde{\tilde{h}}_t$ represent intermediate hidden states.
The final hidden state of the bidirectional GRU at each time step can be obtained by concatenating the forward and backward hidden states:

$$\overleftrightarrow{h}_t = [h_t; \hat{h}_t]$$

### 2.3.6 Transformer

The Transformer is a neural network architecture designed for natural language processing tasks. It was introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017[21], and has since become a popular choice for many NLP and sequential data tasks.

The Transformer is a type of encoder-decoder architecture, where the encoder is used to encode the input sequence into a fixed-length vector, and the decoder is used to decode the vector into an output sequence, both encoder and decoder parts are composed of a stack of one of more layers. Each layer is composed of a multi-head self-attention mechanism, followed by a position-wise fully connected feed-forward network.
We will now go over the different components of the Transformer in more detail.

#### Embedding

The embedding layer precedes the encoder (and decoder) parts, it is used to convert the input sequence (or output sequence) into a vector representation. It is composed of two parts: an embedding matrix, and a positional encoding matrix. The embedding matrix is used to convert the input sequence into a vector representation, and the positional encoding matrix is used to encode the position of each token in the sequence. The embedding layer is defined as follows:

$$E = W_e X + W_p \tag{2.40}$$

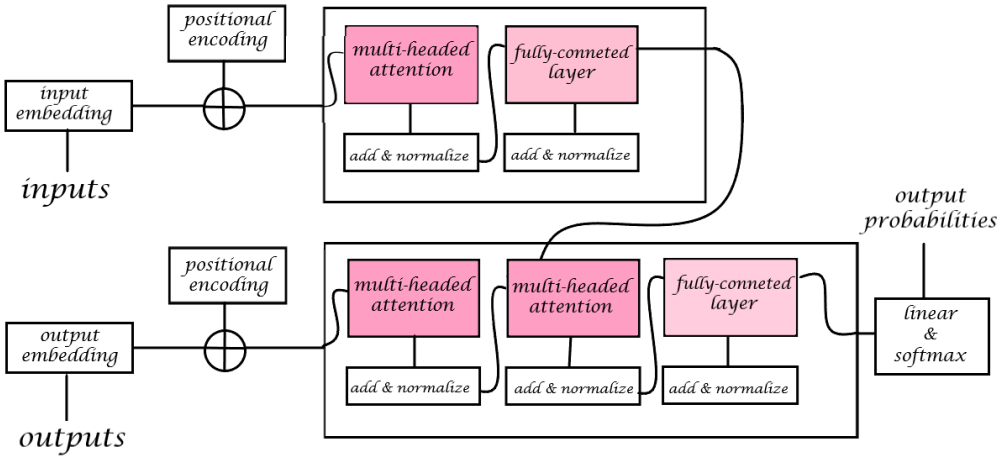


**Figure 2.9.** Transformer Architecture

where $X$ is the input sequence, $W_e$ is the embedding matrix, $W_p$ is the positional encoding matrix, and $E$ is the embedding vector.

### Self-attention

The self-attention mechanism allows the model to attend to different parts of the input sequence. Given an input sequence, we define the query, key, and value vectors as follows:

$$Q = W_q X \tag{2.41}$$

$$K = W_k X \tag{2.42}$$

$$V = W_v X \tag{2.43}$$

where $X$ is the input sequence, $W_q$, $W_k$, and $W_v$ are the weight matrices for the query, key, and value vectors, respectively, and $Q$, $K$, and $V$ are the query, key, and value vectors, respectively.

Those values are then used to compute the attention scores as follows:

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{2.44}$$

where $A$ is the attention matrix, $d_k$ is the dimensionality of the key vectors, and softmax is the softmax function. The attention matrix is then used to compute the output vectors as follows:

$$O = AV \tag{2.45}$$

where $O$ is the output matrix.

### Multi-head attention

The multi-head attention mechanism allows the model to attend to different aspects of the input sequence simultaneously. It does this by computing $h$ different sets of

query, key, and value vectors, each of which is obtained by projecting the input vectors into $d_k$, $d_k$, and $d_v$-dimensional spaces, respectively. The resulting $h$ sets of output vectors are concatenated and linearly transformed to produce the final output.
The equations for the multi-head attention mechanism can be written as follows:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)W^O$$

where:

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

where $W_i^Q$, $W_i^K$, and $W_i^V$ are learned projection matrices for the $i$-th head, each of which has dimensions $d_{model} \times d_k$, $d_{model} \times d_k$, and $d_{model} \times d_v$, respectively. $W^O$ is a learned projection matrix that maps the concatenated output of the attention heads back to the original dimensionality of the input.

### Add and normalize

The add and normalize layer is used to add the input to the output of the multi-head attention mechanism, and then normalize the result. The equations for the add and norm layer can be written as follows:

$$\begin{aligned} \text{AddNorm}(x, y) &= \text{LayerNorm}(x + y) \\ \text{LayerNorm}(x) &= \text{Norm}(x + \text{Sublayer}(x)) \end{aligned}$$

where $x$ is the input vector, $y$ is the output of the multi-head attention mechanism, and $\text{Sublayer}(x)$ is the sublayer that is being normalized.

### Position-wise feed-forward network

The position-wise feed-forward network is a simple two-layer neural network that operates on each position of the input sequence independently. It is applied to each output vector of the self-attention mechanism, and has the effect of transforming the vector into a higher-dimensional space, followed by a projection back into the original dimensionality.
The equations for the position-wise feed-forward network can be written as follows:

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2 \tag{2.46}$$

In this equation, $x$ is the input vector of size $d_{model}$, $W_1$ and $W_2$ are learned weight matrices of sizes $d_{model} \times d_{ff}$ and $d_{ff} \times d_{model}$, respectively, $b_1$ and $b_2$ are learned bias vectors of size $d_{ff}$ and $d_{model}$, respectively, and $\sigma$ is the activation function. The output of the position-wise feed-forward network is also a vector of size $d_{model}$.

### Linear and softmax

The linear and softmax layer is used to convert the output of the decoder into a probability distribution over the target vocabulary. The equations for the linear

and softmax layer can be written as follows:

$$\text{Linear}(x) = xW + b$$
$$\text{Softmax}(x) = \text{softmax}(x)$$

where $x$ is the input vector, $W$ is the learned weight matrix, and $b$ is the learned bias vector.

# Chapter 3

# Methodology

## 3.1 Data

The data used in this thesis are the EEG signals of the EEGdenoiseNet dataset[22]. It is a comprehensive benchmark dataset that proves to be an exemplary resource for training and evaluating deep learning-based EEG denoising models. It is composed of 4514 clean EEG epochs, 3400 EOG epochs, and 5598 EMG epochs, making it an inclusive and extensive dataset that caters to the needs of researchers and practitioners alike.

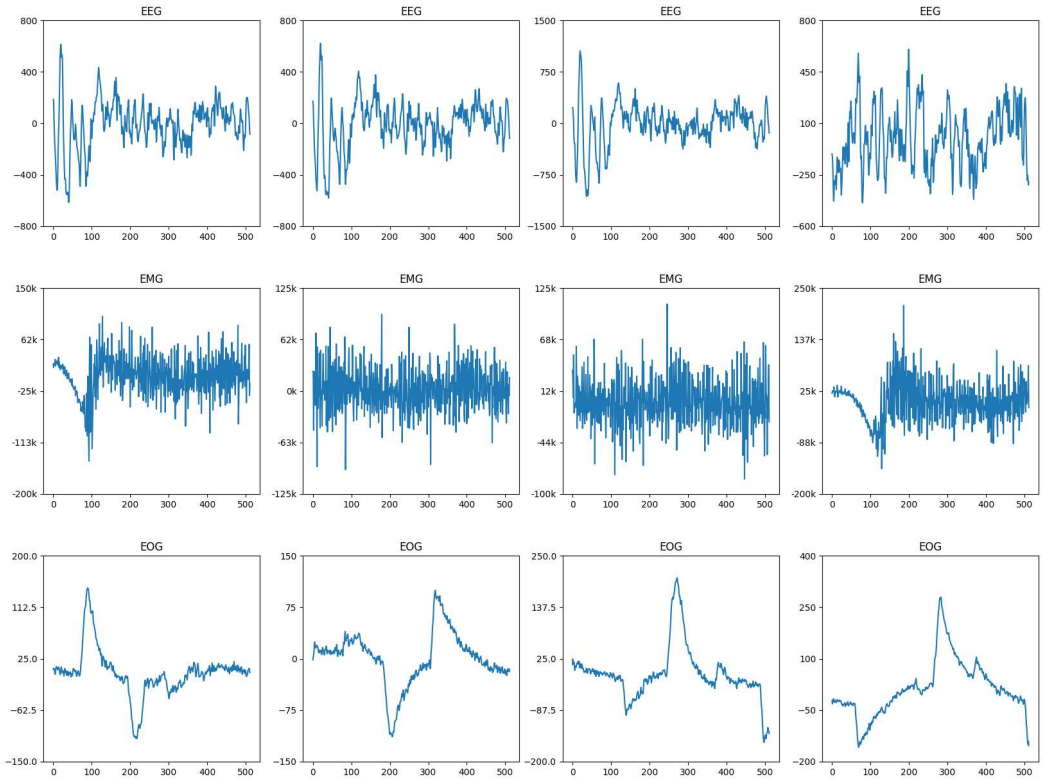


**Figure 3.1.** EEGdenoiseNet dataset

The incorporation of these various types of epochs in the dataset enables to generate a considerable number of noisy EEG epochs with their corresponding ground truth data. As a result, this compilation has proven to be an invaluable resource for model training and testing, facilitating an enhanced understanding of EEG denoising.

## 3.2 Preprocessing

The preprocessing of the EEG signals is an important step in the pipeline of the proposed model. It generates the noisy EEG signals that are used as input to the model. The processing step code used in the proposed model has been written by the authors of the EEGdenoiseNet dataset[22]. It is a simple yet effective method that generates noisy EEG signals from the clean EEG signals and clean EMG and EOG signals present in the dataset. The preprocessing step is performed as follows:

1. The clean EEG signals and noise signals are augmented adding random disturb
2. The EEG signals are partially reused to match the EMG and EOG signals
3. Data are divided in training, validation, and test sets
4. The noise signals are added to the EEG signals
5. The noisy EEG signals are normalized by dividing them by the standard deviation

At the end of the preprocessing step, the dataset is composed of:

- 44780 training EMG contaminated EEG samples
- 5600 validation EMG contaminated EEG samples
- 5600 test EMG contaminated EEG samples
- 27200 training EOG contaminated EEG samples
- 3400 validation EOG contaminated EEG samples
- 3400 test EOG contaminated EEG samples

## 3.3 Comparison models

In this section, each of the models used for comparison with the proposed model is described in detail.

### 3.3.1 FCNN

The proposed FCNN model is taken from[22] and it is composed of:

- 3 dense layers with ReLU activation function of 512 units
- 3 dropout layers eith probability 0.3

- a final dense layer with 512 units

It was trained on 60 epochs with a batch size of 40 for both EOG and EMG.

### 3.3.2 CNN

The proposed CNN model is taken from[22] and it is composed of:

- 4 convolutional layers with relu activation function
- 4 batch normalization layers
- 4 dropout layers with probability 0.3
- a flattening and dense layer with 512 units

It was trained on 40 epochs with a batch size of 40 for EOG and 10 epochs with a batch size of 40 for EMG.

### 3.3.3 LSTM

The proposed LSTM model is taken from[22] and it is composed of:

- 1 LSTM layers with 512 units
- 3 dense layers with ReLU activation function
- 3 dropout layers eith probability 0.3
- a final dense layer with 512 units

It was trained on 100 epochs with a batch size of 40 for EOG and 60 epochs with a batch size of 40 for EMG.

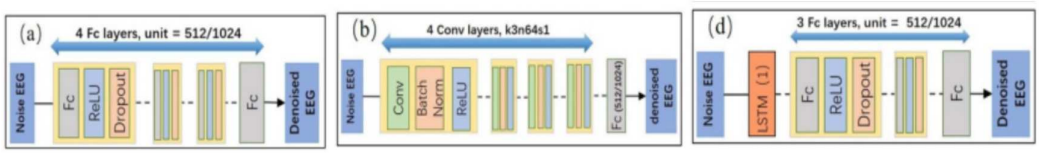


**Figure 3.2.** a) FCNN architecture b) CNN architecture c) LSTM architecture

### 3.3.4 1D-ResCNN

The proposed model is a 1D residual CNN taken from[23] and it is composed of:

- 3 residual blocks each composed of:
  - 6 convolutional layers with relu activation function
  - 6 batch normalization layers
- a final convolutional layer with relu activation function
- a flattening and dense layer

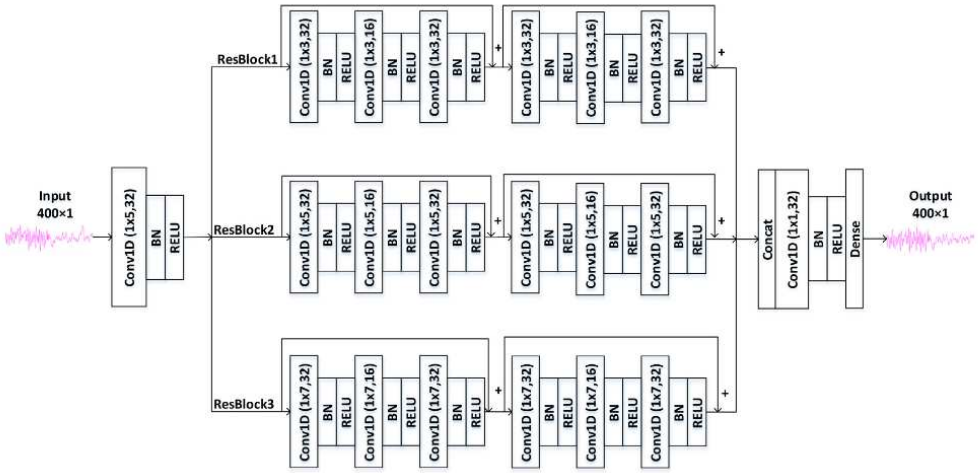


**Figure 3.3.** 1D-ResCNN architecture

### 3.3.5 EEGDNet

The proposed model is taken from[24] and it is composed of four parts including:

- reshape layer
- self-attention block
- feed forward block
- normalization layer

Specifically, the 1-D EEG signal is reshaped into 2-D form firstly and then it was sent into the 2-D transformer encoder, which consists of alternating layers of self-attention blocks and feed forward blocks. Normalization layer and residual connections are applied after every block. The output of transformer encoder is reshaped into one dimension as the final output.

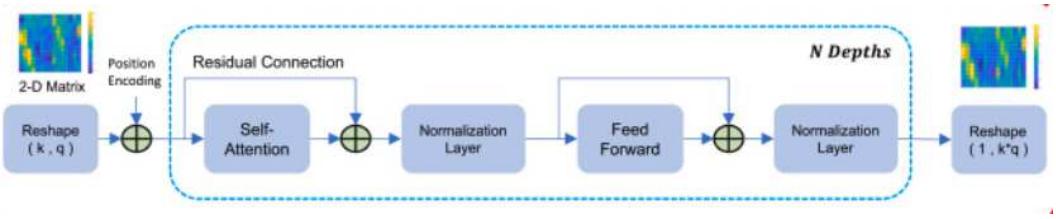


**Figure 3.4.** EEGDNet architecture

## 3.4 Proposed model architecture

In this section I'll provide a detailed explanation of the Encoder, Decoder, and Transformer, along with their components and functionalities.

The transformer has been built using two main entities the encoder and the decoder. The input is fed to the encoder while the decoder takes both the input and the encoder output to perform its calculation. The output of the decoder is then processed through a final fully connected layer.

### Encoder

The Encoder class is responsible for encoding the input sequence. It performs various operations to capture positional information, dependencies between tokens, and long-range dependencies. Firstly, it embeds the positions of the input elements to capture their positional information. Then, it applies self-attention to the input sequence, capturing the dependencies between different words. The output of the attention and feed-forward network layers is normalized using layer normalization. Additionally, dropout regularization is applied to the output of the attention and feed-forward network layers. The Encoder also utilizes a sequence of bidirectional GRU layers with residual connections and batch normalization, known as ResBiGruBlock, to capture long-range dependencies in the input sequence. Lastly, a feed-forward network processes the output of the attention mechanism. The final encoded representation of the input sequence is obtained through these operations. Here it's provided a deeper explanation of the components of the encoder:

- Positional Embedding: Embeds the positions of the input elements to capture their positional information.
- MultiHeadAttention: Performs self-attention on the input sequence to capture the dependencies between different tokens.
- Layer Normalization: Applies normalization to the output of the attention and FFN layers.
- Dropout: Applies dropout regularization to the output of the attention and FFN layers.
- ResBiGruBlock: A sequence of bidirectional GRU layers with residual connections and batch normalization, which helps capture long-range dependencies in the input sequence.
- Feed-Forward Network (FFN): A two-layer fully connected neural network that processes the output of the ResBiGru block.

### Decoder

The Decoder class is responsible for generating the output sequence based on the encoded input. It shares similar components with the Encoder class but includes

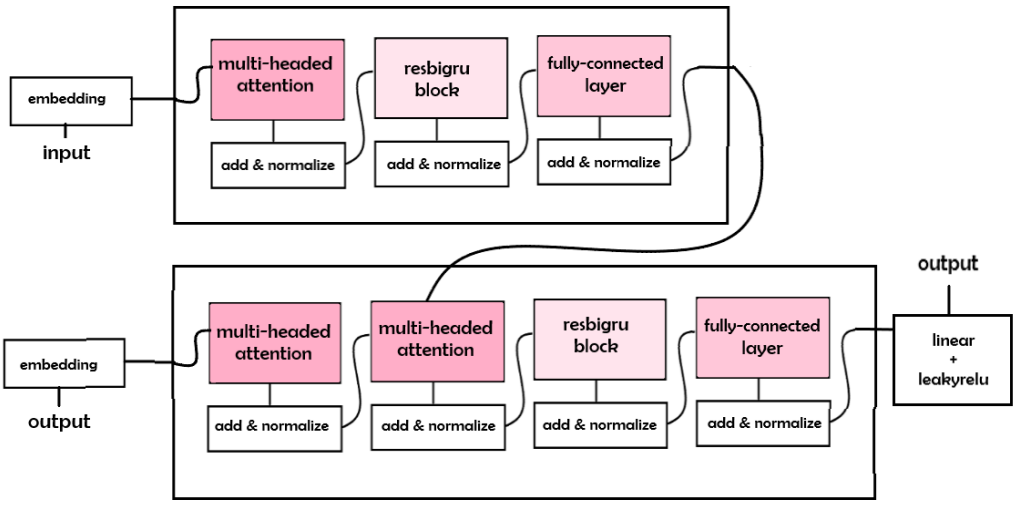


**Figure 3.5.** Proposed model architecture

additional functionality for attention between the encoder and decoder. Similar to the Encoder, positional embedding is applied to the decoder input elements to capture positional information. The Decoder class performs self-attention on the decoder input to capture dependencies between different tokens in the output sequence. It also computes attention between the decoder input and the output of the encoder, incorporating the encoded information. The output of the attention mechanisms is further processed by a ResBiGruBlock and a feed-forward network. ResBiGruBlock, consisting of bidirectional GRU layers with residual connections and batch normalization, is utilized to enhance the model's performance. The Decoder takes the inputs and encoder outputs, performing the aforementioned operations, and returns the final output of the decoder. The Decoder class consists of the following components:

- Positional Embedding: Embeds the positions of the decoder input elements to capture positional information.
- MultiHeadAttention (self-attention): Performs self-attention on the decoder input to capture dependencies between different tokens in the output sequence.
- MultiHeadAttention (encoder-decoder attention): Computes attention between the decoder input and the output of the encoder to incorporate the encoded information.
- Layer Normalization: Normalizes the outputs of the attention layers.
- Dropout: Applies dropout regularization to the output of the attention layers.
- ResBiGruBlock: A sequence of bidirectional GRU layers with residual connections and batch normalization, which helps capture long-range dependencies in the input sequence.

- Feed-Forward Network (FFN): Processes the output of the ResBiGruBlock.

### ResBiGRU Block

The Residual Bidirectional Gated Recurrent Unit (ResBiGRU) serves as a fundamental building block in the proposed model, enabling effective processing of input sequences while capturing long-range dependencies. Comprising multiple ResBiGru layers, the ResBiGRU architecture exhibits several key characteristics crucial to its performance.

Each ResBiGru layer consists of two bidirectional Gated Recurrent Unit (GRU) layers. The bidirectional nature of these GRU layers facilitates comprehensive analysis of the input sequence by considering both the forward and backward temporal contexts of each word. This bi-directionality empowers the model to capture information from both past and future contexts, enhancing its ability to understand the intricate relationships inside the signals at different time stamps.

A vital aspect of the ResBiGru layer is the incorporation of residual connections and batch normalization. Specifically, the output of each bidirectional GRU layer is combined with the input of that layer, forming a residual connection. This strategic design choice mitigates the vanishing gradient problem, allowing the network to retain essential information during training. Furthermore, batch normalization is applied after the bidirectional GRU layers to normalize the output, facilitating stable training and improving the model's generalization capabilities.

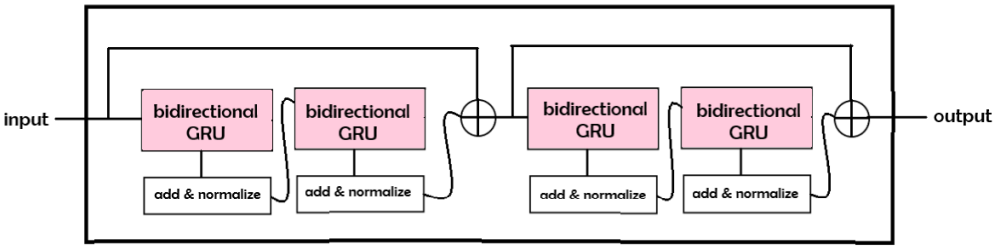


**Figure 3.6.** ResBiGRU Block

To introduce non-linearity and capture complex feature relationships, a leaky rectified linear unit (LeakyReLU) activation function is applied subsequent to the batch normalization step. The Leaky ReLU activation function is a variant of the Rectified Linear Unit (ReLU) function that addresses the issue of "dying ReLU" neurons[6.1.6]. It introduces a small slope for negative input values, allowing for the passing of some information and preventing complete suppression. The Leaky

ReLU function is defined as follows:

$$\text{LeakyReLU}(x) = \begin{cases} x \text{ if } x \geq 0 \\ \alpha x \text{ otherwise} \end{cases}$$

where $\alpha$ is a small positive constant that represents the slope of the function for negative inputs. Typically, $\alpha$ is set to a small value such as 0.01.

This activation function enhances the expressive power of the ResBiGRU architecture, enabling it to capture intricate patterns and nonlinear dependencies within the input sequence.

To further enhance representation learning and extract increasingly complex patterns, the ResBiGru layer is repeated two times in each block. This repetition enables the ResBiGRU model to iteratively learn and refine representations of the input sequence, capturing hierarchical and context-specific information.

# Chapter 4

# Experimental results

This chapter presents the experimental results of the proposed model and the baseline models. The results are presented in the form of comparison tables and graphs.

## 4.1 Evaluation Metrics

The results of the proposed model and the baseline models are compared in terms of the Relative Root Mean Squared Error (RRMSE), the cross-correlation (CC), Signal-to-Noise Ratio (SNR) and Structural Similarity Index Measure (SSIM).

### RRMSE

The RRMSE is a metric that is used to measure the performance of the denoising model. It measures the difference between the original signal and the denoised signal, normalized by the amplitude of the original signal and it is defined as follows:

$$\text{RRMSE} = \frac{\text{RMSE}(x - \hat{x})}{\text{RMSE}(x)} \tag{4.1}$$

where RMS is:

$$\text{RMSE}(x) = \sqrt{\frac{\sum_{i=1}^{N} x_i^2}{N}} \tag{4.2}$$

where $x$ is the original signal, $\hat{x}$ is the denoised signal, and $N$ is the number of samples in the signal.
An RRMSE value of 0 indicates that the estimated signal perfectly matches the reference signal, while a higher RRMSE value indicates a greater deviation between the two signals. As the RRMSE value approaches infinity, it indicates that the estimated signal has a much higher RMSE value than the reference signal, which could mean that the estimated signal is overestimating the true signal power or energy.
In practice, RRMSE values are typically reported as a percentage, which allows for a more intuitive interpretation of the error. Lower RRMSE values indicate a better match between the estimated and reference signals, while higher RRMSE values indicate a larger relative error. A commonly used threshold for RRMSE values is 10%, which means that an RRMSE value of less than 10% is considered a good match between the estimated and reference signals.

### CC

The cross-correlation (CC) is a commonly used metric for evaluating the similarity between two signals. It measures the degree to which the two signals are similar in terms of their shape and magnitude. Here is the equation of CC:

$$\mathrm{CC}(x, \hat{x}) = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(\hat{x}_i - \bar{\hat{x}})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2 \sum_{i=1}^{N}(\hat{x}_i - \bar{\hat{x}})^2}} \tag{4.3}$$

where $x$ is the original signal, $\hat{x}$ is the denoised signal, $n$ is the length of the signals, $\bar{x}$ and $\bar{\hat{x}}$ are the mean values of $x$ and $\hat{x}$, respectively.
The CC ranges from -1 to 1, with 1 indicating perfect correlation between the original and denoised signals, -1 indicating perfect anti-correlation, and 0 indicating no correlation. A higher CC value generally indicates better denoising performance.

### SNR

The Signal-to-Noise Ratio (SNR) is a widely used metric for evaluating the performance of a denoising model. It provides a measure of the ratio between the signal power and the noise power in the denoised output. It can be calculated as follows:

$$\mathrm{SNR} \approx 10 \log_{10} \left( \frac{|x|^2}{\mathrm{MSE}} \right) \tag{4.4}$$

Here, $|x|^2$ represents the power of the original signal $x$, and MSE denotes the mean squared error between $x$ and $\hat{x}$. It is defined as follows:

$$\mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 \tag{4.5}$$

The MSE is a measure of the average squared difference between the original and denoised signals. A lower MSE value generally indicates better denoising performance.

$$\mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 \tag{4.6}$$

The SNR is typically expressed in decibels (dB), and a higher SNR value indicates a higher quality denoising result. For reference, an SNR of 20 dB or lower is generally considered indicative of poor quality, while an SNR of 30 dB or higher is considered indicative of good quality.

### SSIM

The SSIM is a metric that is used to measure the performance of the denoising model. It is designed to capture the similarity between the original signal and the denoised signal in terms of luminance, contrast, and structural information. It is defined as follows:

$$\mathrm{SSIM} = \frac{(2\mu_x \mu_{\hat{x}} + c_1)(2\sigma_{x\hat{x}} + c_2)}{(\mu_x^2 + \mu_{\hat{x}}^2 + c_1)(\sigma_x^2 + \sigma_{\hat{x}}^2 + c_2)} \tag{4.7}$$

where $x$ is the original signal, $\hat{x}$ is the denoised signal, $\mu_x$ and $\mu_{\hat{x}}$ are the mean values of $x$ and $\hat{x}$, $\sigma_x$ and $\sigma_{\hat{x}}$ are their standard deviations, and $\sigma_{x\hat{x}}$ is their covariance. The constants $c_1$ and $c_2$ are small values added to avoid division by zero.
The SSIM ranges from 0 to 1, with 1 indicating perfect similarity between the original and denoised signals. A higher SSIM value generally indicates better denoising performance.

## 4.2 Results and discussion

### 4.2.1 Training details

The model has been trained for a total of 110 epochs on EOG contaminated data and 100 epochs on EMG contaminated data, using a batch size of 32. Each epoch represents a complete iteration through the entire training dataset. During each epoch, the model's parameters are updated multiple times, with each update based on a batch of 32 samples.
A higher number of epochs allows the model to potentially converge to a better solution, but it also increases the computational time. The specific value of 110 and 100 epochs was chosen based on experimentations and may vary depending on the model, the dataset and the specific values of the hyperparameters.

The optimizer that has been used used during the training is an Adam optimizer. The specific configuration of the Adam optimizer is set with various hyperparameters:

- Learning rate: It determines the step size at each iteration of the optimization process. A lower learning rate may lead to slower convergence but better optimization, while a higher learning rate may lead to faster convergence but risk overshooting the optimal solution.
- Beta 1: It controls the exponential decay rate for the first moment estimates. It represents the momentum term and influences the memory of past gradients.
- Beta 2: It controls the exponential decay rate for the second moment estimates. It represents the variance term and influences the adaptivity of the learning rates.
- Epsilon: A small value added to the denominator to improve numerical stability.

The specific values of the hyperparameters for the training were set as follows: Learning rate: 0.0001 Beta 1: 0.9 Beta 2: 0.98 Epsilon: 1e-09 It must be noted that the specific values of the hyperparameters have been chosen based on experimentation and may vary depending on the model, the dataset and the specific values of the other hyperparameters.

The loss function that was used during the training is the mean squared error (MSE), which proved to be the most effective loss function for the task of denoising.

### 4.2.2 Results

The proposed model was trained on the EOG and EMG contaminated data. The loss function quantifies the discrepancy between the predicted outputs and the ground truth labels, serving as a guide for the model to adjust its parameters and optimize its performance. By observing the behavior of the loss function throughout the training process, we can gain insights into the model's learning dynamics and the effectiveness of the chosen optimization algorithm. This analysis enables us to assess the model's training stability, convergence rate, and overall training performance. Through careful examination of the loss function, we can evaluate the effectiveness of our training methodology and gain confidence in the subsequent evaluation of the model's performance. The results of the training are shown in Figure 4.1-4.2

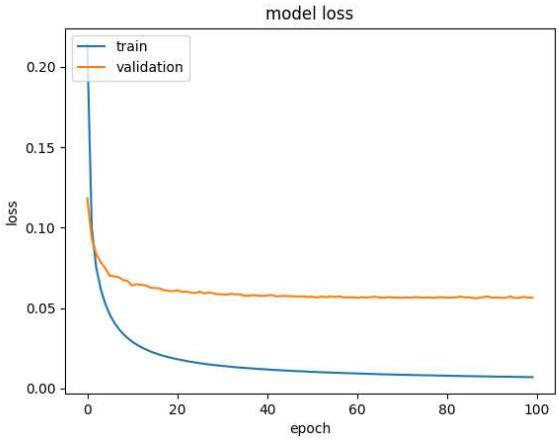


**Figure 4.1.** The loss of proposed model on EMG

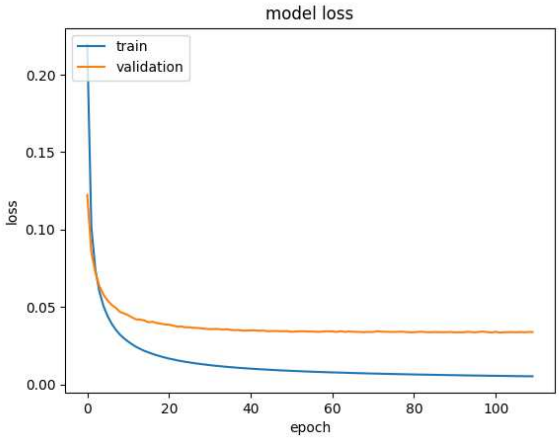


**Figure 4.2.** The loss of proposed model on EOG

### 4.2.3 Comparison with baseline models

The proposed model surpasses the performance of the comparison models, showcasing its exceptional capabilities in handling both EOG and EMG data. It achieves remarkable scores across various evaluation metrics, affirming its effectiveness in generalizing to unseen data and excelling in real-world scenarios. These evaluation metrics act as strong indicators of the model's superior performance, reaffirming its effectiveness in accurately representing and predicting the underlying characteristics of both EOG and EMG signals. By outperforming the comparison models, our proposed model establishes itself as a robust and reliable solution for EOG and EMG data analysis, setting a new benchmark in the field.

**Table 4.1.** Comparison of the results of the proposed model and the baseline models

| | **EMG** | | | | **EOG** | | | |
|---|---|---|---|---|---|---|---|---|
| **Model** | **RRMSE** | **CC** | **SNR** | **SSIM** | **RRMSE** | **CC** | **SNR** | **SSIM** |
| **FCNN** | 0.917 | 0.609 | | | 0.699 | 0.720 | | |
| **CNN** | 0.750 | 0.706 | | | 0.620 | 0.791 | | |
| **LSTM** | 0.785 | 0.636 | | | 0.740 | 0.677 | | |
| **1D-ResCNN** | 0.746 | 0.692 | | | 0.630 | 0.776 | | |
| **EEGDNet** | 0.677 | 0.732 | | | 0.497 | 0.868 | | |
| **Proposed Model** | 0.3719 | 0.9184 | 19.1013 | 0.6097 | 0.2989 | 0.9351 | 21.7110 | 0.6835 |

# Chapter 5

# Conclusions and future work

## 5.1 Conclusions

To conclude, a new model architecture has been proposed for the task of denoising EOG and EMG signals. The proposed model is based on the Transformer architecture, which is a state-of-the-art model for sequence-to-sequence tasks. The Transformer architecture is enhanced with the addition of a ResBiGRU layer, which is a bidirectional Gated Recurrent Unit (GRU) layer with residual connections. The results demonstrate that the proposed model outperforms the baseline models in terms of denoising performance. The RRMSE values indicate a better match between the estimated and reference signals, while the CC values show a higher degree of similarity between the original and denoised signals. The SNR values indicate a higher quality denoising result, and the SSIM values indicate a higher level of similarity in terms of luminance, contrast, and structural information.

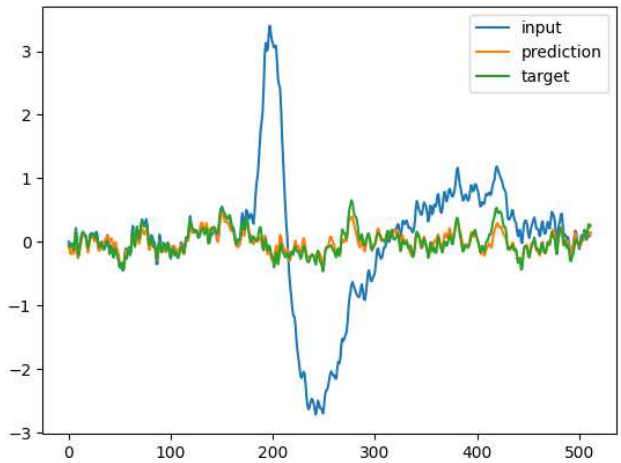


**Figure 5.1.** The denoising result of the proposed model on EMG

Overall, the proposed model architecture, which combines the Transformer and ResBiGRU, proves to be effective in capturing dependencies and long-range information in signal denoising tasks. However, further research and experimentation

can be conducted to explore additional enhancements and optimizations to improve the model's performance.

## 5.2 Future work

Although the proposed model architecture shows promising results, there are several avenues for future work to explore and enhance its capabilities:

- *Optimization techniques*: Investigate and apply advanced optimization techniques such as learning rate scheduling, adaptive learning rates, and different optimizers to improve convergence speed and model performance.
- *Hyperparameter tuning*: Conduct an in-depth hyperparameter search to fine-tune the model's performance. Explore different configurations for the Transformer and ResBiGRU blocks, including the number of layers, hidden units, attention heads, and dropout rates.
- *Data augmentation*: Introduce data augmentation techniques to increase the diversity and quantity of training data. Augmentation methods such as random noise injection, time shifting, and amplitude scaling can help the model generalize better to different noise patterns and improve its denoising performance.
- *Transfer learning*: Investigate the use of transfer learning techniques to leverage pre-trained models on related tasks or domains. Pre-training the model on large-scale datasets or similar denoising tasks can potentially improve its ability to generalize and enhance performance on specific signal denoising tasks.
- *Real-time denoising*: Adapt the model to enable real-time signal denoising by optimizing the architecture and leveraging hardware accelerators or specialized processing units. This can expand the model's applicability to scenarios that require low-latency denoising, such as real-time communication systems or streaming applications.
- *Mixed noise denoising*: Extend the model to handle mixed noise scenarios, where the input signal is corrupted by multiple types of noise. This can be achieved by training the model on a dataset containing mixed noise signals or by combining multiple models trained on different noise types.

By addressing these areas of future work, the proposed model architecture can be further improved and tailored to specific signal denoising tasks, ultimately advancing the field of signal processing and contributing to the development of more robust and accurate denoising techniques.

# Chapter 6

# Appendix

## 6.1 Neural network

### 6.1.1 Neuron

A neuron, also known as a node or a perceptron, is a fundamental building block of a neural network. It is a mathematical function that receives one or more inputs and produces an output based on them.
A typical neuron in a neural network consists of three main components:

Inputs: These are the signals or information that the neuron receives from other neurons or from the outside world. Each input is assigned a weight, which determines how important it is to the neuron's output.

Activation Function: The activation function is a non-linear function that takes the weighted sum of the inputs and produces an output. The output of the activation function is then passed on to the next layer of neurons in the network.

Bias: The bias is an additional input that is used to adjust the output of the activation function. It can be thought of as a threshold that the weighted sum of the inputs must cross before the neuron "fires" or produces an output.

### 6.1.2 Loss function

A loss function is a mathematical function that measures the difference between the predicted output of the network and the actual output (or "ground truth") for a given set of input data.
The goal of a neural network is typically to minimize the difference between the predicted output and the ground truth. The loss function provides a quantitative measure of this difference, which the network can use to adjust its parameters (such as the weights and biases of the neurons) in order to improve its predictions.
There are many different types of loss functions that can be used in a neural network, depending on the specific task the network is trying to perform. Some common examples include:

- Mean Squared Error (MSE): This is a popular loss function for regression problems, where the goal is to predict a continuous output value. It measures the average squared difference between the predicted and actual values.

- Binary Cross-Entropy: This is a loss function that is commonly used for binary classification problems, where the goal is to predict a binary output (e.g., 0 or 1). It measures the difference between the predicted probability of the positive class and the true probability (which is either 0 or 1).

- Categorical Cross-Entropy: This is a loss function that is commonly used for multi-class classification problems, where the goal is to predict one of several possible output classes. It measures the difference between the predicted probability distribution over the classes and the true distribution (which is typically represented as a one-hot encoded vector).

By minimizing the loss function during training, the neural network can adjust its parameters to improve its predictions on the training data. However, it's important to also evaluate the performance of the network on a separate set of validation or test data to ensure that it is not overfitting to the training data.

### 6.1.3 Activation functions

Activation functions play a crucial role in neural networks by introducing non-linearity to the network's output. They are applied to the weighted sum of inputs at each neuron, known as the activation, and determine the neuron's output or its contribution to the next layer. Activation functions enable neural networks to model complex relationships and make non-linear transformations.
Here are some commonly used activation functions:

- **Sigmoid**: The sigmoid function is a smooth, S-shaped curve that maps the input to a range between 0 and 1. It is given by the formula:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6.1}$$

  Sigmoid functions are useful in binary classification tasks as they can squash the output to a probability-like value, interpreting it as the likelihood of belonging to a certain class. However, they suffer from vanishing gradients, making them less suitable for deep neural networks.

- **Hyperbolic tangent (Tanh)**: The hyperbolic tangent function is similar to the sigmoid function but maps the input to a range between -1 and 1. It is given by the formula:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6.2}$$

  Tanh functions also exhibit saturation and vanishing gradients similar to sigmoid functions. They are commonly used in hidden layers of neural networks.

- **Rectified Linear Unit (ReLU)**: The ReLU function applies a threshold at zero, setting negative values to zero and leaving positive values unchanged. Mathematically, it is defined as:

$$f(x) = max(0, x) \tag{6.3}$$

  ReLU has become the most popular activation function for deep neural networks due to its simplicity and effectiveness. It helps alleviate the vanishing gradient problem and promotes sparse activations, which can accelerate training. However, ReLU can also suffer from the "dying ReLU" problem, where neurons become permanently inactive, producing zero outputs.

- **Leaky ReLU**: Leaky ReLU is an extension of the ReLU function that solves the dying ReLU problem by introducing a small slope for negative values. It is defined as:

$$f(x) = max(\alpha x, x) \tag{6.4}$$

  Here, $\alpha$ is a small constant that determines the slope for negative inputs. Leaky ReLU allows a small gradient for negative inputs, preventing the neurons from becoming completely inactive.

- **Softmax**: The softmax function is commonly used in the output layer of a neural network for multi-class classification tasks. It takes a vector of real numbers as input and normalizes them into a probability distribution over multiple classes. The softmax function is defined as:

$$f(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \tag{6.5}$$

  Here, $x_i$ represents the $i$-th element of the input vector, and $K$ is the total number of classes. Softmax ensures that the outputs sum up to 1, allowing them to be interpreted as class probabilities.

These are just a few examples of activation functions used in neural networks. Choosing an appropriate activation function depends on the task at hand, network architecture, and other factors. Experimentation and empirical evaluation are often necessary to determine the most effective activation function for a given scenario.

### 6.1.4 Backpropagation

Backpropagation is a key algorithm used for training neural networks. It enables the network to learn from labeled data and adjust its weights and biases to minimize the prediction error. An explanation of how the forward and backward passes work is given below.

#### Forward Pass

The forward pass is the first step in the backpropagation algorithm. It involves propagating the input data through the neural network to obtain the predicted output. Each neuron in the network performs a weighted sum of its inputs, applies

an activation function to the sum, and passes the result to the next layer.
Let $x$ be the input to the network, and $y$ be the corresponding target output. The forward pass can be described as follows:

1. Compute the activations of the input layer neurons: $a^{(0)} = x$.
2. For each layer $l$ from 1 to the output layer:
   (a) Compute the weighted sum of inputs: $z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$, where $W^{(l)}$ is the weight matrix and $b^{(l)}$ is the bias vector for layer $l$.
   (b) Apply the activation function $\sigma$ to the sum: $a^{(l)} = \sigma(z^{(l)})$.
3. Compute the predicted output: $\hat{y} = a^{(L)}$, where $L$ is the index of the output layer.

**Backward Pass**

The backward pass, also known as backpropagation, is the process of computing the gradients of the loss function with respect to the weights and biases of the network. It allows the network to update its parameters in the opposite direction of the gradient to minimize the loss. The backpropagation algorithm can be summarized as follows:

1. Compute the gradient of the loss with respect to the predicted output: $\delta^{(L)} = \frac{\partial L}{\partial \hat{y}}$.
2. For each layer $l$ from the output layer to the input layer:
   (a) Compute the gradient of the weighted sum: $\delta^{(l)} = \frac{\partial L}{\partial z^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \odot \sigma'(z^{(l)})$, where $\odot$ denotes element-wise multiplication and $\sigma'$ is the derivative of the activation function.
   (b) Compute the gradients of the weights and biases: $\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)}(a^{(l-1)})^T$ and $\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}$.
   (c) Compute the gradient of the previous layer's activations: $\delta^{(l-1)} = (W^{(l)})^T\delta^{(l)}$.

**Parameter Update**

Once the gradients of the loss function with respect to the weights and biases have been computed, the network's parameters can be updated using an optimization algorithm such as gradient descent. The update rule typically involves subtracting a fraction of the gradients from the current parameter values.
Let $\eta$ be the learning rate, a hyperparameter that controls the step size in the parameter update. The update equations are as follows:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}},$$
$$b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}}.$$

These equations are applied for each layer $l$ in the network.
The forward pass, backward pass, and parameter update steps are iterated over multiple training examples until the network converges to a satisfactory level of performance.

### 6.1.5 Vanishing gradient problem

The vanishing gradient problem is a common problem in deep neural networks that occurs when the gradient of the loss function with respect to the weights and biases of the network is very small. This problem can be solved by using a different activation function, such as the rectified linear unit (ReLU) activation function, which does not suffer from the vanishing gradient problem.

### 6.1.6 Dying Neuron Problem in ReLU

The "dying neuron" problem is a phenomenon that can occur when using the Rectified Linear Unit (ReLU) activation function in neural networks. It refers to the situation where a neuron becomes permanently inactive and outputs zero for all inputs, effectively "dying" and contributing no further information to the network. The dying neuron problem typically arises when a ReLU neuron's weights are updated in such a way that the neuron consistently outputs negative values for all inputs during training. When this happens, the gradients flowing through the neuron during backpropagation are also negative, causing the weights to continue adjusting in a way that keeps the neuron inactive.

# Bibliography

[1] Torres, E.P.; Torres, E.A.; Hernández-Álvarez, M.; Yoo, S.G. EEG-Based BCI Emotion Recognition: A Survey. Sensors 2020, 20, 5083. doi: 10.3390/s20185083

[2] Jiang, X.; Bian, G.-B.; Tian, Z. Removal of Artifacts from EEG Signals: A Review. Sensors 2019, 19, 987. doi: 10.3390/s19050987

[3] Wajid Mumtaz, Suleman Rasheed, Alina Irfan, Review of challenges associated with the EEG artifact removal methods, Biomedical Signal Processing and Control, Volume 68, 2021, 102741, ISSN 1746-8094, doi: 10.1016/j.bspc.2021.102741.

[4] Boashash, B., ed. (2003), Time–Frequency Signal Analysis and Processing: A Comprehensive Reference, Oxford: Elsevier Science, ISBN 978-0-08-044335-5.

[5] Cooley, James W. (1987). The Re-Discovery of the Fast Fourier Transform Algorithm (PDF). Microchimica Acta. Vol. III. Vienna, Austria. pp. 33–45. Archived (PDF) from the original on 2016-08-20. doi:10.1007/BF01201681

[6] R. Álvarez, E. Borbor and F. Grijalva, "Comparison of methods for signal analysis in the time-frequency domain," 2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM), 2019, pp. 1-6, doi: 10.1109/ETCM48019.2019.9014860.

[7] W. -k. Lu and Q. Zhang, "Deconvolutive Short-Time Fourier Transform Spectrogram," in IEEE Signal Processing Letters, vol. 16, no. 7, pp. 576-579, July 2009, doi: 10.1109/LSP.2009.2020887.

[8] E. Chassande-Mottin and A. Pai, "Discrete time and frequency Wigner-Ville distribution: Moyal's formula and aliasing," in IEEE Signal Processing Letters, vol. 12, no. 7, pp. 508-511, July 2005, doi: 10.1109/LSP.2005.849493

[9] S. Zhou, B. Tang and R. Chen, "Comparison between Non-stationary Signals Fast Fourier Transform and Wavelet Analysis," 2009 International Asia Symposium on Intelligent Interaction and Affective Computing, Wuhan, China, 2009, pp. 128-129, doi: 10.1109/ASIA.2009.31.

[10] V. M. Pukhova and M. S. Stepanova, "Up-Chirplet and Down-Chirplet Transforms of Non-Stationary Signals," 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), Saint Petersburg and Moscow, Russia, 2019, pp. 1221-1225, doi: 10.1109/EIConRus.2019.8657179.

[11] S. G. Mallat and Zhifeng Zhang, "Matching pursuits with time-frequency dictionaries," in IEEE Transactions on Signal Processing, vol. 41, no. 12, pp. 3397-3415, Dec. 1993, doi: 10.1109/78.258082.

[12] Klados, M.A.; Papadelis, C.; Braun, C.; Bamidis, P.D. REG-ICA: A hybrid methodology combining blind source separation and regression techniques for the rejection of ocular artifacts.Biomed. Signal Process Control doi:10.1016/j.bspc.2011.02.001

[13] J. A. Nascimento, J. M. S. Nascimento, and J. M. C. Nascimento, "Blind source separation: A review," in IEEE Signal Processing Magazine, vol. 26, no. 6, pp. 22-38, Nov. 2009, doi: 10.1109/MSP.2009.937000.

[14] Casarotto, S.; Bianchi, A.M.; Cerutti, S.; Chiarenza, G.A. Principal component analysis for reduction of ocular artefacts in event-related potentials of normal and dyslexic children. Clin. Neurophysiol. 2004, 115, 609–619. doi: 10.1016/j.clinph.2003.10.018.

[15] Jung, T.P.; Makieg, S.; Bell, A.J.; Sejnowski, T.J. Independent component analysis of electroencephalographic and event-related potential data. Cent. Audit. Process. Neural Model. 1996, 2, 1548–1551.

[16] Vigário, R. Extraction of ocular artifacts from EEG using independent component analysis. Electroencephalogr. Clin. Neurophysiol. 1997, 103, 395–404, doi: 10.1016/S0013-4694(97)00042-8.

[17] N. Robinson, K. P. Thomas and A. P. Vinod, "Canonical correlation analysis of EEG for classification of motor imagery," 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, Canada, 2017, pp. 2317-2321, doi: 10.1109/SMC.2017.8122967.

[18] V. Bajaj and R. B. Pachori, "Classification of Seizure and Nonseizure EEG Signals Using Empirical Mode Decomposition," in IEEE Transactions on Information Technology in Biomedicine, vol. 16, no. 6, pp. 1135-1142, Nov. 2012, doi: 10.1109/TITB.2011.2181403.

[19] He, P., Wilson, G. Russell, C. Removal of ocular artifacts from electroencephalogram by adaptive filtering. Med. Biol. Eng. Comput. 42, 407–412 (2004). doi: 10.1007/BF02344717

[20] Somers, B., Francart, T., Bertrand, A. (2018). A generic EEG artifact removal algorithm based on the multi-channel Wiener filter. Journal of neural engineering, 15(3), 036007. doi:10.1088/1741-2552/aaac92

[21] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I. (2017). Attention is All you Need. ArXiv, abs/1706.03762. doi: 10.48550/arXiv.1706.03762

[22] Zhang, Haoming Zhao, Mingqi Wei, Chen Mantini, Dante Li, Zherui Liu, Quanying. (2021). EEGdenoiseNet: A benchmark dataset for end-to-end deep learning solutions of EEG denoising. Journal of Neural Engineering. 18.doi: 10.1088/1741-2552/ac2bf8.

[23] Yang, Banghua Duan, Kaiwen Fan, Chengcheng Hu, Chenxiao Wang, Jinlong. (2018). Automatic ocular artifacts removal in EEG using deep learning. Biomedical Signal Processing and Control. 43. 148-158. doi: 10.1016/j.bspc.2018.02.021.

[24] Xiaorong Pu, Peng Yi, Kecheng Chen, Zhaoqi Ma, Di Zhao, Yazhou Ren, EEGDnet: Fusing non-local and local self-similarity for EEG signal denoising with transformer, Computers in Biology and Medicine, ISSN 0010-4825, doi: 10.1016/j.compbiomed.2022.106248.

[25] Wenlong Wang, Baojiang Li, Haiyan Wang, A Novel End-to-end Network Based on a bidirectional GRU and a Self-Attention Mechanism for Denoising of Electroencephalography Signals, Neuroscience, doi: 10.1016/j.neuroscience.2022.10.006.