KU LEUVEN

Unsupervised deep feature extraction for neonatal sleep stage classification

Nick Seeuws

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Big Data Analytics

Thesis supervisors:
Prof. dr. ir. Sabine Van Huffel
Prof. dr. Gunnar Naulaers

Assessors:
Prof. dr. Katrien Jansen
Ir. Ofelie De Wel

Mentor:
Dr. ir. Amir Hossein Ansari

Academic year 2018 – 2019

Preface

A thesis is influenced by many people during the year. At this final stage I would like to express my gratitude to some of them.
First, I would like to thank my supervisors for offering me the chance to research such an interesting topic. My coursework always proposed models in a nice, clean theoretic framework and viewing the implications of using data from the real world has been eye opening. Interacting with the end users of algorithms turned out to be a refreshing experience.
Second, my mentor Amir also deserves his share of praise. Many interesting discussions stood at the basis of the directions my research took. He was always at the ready to temper my crazy ideas or offer a down to earth view on my proposals and kept pushing me throughout the year.
Finally, I have to thank my parents. They stood by me for my past six years of study and motivated me to keep pursuing my interests. Their unwavering support was something I could rely on every day of my life.

Nick Seeuws

Contents

Abstract

Sleep stage monitoring is an important tool in the clinician's arsenal to provide care for neonates and assess normal brain maturity. Correctly assessing the sleep wake cycles can direct interventions in the NICU and provide better understanding of neurophysiological processes in the earliest stages of life. EEG recordings carry substantial information on these sleep stages and are used to perform sleep stage monitoring in a noninvasive way. Correctly identifying sleep stages from EEG recordings is a challenging task even for experts in the field. Using traditional supervised deep learning approaches to support clinicians demands major effort up front to provide annotated EEG segments for the machine learning model to train from.

This thesis investigates methods to improve the current state of the art in neonatal sleep stage monitoring based on supervised learning by using unannotated EEG recordings. An unsupervised method is proposed to automatically extract relevant features from data and a machine learning classifier is later trained on the extracted features. A semi-supervised model is proposed to directly combine information from annotated and unannotated data. Classification using features from unsupervised learning on the C3-C4 EEG signal reported an average kappa coefficient of 0.47 and classification using semi-supervised learning on the same data reported an average kappa coefficient of 0.64.

The proposed unsupervised learning model was not successful at competing with the state of the art in automated sleep stage classification and is shown to not disentangle factors of variation in the data corresponding to sleep stages. Semi-supervised learning proved to be very competitive with the current state of the art outperforming a supervised deep learning approach using the same data.

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

CFM	Cerebral Function Monitoring
CNN	Convolutional Neural Network
EEG	Electroencephalogram
GAN	Generative Adversarial Networks
MSE	Mean Square Error
MLP	Multilayer Perceptron
NICU	Neonatal Intensive Care Unit
PCA	Principal Component Analysis
PMA	Postmenstrual Age
PP	Postprocessing
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
t-SNE	t-Distributed Stochastic Neighbor Embedding
VAE	Variational Autoencoder
TP	True Positive rate
FP	False Positive rate
TN	True Negative rate
FN	False Negative rate

Symbols

x	Scalar
$\boldsymbol{x}$	Vector
x_i	i-th component of a vector
X	Matrix
$(x_j)_{j=1}^{J}$	A collection of J elements x

Chapter 1

Introduction

This thesis will study unsupervised deep feature extraction for neonatal sleep stage classification and and extend the study to semi-supervised methods. This introductory chapter will shortly illustrate the background of the research problem, will formulate the goal of this work and will provide an overview of the following chapters.

1.1 Background

Preterm birth carries many risks for an infant. One such risk is related to neurophysiological development with longlasting consequences to brain structure as far as early adulthood. [1] The central nervous system undergoes major changes during the final weeks of pregnancy, especially the area of the prefrontal cortex. This part of the brain influences the higher-order cognitive abilities and is late to mature. The longlasting consequences can manifest themselves in problems related to these higher-order cognitive abilities with attention-deficit disorder, learning difficulties and autism as examples. [2]

Adding to the problems is the often necessary stay in the Neonatal Intensive Care Unit, NICU, of a hospital. The light, sound, diverse set of caregivers and interventions for potential complications after birth all contribute to a very stressful environment. The NICU does allow for monitoring of the biophysical processes in the neonate to adapt care and direct interventions. One such monitoring process is the electroencephalogram, EEG, measuring voltage fluctuations originating from neurons firing in the brain. A major benefit of EEG monitoring is the noninvasive nature of most monitoring setups, consisting of a head cap with electrodes. Analysis of the EEG signal during sleep can be used to monitor brain maturation: neurophysiological development is linked with functional reorganization of the different sleep stages. Identifying sleep stages in neonatal EEG recording constitutes a major challenge and clinicians have already hinted to applying computer-based analysis routines. [3]

Dereymaeker et al. [4] provide an extensive overview of sleep stage classification using neonatal sleep-EEG. They list several expected characteristics or features of

the EEG signal for the relevant sleep stages over a wide range of gestational age. The publication also included several examples of EEG recordings corresponding to the relevant sleep stages. They also indicate the challenge of sleep assessment due to external factors influencing EEG recording and call for automated analysis tools to further improve the understanding of brain development in neonates.

Development of such automated analysis tools has been investigated in the past by looking for patterns in spectral features of EEG recordings [5] [6] or by exploiting the discontinuous nature of neonatal quiet sleep. [7] A recent publication by Ansari et al. [8] reported successful development of an end-to-end machine learning algorithm for neonatal sleep stage classification in quiet sleep and non-quiet sleep. Their work involved training a supervised convolutional neural network on 500 hours of EEG recordings annotated by experts. Succesful application of deep learning hinges on the availability of vast datasets. Substantially improving the performance of existing algorithms would require expert clinicians to manually label hundreds of hours of EEG recordings which is a time-consuming process.

1.2 Goal

This thesis aims to investigate the potential for applying unsupervised learning models to alleviate the reliance on human effort in a machine learning pipeline. Unsupervised feature extraction aims to consume vast amounts of unlabeled EEG recordings and to automatically identify relevant characteristics or features of the data. Whether or not such unsupervised models extract meaningful features from data can be assessed by using the extracted features to train a classifier on the limited labeled dataset.

The potential of semi-supervised learning is investigated as well. These models make use of a combination of labeled and unlabeled data and fullfil the same goal as unsupervised learning: increase classification performance by using unlabeled recordings. However, in contrast to unsupervised feature learning the semi-supervised models directly result in a classification algorithm.

The main goal of this work is using machine learning to leverage unlabeled EEG recordings to improve on existing sleep stage classification algorithms paving the way for further developments in automated EEG analysis with a minimal reliance on labels provided by the medical field.

1.3 Thesis Overview

The remainder of this thesis is organized as follows: Chapter 2 will provide the necessary background on machine learning, neural networks and two advanced models: Variational Auto-Encoders and Generative Adversarial Networks. Chapter 3 will discuss previous work on neonatal sleep stage classification and unsupervised learning

on EEG signals. This chapter will also introduce recent advances in unsupervised and semi-supervised learning. Chapter 4 will introduce the EEG recordings used for training and testing the models. Chapter 5 will illustrate the proposed unsupervised and semi-supervised pipelines. Chapter 6 will contain the classification results when using the trained unsupervised and semi-supervised model. Chapter 7 will compare the classification performance with other approaches and discuss the extracted features. Finally, chapter 8 will conclude the work and propose paths for future study.

Chapter 2

Machine Learning

This chapter will introduce the theoretical foundation used by the remainder of this work. It will first discuss the fundamentals of machine learning by using a mathematical parametric model and how to estimate the parameters of such models. Next, it will touch upon a popular machine learning model: gradient tree boosting. Neural networks will be introduced next, starting with the basic building blocks and how to use them to build more complex architectures. Parameter estimation for neural networks, the so-called "training", is discussed more in depth for this specific type of machine learning model. To conclude, two important advanced machine learning models will be introduced which make heavy use of neural networks: the Variational Auto-Encoder by Kingma et al. [9] and the Generative Adversarial Networks as proposed by Goodfellow et al. [10] This chapter is heavily inspired by the book on Statistical Learning by Hastie et al. [11] and the book on Deep Learning by Goodfellow et al. [12]

2.1 Learning

Almost all machine learning problems involve a mathematical model with which to model data. In this thesis such models will be formulated as a function acting on input x producing output y through the function $f(\cdot)$. The inputs and outputs can represent scalars, vectors, matrices or higher order tensors and the function $f(\cdot)$ can produce a deterministic result or have stochastic elements. Many such models exist in literature with varying degrees of flexibility in adapting to given data with a wide variety in properties. However, a usual constant is the reliance on parameters θ to incorporate the flexible nature of the machine learning models. To summarise, data is modeled as follows.

$$y = f(x; \theta)$$

A basic illustrative model is a polynomial: $y = \theta_2 x^2 + \theta_1 x + \theta_0$. This polynomial is a deterministic function of the input x and the behavior can be adapted through the components of the parameter vector $\boldsymbol{\theta} = [\theta_2 \ \theta_1 \ \theta_0]^T$.

2.1.1 Parameter estimation

Models can learn from data by finding the best parameters given a training set. The notion of "best" parameters is a difficult one. To quantify this notion of "best" a loss function is introduced. The loss function $\mathcal{L}[y, f(x;\theta)]$ takes as input both the real expected output of the model y provided by the training set and the produced output $f(x;\theta)$. It assigns a penalty to quantify the error between the desired result and the outputted result. The problem of finding the best parameters can be formulated as follows:

$$\arg\min_{\theta} \mathbb{E}_{p(x,y)} \mathcal{L}[y, f(x;\theta)] \tag{2.1}$$

The expectation is taken over the joint distribution $p(x, y)$ representing the true data distribution of the population for the problem at hand. Calculating the full expectation is in many cases infeasible. One can approximate Monte Carlo integration of the expectation by using a training dataset $(x_n, y_n)_{n=1}^{N}$. This in turn changes the parameter estimation problem to

$$\arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}[y_n, f(x_n;\theta)] \tag{2.2}$$

i.e. minimizing the mean loss calculated over the N datapoints in the training dataset. Changing from minimizing an expectation of the loss of the population to a sum over all datapoints allows for easier computation of the loss and a more straightforward formulation as a standard optimization problem.

Many different formulations of the loss $\mathcal{L}[y_n, f(x_n;\theta)]$ exist. What should be considered "good" values for parameters changes from application to application and the myriad of loss functions reflects this diversity. However, some broad categories exist with a standard loss functions, two of which will be introduced below. A common problem consists of predicting a real output, $y \in \mathbb{R}$. This problem is called regression and its most basic loss function is the mean squared error, MSE.

$$\text{MSE} = \frac{1}{N} \sum_{n=1}^{N} (y_n - f(x_n;\theta))^2$$

The output y can also be limited to discrete values. In this case the problems are regarded as classification. For a two-class problem one can assume without loss of generality that $y \in (0, 1)$. The mathematical model $f(x;\theta)$ can be stated to output a probability value of y having the value 1 given the input x: $f(x;\theta) = p(y = 1|x;\theta)$. For this classification model, a popular loss function is the cross-entropy loss, or log loss:

$$-\text{Log loss} = \frac{1}{N} \sum_{n=1}^{N} y_n \log_e f(x_n;\theta) + (1 - y_n) \log_e (1 - f(x_n;\theta))$$

2.1.2 Generalization

Estimating the population loss in equation 2.1 by the training loss in equation 2.2 introduces a statistical error. Minimizing the average loss of the training data

is the only way to estimate the parameters of a model when the true population distribution $p(x, y)$ is unknown. The best parameters for the training set, however, do not necessarily minimize the loss on the population. A model that can estimate its parameters from training data and still perform well on unseen data sampled from the data population is said to generalize well.

To test for generalization capabilities during training of a model a separate dataset is usually introduced for validating the model. This validation set is kept separate from the core training process and only after completion of training is the validation dataset fed to the model to calculate its average loss. The validation loss can be compared to the training loss at the end of the training process to gauge the generalization capabilities of a model. When the training loss is significantly lower than the validation loss the model is said to overfit to the training data. This happens when the model mainly has memorized the training dataset and failed to find general patterns generalizable to the data population. When the validation loss is less than or somewhat equal to the training loss it is considered an indication that the model generalizes. Another important notion related to training is underfitting. A trivial solution to the problem of overfitting is to only have a superficial look at the data and not allow the model to extract difficult patterns. In this case the training and validation loss might be alike but the model does not fit sufficiently to any data, training or validation.

Figure 2.1 illustrates these concepts for fitting a polynomial on noisy data. Data has been generated by using a third order polynomial and applying normally distributed noise to points on the polynomial. The first illustration contains a model underfitting to the data; a linear regression model is clearly not suited to represent polynomial data. The second illustration estimates parameters for a third order polynomial which is clearly a good fit to the data and expected to perform well on out of sample data points. The final illustration performs parameter estimation for a polynomial of order 15 which clearly deviates from the underlying data model and overfits to the training data.

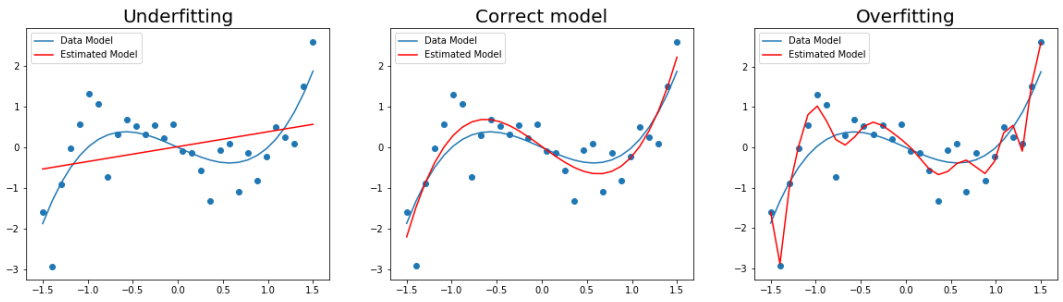

Figure 2.1: Illustration of Under- and Overfitting

One has to take care when using a validation set to gauge generalization capabil-

ities. In most machine learning pipelines an iterative approach is taken to model building. A model is proposed and the validation loss for the model is calculated. This loss is compared to previous models and the model with the best validation loss is retained. Modifications are made to the current best model and tested again. This process falls prey to a data snooping bias in the sense that the machine learning engineer is fitting the model to the validation data by manually performing optimization steps for the validation loss. This has a negative impact on the generalization capabilities of the model. To this end a given dataset is usually split in three parts, not just training and validation. One part is the training set, a second part the validation set an a third part is called the test set. At the end of all model iterations a final test of the generalization power of the model is performed with this test set. It is imperative that this dataset is not used in any other part of the pipeline or the test result will also fall prey to the same data snooping bias.

2.1.3 Performance Metrics

While the loss function for a given learning process can already be perceived as a measure of performance more informative metrics exist. A classifier cannot be adequately assessed on one evaluation dimension alone; multiple performance metrics are usually reported. This section will discuss common performance metrics for a binary classification into two categories: true and false. Different proportions can be considered for such a problem. The true positives, TP, which is the proportion of samples corresponding to true which the learner correctly classifies. The true negatives, TN, which is the proportion of samples corresponding to false which the learner correctly classifies. The false negatives, FN, which is the proportion of samples corresponding to true which are incorrectly labeled false and finally the false positives, FP, which is the proportion of samples corresponding to false incorrectly given the label true by the learner.

- Accuracy: corresponds to the amount of samples labeled correctly.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

- Precision: the proportion of correctly labeled samples in all positively classified samples.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall: the proportion of samples corresponding to true that are correctly classified

$$\text{Recall} = \frac{TP}{TP + FN}$$

- F1-score: a more abstract measure calculated as the harmonic mean of precision and recall

$$\text{F1} = \frac{2TP}{2TP + FP + FN}$$

- Cohen's kappa coefficient: a statistical measure of inter-rater agreement for categorical variables. In a machine learning context the inter-rater agreement can be viewed as the agreement between the ground truth and the outputted labels by the classifier. It takes into account the label distribution to better cope with an unbalanced dataset. To calculate it two values have to be computed. Firstly, the observed agreement p_o wich corresponds to the accuracy of the classifier. Secondly, the expected agreement by chance p_e. This value is calculated taking into account the distribution of the labels hence the coefficent naturally takes the unbalance in a dataset into account.

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

- Sensitivity: is equal to recall but is mentioned here again explicitly due to the widespread use in medical applications.

- Specificity: the proportion of samples corresponding to false that are correctly classified.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

2.1.4 Feature Space Visualization

Performance metrics as described above rely on a ground truth for comparison. Unsupervised learning models lack such a ground truth and are more intiutively assessed on qualitative results. One such evaluation approach is visualizing the extracted feature space and checking for structure. Two methods are used later in this thesis and are introduced below.

Principal Component Analysis, PCA, is a wellknown technique for dimensionality reduction. For visualization in a 2-D plane the problem is stated as dimensionality reduction using the first two principal components. This corresponds to projecting the high-dimensional data on the plane that captures most of the variance of the total dataset out of all orientations of a plane in the high-dimensional space. It is a linear embedding method.

t-Distributed Stochastic Neighbor Embedding, t-SNE for short, is a nonlinear dimensionality reduction algorithm introduced by van der Maaten et al. [13] It is mainly designed for visualization of high-dimensional space, transforming data from its original space to two or three dimensions. t-SNE aims to find a nonparametric mapping from data in the original high-dimensional space to the lower-dimensional visualization space respecting the relative pairwise distances. Clusters in the high-dimensional space should show up as clusters in the mapping and separated points in high-dimensional space stay separated in visualization space. It is a stochastic iterative algorithm without a guarantee that these properties hold when the algorithm terminates. In practice t-SNE is a very popular visualization algorithm and it often succeeds in finding the natural clusters present in the data. [14] [15] [16]

2.1.5 Gradient Tree Boosting

Gradient tree boosting is a highly effective and widely applied machine learning algorithm. [17] For a tree boosting classifier the method combines many classifiers which on their own do not perform well but when combined in a boosting ensemble achieve impressive results. For base learners f_k the total tree boosting classifier F calculates its result as

$$F(x) = \sum_{k=1}^{K} \gamma_k f_k(x)$$

The set of base learners $(f_k)_{k=1}^{K}$ is constructed iteratively incorporating the results of the previous classifier. The first learner f_1 is constructed in a regular fashion and its performance on the training set is calculated. The following learner is given a weighted training set with more weight placed on the samples classified incorrectly by the first learner. This process is repeated for all learners, each focussing on the errors of the previous one. Building an ensemble in this way this allows to combine many weak learners to form a well performing total classifier.
The base learners in gradient tree boosting are decision trees. These algorithms perform binary partitions on the training set based on the values of its features a fixed number of times. A new data sample being classified is moved to one of the two partitions at every step in the decision tree and is assigned the label of the majority class of the training data of the final partition it is placed in. The depth of a decision tree is an important hyperparameter for a tree boosting classifier and specifies the amount of partitioning steps a base learner is allowed to perform.

2.2 Neural Networks

Neural networks are highly flexible models that consist of a hierarchy of basic building blocks: neurons. Two main kinds of neurons exist: densely connected ones and convolutional ones. Especially the convolutional neural networks, CNNs, have been used to great succes in recent image processing tasks, both on general image classification [18] [19] and in more specialized medical imaging tasks [20].

2.2.1 Dense Neuron

The basic building blocks of neural networks are based on the classical perceptron classifier. An input vector $\boldsymbol{x} \in \mathbb{R}^I$ is fed as input to these basic blocks and the neuron outputs a single scalar y based on a nonlinear transformation. The specific mathematical formulation goes as follows

$$y = f(\boldsymbol{w}^T \boldsymbol{x} + b)$$

Trainable parameters for the neuron are the weight vector $\boldsymbol{w}$ and the bias term b. For notational ease the input vector is usually appended with a constant scalar value to include the bias term as a component of the weight vector and express the mathematical model as a single inner product: $y = f(\boldsymbol{w}^T \boldsymbol{x})$

The nonlinear character of the transformation stems from the use of an activation function $f(\cdot)$. Many different activation functions have been proposed in literature and are still in use. In this work the Rectified Linear Unit, ReLU, is used by most of the learners.

$$\text{ReLU}(x) = max(0, x)$$

It is the activation function of choice for building very deep architectures as illustrated in [12]. The sigmoid activation function was originally the main nonlinearity in neural networks but fell out of favor. Currently it is still in use for the final part of a network architecture when the learner has to represent a probability due to its range being contained to $(0, 1)$.

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^x}$$

A third activation function is the identity function despite removing the nonlinear character of the neuron. This activation function occurs infrequently and is used when a linear transformation of the input is desired while still fitting into the formulation of a neuron.

2.2.2 Convolutional Neuron

Discrete convolutions arise in neural network models due to the need of locally connected units. The basic neurons are very flexible building blocks but use the entire input to compute the outputted values. In both signal processing and image processing it is interesting to use local information instead of relying on models that are forced to cope with the global context. Instead of using a complete inner product one can rely on discrete convolutions to compute the output resulting in locally connected units compared to the global connections of the normal neuron. For an input signal, the vector $\boldsymbol{x} \in \mathbb{R}^I$, the convolutional neuron applies the filter $S \in \mathbb{R}^L$ to result in the output signal, $\hat{\boldsymbol{x}} \in \mathbb{R}^I$. To ensure the edge values can be computed the input vector is padded, either with zero values or the edge components are copied.

$$\hat{\boldsymbol{x}} = S * \boldsymbol{x}$$

The neuron output can be formulated as

$$\boldsymbol{y} = f(S * \boldsymbol{x})$$

For a convolutional neuron the filter components are the trainable parameters. The method can also be applied when the input consists of multiple channels. In this case the filter S is generalized to a spatio-temporal filter instead of a purely temporal filter as introduced above.

In the neural network context several hyperparameter of the building blocks can be tweaked. A clear hyperparameter is the filter size L. However, convolutional

layers usually also operate with a specific stride k. Instead of the usual discrete convolution operation one can calculate an output signal $\hat{\boldsymbol{x}} \in \mathbb{R}^{I/k}$ as follows:

$$\hat{\boldsymbol{x}}[n] = \sum_{l=-L/2}^{L/2} S[l]\boldsymbol{x}[kn - l]$$

Using this calculation the normal discrete convolution corresponds to a stride of 1. Strided convolutions can be used to downsample an input and aggregate information. Many recent network architectures follow the all-convolutional approach proposed by Springenberg et al. [21] and use strided convolutions as replacement for the traditional pooling operation.

2.2.3 Layers

Neurons on their own are not flexible models. The power of neural networks only arises when neurons are stacked and combined into deep architectures with hidden layers. Multiple neurons can operate on the same input and their outputs can be combined in an output vector $\boldsymbol{y}$ in the case of a densely connected neuron. For convolutional units acting on a signal multiple neurons result in a multi-channel output. Multiple such layers can be appended together with the output of a layer used as the input of the next one. For densely connected networks it can be showed that neural networks making use of at least one hidden layer between input and output are universal approximators. [22]
Figure 2.2 illustrates one such architecture for two hidden layers. The input $\boldsymbol{x} \in \mathbb{R}^3$ is transformed to the output $\boldsymbol{y} \in \mathbb{R}^2$ by

$$\boldsymbol{y} = f(W_2 f(W_1 \boldsymbol{x} + b_1) + b_2)$$

with $W_1 \in \mathbb{R}^{4\times3}, W_2 \in \mathbb{R}^{2\times4}$ and the nonlinearities $f(\cdot)$ applied elementwise. It should be noted that the identity function should not be used throughout the entire network since the model would then reduce to a single linear operation.

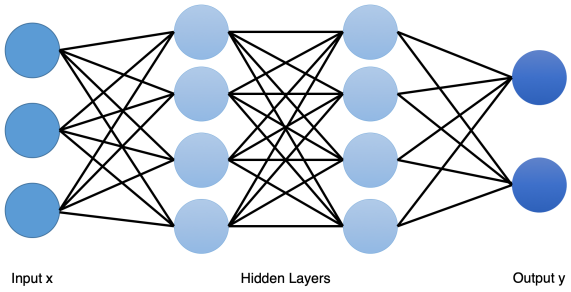

Figure 2.2: A Multilayer Architecture

2.2.4 Training

The training process of a neural network involves minimizing the average loss on the training data as formulated in equation 2.2. Neural networks are usually trained using first-order derivative based optimization routines. The step size for the optimization routine is commonly called the learning rate. For these gradient descent methods the gradient of the loss L with respect to the model parameters θ_i is needed. Backpropagation is a commonly known technique based on the chain rule for derivatives to calculate this gradient starting from the final layer of a network. The derivatives of the parameters of the layers closer to the input of the network are formulated as the product of many terms due to the chain rule and can suffer from exploding or vanishing gradients. In very deep networks this repeated multiplication can result in numerical over- or underflow if left unchecked.
Using the complete training set to calculate the gradient can result in a prohibitive computational cost due to neural networks having a large number of trainable parameters and datasets growing into the millions and billions of datapoints. The common solution is using Stochastic Gradient Descent, SGD, with minibatches of size M much smaller than the complete dataset. The average loss is calculated for the minibatches and the gradient computed and applied to the model. An added benefit, apart from faster convergence in total time, is the regularizing effect of SGD. The network sees new datapoints in every minibatch making it harder to overfit to the training data.
SGD is, however, not without its limitations. Using minibatches can result in oscillitations in parameter space that do not settle down to a desirable result and is very dependent to the user-provided learning rate to converge to a desirable region in parameter space. Kingma et al. [23] propose an extension to classical SGD called Adam, short for adaptive moment estimation. By keeping track of the first and second moments of the gradient during minibatch updates the oscillatory nature of SGD is mitigated and the method accepts a wide range of learning rates.

2.2.5 Advanced Layer Types

Several specialized layer types have been introduced and can be situated throughout a network. For this thesis two specific layer types are of interest, mainly aimed at increasing generalization capabilities.

Batchnormalization Ioffe et al. [24] introduced batchnormalization as a way of reducing internal covariate shift during training. Due to the highly nonlinear nature of neural networks and the use of stochastic gradient descent the distribution of the inputs to an intermediate layer in the network can change drastically from minibatch to minibatch. The layer not only has to learn patterns from the minibatch but also has to cope with this change in distribution of the input data. Batchnormalization aims to solve the problem of internal covariate shift by normalizing layer inputs using minibatch statistics. Ioffe et al. report faster training, less dependence on choosing a specific learning rate and a regularizing effect due to batchnormalization. For this

thesis batchnormalization is applied within a layer by normalizing the inputs to the activation function of a layer as described in the original paper.

Dropout Dropout is another specialized layer introduced by Srivastava et al. [25] Their approach consists of randomly dropping a fraction of the units in a neural network during training to force the network to learn robust features. During testing all units remain active and the results are averaged. One can think of this method as being an ensemble of many different network architectures all training at once. It is a very popular method of preventing overfitting in a neural network.

2.3 Advanced Models

This thesis focusses on two advanced models making use of neural networks. The Variational Auto-Encoder [9] is later used for unsupervised learning and the Generative Adversarial Network [10] is used in adapted form in the context of semi-supervised learning.

2.3.1 Variational auto-encoders

Kingma et al. [9] introduced a class of models suitable for unsupervised learning: Variational Autoencoders, VAEs. Their aim was to provide methods of efficient learning for directed graphical models and efficient posterior inference. General parametrizations for the different distributions can be used in their approach but the experiments were carried out using neural networks; further publications also leaned heavily on neural networks.

Data points $\boldsymbol{x}$ are assumed to be drawn from a distribution with a hidden latent random variable $\boldsymbol{z}$. The true prior and likelihood distributions are assumed to correspond to the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ with parameter vector $\boldsymbol{\theta}$. Barring very simplistic models the true posterior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ is intractable. They approximate the true posterior by a new distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. In the context of feature extraction the approximate posterior can be viewed as a probabilistic encoder defining a conditional distribution over the feature space $\boldsymbol{z}$. The approximate model parameters $\boldsymbol{\phi}$ are trained jointly with the model parameters $\boldsymbol{\theta}$.

The approach taken by the authors to estimate the parameters of their models is linked to maximum loglikelihood estimation.

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) = D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})) + \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)})$$

The full loglikelihood is not maximized but the variational lower bound $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$ is maximized instead.

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})]$$

When factorizing the joint and working out the expectation operator a more convenient form can be found for the variational bound. This formulation of the loss function can be linked to the training of regular auto-encoders. The second term is a measure for the reconstruction error and the first term can be interpreted as a regularization term on the encoder.

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})] \tag{2.3}$$

In a variational auto-encoder the approximate posterior is formulated as

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)} I)$$

with $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{2(i)}$ both being the result of applying a neural network to the datapoint $\boldsymbol{x}^{(i)}$. When the prior is defined as

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) \sim \mathcal{N}(0, I)$$

the KL-divergence term can be worked out analytically and an estimator for equation 2.3 can be formulated,

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} (1 + \log{(\sigma_j^{(i)})^2}) - (\mu_j^{(i)})^2 - \sigma_j^{(i)})^2) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)}) \tag{2.4}$$

with $\boldsymbol{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim \mathcal{N}(0, I)$. The summation in the first term is over all J dimensions of the vector $\boldsymbol{z}$ and the second over L samples being drawn from the approximate posterior distribution. When minimizing $-\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$ as loss function for the samples in a minibatch the usual training procedures described above for regular neural networks can be followed. As described by Kingma et al. setting $L = 1$ is sufficient to train the VAE.

2.3.2 Generative Adversarial Networks

Generative Adversarial Networks, GANs for short, were introduced by Goodfellow et al. [10] as a way of training deep generative models. They learn a generator distribution $p_g(\boldsymbol{x})$ by first defining a prior noise distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$, usually $p_{\boldsymbol{z}}(\boldsymbol{z}) \sim \mathcal{N}(0, I)$, as input to a generator neural network $G(\boldsymbol{z}; \theta_g)$ with parameters θ_g. In order to train the generator network a second network is introduced, the discriminator $D(\boldsymbol{x}; \theta_d)$ with parameters θ_d. This discriminator acts as a classifier and represents the probability that a sample $\boldsymbol{x}$ originates from the training data instead of being drawn from the generator distribution p_g. The discriminator is trained to maximally discern real samples from generated ones while the generator aims to "fool" the generator by minimizing $\log{(1 - D(G(\boldsymbol{z}))}$. This can be summarized in a two-player minimax game.

$$\min_G \max_D V(D, G) = \mathbb{E}_{p_{data}(\boldsymbol{x})} \log D(\boldsymbol{x}) + \mathbb{E}_{p_{\boldsymbol{z}}(\boldsymbol{z})} \log{(1 - D(G(\boldsymbol{z})))} \tag{2.5}$$

Training the GAN involves an iterative process alternating between training the discriminator and generator. The discriminator is shown a batch of real data samples

and a batch of generated samples and updates its weights to better classify the two. Afterwards the discriminator's weights are kept fixed and the generator updates its own weights to better fool the discriminator in the next training iteration. Goodfellow et al. show that this training scheme has a fixed point for D and G where $p_g = p_{data}$ when both functions have sufficient capacity. At the fixed point the discriminator is as accurate as a coin flip and the generator outputs perfect data samples. Figure 2.3 illustrates the GAN training setup.

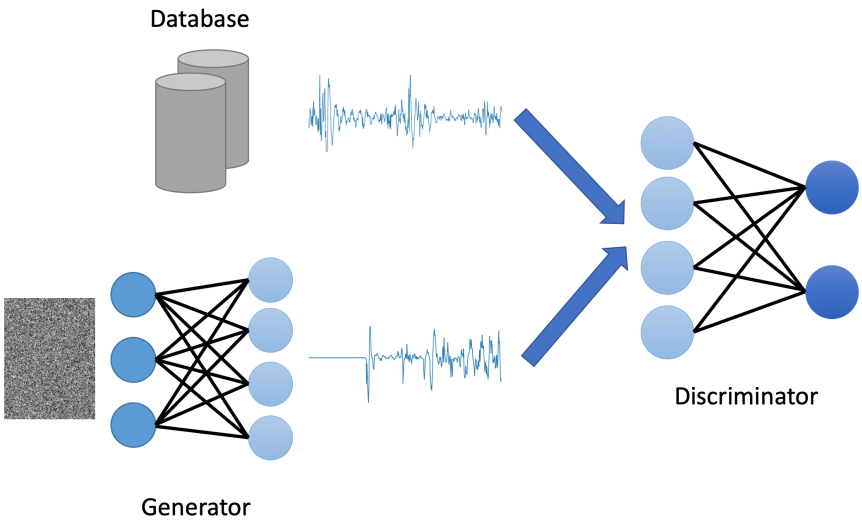

Figure 2.3: Illustration of the GAN setup. The generator transforms noise into a data sample and the discriminator has to distinguish between samples from a database and generated samples.

Chapter 3

Related work

This chapter will introduce previous related work, both on deep learning for EEG and on more general machine learning applications. The part on deep learning for EEG applications will discuss sleep stage classification for neonates and unsupervised feature learning for EEG in adults. Next several general works on unsupervised feature learning will be elaborated on and the chapter will conclude with a discussion on recent works in semi-supervised learning.

3.1 Deep Learning for EEG

Sleep Stage Classification The main inspiration for this thesis was the work performed by Ansari et al. [8] This work applied a convolutional neural network on multichannel EEG recordings for sleep stage classification in preterm neonates using the same dataset as the one used in this thesis. Eight channels were used in the study, corresponding to the F1, F2, C3, C4, T3, T4, O1 and O2 electrodes compared to the reference Cz. EEG segments of 30s from all the different channels were combined in a matrix to operate on a temporal dimension and a channel dimension. The proposed CNN architecture alternates applying 1D-convolutions along the temporal and channel dimension.

The authors' proposed CNN forms the first application of CNNs for neonatal sleep stage classification. Such end-to-end machine learning pipelines are still rare compared to the more classical machine learning pipeline where authors define handcrafted features and train a machine learning algorithm on top of these features, an example being the algorithm proposed by Koolen et al. [6] Ansari et al. also reported their own results for applying a classification model on top of handcrafted features. The used features are the spectral features identified as relevant to neonatal sleep stage classification by Piryatinska et al. [5] These features consisted of the spectral power in the δ, θ, α and β bands corresponding to $0.5 - 4$Hz, $4 - 8$ Hz, $8 - 12$Hz and $12 - 15(30)$Hz respectively; the 90% and 75% spectral edge frequency; the first spectral density moment; the spectral density entropy and the amplitude entropy. Ansari et al. applied a radial basis function support vector machine, SVM, to these

features computed for every channel and reported results for both the CNN and SVM. A postprocessing step was applied and consisted of a moving average filter of length 6.

Ansari et al. provided an updated algorithm [26] which performs sleep stage classification for both term and preterm infants and extended the classification to four sleep stages for term babies. The effect of decreasing the number of channels used by the classification algorithm is investigated and results for preterm sleep stage classification are reported for single channel (C3-C4), two channel (C3,C4), four channel (C3,C4,O1,O2) and the full eight channel data.

Unsupervised EEG Feature Extraction Previous works have applied deep unsupervised learning algorithms to automatically extract features from EEG data. [27] [28] [29] These previous applications had mixed success. Lin et al. [27] used a stacked sparse auto-encoder to perform feature learning and applied logistic regression on their extracted features to perform the final classification. While the final classifier performed well, it failed to outperform the state of the art algorithms they referenced.

Ren et al. [29] applied a convolutional deep belief network to learn representations. The network input was not the raw signal but rather the Fourier transform of the signal segments using only the signal in the $8 - 30$Hz frequency band. They used a high number of channels, 118 in one case, and used Principal Component Analysis, PCA, to extract lower dimensional signals. The learned features were compared with relevant handcrafted features from literature using an SVM on a BCI competition dataset and the number of training data points was varied through their experiments. The learned features consistently outperformed the handcrafted features when enough training data was available for the feature learning algorithm.

All these applications performed unsupervised learning on EEG data recorded from adults. No publications were found that perform feature learning on neonatal EEG. Migrating the models applied to adult EEG to neonatal EEG is not straightforward due to high rate of change in the EEG characteristics during infancy. [3]

3.2 Unsupervised Feature Learning

Feature learning in images is a more widely studied subject. Kingma et al. [9] explored the feature space of the MNIST dataset in their original paper introducing the VAE. Pu et al. [30] applied a VAE to extract a latent code out of images and applied an SVM to classify images into different classes. They also used the extracted features as input to a Recurrent Neural Network to automatically generate captioning belonging to an image. GANs implicitly learn data features in their discriminator network and these are used by Radford et al. [31] as input to a linear SVM to classify the CIFAR-10 and SVHN image dataset.

Other models have also been applied to great success. Convolutional deep belief networks as introduced by Lee et al. [32] are a classical approach to representation learning and suffer from a convoluted training process. The method does allow for explicit modeling of a (non-normalized) probability distribution and allows for efficient exact inference both from data to latent space and back in comparison to VAEs which only allow for approximate inference.

Valpola [33] introduced an extension to deep autoencoders by allowing shortcut connections between corresponding layers in the encoder and decoder. Valpola showed that even a regular autoencoder can be seen as a directed graphical model with a single latent variable vector and that his Ladder network is able to learn a hierarchy of latent stochastic variables where the top latent variables represented high-level abstract features and by moving down the ladder the stochastic layers could add fine-grained information to the internal representation, moving from abstract to detailed information. Valpola also showed an intuitive link between his unsupervised learning approach and supervised learning. He hinted to extending the Ladder network with supervised learning to steer the representation learning process to features relevant for the supervised learning problem.

Salimans et al. [34] improved upon the existing GAN training process with several innovations. One in particular is relevant in the context of this thesis. They observed that one of the major modes of failure in GANs is the collapse to similar generator outputs. Every generator input in a mini-batch is processed independently so the generator can transform all its inputs to a point in data space that is rated highly by the discriminator. When this occurs the following discriminator update recognizes the collapsed generator sample as fake and the generator update starts moving the single output around in data space. This common output is never disentangled due to the independent processing of samples. Salimans et al. propose a method for the discriminator to account for the similarity in a mini-batch allowing for easy identification of generator collapse and potentially a means to escape this collapsed mode.
Their method involved taking the features outputted by an intermediate layer of the discriminator and stacking them in the vector $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^A$. B different matrices $M_b \in \mathbb{R}^{C \times A}$ are applied to the feature vectors for samples $\boldsymbol{x}_n$ in the mini-batch. A similarity function is associated with every matrix defined as follows:

$$c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-||M_b(\boldsymbol{x}_i - \boldsymbol{x}_j)||_{L1}) \in \mathbb{R}$$

The mini-batch discrimination layer defines its output component as follows for a mini-batch of size J: $o(\boldsymbol{x}_i)_b = \sum_{j=1}^{J} c_b(\boldsymbol{x}_i, \boldsymbol{x}_j)$ The total mini-batch discrimination output vector is then defined as the B different components:

$$o(\boldsymbol{x_i}) = [o(\boldsymbol{x}_i)_1 \; o(\boldsymbol{x}_i)_2 \cdots o(\boldsymbol{x}_i)_B] \in \mathbb{R}^B$$

The mini-batch discrimination layer is combined with the intermediate features corresponding to sample $\boldsymbol{x}_i$ and is fed to the next layer in the neural network as one

single vector $\boldsymbol{v}_i$:

$$\boldsymbol{v}_i = [\boldsymbol{f}(\boldsymbol{x}_i)^T \; o(\boldsymbol{x_i})^T]^T$$

In the experiments by Salimans et al. on learning generative models for images the mini-batch discrimination allowed the generator to output visually pleasing results in only a few epochs.

3.3 Semi-supervised Learning

A common application of unsupervised feature learning is learning informative representations from a large database of unlabeled data to improve a supervised learning task. This stands in contrast with directly training a supervised model on the data samples provided with labels. Common in these applications is the limited amount of labeled data being available compared to the unlabeled data set size. However, one can directly combine the two tasks and use a single model to both learn relevant features from the large amount of unlabeled data and perform classification based on the small amount of labels. An intuitive way of looking at this machine learning problem is to use available labels to steer the feature extraction model to features relevant for the classification task compared to learning features without any guidance. This problem is known as semi-supervised learning in literature.

Kingma et al. [35] extended the original VAE for use in such a semi-supervised learning context. The first approach applies a regular VAE to the unlabeled data and later uses the encoder to extract features for the labeled data points to train a classifier. Kingma et al. named this approach M1 in their publication and does not differ from the regular pipeline of unsupervised feature learning and training a classifier on the extracted features. The second approach involved a minor change to the graphical model underlying the VAE formulation. The regular formulation $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$ is changed to use an additional stochastic variable y representing the class label for use in a classification task:

$$p(\boldsymbol{x}, \boldsymbol{z}, y) = p(\boldsymbol{x}|\boldsymbol{z}, y)p(\boldsymbol{z})p(y)$$

The original approximate posterior distribution $q(\boldsymbol{z}|\boldsymbol{x})$ is extended to include the class label and is factorized as follows:

$$q(\boldsymbol{z}, y|\boldsymbol{x}) = q(\boldsymbol{z}|\boldsymbol{x})q(y|\boldsymbol{x})$$

Kingma et al. adapted the variational lower bound to incorporate the new formulation for the posterior distribution. After training the model can still be used as a generative model by generating a sample for $\boldsymbol{z}$ and specifying a class label. The distribution $q(y|\boldsymbol{x})$ can be used as classifier for new data points. This approach is named M2. Kingma et al. also provided an extension to the M2 model called M1+M2 by adding an additional stochastic layer. They reported very competitive results outperforming the state of the art semi-supervised learning algorithms at the time of publication.

Salimans et al. [34] extended the regular GAN formulation to also include a classifier. This allows using GANs for semi-supervised learning. In their work the discriminator $D(\boldsymbol{x})$ was extended to produce two outputs: one output corresponding to the regular probability of a sample being real and another output corresponding to the desired classifier. One can change the final loss function to jointly train the GAN and the classifier allowing both training modalities to share information. The training process is changed to include the classifier and it now consists of three steps. The generator and discriminator still compete which each other to learn a generating distribution corresponding to the unlabeled dataset but are alternated by a training step for the classifier which shares part of its network with the discriminator. The classifier is able to leverage the features learned by the discriminator during regular GAN training to improve its own performance. This approach is still considered to be a very competitive semi-supervised learning algorithm.

Rasmus et al. [36] further built upon the ladder networks introduced by Valpola and extended them for semi-supervised learning. They reported even better classification results than using GANs. Introducing the algorithm is considered to be outside of the scope of this thesis due to the complexity of their method. Semi-supervised ladder networks and the Virtual Adversarial Training routine introduced by Miyato et al. [37] are considered the current state of the art in semi-supervised learning.

Chapter 4

Data

This chapter discusses the dataset used for later model training. The dataset consists of a part labeled by expert clinicians and a part that only contains the EEG recordings without labels. First, the labeled data is discussed and secondly the unlabeled data. The chapter concludes with an explanation on the split in training, validation and testing data.

4.1 Labeled Data

The labeled data used to train models was the same as utilized by Ansari et al. [8] in their study of an end-to-end supervised learning approach to sleep stage classification. The data was recorded at the Neonatal Intensive Care Unit, NICU, of the University Hospitals in Leuven, Belgium. The Ethics Committee of the University Hospitals provided approval for the recordings and informed parental consent was obtained. In total the recordings of 26 preterm infants were used all born before 32 weeks of gestational age. For each infant at least two recordings were performed during the stay in the NICU with the number of recordings resulting in a total of 97 multichannel recordings and 492 hours of EEG data. Signals were recorded for infants in a range of 27 and 42 weeks of postmenstrual age, PMA. As reported by Dereymaeker et al. [7] the infants had a normal neurodevelopmental outcome score at 9 and 24 months corrected age (Bayley Scales of Infant Development-II, mental and motor score >85). No subjects were under the influence of sedative or anti-epileptic medication during recording or suffered from severe cerebral lesions (normal cerebral ultra-sonography or intraventricular haemorrhage grade ≤II, no periventricular leukomalacia or ventricular dilatation >p97).

The recordings were provided as an 8-channel signal making use of the F1, F2, C3, C4, T3, T4, O1 and O2 electrode with Cz as reference according to the modified international 10-20 system as explained in the work by Cherian et al. [38] Their publication included an illustration of the placement of the electrodes. The data was acquired using BrainRT equipment from OSG bvba, Belgium. Initial filtering of the recordings was performed with a bandpass filter between 0.3 and 70Hz using a

sample rate of 250Hz. Two independent expert clinicians annotated the quiet sleep segments upon consensus. Wakefulness, active sleep and indeterminate sleep were merged and labelled as non-quiet sleep. This results in a severely unbalanced dataset with only 122 hours of recordings corresponding to quiet sleep and 370 hours with the combined non-quiet sleep. The challenge for the machine learning algorithm will be to classify a given recording segment into quiet sleep or non-quiet sleep with this unbalanced dataset.

Additional preprocessing is performed on the data to make it more suited for use in a machine learning context. The sampling rate of 250Hz results in data with a dimensionality too high for realistic computing times. Another bandpass filter is applied to the data between 1 and 15Hz before downsampling to 30Hz. All frequency bands identified as relevant for sleep stage classification by Piryatinska et al. [5] remain present in the decimated signal, only the β band $(12 - 30\text{Hz})$ is severely affected. The EEG recordings are split into 30s segments resulting in a signal of $30\text{Hz} \times 30\text{s} = 900$ datapoints to be classified by a machine learning algorithm. Figure 4.1 shows a quiet sleep and non-quiet sleep segment with the preprocessing steps applied to it. Finally, the quiet and non-quiet sleep annotations are mapped to a scalar $y \in (0, 1)$ with 0 corresponding to non-quiet sleep and 1 to quiet sleep.

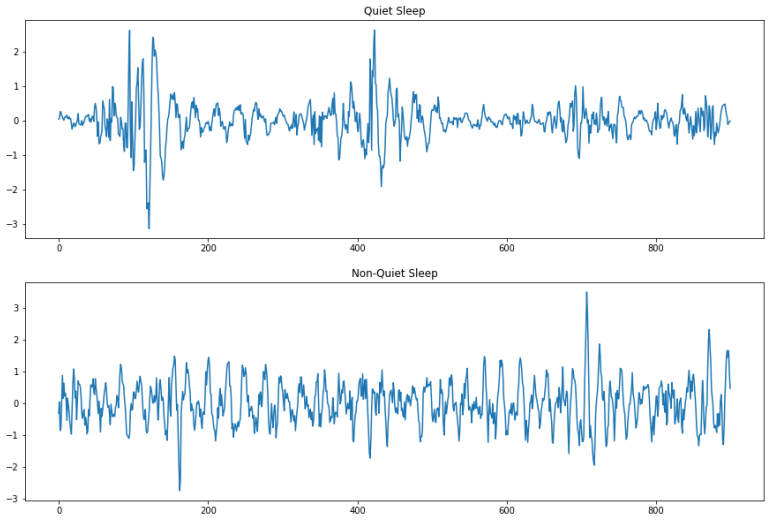

Figure 4.1: Quiet Sleep segment compared to Non-Quiet Sleep

4.2 Unlabeled Data

Seven additional recordings are included in the dataset for this thesis. These recordings combine to form an extra 109 hours worth of EEG signals. They also originate from the NICU of the University Hospitals in Leuven and were recorded under the same modalities as the labeled data above. The 8-channel signal was subject to all preprocessing steps as described above: bandpass filtering and downsampling to 30Hz. Nonoverlapping 30s segments were formed resulting into an additional 13 154 datapoints to leverage in learning algorithms to improve sleep stage classification. It should be explicitly noted that no information is known on the balance between quiet and non-quiet sleep for these recordings as they are unlabeled, these are raw EEG recordings. None of the infants recorded for the unlabeled dataset are included in the original labeled dataset.

4.3 Training/Validation/Test Split

Ansari et al. [8] already split up their data into a partition used for building the model and for testing. They made their split based on the patient being recorded and not purely based on the recordings to avoid leakage of patient-specific information from training set to test set. 54 recordings from 13 patients were used for training and 43 recordings from 13 different patients used for testing. This thesis makes use of the same split to make direct comparison possible. The split results in 69h quiet sleep and 200h non-quiet sleep for the training set. The test set consisted of 53h quiet sleep and 170h non-quiet sleep.

Both the training and testing set cover a wide range of infant PMA. Histograms to illustrate the available data at a specific PMA are computed. Due to the high diversity in recording duration the histograms are not computed based on the number of recordings but they take into account the number of segments at a specific PMA. Figure 4.2 shows the PMA histogram for both the training data and the test data. Both datasets span the range of $28 - 42$ weeks PMA but are clearly centered in the middle of this range.

A good machine learning pipeline also makes use of a validation set to test intermediate results. Ansari et al. made a random 80/20 split into training and validation from their original dataset respecting the ratio between segments corresponding to quiet and non-quiet sleep. This thesis makes use of another training/validation split. Out of the 13 total infants in the training set three are separated from the training set and their recordings are used for validation. This new split is made to avoid the potential leakage of patient- or even recording-specific information from training to validation and make the validation results more representative of the general recording population. Unlabeled recordings are added to the training set for unsupervised and semi-supervised learning. The full overview of amount of quiet and non-quiet segments for each of the training, validation and test set can be found

in table 4.1.

	Training	Validation	Test	Unlabeled
Quiet Sleep	6 319	1 908	6 303	/
Non-Quiet Sleep	18 780	5 265	20 433	/
Total	25 099	7 173	26 736	13 154

Table 4.1: Training, Validation and Test Segments

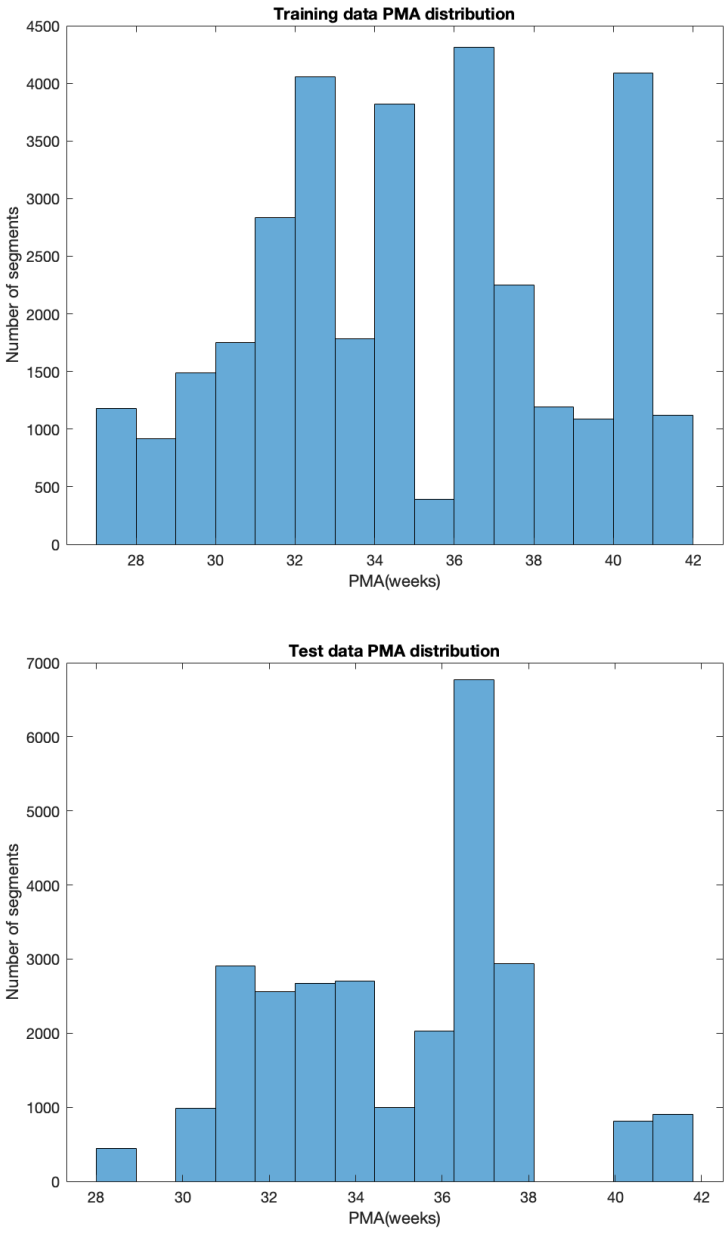

Figure 4.2: PMA Distribution

Chapter 5

Proposed Models

This chapter will discuss the proposed models for unsupervised feature extraction and semi-supervised learning. The main focus has been on single channel data to allow for faster experimentation and visualization. To this end, the algorithms were trained on the difference between electrodes C3 and C4. Training on single channel data also benefits from better numerical stability: fewer problems with exploding or vanishing gradients were encountered. The focus on C3-C4 also has clinical relevance. While the data was recorded in the University Hospitals in Leuven using recording equipment with nine electrodes not all medical centers can make use of such equipment. C3-C4 is a commonly used signal in cerebral function monitoring setups, CFM, [26] making the proposed algorithms relevant for such an environment. First, the data preprocessing steps are introduced. Secondly, the unsupervised learning algorithm is discussed. This chapter concludes with proposing a semi-supervised learning algorithm.

5.1 Data Preprocessing

Segmentation The original labeled dataset and additional unlabeled recordings are combined into 38 253 segments for the training set. This amount of datapoints was insufficient for training the unsupervised models and resulted in severe overfitting. The solution was a preprocessing step on the dataset to artificially increase the number of datapoints. Segments belonging to the same recording were all concatenated to recreate the original recordings. A window of 900 points is applied to the recordings to obtain datapoints with the same dimensionality as the originally provided segments as shown in figure 5.1. The window is applied with a stride < 900 to obtain overlapping segments resulting in a large increase of available datapoints while still only using the original recordings. Setting a stride of 900 results in the original recordings being extracted while setting a stride of 1 would result in an approximate 900 times increase in the number of datapoints, albeit with severe overlap. For the following models a stride of 10 was found to be a good tradeoff between increasing the dataset size sufficiently for training proper representations and increasing computation time. This resulted in an approximate 90 times increase in training set segments to about

3 million segments.

Normalizing The data already is centered by the filtering applied beforehand. The standard deviation however remained untouched compared to the original raw recording. For all models below the total standard deviation is calculated across the training set and all data (training, validation and testing) is divided by this standard deviation in an effort to normalize the recordings.

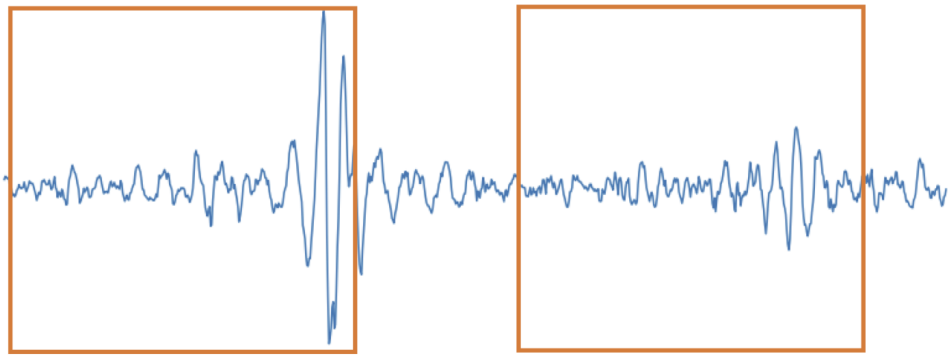

Figure 5.1: Window applied to an EEG recording

5.2 Unsupervised Feature Extraction

For unsupervised learning a variational auto-encoder is proposed. Experiments showed it performed better on later sleep stage classification compared to features extracted by an unsupervised Generative Adversarial Network, even using the improvements to training proposed by Salimans et al. [34]

5.2.1 Setup

Before discussing the specific architectures and training routines the more general setup of the utilized VAE is introduced. A VAE needs initial choices to be made before it can be cast into a general training routine, namely the latent space dimension and the specification of the likelihood distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. As indicated by Kingma et al. [9] a VAE is quite insensitive to the latent space dimensionality as long as the space is large enough to allow for storing the necessary information for reconstruction. Experiments showed a latent space $\boldsymbol{z} \in \mathbb{R}^{50}$ to have a slight edge in later sleep stage classification while being large enough to represent data segments.
The likelihood distribution can take any general form to model the data and in this thesis is specified as a Gaussian with parameters based on the latent variables:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}), \boldsymbol{\sigma}_{\boldsymbol{\theta}}(\boldsymbol{z})^2 I)$$

Specifying the likelihood in this way allows for easy calculation of the $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)})$ term in Equation 2.4 for the training loss.

5.2.2 Network Architectures

Encoder

The probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is specified as $\mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x})^2 I)$. Both $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})$ and $\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x})$ are parametrized by a CNN with a dense layer as the final layer. The networks share a common feature extraction part and have a separate final dense layer. The common part is specified in Table 5.1. Every convolution is followed by a batchnormalization layer and afterwards a ReLU nonlinearity is applied to the features.
For $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})$ a dense linear layer with 50 neurons is applied to match the dimensionality of $\boldsymbol{z}$. A separate dense linear layer with the same dimensionality is applied for $\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x})$. A linear activation however cannot directly be used to model a standard deviation since this output would have to be guaranteed to be nonnegative. The network instead outputs the logarithm of the variance used for the specification of $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ for an easier implementation while still following the theoretical VAE formulation.

Decoder

The decoder or likelihood distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ also uses two neural networks in its formulation. Following the idea of the encoder a common part is used for $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $\boldsymbol{\sigma}_{\boldsymbol{\theta}}(\boldsymbol{z})$. This common part is specified in Table 5.2. The upsampling is performed by copying the values present in the original tensor, no bilinear or cubic interpolation is performed, and is not followed by batchnormalization or an activation function. The convolutional layers are followed by batchnormalization and a ReLU activation.
The decoder follows the same idea of the encoder with a large shared network and the final layer performing the split between the parameters for the required Gaussian. For the decoder this is performed by adding a convolutional layer for both $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $\boldsymbol{\sigma}_{\boldsymbol{\theta}}(\boldsymbol{z})$ with one filter of size 1 followed by a linear activation. Following the encoder the final layer responsible for modeling $\boldsymbol{\sigma}_{\boldsymbol{\theta}}(\boldsymbol{z})$ is considered to output the logarithm of the variance instead of the standard deviation directly.

Layer	# Filters	Filter size	Stride	Output shape
1	16	15	1	$(900, 16)$
2	32	15	15	$(60, 32)$
3	32	15	1	$(60, 32)$
4	64	15	15	$(4, 64)$
5	128	4	4	$(1, 128)$

Table 5.1: VAE Encoder Common Architecture

Layer	# Filters	Filter size	Stride	Output shape
		4× upsampling		$(4, 50)$
1	128	4	1	$(4, 128)$
		15× upsampling		$(60, 128)$
2	64	15	1	$(60, 64)$
		15× upsampling		$(900, 64)$
3	32	15	1	$(900, 32)$
4	32	5	1	$(900, 32)$

Table 5.2: VAE Decoder Common Architecture

5.2.3 Training

Basic training process

The training process involves calculating Equation 2.4 for every segment in a mini-batch. For the encoder $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})$ and $\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x})$ are computed for a data segment and are used to compute the KL-divergence term between the encoder distribution and the latent prior. A sample from latent space is drawn from $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and used to calculate the second term of Equation 2.4. Such a setup allows for easily applying backpropagation through the encoder and decoder network.

Warmup

Directly using the full variational lower bound as a loss function does not lead to a useful model. In all experiments this led to the decoder collapsing to a zero-centered Gaussian and the encoder failing to extract meaningful features for later classification, the classifiers barely performed better than random. An example of such a collapsed reconstruction together with a random sample drawn from the decoder distribution can be found in figure 5.2. Sønderby et al. [39] encountered a similar collapse in working with directed graphical models similar to a VAE but with more stochastic layers. In their work the deeper stochastic layers all collapsed to the prior and failed to learn meaningful representations. Changing the loss function in the first N_t training epochs alleviated the problem and avoided collapse. The variational lower bound was formulated with an additional parameter β:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = -\beta \cdot D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})]$$

Sønderby et al. linearly increased β from 0 to 1 in the first N_t training epochs as a "warmup" stage and reported a substantial drop in feature collapse during training in the stochastic layers.
In this work the training process also involved 5 epochs of using warmup. Slowly increasing the influence of the KL-divergence term had a positive effect on the training process and allowed the decoder distribution to better follow the original input segment; the mean was observed to mimic a smoothed copy of the EEG segment compared to being zero-centered.

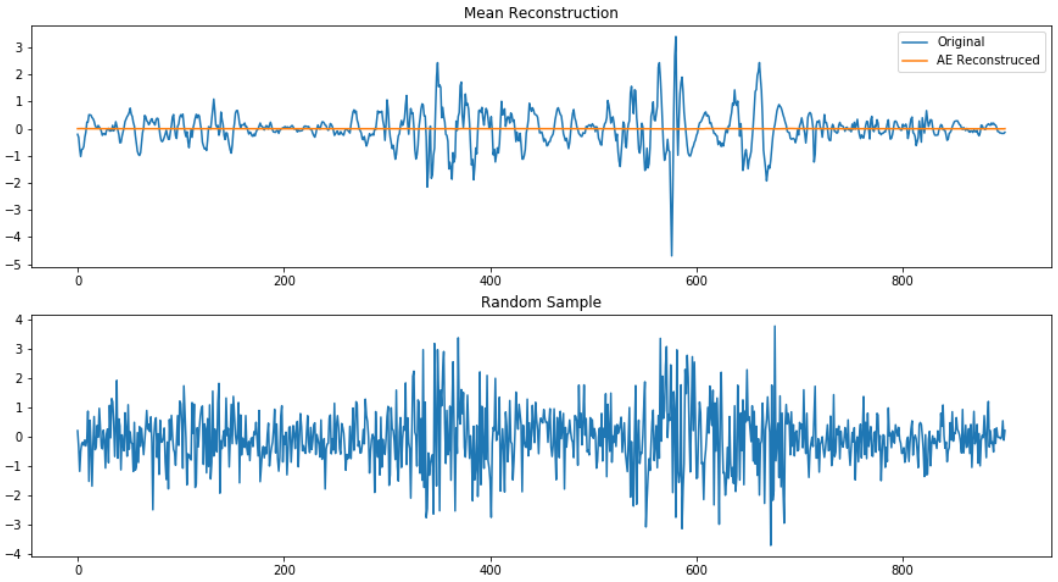

Figure 5.2: VAE Decoder Collapse

Optimization

Training was performed using the Adam optimizer with a default learning rate of 0.001. A significantly higher or lower learning rate resulted in the same collapse as reported above for training without warm-up. The gradient L2-norm was clipped to a value of 1 during training and was also a necessity to avoid collapse of the decoder. The idea of clipping the gradient to stabilize training was originally proposed in the work by Sutskever et al. [40] A single batch consisted of 512 segments.
After performing five warm-up epochs the networks were trained for 100 epochs while monitoring the variational lower bound of the validation data. Early stopping was not used but the network parameters corresponding to the lowest validation loss were stored and used for future calculations.

5.2.4 Results

The final encoder and decoder can be assessed on their ability to reconstruct a given segment. Sampling a point in latent space from the encoder distribution and taking the mean values of the decoder distribution given this latent point should in an ideal case mimick the original segment when the VAE has finished training. Figure 5.3 shows several reconstructions for segments in the training set and figure 5.4 for segments in the validation set. Both figures show reconstructions as smoothed versions of the original EEG segments.
The decoder network should in theory also be able to generate new EEG samples when provided with inputs drawn from a unit Gaussian. Figure 5.5 shows two such samples, each with the decoder mean and a random sample drawn from the decoder distribution. The mean of the random samples can be regarded as a smooth

approximation of a real EEG signal and as such the generative character of the decoder is quite decent. The true samples drawn from the decoder distribution leave a lot to be desired. These samples are very noisy and appear to contain a larger high frequency component compared to the ground truth signal segments.

5.3 Semi-Supervised Learning

The proposed semi-supervised learning algorithm is a GAN inspired by the work by Salimans et al. [34] making use of Mini-Batch Discrimination. Salimans et al. introduced two different approaches to improving and stabilizing the GAN training process: Feature Matching and Mini-Batch Discrimination. They reported better semi-supervised performance using Feature Matching but experiments showed that in the context of sleep stage classification Mini-Batch Discrimination provided a better performing classifier. The final architecture for use in semi-supervised learning is elaborated on below.

5.3.1 Mini-Batch Discrimination Implementation

Before introducing the semi-supervised learning algorithm a software implementation aspect has to be discussed. The Keras library which is used extensively throughout this thesis does not contain a layer type to perform mini-batch discrimination. GitHub user mcgibbon has extended the Keras Layer class with his own implementation [41]. A small modification had to be made to make it work in the context of this thesis. The minibatch discrimination layer is added after the convolutional part of the discriminator, it thus acts on the highest extracted features which are also used later for sleep stage classification. Setting the number of mini-batch kernels to 10 and their dimension to 20 was found to be a good setting for generating visually pleasing samples. The number of mini-batch kernels corresponds to the B matrices being generated and the kernel dimension to the dimension C of feature vectors after transformation with a matrix M_b as explained in Chapter 3.

5.3.2 Network Architectures

Discriminator

The discriminator network is split up into a feature extraction part making use of convolutional layers which is shared for sleep stage classification and distinguishing between real and generated samples. Table 5.3 shows the architecture of this feature extraction part. As proposed by Radford et al. [31] a LeakyReLU activation function is used throughout this part of the discriminator with a slope of 0.2 and dropout with a dropout rate of 0.2 is applied after every activation function. They also proposed to perform batchnormalization with momentum of 0.8 before every activation function except for the first layer of the network. Sleep stage classification is performed by placing a densely connected neuron with sigmoid activation function on top of the feature extraction part without using batchnormalization or dropout. The feature

extraction part is also appended with a mini-batch discrimination layer which adds 10 additional mini-batch features. A second densely connected neuron with sigmoid activation function acting on the new feature vector including the mini-batch features performs the classification between real and generated samples.

Generator

The generator makes use of a sample $\boldsymbol{z} \in \mathbb{R}^{500}$ drawn from a Gaussian with an identity covariance matrix. Table 5.4 contains the generator architecture. The network uses a regular ReLU activation function throughout except for the final layer which uses an identity activation function. Batchnormalization with momentum of 0.8 is applied before every activation function except in the final layer.

Layer	# Filters	Filter size	Stride	Output shape
1	16	15	1	$(900, 16)$
2	32	15	15	$(60, 32)$
3	32	15	1	$(60, 32)$
4	64	15	15	$(4, 64)$
5	128	4	4	$(1, 128)$

Table 5.3: GAN Discriminator Architecture

Layer	# Filters	Filter size	Stride	Output shape
	$4\times$ upsampling			$(4, 500)$
1	128	4	1	$(4, 128)$
	$15\times$ upsampling			$(60, 128)$
2	64	15	1	$(60, 64)$
	$15\times$ upsampling			$(900, 64)$
3	32	15	1	$(900, 32)$
4	32	5	1	$(900, 32)$
5	1	1	1	$(900, 1)$

Table 5.4: GAN Generator Architecture

5.3.3 Training

Two different kinds of minibatches have to be provided during training: labeled and unlabeled. Both the labeled and unlabeled batches had a size of 512. The labeled batches respected the label proportion in the larger dataset, every batch contained one Quiet Sleep segment for every three Non-Quiet Sleep segments. A single epoch is finished when the full unlabeled dataset has been presented to the networks, the labeled dataset is shuffled whenever it has been traversed completely to generate

new batches during the entire unlabeled epoch.

As advised by Salimans et al. the positive labels are smoothed to 0.9 to avoid overconfidence in the discriminator. Training makes use of the Adam optimizer with learning rate set to 0.0002 and the β_1 momentum term set to 0.5 as advised by Radford et al. [31] Training was performed for 20 epochs and during training Cohen's Kappa coefficient is monitored on the validation set. The weights corresponding to the best Kappa coefficient during training are stored.

The training process of the generator is revised compared to the original algorithm by Goodfellow et al. [10] During training the generator would often struggle to maintain a diverse mix of output modes and collapse to a single mode despite using mini-batch discrimination with a negative effect on the final classification performance. Performing multiple generator optimization steps on newly generated noise samples for every discriminator step lessened the likelihood of mode collapse and improved classification performance. Regular GAN training resulted in a validation kappa coefficient of 0.637 and the revised training scheme with three generator iterations resulted in 0.643.

Results

The main goal of GAN training in this work is obtaining a good classifier. Despite this the GAN can still be assessed on its ability to generate realistic samples. Figure 5.6 shows generated samples for regular GAN training and Figure 5.7 shows samples for the revised training scheme. The revised scheme overall resulted in a larger amount of realistically looking samples to an untrained eye but both training routines should clearly be substantially improved when the goal would be to generate realistic EEG segments.

5.4 Implementation

All models and auxiliary routines are implemented in Python. The Keras [42] and Tensorflow [43] libraries were used to build models and test algorithms. Performance metrics were calculated making use of the methods provided by the scikit-learn library [44]. Tree boosting was performed using the XGBoost library [17] using the Python wrapper.
Computations were performed on a system with an Intel i5-6500 CPU and 8GB of RAM. Neural networks were trained on a Nvidia GTX 1080Ti GPU. Using this hardware the VAE epochs ran for 40 minutes each and the GAN epochs took 30 minutes each.

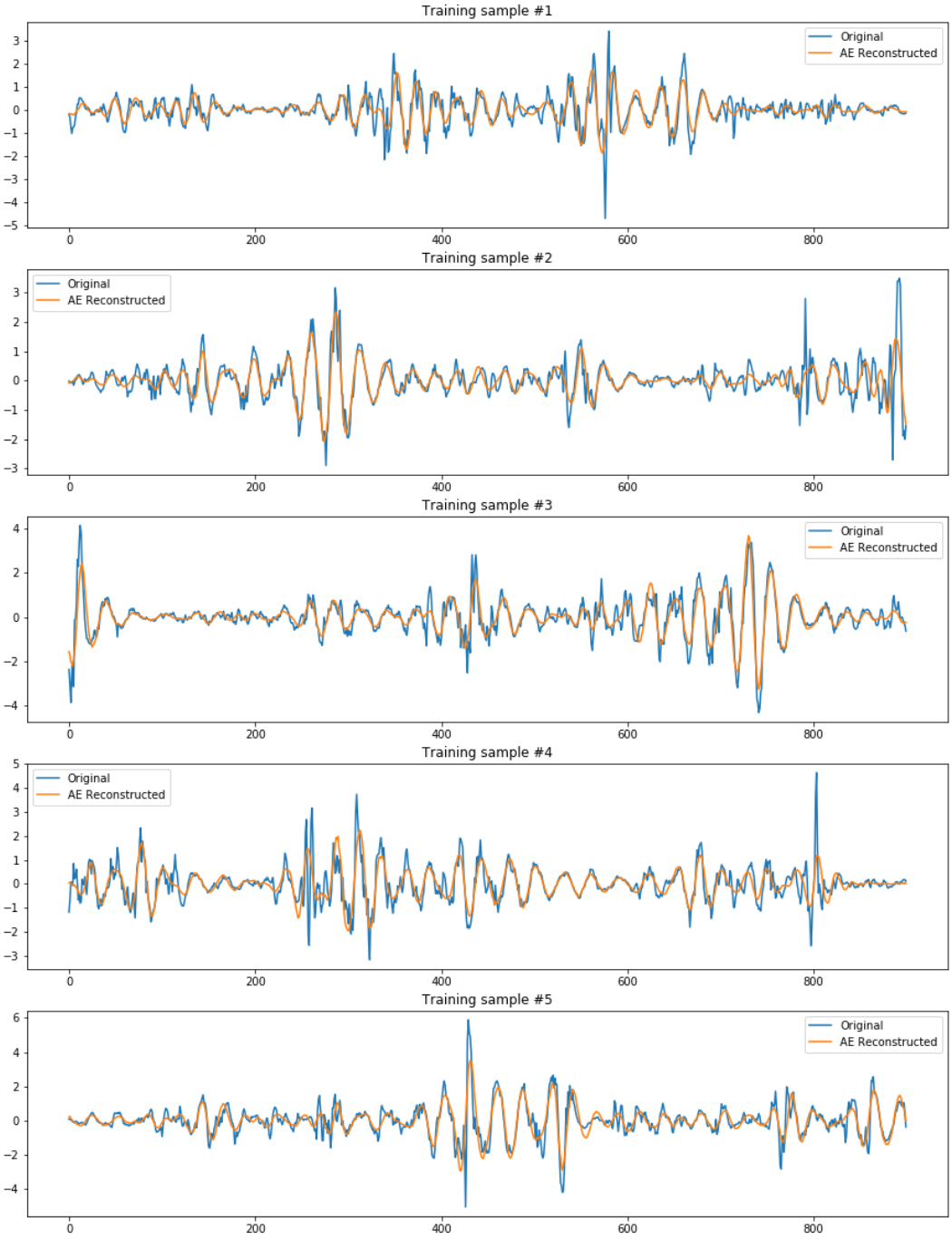

Figure 5.3: VAE Training Reconstructions

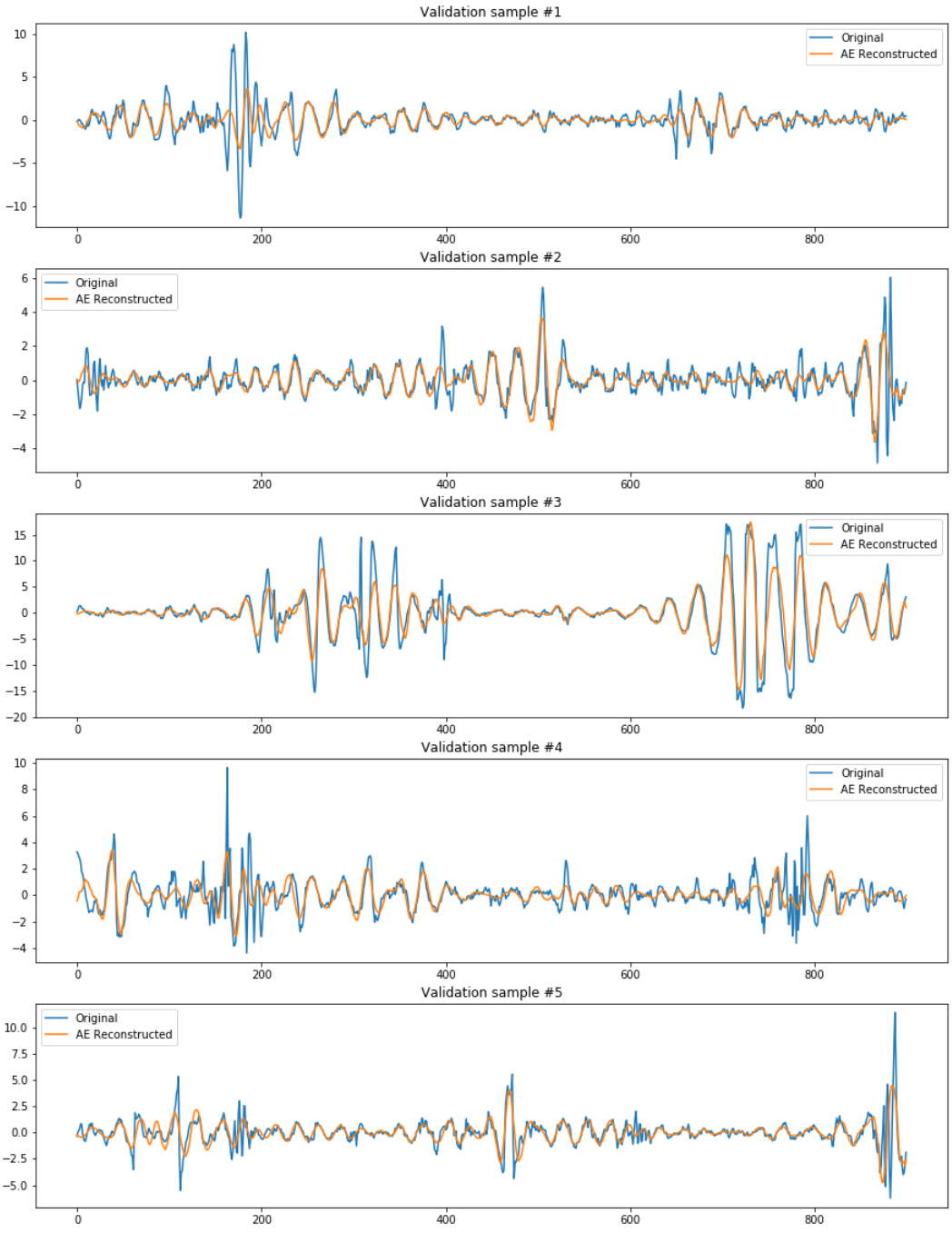

Figure 5.4: VAE Validation Reconstructions

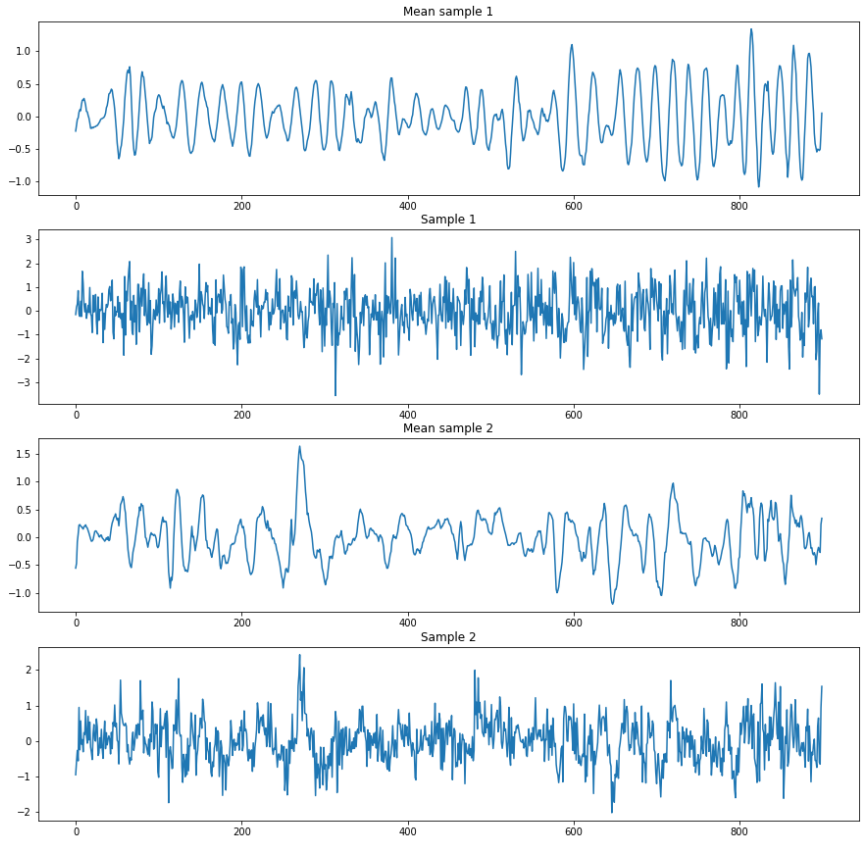

Figure 5.5: VAE Random Samples

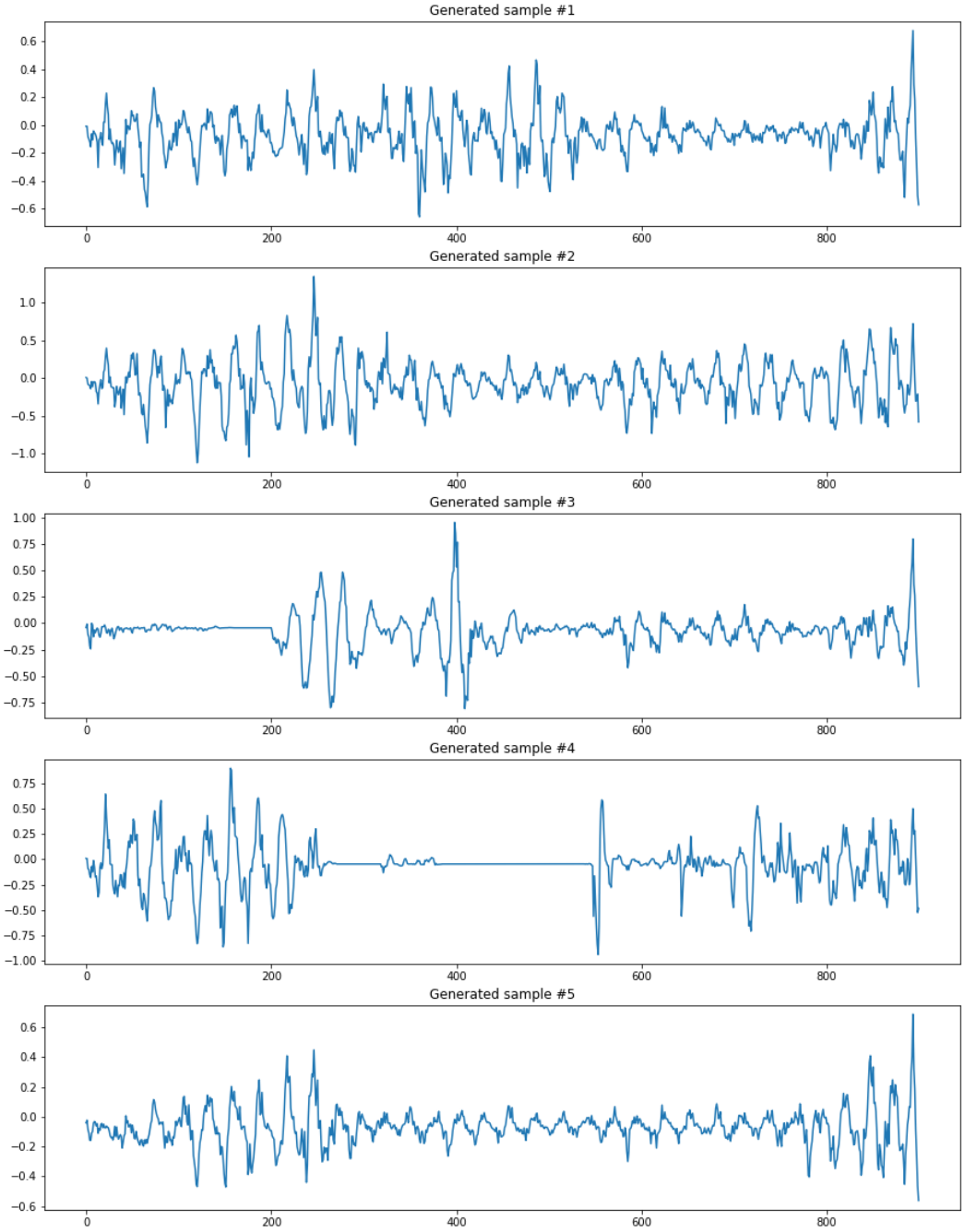

Figure 5.6: Results for Regular GAN Training

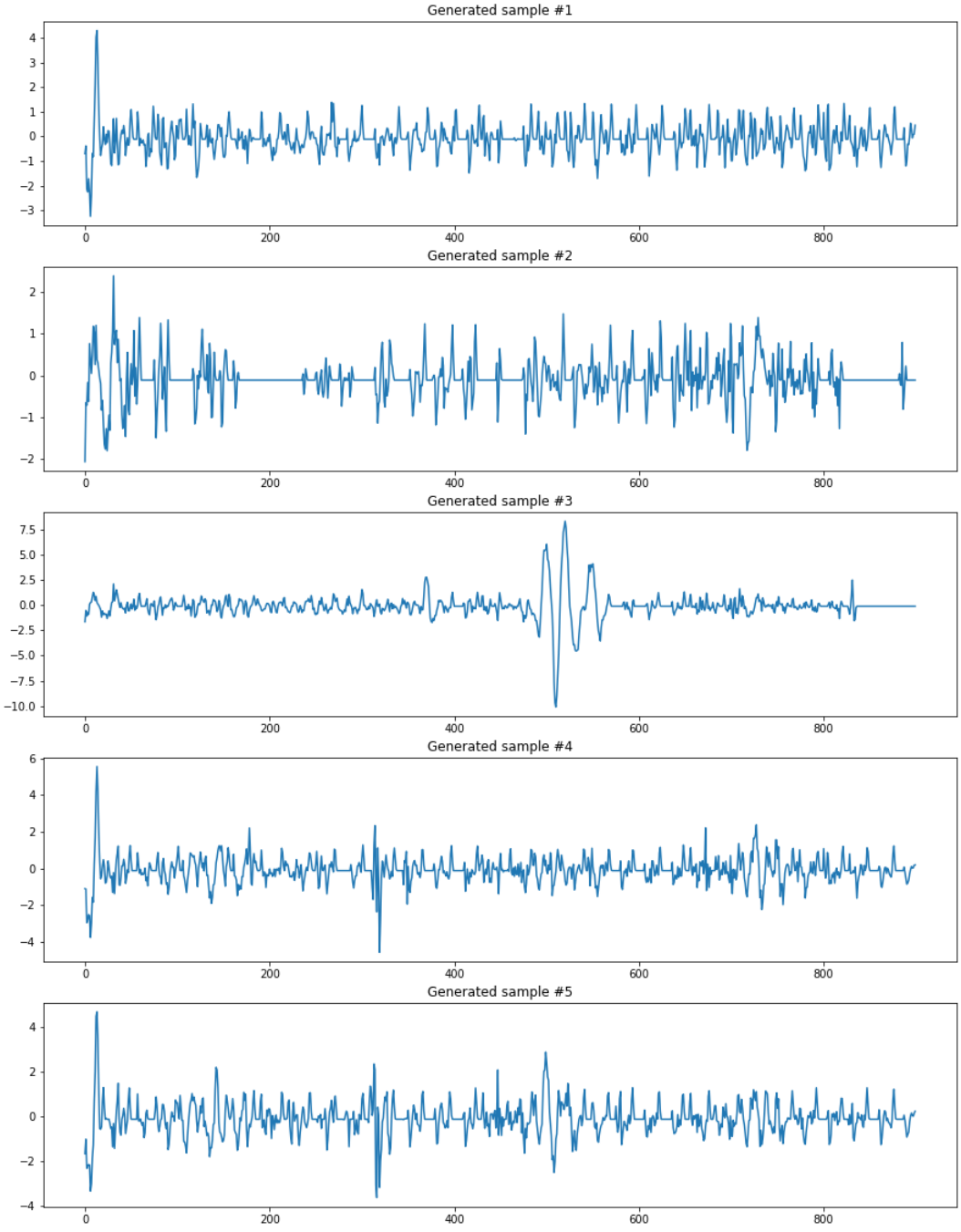

Figure 5.7: Results for Revised GAN Training

Chapter 6

Sleep Stage Classification Results

This chapter will show the final sleep stage classification using the features extracted by the unsupervised training routine and the classification performance of the semi-supervised routine. Both algorithms were trained on the difference between the C3 and C4 channel. The classification performance using only C3-C4 information is reported and an extension is proposed for use in the full 8-channel context.

6.1 C3-C4 Classification Results

This section will discuss the classification results using only signals coming from the difference between the C3 and C4 electrode. The GAN classifier is used as-is since it directly learns to classify segments. For the VAE the probabilistic encoder is used to extract features. The mean value of the approximate posterior distribution $\boldsymbol{\mu}_\phi(\boldsymbol{x})$ represent the extracted features for a given segment. Features are calculated for both the training set and test set. A tree boosting classifier [17] is trained on the training features and used for the final classification. The tree ensemble consists of 5 000 trees with a maximum depth of 6 nodes. Similar to the work by Ansari et al. [8] a moving average filter is used for both algorithms as postprocessing, PP. Here a filter of length 5 is used on the estimated probability values. Table 6.1 contains the results for classification only using C3-C4 for the full test set. Results are computed for every recording and reported as *average(standard deviation)* over all recordings. Both algorithms show a substantial boost in performance using the moving average filter. The features extracted by the VAE do still not result in a strong classifier. The semi-supervised algorithm performs very well with AUC over 0.90 and a kappa coefficient in excess of 0.60.

6.2 8-Channel Classification Results

Despite not being trained on the full dataset the trained networks can still be applied to a multichannel input. For the VAE the probabilistic encoder is applied individ-

Metric	No PP		PP	
	VAE	GAN	VAE	GAN
F1	0.52(0.14)	0.66(0.16)	0.59(0.20)	0.71(0.21)
Accuracy	0.77(0.09)	0.85(0.08)	0.81(0.11)	0.86(0.08)
Precision	0.55(0.14)	0.75(0.16)	0.69(0.21)	0.84(0.16)
Recall	0.53(0.19)	0.61(0.19)	0.61(0.26)	0.65(0.23)
AUC	0.78(0.09)	0.88(0.11)	0.85(0.10)	0.92(0.12)
Kappa	0.37(0.16)	0.57(0.19)	0.47(0.22)	0.64(0.23)
Sensitivity	0.53(0.19)	0.61(0.19)	0.61(0.26)	0.65(0.23)
Specificity	0.84(0.12)	0.93(0.07)	0.85(0.18)	0.95(0.05)

Table 6.1: C3-C4 Classification Performance Metrics

ually to every channel and the extracted features are combined in a single vector. A tree boosting ensemble is used for final classification containing 5 000 trees with a maximum of 15 nodes. The GAN classifier is also applied individually to every channel and the final predicted value is obtained by averaging the output for every channel. The same moving averaging filter of size 5 as for the single channel case is used here. Table 6.2 reports the performance metrics averaged over the recordings of the test set.

Using multiple channels has a strong positive effect on the unsupervised feature extraction algorithm with an increase of 0.10 for the kappa coefficient after post-processing. It remains still a weaker classifier than the GAN on C3-C4 data. The gain for GAN-based classification is negligible with only an 0.01 increase in kappa coefficient after postprocessing.

Metric	No PP		PP	
	VAE	GAN	VAE	GAN
F1	0.64(0.12)	0.71(0.15)	0.67(0.18)	0.73(0.16)
Accuracy	0.82(0.09)	0.87(0.08)	0.85(0.10)	0.88(0.07)
Precision	0.64(0.15)	0.77(0.18)	0.75(0.18)	0.83(0.16)
Recall	0.69(0.17)	0.68(0.18)	0.67(0.23)	0.68(0.21)
AUC	0.86(0.10)	0.90(0.12)	0.90(0.10)	0.92(0.12)
Kappa	0.52(0.17)	0.62(0.19)	0.57(0.22)	0.65(0.19)
Sensitivity	0.69(0.17)	0.68(0.18)	0.67(0.23)	0.68(0.21)
Specificity	0.85(0.13)	0.92(0.08)	0.90(0.14)	0.95(0.06)

Table 6.2: 8-Channel Classification Performance Metrics

Chapter 7

Discussion

This chapter will place the classification performance reported in chapter 6 in more context. First, it will compare the performance of classification using C3-C4 and the 8-channel signal with the current state of the art models as proposed by Ansari et al. [8] [26] Secondly, it will explore the feature space of handcrafted spectral features, features extracted by the VAE probabilistic encoder and intermediate features extracted by the GAN. Thirdly, it will discuss the importance of the VAE features for sleep stage classification. Finally, the performance of the specific methods will be linked with the PMA of the neonates.

7.1 Benchmarking

The final classification results are compared to two approaches. Firstly, the spectral features proposed by Piryatinska et al. [5] are used as example of handcrafting features. Secondly, the end-to-end machine learning algorithm by Ansari et al. [26] is used to represent a more recent machine learning pipeline. Their work presents the results of applying a CNN on both the C3-C4 signal and the full 8-channel dataset using the same data as was provided for this thesis.
The spectral features involve the spectral power in the δ, θ, α and β frequency bands of the EEG, the spectral edge frequency at 90% and 75%, the first spectral density moment, the spectral density entropy and the amplitude entropy. For C3-C4 these features are classified using a tree boosting ensemble consisting of 1 000 trees with a maximum of 2 nodes. For the full 8-channel dataset the results of the SVM applied to these same spectral features by Ansari et al. [8] is used. All reported results use a moving average filter for postprocessing. The proposed VAE and GAN based models use a length of 5 segments for the moving average, as well as the tree boosting ensemble on C3-C4. The results by Ansari et al. are reported for a filter of length 6.

The overview of different approaches to classifying C3-C4 can be found in table 7.1. While implementing an unsupervised feature extraction step clearly is not competitive with end-to-end machine learning pipelines it slightly outperforms a classifier based on hand-crafted features. One can also expect the unsupervised feature extraction

capabilities of the algorithm to improve given more data where the performance of a classifier based on spectral features will always be limited by the expressiveness of these features. Unfortunately, unsupervised feature extraction does not seem to be on the same level as the current state of the art in preterm sleep stage classification.

Semi-supervised learning using GANs on the other hand is shown to be very competitive and is even shown to outperform the CNN by Ansari et al. This supports the conclusion that using a combination of labeled and unlabeled data in a semi-supervised learning setting allows for improving sleep stage classification compared to a pure supervised learning setting. When removing the final layer of the classifier the semi-supervised algorithm is also able to extract significantly more meaningful features than the ones known in literature. [5]

As already shown in Chapter 6 classification performance does not increase substantially when applying the proposed algorithms to all eight channels. Using spectral features on eight channels does allow to formulate a powerful classifier and both the CNN by Ansari et al. and the SVM using spectral features outperform the proposed algorithms on the full channel data.

Metric	VAE	GAN	Spectral features	Ansari et al. [26]
AUC	0.85(0.10)	0.92(0.12)	0.83(0.11)	/
Kappa	0.47(0.22)	0.64(0.23)	0.43(0.20)	0.60(0.31)
Sensitivity	0.61(0.26)	0.65(0.23)	0.66(0.23)	/
Specificity	0.85(0.18)	0.95(0.05)	0.79(0.20)	/

Table 7.1: C3-C4 Benchmarking Results

Metric	VAE	GAN	Spectral SVM [8]	Ansari et al. [26]
AUC	0.90(0.10)	0.92(0.12)	0.93	0.95
Kappa	0.57(0.22)	0.65(0.19)	0.77(0.23)	0.76(0.22)
Sensitivity	0.67(0.23)	0.68(0.21)	0.83(0.28)	0.90(0.22)
Specificity	0.90(0.14)	0.95(0.06)	0.97(0.07)	0.88(0.16)

Table 7.2: 8-Channel Benchmarking Results

7.2 C3-C4 Feature Space Exploration

The feature space used by the different methods for classification of the C3-C4 signal is discussed next. 2-D visualizations are shown for the spectral features, the features extracted by the VAE and the inputs to the final layer of the classifier used by the GAN training procedure. These different feature spaces are projected

from high-dimensional space unto two-dimensional manifolds through PCA and t-SNE. This projection is performed for the training data and testing data with the datapoints getting their respective labels: Quiet Sleep and Non-Quiet Sleep. These plots show the ability of the feature space to naturally separate the two sleep stages. It should be noted that the t-SNE visualizations are not directly comparable, they only show the relative structure in the data the mapping was derived from.

7.2.1 Spectral Feature Space

The spectral features by Piryatinska et al. [5] represent data in nine dimensional space. Figure 7.1 shows the 2-D embeddings for the training data and figure 7.2 shows the embeddings for the test data. No significant differences between the two visualizations should be expected since the features are not learned from data and cannot overfit in the training set. Note that in the PCA plots a substantial portion of points corresponding to non-quiet sleep is hidden behind the quiet sleep projections. PCA shows the impossibility to classify sleep stages using a linear 2-D projection. The t-SNE embedding shows some amount of quiet sleep clusters naturally emerging from the data. These clusters are small or contaminated by a large portion of non-quiet sleep segments. Note that the embeddings do not imply classification to be impossible or hard, they just show that a classification algorithm cannot rely on the natural structure of the data.

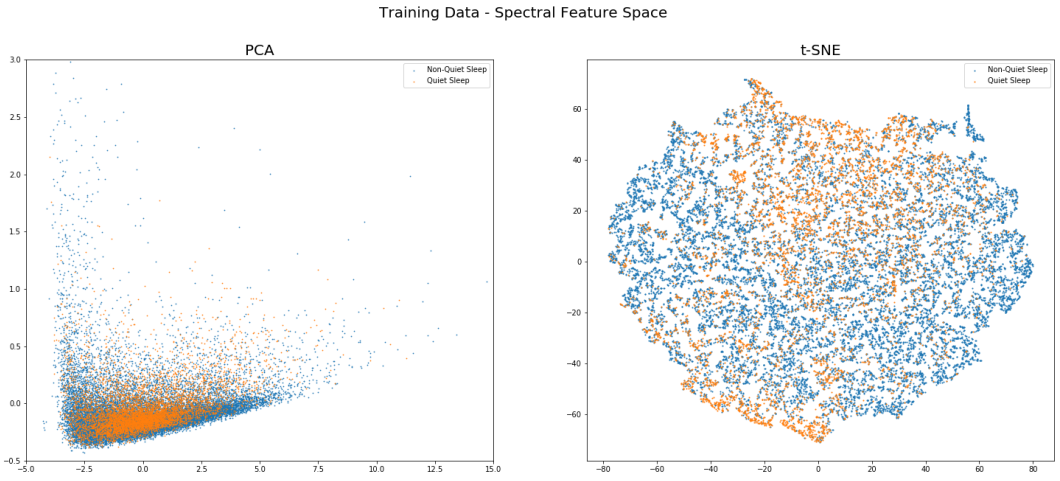

Figure 7.1: C3-C4 Spectral Feature Space for Training Data

7.2.2 VAE Feature Space

The VAE feature space contains points in $\mathbb{R}^{50}$. These points are used directly in the t-SNE algorithm. The embeddings for the training data can be found in figure 7.3 and the embeddings for the test data in figure 7.4. No natural clusters seem to appear, not in a linear or nonlinear embedding. This seems to indicate that

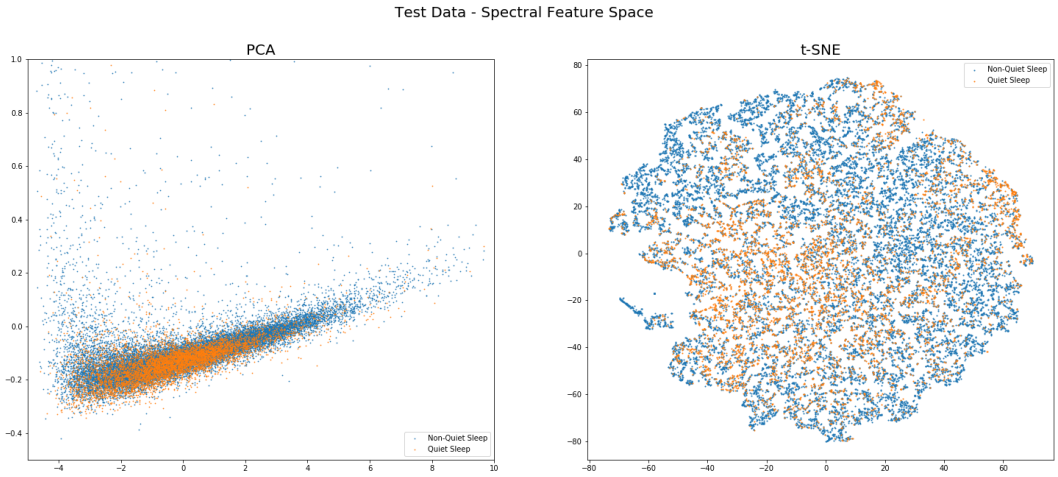

Figure 7.2: C3-C4 Spectral Feature Space for Test Data

classification cannot rely on the natural structure of the data, even more so than the spectral feature space.

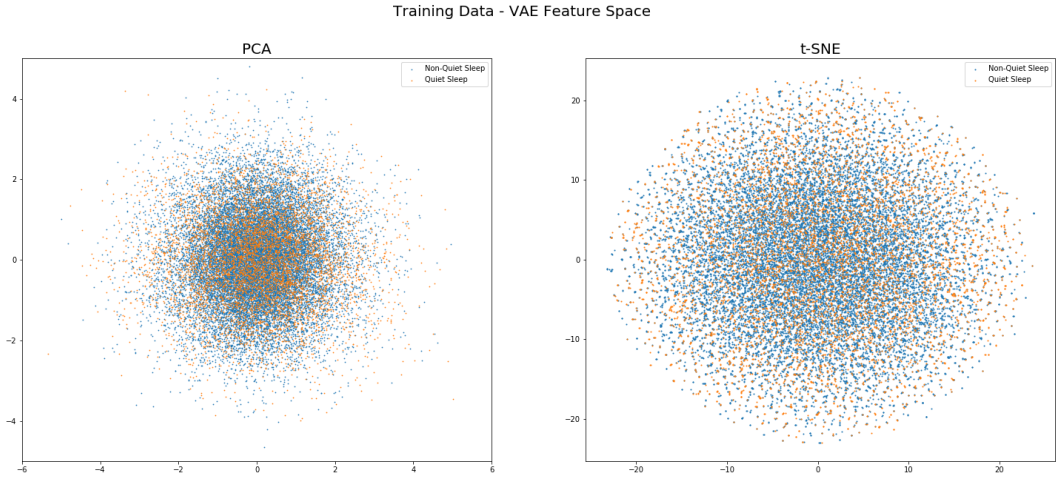

Figure 7.3: C3-C4 VAE Feature Space for Training Data

7.2.3 GAN Feature Space

The GAN classifier represents signal segments in $\mathbb{R}^{128}$ in its final layer. This space is regarded as feature space for the GAN. The PCA embedding is directly computed on the 128-dimensional dataset while for t-SNE it is first reduced to $\mathbb{R}^{50}$ using PCA as advised by the original publication. [13] One can apply t-SNE directly on the original features but as indicated by the authors the additional runtime of the algorithm is often not warranted. Figure 7.5 and figure 7.6 hold the results for the training set and test set respectively. Contrary to the previous results the visualizations show natural

Figure 7.4: C3-C4 VAE Feature Space for Test Data

structure in the feature space splitting quiet sleep and non-quiet sleep segments, even in the test data. This is partly unsurprising due to the strong classification results in the GAN classifier with what is in essence a logistic regression model on top of these features.

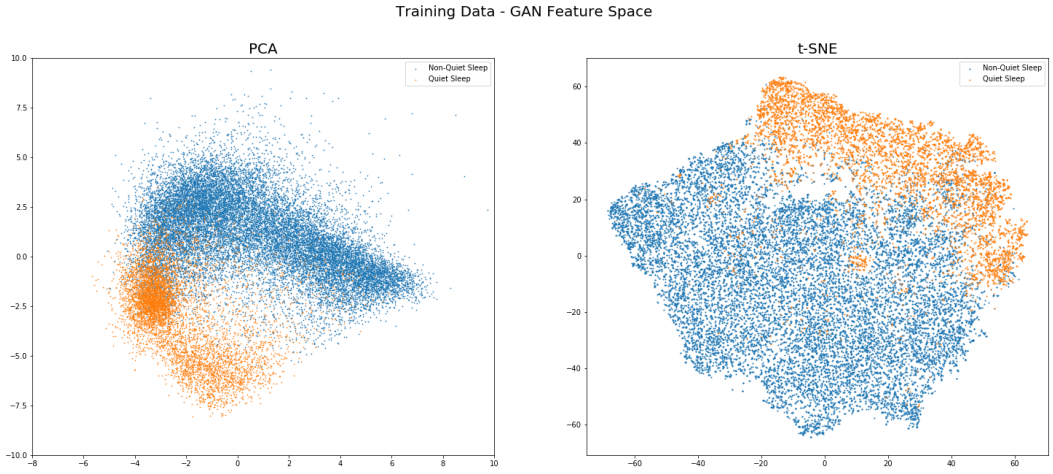

Figure 7.5: C3-C4 GAN Feature Space for Training Data

7.3 Feature Importance

Sleep stage classification based on the VAE model makes use of tree boosting. Such models allow for easy assessment of relative feature importance in the final classification. Figure 7.7 shows such a plot of the relative importance for the features extracted by the VAE probabilistic encoder. The horizontal axis in this plot represents the different extracted features. The importance is calculated as the proportion of total

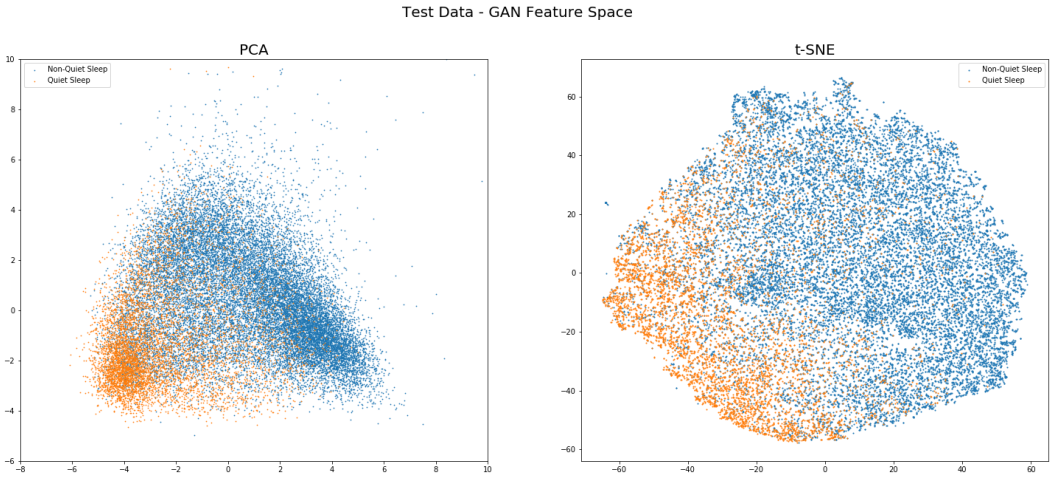

Figure 7.6: C3-C4 GAN Feature Space for Test Data

accuracy gained by performing splits on this feature. Feature #46 clearly is the most informative feature for sleep stage classification but is not solely responsible for the classification performance. Training a logistic regression model on this feature resulted in a kappa coefficient of 0.14(0.22) averaged over the test set indicating the need for the additional features. Nonlinear classification models did not increase the kappa coefficient.

All features being important for sleep stage classification combined with the feature space plots leads to the conclusion that the VAE struggles to disentangle factors of variation linked to sleep stage classification from other factors of variation present in an EEG signal. The GAN succeeds better in this task as evidenced by the stronger performance on sleep stage classification and the natural clustering in the data for segments representing the same sleep stage.

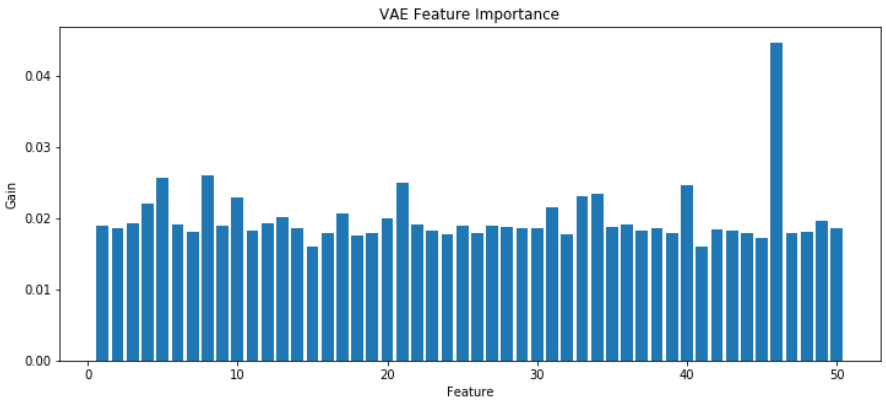

Figure 7.7: VAE Feature Importance

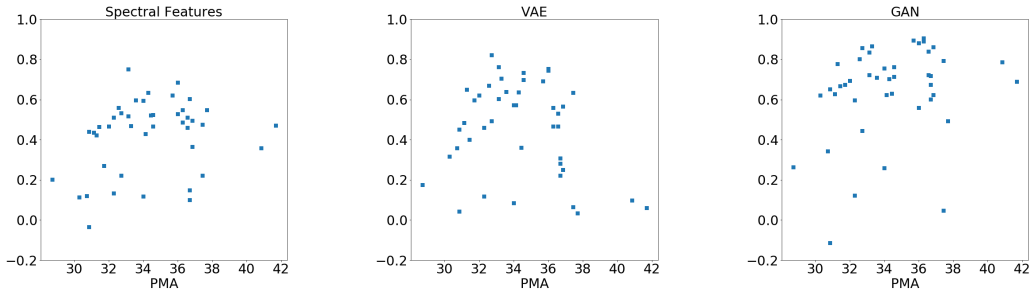

Figure 7.8: Kappa Coefficient for Neonate Age

7.4 Age Factor

The review by Dereymaeker et al. [4] indicates major differences in neonatal EEG patterns related to age. One can expect age to also have a substantial impact on the feature extraction step and the final sleep stage classifications due to the changing patterns. To test the effect age has on classification the kappa coefficient for the different recordings is plotted against PMA of the recorded subject in figure 7.8. This figure is based on classification using C3-C4 for the tree boosting ensemble on the spectral features, for the tree boosting ensemble using VAE features and on the GAN classifier. The results are very scattered but a constant can be identified: average performance is higher for the middle range of PMA values. This middle range also corresponds to most of the training segments as seen in figure 4.2. Whether these results can be attributed to the middle range of PMA being easier to classify or to the inbalance of the training set remains to be seen.

Chapter 8

Conclusion

This final chapter formulates a conclusion to the work and proposes several ideas for future work.

8.1 Conclusion

This thesis proposed an unsupervised deep feature extraction algorithm for neonatal sleep stage classification. It discussed the necessary theoretical background in machine learning, neural networks and variational auto-encoders. Related work was introduced, both on EEG processing and on more general machine learning problems. The dataset used for training is illustrated and split up into a training, validation and test set. A variational auto-encoder making use of convolutional neural networks is trained on the dataset and used to extract descriptive features from the data. Sleep stage classification is carried out using a tree boosting ensemble. Unfortunately, this approach failed to outperform the current state of the art for sleep stage classification using the C3-C4 signal.

Motivated by the underlying goal of the project, using unlabeled recordings to improve classification performance, an additional algorithm is proposed making use of semi-supervised learning. A generative adversarial network is trained on the combination of labeled and unlabeled data and shows superior performance on C3-C4 compared to the state of the art as proposed by Ansari et al. [26]

The lackluster performance of the unsupervised training routine compared to the strong performance of semi-supervised learning leads to the conclusion that the unsupervised learning routine struggles to extract features from the raw data that correspond well with the sleep stage and that providing label information allows a learner to better focus on relevant features. It remains to be investigated whether the unsupervised learning performance stays the same when more data is available to train from. Currently the possibility still exists that the unsupervised algorithm lacks a patient pool diverse enough to properly learn invariant EEG features.

8.2 Future Work

The effect of using the unsupervised training routine as pretraining for a later neural network has not been investigated and can be a first avenue to investigate. While having fallen out of favor recently, unsupervised pretraining stood at the basis of the deep learning boom starting in 2006. [12]

Current models were only trained on C3-C4. All training routines can be extended to include all channels available in a dataset. It should be noted however that applying a comparable variational auto-encoder architecture to the full dataset has been attempted and always led to exploding gradients or collapse as reported in chapter 5. Direct application of the proposed GAN architecture to the full dataset resulted in comparable performance as a GAN trained on C3-C4 resulting in a kappa coefficient of around 0.65 leaving much room for improvement.

The temporal dimension of EEG recordings makes them a natural candidate for applying recurrent neural networks in a processing pipeline. Using LSTMs in an auto-encoder based architecture has been attempted and showed the same decoder collapse as reported in chapter 5. Using shorter segments or applying LSTM modules for semi-supervised learning might still prove to be successful.

The current dataset provides sleep stage labels for a 30s segment sampled at 30Hz. Longer or shorter segments and a higher or lower sampling rate might prove to be more suited for automated sleep stage classification. The current VAE used for unsupervised learning can struggle with processing larger segments resulting in more datapoints due to its inherent need for compressing the data to feature space and relying on gradient flow through the combination of decoder and encoder. More complex networks turn out to be infeasible to use for a VAE. The semi-supervised GAN is expected to show less problems for higher-dimensional inputs. The generator and discriminator are trained separately and the same compression to feature space as for the VAE case is not inherent to the model. The noise distribution used to generate samples needs a high enough entropy to be transformed in a meaningful generator distribution but its easy to sample from a higher-dimensional noise distribution.

Variational auto-encoders are far from the only unsupervised learning algorithms in literature. GANs have also been applied on images for unsupervised automated feature extraction [31] but applying them to the EEG signals proved unsuccessful. A straightforward candidate is the Ladder network by Valpola [33] due to its ability to extract a hierarchy of features. Sleep stage classification might turn out to mainly rely on high-level or fine-grained features and the ladder network should be able to disentangle the two and identify a spectrum of features going from abstract to fine-grained. In the context of semi-supervised learning other algorithms exist that consistently outperform GANs on the artificial benchmarks used by the machine learning community. Both semi-supervised Ladder networks [36] and virtual adversarial training [37] are straightforward candidates. It remains to be seen whether

these algorithms also perform better on EEG data.

Chapter 2 already discussed the lack of good performance metrics for unsupervised learning algorithms. Domain-specific analysis can be investigated in collaboration with medical experts in an effort to assess performance for a clinical context. A first step in such joint analysis might be found in identifying and visualizing the structures a neural networks looks for in the data. Zeiler et al. [45] investigated the intermediate activations in a CNN aimed at image processing. The methodology can be adapted to fit EEG processing and specific features can be visualized.

A final point of future work is investigating the factor of patient characteristics in feature extraction and sleep stage classification. Models can be extended to incorporate this additional information and move to multimodal representations or different models can be considered for specific types of patients after identifying major groups in the patient population.

Bibliography

[1] Chiara Nosarti, Kie Woo Nam, Muriel Walshe, Robin M Murray, Marion Cuddy, Larry Rifkin, and Matthew PG Allin. Preterm birth and structural brain alterations in early adulthood. *NeuroImage: Clinical*, 6:180–191, 2014.

[2] Rita H Pickler, Jacqueline M McGrath, Ms Barbara A Reyna, Nancy McCain, Ms Mary Lewis, Ms Sharon Cone, Paul Wetzel, and Al Best. A model of neurodevelopmental risk and protection for preterm infants. *The Journal of perinatal & neonatal nursing*, 24(4):356, 2010.

[3] Mark S Scher. Ontogeny of eeg-sleep from neonatal through infancy periods. *Sleep medicine*, 9(6):615–636, 2008.

[4] Anneleen Dereymaeker, Kirubin Pillay, Jan Vervisch, Maarten De Vos, Sabine Van Huffel, Katrien Jansen, and Gunnar Naulaers. Review of sleep-eeg in preterm and term neonates. *Early human development*, 113:87–103, 2017.

[5] Alexandra Piryatinska, Gyorgy Terdik, Wojbor A Woyczynski, Kenneth A Loparo, Mark S Scher, and Anatoly Zlotnik. Automated detection of neonate eeg sleep stages. *Computer methods and programs in biomedicine*, 95(1):31–46, 2009.

[6] Ninah Koolen, Lisa Oberdorfer, Zsofia Rona, Vito Giordano, Tobias Werther, Katrin Klebermass-Schrehof, Nathan Stevenson, and Sampsa Vanhatalo. Automated classification of neonatal sleep states using eeg. *Clinical Neurophysiology*, 128(6):1100–1108, 2017.

[7] Anneleen Dereymaeker, Kirubin Pillay, Jan Vervisch, Sabine Van Huffel, Gunnar Naulaers, Katrien Jansen, and Maarten De Vos. An automated quiet sleep detection approach in preterm infants as a gateway to assess brain maturation. *International journal of neural systems*, 27(06):1750023, 2017.

[8] Amir Hossein Ansari, Ofelie De Wel, Mario Lavanga, Alexander Caicedo, Anneleen Dereymaeker, Katrien Jansen, Jan Vervisch, Maarten De Vos, Gunnar Naulaers, and Sabine Van Huffel. Quiet sleep detection in preterm infants using deep convolutional neural networks. *Journal of neural engineering*, 15(6):066006, 2018.

[9] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[11] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[15] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[16] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[17] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention*, 9351:234–241, 2015.

[21] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[22] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[26] A.H. Ansari, O. De Wel, K. Pillay, A. Dereymaeker, K. Jansen, S. Van Huffel, G. Naulaers, and M. De Vos. A convolutional neural network outperforming state-of-the-art sleep staging algorithms for both preterm and term infants. *Internal Report 19-65, ESAT-STADIUS, KU Leuven (Leuven,Belgium)*, 2019.

[27] Qin Lin, Shu-qun Ye, Xiu-mei Huang, Si-you Li, Mei-zhen Zhang, Yun Xue, and Wen-Sheng Chen. Classification of epileptic eeg signals with stacked sparse autoencoder based on deep learning. In *International Conference on Intelligent Computing*, pages 802–810. Springer, 2016.

[28] Junhua Li, Zbigniew Struzik, Liqing Zhang, and Andrzej Cichocki. Feature learning from incomplete eeg with denoising autoencoder. *Neurocomputing*, 165:23–31, 2015.

[29] Yuanfang Ren and Yan Wu. Convolutional deep belief networks for feature extraction of eeg signal. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 2850–2853. IEEE, 2014.

[30] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

[31] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[32] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

[33] Harri Valpola. From neural pca to deep unsupervised learning. In *Advances in Independent Component Analysis and Learning Machines*, pages 143–171. Elsevier, 2015.

[34] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[35] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

[36] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

[37] Takeru Miyato, Shin-ichi Maeda, Shin Ishii, and Masanori Koyama. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[38] Perumpillichira J Cherian, Renate M Swarte, and Gerhard H Visser. Technical standards for recording and interpretation of neonatal electroencephalogram in clinical practice. *Annals of Indian Academy of Neurology*, 12(1):58, 2009.

[39] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

[40] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[41] https://github.com/mcgibbon. Implementation for mini-batch discrimination for keras. `https://github.com/keras-team/keras/pull/3677`. Accessed: 2019-05-28.

[42] François Chollet et al. Keras. `https://keras.io`, 2015.

[43] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[45] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Master's thesis filing card

Student: Nick Seeuws

Title: Unsupervised deep feature extraction for neonatal sleep stage classification

UDC: 681.3*I20

Abstract:
Sleep stage monitoring is an important tool in the clinician's arsenal to provide care for neonates and assess normal brain maturity. Correctly assessing the sleep wake cycles can direct interventions in the NICU and provide better understanding of neurophysiological processes in the earliest stages of life. EEG recordings carry substantial information on these sleep stages and are used to perform sleep stage monitoring in a noninvasive way. Correctly identifying sleep stages from EEG recordings is a challenging task even for experts in the field. Using traditional supervised deep learning approaches to support clinicians demands major effort up front to provide annotated EEG segments for the machine learning model to train from.
This thesis investigates methods to improve the current state of the art in neonatal sleep stage monitoring based on supervised learning by using unannotated EEG recordings. An unsupervised method is proposed to automatically extract relevant features from data and a machine learning classifier is later trained on the extracted features. A semi-supervised model is proposed to directly combine information from annotated and unannotated data. Classification using features from unsupervised learning on the C3-C4 EEG signal reported an average kappa coefficient of 0.47 and classification using semi-supervised learning on the same data reported an average kappa coefficient of 0.64.
The proposed unsupervised learning model was not successful at competing with the state of the art in automated sleep stage classification and is shown to not disentangle factors of variation in the data corresponding to sleep stages. Semi-supervised learning proved to be very competitive with the current state of the art outperforming a supervised deep learning approach using the same data.

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Big Data Analytics

Thesis supervisors: Prof. dr. ir. Sabine Van Huffel
Prof. dr. Gunnar Naulaers

Assessors: Prof. dr. Katrien Jansen
Ir. Ofelie De Wel

Mentor: Dr. ir. Amir Hossein Ansari