FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Detecting abnormalities in clinical data with Generative Adversarial Networks

Bachelorarbeit

im Bachelorstudiengang Angewandte Mathematik und Informatik

von

Matthias Wright

Aachen, August 2018

Fachhochschule Aachen, Campus Jülich

Fachbereich Medizintechnik und Technomathematik

Eigenständigkeitserklärung

Diese Arbeit ist von mir selbständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Aachen, 24. August 2018,

Matthias Wright

Diese Arbeit wurde betreut von:

1. Prüfer: Prof. Dr. Melanie Hollstein
2. Prüfer: Dr. Stephan Jonas

Acknowledgments

I would like to thank my supervisors Prof. Dr. Melanie Hollstein and Dr. Stephan Jonas for their advice as well as for taking the time to read and grade my thesis. Furthermore, I would like to thank my parents for their financial support.

ABSTRACT

In an intensive care unit, the vital functions of a patient are constantly carefully monitored so that doctors can quickly and appropriately respond to life-threatening changes in the patient's condition. In the US, 39.5% of intensive care unit patients are receiving mechanical ventilation. 44.4% of the patients, who were suffering from respiratory failure and received mechanical ventilation died in the hospital. When it comes to mechanical ventilation, it is essential to take measures at the right time, but in order to do that we have to be able to detect changes in the patient's condition.

Another frequent complication that occurs among hospitalized patients is cardiac arrest. In order to detect cardiac complications, patients are monitored with the electrocardiogram (ECG). However, the interpretation of ECG recordings is not a trivial task.

In this study, we employed *Generative Adversarial Networks* to examine whether they are capable of distinguishing between the data of patients who are suffering from respiratory failure or cardiac arrhythmia and the data of healthy patients. Specifically, we looked at the ventilation parameter PEEP and the blood gases pC02 and PO2, which we extracted from the freely available *MIMIC-III Database*. We trained the generative adversarial network with the data of patients who were not suffering from respiratory failure. In order to validate the results, we first fed test data of patients suffering from respiratory failure and then test data of patients not suffering from respiratory failure into the discriminative model and compared the cross entropy losses. The test data of patients suffering from respiratory failure caused an average cross entropy loss of 0.7508 and the test data of patients not suffering from respiratory failure caused an average cross entropy loss of 0.5034. The best performance was achieved with the PO2 data. However, for the test data the area under the ROC curve was 0.57, leading us to the conclusion that either *Generative Adversarial Networks* are not capable of diagnosing respiratory failure or the used data did not contain the necessary information to perform

this distinction.

Furthermore, we trained a generative adversarial network with normal ECG recordings that we extracted from the *The Long-Term ST Database*. We tested the model on the *MIT-BIH Arrhythmia Database*. We fed test data containing normal ECG recordings and test data containing arrhythmic ECG recordings into the discriminative model and compared the cross entropy losses. The test data containing normal ECG recordings caused an average cross entropy loss of 0.0021 and the test data containing arrhythmic ECG recordings caused an average cross entropy loss of 0.0807. The area under the ROC curve was 0.71, leading us to the conclusion that *Generative Adversarial Networks* are capable of distinguishing between the two groups.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Severely ill or injured patients are in need of intensive care. In an intensive care unit, the vital functions of a patient are constantly carefully monitored so that doctors can quickly and appropriately respond to life-threatening changes in the patient's condition. In 70% of the cases, the patient in intensive care only depends on monitoring and vital support for a couple of days with a high chance of survival. However, 30% of intensive care patients stay for longer periods, sometimes even months and the likelihood of survival decreases over time [1]. In the US, 39.5% of intensive care unit patients are receiving mechanical ventilation [2]. A study, conducted in 1995, examined 1426 patients, who were suffering from respiratory failure and received mechanical ventilation. While 55.6% survived their stay in intensive care unit, 44.4% died in the hospital [3]. Respiratory failure is also one of the two commonest causes of death after discharge from intensive care [4]. When it comes to mechanical ventilation, it is essential to take measures at the right time, but in order to do that we have to be able to detect changes in the patient's condition. Usually, this is done by a medical expert, who analyses the available clinical data. However, this data may contain much more information than a human can extract from it. Studies have shown that the human mind is only capable of dealing with seven parameters at the same time [5] but according to estimates, there are approximately 250 parameters for a intensive care unit patient [6].

Another frequent complication that occurs among hospitalized patients is cardiac arrest [7]. Especially unexpected cardiac arrests in the intensive care unit lead to high mortal-

ity rates, only 15% of the patients survive [8]. In order to detect cardiac complications, patients are monitored with the electrocardiogram (ECG) [9]. However, the interpretation of ECG signals is not a trivial task.

Employing machine learning models in order to leverage the massive amounts of available data could benefit the medical personnel as well as the patients.

The goal of this thesis is to develop a *Generative Adversarial Network* and examine whether it is capable of distinguishing between the data of patients suffering from respiratory failure or cardiac arrhythmia and the data of healthy patients. We will train our model only on the data of healthy patients because there is much more data of healthy patients available.

Chapter 2

Background and Related Work

2.1 Generative Models vs. Discriminative Models

There are two basic approaches when it comes to classifying data of any kind. Subsequently, x will denote the input and y the corresponding label. Generative models learn the joint probability $P(x, y)$. The y with highest probability is yielded by computing $P(y|x)$ via Bayes rule. In contrast, discriminative models infer the posterior $P(y|x)$ directly or learn a direct mapping from x to the corresponding label y [11]. In the past, discriminate models have been preferred over generative models because they solve the problem of classification directly instead of solving a more general problem [15]. However, this generalization entails the advantage that we gain knowledge about the data distribution and additionally are able to generate samples.

2.2 Gibbs Sampling for Generating Health Records

An example of a generative model is *Gibbs sampling*. In 2014, Park et al. [12] introduced a variation of *Gibbs sampling* in order to generate synthetic health data on the basis of original data. This so called *Perturbed Gibbs Sampler* is structured in three steps.

Firstly, the original data D is disintegrated into statistical building blocks. These statistical building blocks are the conditional probabilities of a particular feature, conditioned

on the hash value $h(x)$ of the rest of the features for that particular record. For a dataset with four features (x_1, x_2, x_3, x_4) for every record the statistical building blocks would be: $P(x_1|h(x_2, x_3, x_4))$, $P(x_2|h(x_1, x_3, x_4))$, $P(x_3|h(x_1, x_2, x_4))$, and $P(x_4|h(x_1, x_2, x_3))$. These conditional probabilities are estimated by counting the occurrences in the original data. The use of a hash function allows the use of high-dimensional categorical data and also provides computational efficiency.

In the second step, noise is added to the conditional probabilities. This is primarily a measure to ensure privacy.

In the third and final step, a pool of seeds is defined. Starting with a random seed from the pool, a new synthetic sample is generated by iteratively sampling each feature according to the conditional probabilities using *Gibbs sampling*.

A major limitation of this approach is the *Perturbed Gibbs Sampler*'s inability to handle numerical features. Furthermore, this approach also does not provide us with a mean of distinguishing between data of patients with a disease and healthy patients.

2.3 Artificial Neural Networks

An example for a discriminative model is an *Artificial Neural Network*. It is an information-processing paradigm [16] that is inspired from the way the human brain processes information. The goal is to approximate a universal function f. A neural network defines the mapping $y = \hat{f}(x; \theta)$ and learns the value of the parameters θ that minimize the error between the function f and its approximation $\hat{f}$. They are composed of artificial neurons, which are functions that receive one or more inputs and sum them to produce an output. We distinguish between linear neurons and non-linear neurons. Linear neurons solely sum up the inputs and output the result and non-linear neurons additionally apply an activation function ϕ (section 2.3.1) to the result.

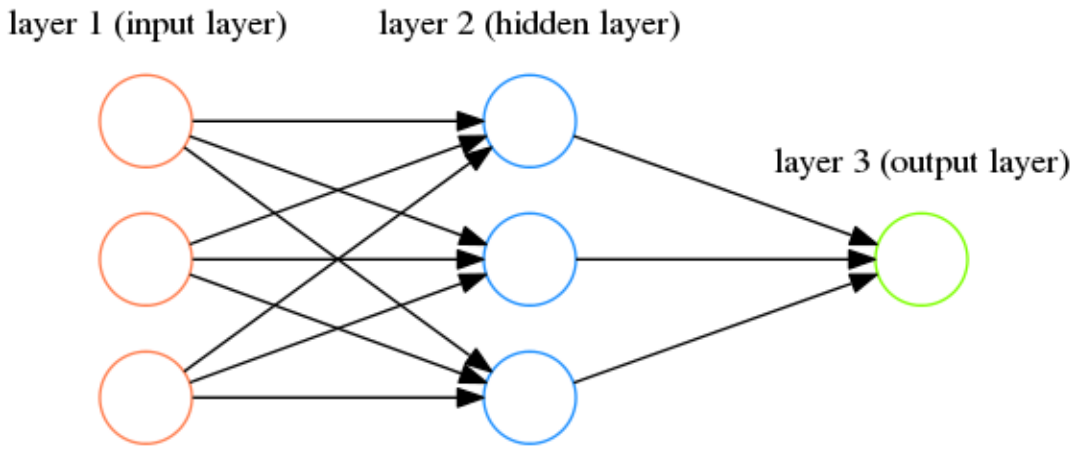

Fig. 2.1: A simple neural network with three input neurons, one output neuron, and one hidden layer with three neurons.

A single artificial neuron is a neural network in its simplest form. However, in practice neural networks are composed of a plethora of neurons that are divided into different layers. The first layer is the input layer that receives the input x, the last layer is the output layer that produces the result $\hat{f}(x)$, and the layers in between are called hidden layers. Figure 2.1 shows a simple neural network. The connections between the neurons represent the parameters θ, which are also referred to as the weights w. The formulas for the following section are taken from chapter 1 of the the book *Neural networks and deep learning* (Nielsen, 2015) [20].

The process of computing $\hat{f}(x)$ for a given x is usually called *feed-forward propagation*. Each neuron output is multiplied with its corresponding weight and *fed* into every neuron in the next layer. Subsequently, $a_i^{(l)}$ will be used to denote the output of neuron i in layer l. The weights between two layers l and $l+1$ with three neurons each can be

represented by a matrix $W^{(l+1)}$. Figure 2.2 explains the notation of the weights.

$$W^{(l+1)} = \begin{bmatrix} w_{11}^{(l+1)} & w_{12}^{(l+1)} & w_{13}^{(l+1)} \\ w_{21}^{(l+1)} & w_{22}^{(l+1)} & w_{23}^{(l+1)} \\ w_{31}^{(l+1)} & w_{32}^{(l+1)} & w_{33}^{(l+1)} \end{bmatrix} \tag{2.1}$$

This process of *feeding* the outputs of layer l into the next layer $l+1$ and computing the outputs of $l+1$ can be condensed into a single multiplication between the weight matrix $W^{(l+1)}$ and the vector $a^{(l)}$, where $a^{(l)}$ denotes the outputs from layer l.

$$\begin{bmatrix} z_1^{(l+1)} \\ z_2^{(l+1)} \\ z_3^{(l+1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(l+1)} & w_{12}^{(l+1)} & w_{13}^{(l+1)} \\ w_{21}^{(l+1)} & w_{22}^{(l+1)} & w_{23}^{(l+1)} \\ w_{31}^{(l+1)} & w_{32}^{(l+1)} & w_{33}^{(l+1)} \end{bmatrix} \cdot \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ a_3^{(l)} \end{bmatrix} \tag{2.2}$$

Applying the activation function ϕ to the vector $z^{(l+1)}$ yields the output vector $a^{(l+1)}$.

$$a^{(l+1)} = \phi(z^{(l+1)}) \tag{2.3}$$

In practice, every layer has an additional neuron, which is not connected to the previous layer. This so called *bias unit* functions like an input node with a constant value of 1. It is connected to every neuron in the next layer, except the bias unit of that layer. The bias unit has essentially the same purpose as the constant b in the linear equation $y = ax + b$, it shifts the function output (of the activation function). Figure 2.3 shows a neural network architecture with bias units.

The bias unit of layer l can be represented by a vector $b^{(l+1)}$, where the component $b_i^{(l+1)}$ is the bias that is connected to the i^{th} neuron in the layer $l+1$. Equation 2.2 is thus

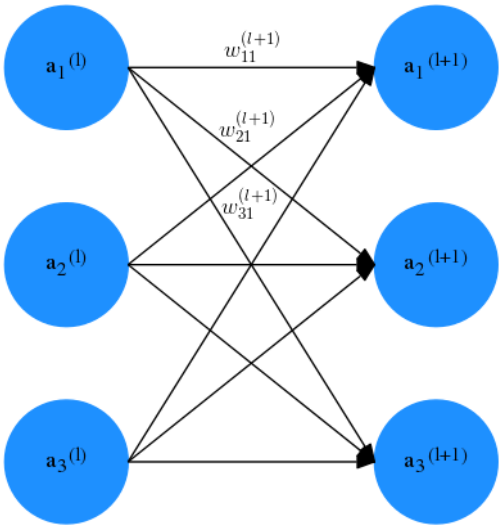

Fig. 2.2: $a_1^{(l)}$ is multiplied with the weight $w_{11}^{(l+1)}$ before it is *fed* into neuron 1 in layer $l+1$, $a_1^{(l)}$ is multiplied with the weight $w_{21}^{(l+1)}$ before it is *fed* into neuron 2 in layer $l+1$, and $a_1^{(l)}$ is multiplied with the weight $w_{31}^{(l+1)}$ before it is *fed* into neuron 3 in layer $l+1$ (the rest of the weights were purposely left out for clarity).

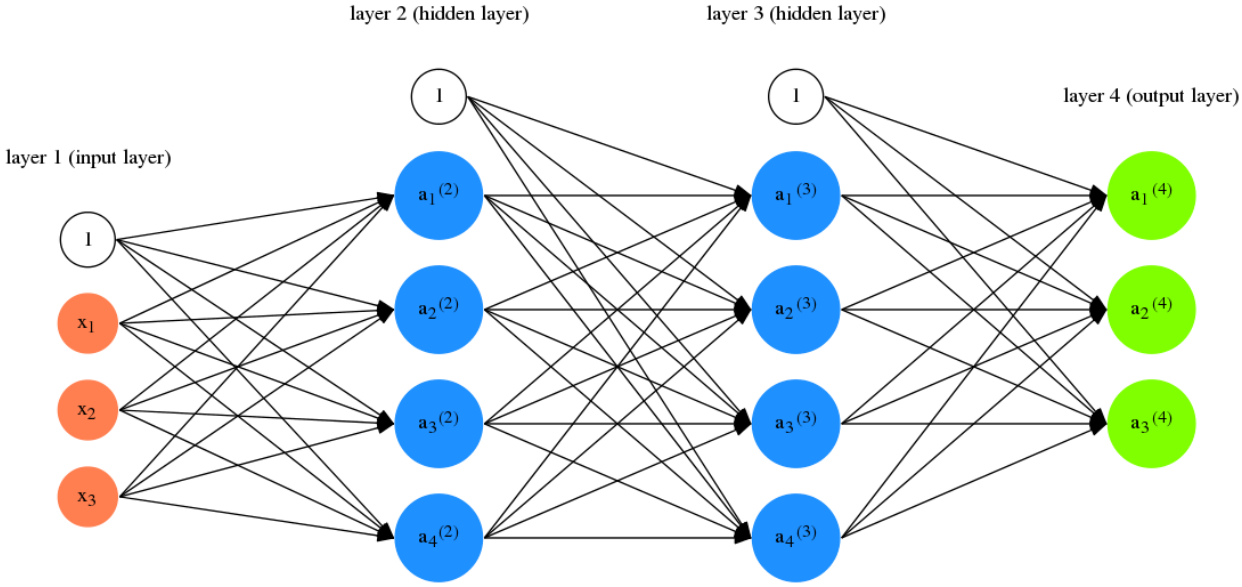

Fig. 2.3: A complete neural network with 3 input neurons, 2 hidden layers with each 4 neurons, and 3 output neurons. The bias units are depicted in white.

updated to:

$$\begin{bmatrix} z_1^{(l+1)} \\ z_2^{(l+1)} \\ z_3^{(l+1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(l+1)} & w_{12}^{(l+1)} & w_{13}^{(l+1)} \\ w_{21}^{(l+1)} & w_{22}^{(l+1)} & w_{23}^{(l+1)} \\ w_{31}^{(l+1)} & w_{32}^{(l+1)} & w_{33}^{(l+1)} \end{bmatrix} \cdot \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ a_3^{(l)} \end{bmatrix} + \begin{bmatrix} b_1^{(l+1)} \\ b_2^{(l+1)} \\ b_3^{(l+1)} \end{bmatrix}. \tag{2.4}$$

2.3.1 Activation Functions

The content of this section is taken from chapters 3 and 6 of the book *Deep Learning* (Goodfellow, 2016) [17].

Sigmoid

The sigmoid function maps real values to the range $[0, 1]$. Figure 2.4 shows the plot of the sigmoid function.

$$f(x) = \frac{1}{1 + e^{-x}}, x \in \mathbb{R} \tag{2.5}$$

Tanh

The tanh function maps real values to the range $[-1, 1]$. Figure 2.5 shows the plot of the tanh function.

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, x \in \mathbb{R} \tag{2.6}$$

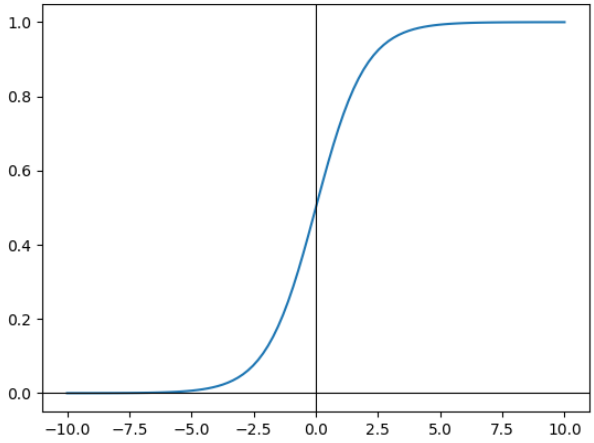

Fig. 2.4: Plot of the sigmoid function.

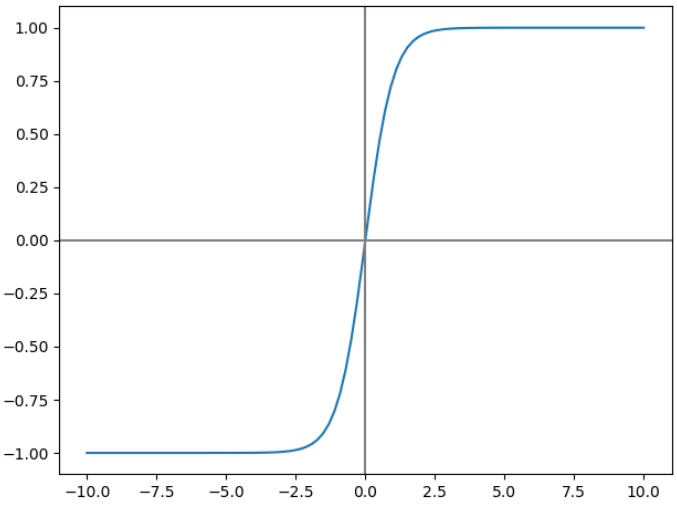

Fig. 2.5: Plot of the tanh function.

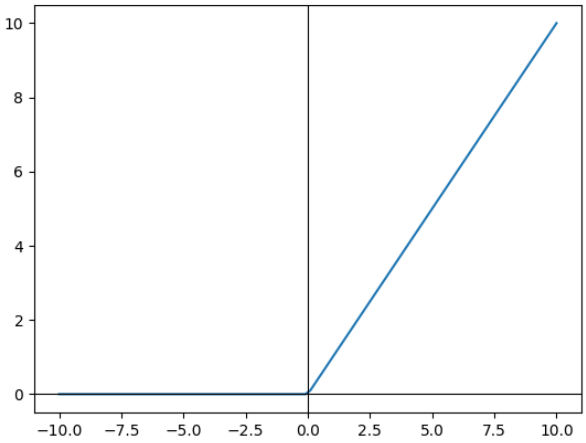

Fig. 2.6: Plot of the rectifier function.

Rectifier

The rectifier is a linear function for positive values. For negative inputs, the function outputs zero. Figure 2.6 shows the plot of the rectifier function.

$$f(x) = max\{0, x\}, x \in \mathbb{R} \tag{2.7}$$

2.3.2 Neural Network Learning

The goal is to find the weights that yield the optimal approximation $\hat{f}$ of f. This process is commonly referred to as *training*. When a neural network only consists of a single neuron, we can infer the amount of change that has to happen to the corresponding weight by looking at the error, where the error is the difference between the desired output and the output of the neuron. However, when the neural network consists of multiple neurons, this method can not be applied because the output is influenced by different neurons with different weights. In order to determine the amount by which

a weight of the neural network has to change, we need to quantify the influence this weight has on the output. First, we define a cost function C that takes the weights of a neural network as inputs and outputs a single value. The examples from a given dataset are the parameters of C.

$$C = \frac{1}{2m} \sum_{i=1}^{m} (a^{i,(L)} - y^i)^2 \tag{2.8}$$

In equation 2.8, m is the size of the dataset, $a^{i,(L)}$ denotes the output vector (of the output layer L) of the i^{th} example from the dataset, and y^i the associated output. The cost function C is just the average of the cost function C_i for every individual training example (x^i, y^i):

$$C_i = \frac{1}{2}(a^{i,(L)} - y^i)^2. \tag{2.9}$$

For simplicity, we will omit the superscript i:

$$C_i = \frac{1}{2}(a^{(L)} - y)^2. \tag{2.10}$$

Or in component form:

$$C_i = \sum_k \frac{1}{2}(a_k^{(L)} - y_k)^2. \tag{2.11}$$

Computing the minimum of C corresponds to approximating the optimal values for every weight of $\hat{f}$. In practice, this task is carried out iteratively because calculating the minimum explicitly is generally not feasible. A commonly used iterative optimization algorithm is *Gradient descent*.

Gradient descent

The gradient ∇f of a function f that maps an input vector $x = [x_1, x_2, ..., x_n]^T$ to a scalar output is the derivative of f with respect to the input vector x. The i^{th} component of ∇f is the partial derivative of f with respect to x_i, denoted as $\frac{\partial f}{\partial x_i}$. The gradient of a function indicates the direction of the steepest ascend (Strang, Calculus, chapter 13) [18]. Conversely, the negative gradient indicates the direction of the steepest descent. In order to compute a local minimum of a function, we iteratively step in the direction of the negative gradient.

$$x^{(k+1)} = x^{(k)} - \alpha \nabla f(x^{(k)}) \tag{2.12}$$

The parameter α is called the *learning rate* and specifies the step size.

Backpropagation

The formulas for this section are taken from chapter 2 of the the book *Neural networks and deep learning* (Nielsen, 2015) [20]. In order to find the minimum of the cost function C of a neural network with *Gradient descent*, we have to be able to compute the gradient. This can be achieved by using the *Backpropagation* algorithm [19]. The gradient indicates the rate of change of the cost function C as a function of every weight w and every bias unit b in the network. More concisely, the gradient consists of the partial derivative $\frac{\partial C}{\partial w}$ of the cost function C with respect to any weight w and the partial derivative $\frac{\partial C}{\partial b}$ of the cost function C with respect to any bias unit b. We compute the gradient by computing the partial derivatives $\frac{\partial C_i}{\partial w}$ and $\frac{\partial C_i}{\partial b}$ for every individual training

example (x^i, y^i) and then averaging over all of the training examples in the dataset:

$$\frac{\partial C}{\partial w} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial C_i}{\partial w}, \tag{2.13}$$

$$\frac{\partial C}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial C_i}{\partial b}. \tag{2.14}$$

In order to compute the partial derivatives $\frac{\partial C_i}{\partial w}$ and $\frac{\partial C_i}{\partial b}$ we have to compute the error $\delta_j^{(l)}$ in the j^{th} neuron in the l^{th} layer. The influence of a change $\Delta z_j^{(l)}$ to the input $z_j^{(l)}$ of the neuron on the cost function C_i (for a particular training example) can be inferred by taking the partial derivative $\frac{\partial C_i}{\partial z_j^{(l)}}$ of C_i with respect to the input $z_j^{(l)}$ and multiplying the two quantities $\frac{\partial C_i}{\partial z_j^{(l)}} \Delta z_j^{(l)}$. Consequently, the error $\delta_j^{(l)}$ in the j^{th} neuron in the l^{th} layer is defined by:

$$\delta_j^{(l)} := \frac{\partial C_i}{\partial z_j^{(l)}}. \tag{2.15}$$

The error for the output layer: For a neuron j in the output layer L this quantity equates to (see proof 1 in chapter A of the appendix):

$$\delta_j^{(L)} = \frac{\partial C_i}{\partial a_j^{(L)}} \phi'(z_j^{(L)}). \tag{2.16}$$

ϕ' denotes the derivative of the activation function ϕ and $a_j^{(L)}$ the output of the neuron. With the cost function C_i defined as in equation 2.11, $\frac{\partial C_i}{\partial a_j^{(L)}}$ equates to $(a_j^{(L)} - y_j)$ (see proof 2 in chapter A of the appendix).

Consequently, the error $\delta^{(L)}$ for the whole output layer L can be expressed as:

$$\delta^{(L)} = \nabla_{a^{(L)}} C_i \circ \phi'(z^{(L)}). \tag{2.17}$$

$\nabla_{a^{(L)}} C_i$ is a vector where the k^{th} component is the partial derivative $\frac{\partial C_i}{\partial a_k^{(L)}}$ of C_i with respect to $a_k^{(L)}$, the output of the k^{th} neuron in the output layer L (the gradient of C_i with respect to $a^{(L)}$). The operation $\circ$ denotes the *Hadamard product* (see definition 1 in chapter B of the appendix).

With the cost function C_i defined as in equation 2.11, equation 2.17 can be expressed as:

$$\delta^{(L)} = (a^{(L)} - y) \circ \phi'(z^{(L)}). \tag{2.18}$$

The error for any hidden layer: For a neuron j in any hidden layer l the error $\delta_j^{(l)}$ equates to (see proof 3 in chapter A of the appendix):

$$\delta_j^{(l)} = \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)} \phi'(z_j^{(l)}). \tag{2.19}$$

Consequently, the error for the whole hidden layer l can be expressed as:

$$\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \circ \phi'(z^{(l)}). \tag{2.20}$$

$(W^{(l+1)})^T$ is the transpose of the weight matrix $W^{(l+1)}$.

This is a recursive expression where the base case is the error of the output layer, which was defined in the previous paragraph.

The partial derivative of the cost function with respect to any bias unit: The partial derivative $\frac{\partial C_i}{\partial b_j^{(l)}}$ of the cost function with respect to the bias unit for the j^{th} neuron in the l^{th} layer is equal to that neuron's error (see proof 4 in chapter A of the appendix):

$$\frac{\partial C_i}{\partial b_j^{(l)}} = \delta_j^{(l)}. \tag{2.21}$$

The partial derivative of the cost function with respect too any weight: The partial derivative $\frac{\partial C_i}{\partial w_{jk}^{(l)}}$ of the cost function with respect to the weight between the k^{th} neuron in layer $l-1$ and the j^{th} neuron in layer l can be expressed as (see proof 5 in chapter A of the appendix):

$$\frac{\partial C_i}{\partial w_{jk}^{(l)}} = a_k^{(l-1)}\delta_j^{(l)}. \tag{2.22}$$

Algorithm 1 shows the *Backpropagation* algorithm for a single training example. This routine computes the gradient of the cost function C_i for a single example (x^i, y^i). In order to compute the gradient for the overall cost function C for the entire dataset we simply average over all of the examples. Algorithm 2 shows the complete *Backpropagation* algorithm.

Initialize $a^{(1)}$ with input x:
Feed-forward propagation:
for $l = 2$ *to* L **do**
  • compute $z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$
  • compute $a^{(l)} = \phi(z^{(l)})$
end
• compute output error: $\delta^{(L)} = (a^{(L)} - y) \circ \phi'(z^{(L)})$
Backpropagation:
for $l = L - 1$ *to* 2 **do**
  • compute $\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \circ \phi'(z^{(l)})$
end
Gradient is given by: $\frac{\partial C_i}{b_j^{(l)}} = \delta_j^{(l)}$ and $\frac{\partial C_i}{\partial w_{jk}^{(l)}} = a_k^{(l-1)}\delta_j^{(l)}$

Algorithm 1: One iteration of the Backpropagation algorithm for a single training example (x, y).

for *each* (x^i, y^i) *in* D **do**
  Initialize $a^{(1)}$ with input x^i:
  Feed-forward propagation:
  for $l = 2$ *to* L **do**
    • compute $z^{i,(l)} = W^{(l)}a^{i,(l-1)} + b^{(l)}$
    • compute $a^{i,(l)} = \phi(z^{i,(l)})$
  end
  • compute output error: $\delta^{i,(L)} = (a^{i,(L)} - y^i) \circ \phi'(z^{i,(L)})$
  Backpropagation:
  for $l = L - 1, L - 2$ *to* 2 **do**
    • compute $\delta^{i,(l)} = ((W^{(l+1)})^T \delta^{i,(l+1)}) \circ \phi'(z^{i,(l)})$
  end
end
Gradient descent:
for $l = L, L - 1$ *to* 2 **do**
  • update weights: $W^{(l)} = W^{(l)} - \frac{1}{m}\sum_{i=1}^{m} \delta^{i,(l)}(a^{i,(l-1)})^T$
  • update bias units: $b^{(l)} = b^{(l)} - \frac{1}{m}\sum_{i=1}^{m} \delta^{i,(l)}$
end

Algorithm 2: One iteration of the Backpropagation algorithm for the whole dataset D with m examples.

It is common practice not to iterate over the entire dataset D for a single *Gradient descent* step (as algorithm 2 shows) but instead D is divided into *mini-batches* of k examples. The gradient for a single step is computed using only a *mini-batch*. This variation is referred to as *Mini-batch gradient descent*.

2.4 Discriminative Models for ECG Arrhythmia Detection

Discriminative models have been used in the past for classifying ECG Arrhythmia. Polat et al. [22] employed *Support Vector Machines* (Cortes et al., 1995) [23] for detecting arrhythmia in ECG recordings and achieved a classification accuracy of 96.86% on a dataset from the University of California. Vishwa et al. [24] employed artificial neural networks for classifying arrhythmia in ECG recordings. The model was tested on the *MIT-BIH Arrhythmia Database* [33] and achieved an accuracy of 96.77%. However, both models were trained using normal ECG data as well as arrhythmic ECG data. As stated in chapter 1, our goal is to build a model for detecting cardiac arrhythmia that is trained only on normal ECG data. Thus, we will explore generative modelling.

2.5 Generative Adversarial Networks

A *Generative Adversarial Network* (Goodfellow, 2014) [25] is a framework for estimating generative models by employing an adversarial training process where a generative model G and a discriminative model D are simultaneously trained. The goal is for G to capture the distribution of the training data and for D to estimate the probability that a given sample originates from the training data and was not generated by G. In *Game theory*

terms, this process can be described as a minimax two-player game. According to the framework, both G and D have to be differentiable functions. In practice, G and D are commonly represented by neural networks and can be trained using the *Backpropagation* algorithm (see section 2.3.2).

2.5.1 Training Process

The generative model G takes in a random input vector z and outputs a sample $G(z)$ that has the same shape as the training examples in the dataset p_{data}. $D(x)$ denotes the probability that x came from the dataset. The goal of the discriminative model D is to correctly decide whether a given example came from the dataset or was generated by G. Simultaneously, G is trained to minimize the probability $\log(1 - D(G(z)))$ that a generated sample did not come from the dataset, where z is a random input vector. This process corresponds to a two-player minimax game with the value function $V(D, G)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log(D(x))] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]. \tag{2.23}$$

$\mathbb{E}_{x \sim p_{data}(x)}[\log(D(x))]$ denotes the expected value of $\log(D(x))$, averaged over the distribution p_{data} and $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$ denotes the expected value of $\log(1 - D(G(z)))$, averaged over the distribution p_z, where the random input vector z is sampled from. Early in the learning process, the probability $D(G(z))$ that a generated sample came from the training data is close to 0. As a result, the partial derivative $\frac{\partial \log(1-D(G(z)))}{\partial G(z)}$ of the cost function $\log(1 - D(G(z)))$ with respect to the generated sample $G(z)$ is also close to 0. A small gradient leads to small step sizes for *Gradient descent*, which slows down the learning process. To prevent this, Goodfellow et al. have proposed to train the generative model G by maximizing $\log(D(G(z)))$ instead of minimizing $\log(1 - D(G(z)))$. Furthermore, Goodfellow et al. proposed to alternate the

training of D and G. One entire training iteration comprises j steps of optimizing D and one step of optimizing G. Algorithm 3 shows *Mini-batch stochastic gradient descent* training for a generative adversarial network.

for *number of training iterations* **do**
  for j *steps* **do**
    • sample mini-batch of k random input vectors $(z^{(1)}, ..., z^{(k)})$ from p_z
    • sample mini-batch of k examples $(x^{(1)}, ..., x^{(k)})$ from the training data p_{data}
    • update discriminative model by ascending its stochastic gradient:
      $\nabla \frac{1}{m} \sum_{i=1}^{k} [\log(D(x^{(i)})) + \log(1 - D((G(z^{(i)}))))]$.
  end
  • sample mini-batch of k random input vectors $(z^{(1)}, ..., z^{(k)})$ from p_z.
  • update generative model by ascending its stochastic gradient:
    $\nabla \frac{1}{m} \sum_{i=1}^{k} \log(D(G(z^{(i)})))$.
end

Algorithm 3: Mini-batch gradient descent for a generative adversarial network.

Chapter 3
Materials and Methods

3.1 Mechanical Ventilation

Mechanical ventilation describes the process of assisting or replacing a person's breathing [27]. The primary goal is to provide oxygen and remove carbon dioxide. It is also applied during surgery in order to assure that the patient continues breathing, or in intensive care. We generally distinguish between invasive ventilation and non-invasive ventilation. While invasive ventilation involves the penetration of an instrument via the mouth (e.g. endotracheal tube), nose, or the skin [28], non-invasive ventilation does not (e.g. mask).

3.2 Electrocardiogram

The electrocardiogram (ECG) is a tool that is employed for measuring the electrical and muscular functions of the heart [29]. The measuring is non-invasive and is performed by placing electrodes on the skin.

3.3 Data

3.3.1 Ventilation Data

The data used in the experiments for detecting respiratory failure was extracted from the freely available *Medical Information Mart for Intensive Care III Database (MIMIC-III Database)*[13] made available by the *PhysioNet*[14] project. The *MIMIC-III Database* comprises over forty thousand records of critical care patients of the *Beth Israel Deaconess Medical Center* that were collected over a period from 2001 to 2012. The database includes demographical information, vital sign measurements, laboratory test results, procedures, medications, caregiver notes, imaging reports, and mortality. The data was recorded during routine clinical care and was not particularly intended for the purpose of analysis. It is divided into two types, static data and dynamic data. Static data is recorded once and does not change over time (e.g. date of birth). Dynamic data is recorded routinely and is always associated with a corresponding timestamp (e.g. blood pressure). One thing to note is that all dates in the database have been shifted to the future in order to protect patient confidentiality. However, this does not impair the results because the dates are still consistent.

In order to gain access to the *MIMIC-III Database* you have to complete the course *Data or Specimens Only Research* by *CITI* [1]. The database is provided in the form of 26 comma-separated-value (CSV) files which were imported into a *PostgreSQL* database.

Meta Information

The database comprises 26 tables which are listed in Table 3.1. It contains 58976 admissions for 46520 patients and 61532 stays in intensive care. The average time a

[1] https://about.citiprogram.org/en/homepage/

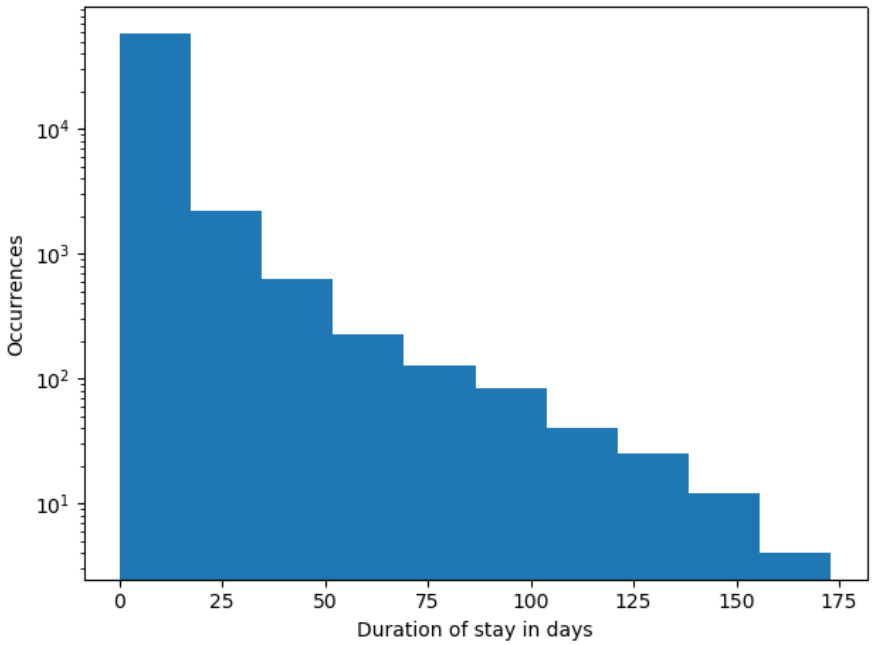

Fig. 3.1: Histogram of the different durations of a stay in the ICU. Note that we used a log scale.

patient stayed in the intensive care unit (ICU) is 4 days. The longest recorded stay is 173 days and the shortest recorded stay is 12 seconds. The latter is likely to be an error. Figure 3.1 shows a histogram of the different durations of a stay in the ICU.

Measurements

During the process of mechanical ventilation, various measurements are being recorded.

PEEP The Positive End-Expiratory Pressure (PEEP) is the positive pressure that remains in the airways at the end of expiration [30]. The *MIMIC-III Database* contains PEEP measurements for 12630 different patients. Of the 12630 patients, 6861 were suffering from respiratory failure and 5769 were not. On average, there are 6.88 recorded values per patient.

Table name	Purpose
admissions	Define a patient's hospital admission
callout	Provides information when a patient was ready for discharge
caregivers	Defines the role of caregivers
chartevents	Contains all charted data for all patients
cptevents	Contains CPT codes, which facilitate billing for procedures
d_cpt	High-level definitions for CPT codes
d_icd_diagnoses	Definition table for ICD diagnoses
d_icd_procedures	Definition table for ICD procedures
d_items	Definition table for all items in the ICU databases
d_labitems	Definition table for all laboratory measurements
datetimeevents	Contains all date formatted data
diagnoses_icd	Contains ICD diagnoses for patients
drgcodes	Contains diagnosis related groups codes for patients
icustays	Defines each *icustay_id* in the database
inputevents_cv	Input data for patients
inputevents_mv	Input data for patients
labevents	Contains all laboratory measurements for a given patient
microbiologyevents	Contains microbiology information, including tests
noteevents	Contains all notes for patients
outputevents	Output data for patients
patients	Contains all charted data for all patients
prescriptions	Contains medication related order entries
procedureevents_mv	Contains procedures for patients
procedures_icd	Contains ICD procedures for patients
services	Lists services that a patient was admitted/transferred under
transfers	Physical locations for patients throughout their hospital stay

Table 3.1: Tables of the *MIMIC-III Database*.

PO2 The Partial Pressure of Oxygen (PO2) indicates the amount of oxygen gas dissolved in the blood [31]. The *MIMIC-III Database* contains CO2 measurements for 31424 different patients, 9403 were suffering from respiratory failure and 22021 were not. On average, there are 15.61 recorded values per patient.

pCO2 The Partial Pressure of Carbon Dioxide (pCO2) indicates the amount of carbon dioxide gas dissolved in the blood [31]. The *MIMIC-III Database* contains pCO2 measurements for 31424 different patients, 9403 were suffering from respiratory failure and 22021 were not. On average, there are 41.72 recorded values per patient.

The intervals between the measurements are not consistent and the amount of measurements recorded differs widely for different patients. Figure 3.2 shows scatter plots where the x-axis represents the time interval between the first and last measurement for a patient and the y-axis represents the total amount of measurements for the patient. The plots show how in general the less time there is between the first and the last recording, the more measurements were recorded, independently of respiratory failure.

Figure 3.3 shows scatter plots where the x-axis represents the length of stay in the ICU and the y-axis represents the total amount of measurements for the patient. The plots show that there are in general more measurements for patients who stayed between 25.000 and 125.000 minutes in the ICU. Patients who stayed longer tend to have less measurements recorded. The plots also show that there are more measurements of patients who were suffering from respiratory failure.

3.3.2 ECG Data

The data used for the ECG experiments was also provided by the *PhysioNet*[14] project. For the training of the *Generative Adversarial Network* we used the *Long-Term ST*

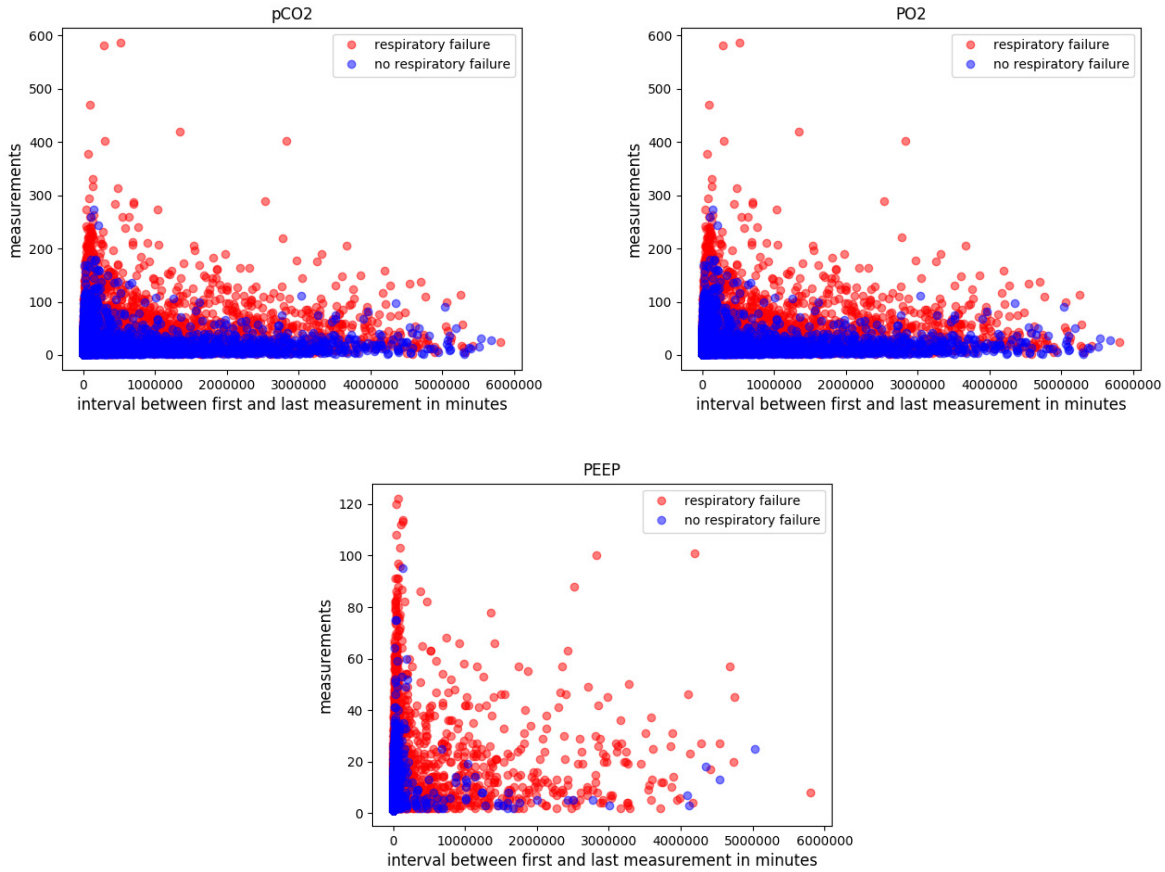

Fig. 3.2: Scatter plot where the x-axis represents the time interval between the first and last measurement and the y-axis represents the total amount of measurements for the patient. One point represents one patient.

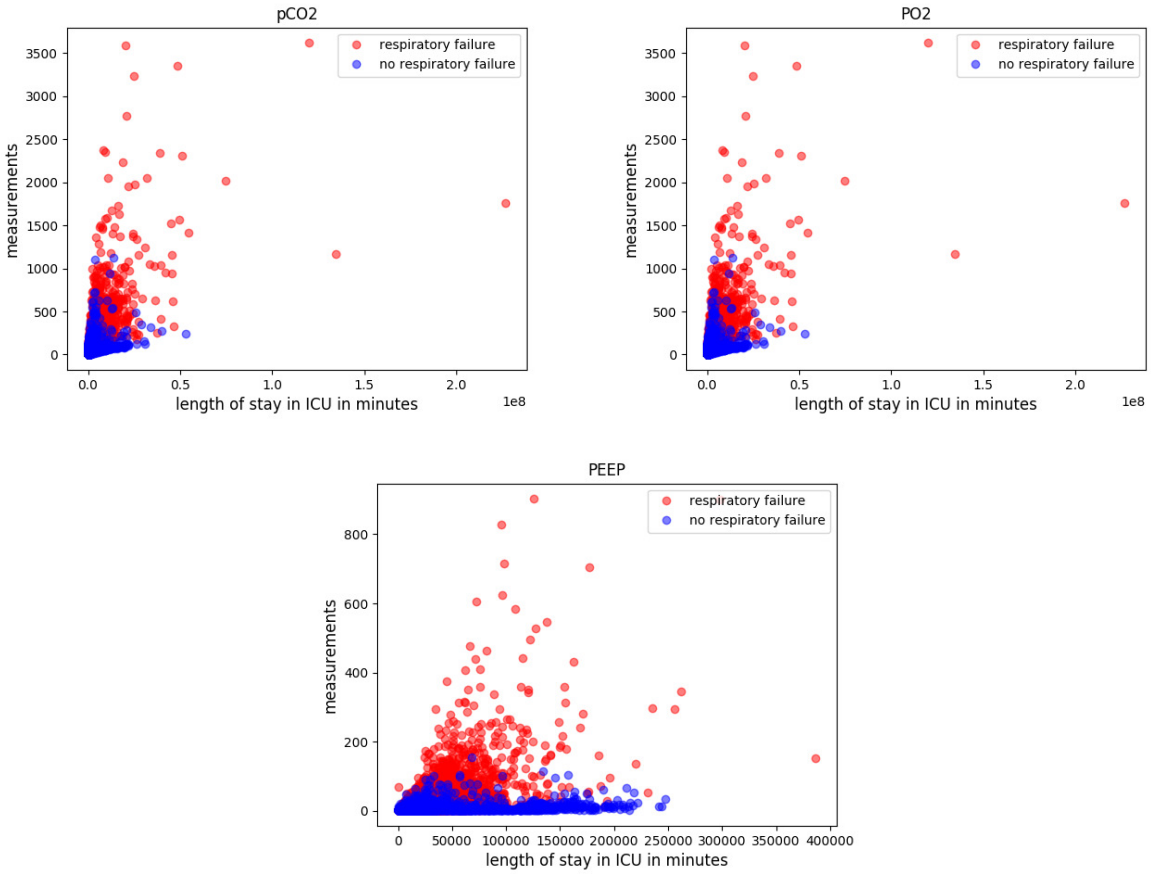

Fig. 3.3: Scatter plot where the x-axis represents the length of stay in ICU and the y-axis represents the total amount of measurements for the patient. One point represents one patient.

Database [32]. It contains 86 ECG recordings obtained from 80 patients. The recordings originate from different hospitals over a period from 1995 to 2002. For testing we used the *MIT-BIH Arrhythmia Database* [33]. It contains 48 ECG recordings obtained from 47 patients, which were studied by the BIH Arrhythmia Laboratory between 1975 and 1979.

3.4 Data Preprocessing

3.4.1 Ventilation Data

The data was queried from the database with SQL and the preprocessing of the data was implemented in Java 8[2].

The measurements are stored in the *labevents* table (table 3.1). The *subject_id* column identifies a patient, the *valuenum* column holds the measurement value, the *charttime* column holds the corresponding timestamp and the *itemid* column identifies the measurement (PEEP, pCO2, ...). The following SQL query returned the subject_id and all of the pCO2 measurements for every patient in one row, separated by commas:

```
select subject_id, string_agg(valuenum::text, ',' order by charttime)
from mimiciii.labevents
where itemid=50818
group by subject_id;
```

The results of the query where then exported to a text file. The first value in a row is the subject_id, followed by all of the measurements (subject_id's and values were changed for privacy reasons):

[2]https://java.com/en/

```
5,39,40,45,41,43,34,41,28,28,33,30,29,26,33,27,28,36,30,37
34,28,28,27
55,46,35,34,41,32
91,45,37,41,38,39,36,34,34,31,23,28,38,38,67
...
```

The following SQL query returned the subject_id and all of the corresponding timestamps for the pCO2 measurements for every patient in one row, separated by commas:

```
select subject_id,string_agg(charttime::text, ',' order by charttime)
from mimiciii.labevents
where itemid=50818
group by subject_id;
```

The results of the query where then exported to a text file. The first value in a row is the subject_id, followed by all of the timestamps:

```
5,2101-10-12 09:18:00,2101-10-12 10:51:00,2101-10-12 12:05:00,...
34,2175-05-30 07:21:00,2175-05-30 10:15:00,2175-05-30 11:52:00
55,2149-11-09 17:47:00,2149-11-09 19:42:00,2149-11-10 03:04:00,...
91,2149-11-09 17:47:00,2149-11-09 19:42:00,2149-11-10 03:04:00,...
...
```

In order to be able to feed it into a neural network, the data has to be unified. For every patient there has to be the same amount of values (for a specific measurement). The values should also originate from a specified interval, thus for every patient we took the measurements from a k hour span. Because different patients had different amount of measurements taken in a specific interval, we had to augment the measurements for some patients so that the k hour interval was filled with measurements with a gap of l minutes. Firstly, for every patient we searched for the first timestamp of a k hour interval that contained the most measurements. Starting with the first timestamp, we continually added l minutes to the current timestamp. If there existed a new measurement at the

new time, we took the new value. If not we took the value from the previous timestamp. In order to do this we had to assume that a new value was only recorded in case the value changes.

For the training process, we only used data of patients who were not suffering from respiratory failure. The distinction can be made by means of the ICD-9 codes in the *diagnoses_icd* table (see section 3.3.1).

3.4.2 ECG Data

The databases can be downloaded from *PhysioNet* by means of a Python library[3]. We then wrote our own tool for further processing.

A recording consists of several channels. One channel is represented by a sequence of measurement values, which is divided into intervals. One interval consists of the measurements between two heartbeats. One interval was used as one training example. With the aid of the annotations channel, we classified the intervals in normal and abnormal (arrhythmic) intervals. Because the intervals were consisting of different amount of values, we linearly interpolated them to have 100 values each. Figure 3.4 shows an example comparison between an original interval and its interpolated version. We also added the length of the original interval as a feature. For the training process, we only used data of normal ECG recordings.

[3]https://github.com/MIT-LCP/wfdb-python

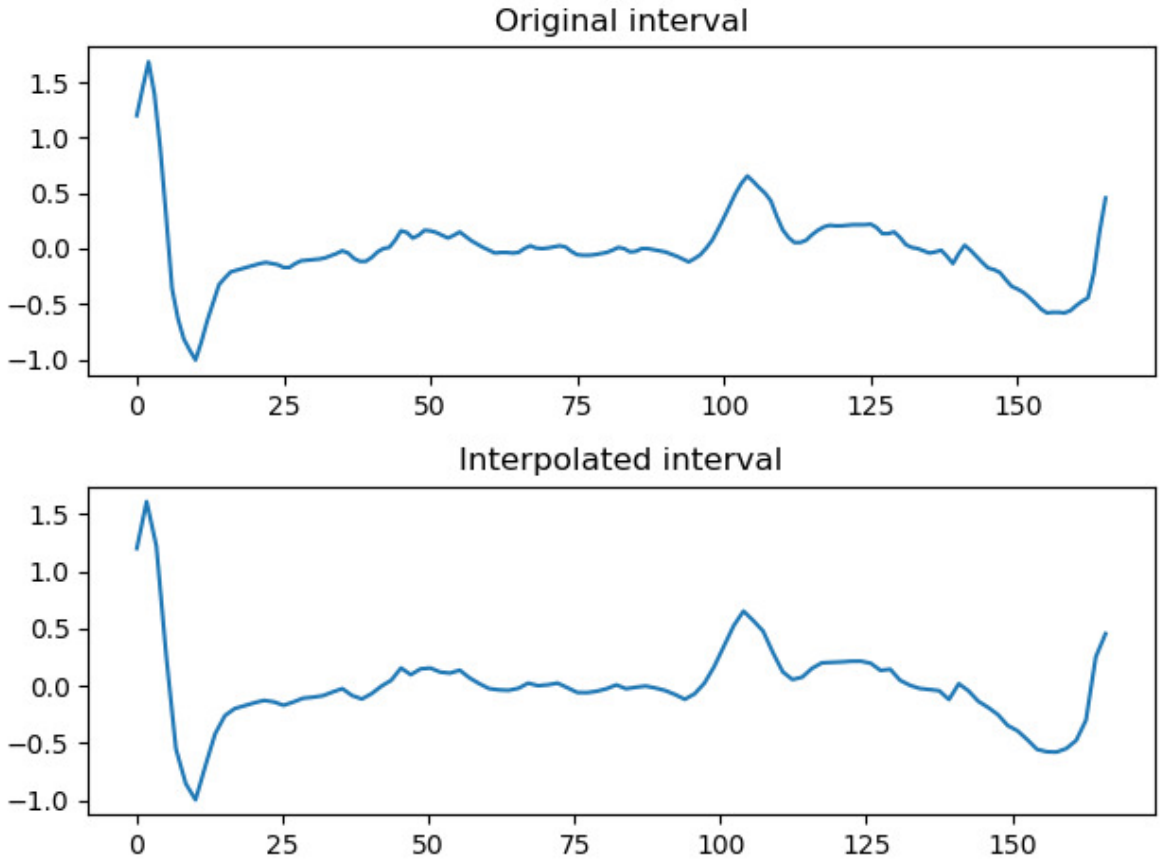

Fig. 3.4: Example of the plots of an original interval with 166 values and the interpolated interval with 100 values.

3.5 Experiments

Python 3.6.3[4] (Anaconda[5]) and TensorFlow 1.8.0 [34] were used for the implementation. All experiments were carried out on the *RWTH Compute Cluster*[6] running under the CentOS 7.4 [7] operating system. We implemented the generative adversarial networks as described in section 2.5.

3.5.1 Ventilation Data

During the preprocessing (section 3.4) we empirically used an interval of $k = 24$ hours with a gap of $l = 30$ minutes between the measurements, this gave us 49 values per patient. We tried other configurations (e.g. 145 values per patient), however, because the results were comparable, we omitted them. We used pCO2, PO2, and PEEP measurements (section 3.3.1).

For the discriminative model D we used a neural network (section 2.3) with three layers, an input layer with 49 neurons, one hidden layer with 24 neurons, and a output layer with one neuron. We applied the sigmoid function (section 2.3.1) to the output neuron because we want it to output a probability and the rectifier function (section 2.3.1) to the hidden layer. For the generative model G we used a neural network (section 2.3) with three layers, an input layer with 20 neurons, one hidden layer with 40 neurons, and one output layer with 49 neurons. We applied the rectifier function to the hidden layer and the output layer because the measurements take on only positive values. For the training process we used *Mini-batch gradient descent* (section 2.3.2) where the size of the mini-batches was 25, in each iteration we performed one step of optimizing D and one step of optimizing G. In order to stabilize the training process we added noise

[4]https://www.python.org/
[5]https://anaconda.org/
[6]https://doc.itc.rwth-aachen.de/display/CC/Home
[7]https://www.centos.org/

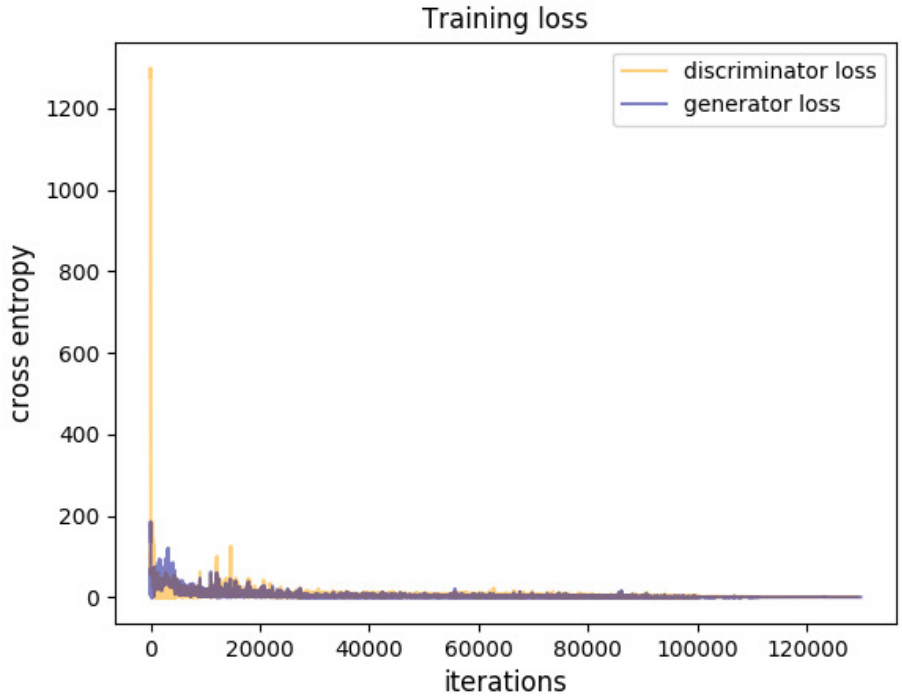

Fig. 3.5: Plots for the training loss for pCO2 data of the discriminator D and the generator G as a function of the number of iterations.

	pCO2 Data	PO2 Data	PEEP Data
Training set	15525 examples	15530 examples	2638 examples
Test sets	2000 examples	2000 examples	500 examples

Table 3.2: Sizes of the training and test sets for ventilation data.

ϵ to the input of D (proposed by Arjovsky and Bottou [35]), with $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Figure 3.5 shows the training loss exemplary for pCO2 data. Table 3.2 shows the sizes of the training and test sets.

3.5.2 ECG Data

For the discriminative model D we used a neural network (section 2.3) with five layers, an input layer with 101 neurons, three hidden layers with 50 neurons, and an output layer with one neuron. We applied the sigmoid function (section 2.3.1) to the output

neuron because we want it to output a probability and the tanh function (section 2.3.1) to the hidden layers. For the generative model G we used a neural network (section 2.3) with five layers, an input layer with 60 neurons, three hidden layers with 80 neurons, and an output layer with 101 neurons. We applied the tanh function to the hidden layers and no activation function to the output layer because the measurements can take on positive and negative values. For the training process we used *Mini-batch gradient descent* (section 2.3.2) where the size of the mini-batches was 512, in each iteration we did one step of optimizing D and one step of optimizing G. As described in section 3.5.1, we stabilized the training process by adding noise to the input of D. In order to prevent the neural networks from overfitting, we employed *Dropout* regularization (Srivastava et al., 2014) [36]. When we apply dropout regularization to a layer, every neuron output in that layer is set to zero with a specified probability p. We set $p = 0.5$ for every hidden layer of D and every hidden layer of G.

The training set consisted 1.594.410 examples, the test set containing normal ECG recordings consisted of 66.641 examples, and the test set containing abnormal/arrhythmic ECG recordings consisted of 41.457 examples.

Chapter 4

Results

As mentioned in chapter 1, we only used the measurements of healthy patients for training the models. In order to examine whether or not *Generative Adversarial Networks* can be used to diagnose respiratory diseases or cardiac arrhythmia, we used two test datasets for each model. One dataset containing data of healthy patients and one dataset containing data of patients with respiratory failure and cardiac arrhythmia, respectively. We separately fed both of the test datasets into the discriminator D and compared the resulting losses.

In order to test the difference of the means of the two test sets, we applied the t-test. We formulated the null hypothesis $H_0 : \mu_1 = \mu_2$ and the alternative hypothesis $H_a : \mu_1 > \mu_2$, where μ_1 is the mean of the test set of patients who were suffering from respiratory failure or cardiac arrhythmia and μ_2 is the mean of the test set containing data of healthy patients.

We plotted box plots to visualize the value range of the respective losses.

In order to examine the diagnostic ability of the models, we constructed *Receiver Operating Characteristic* (ROC) curves. A ROC curve represents the relationship between sensitivity and specificity [37]. The area under the ROC curve (AUC) gives measure to the accuracy of a test, where 1 is the maximum. For medical diagnosis, an AUC of 0.7 to 0.8 is considered acceptable, 0.8 to 0.9 is considered excellent, and greater than 0.9 is considered outstanding [38].

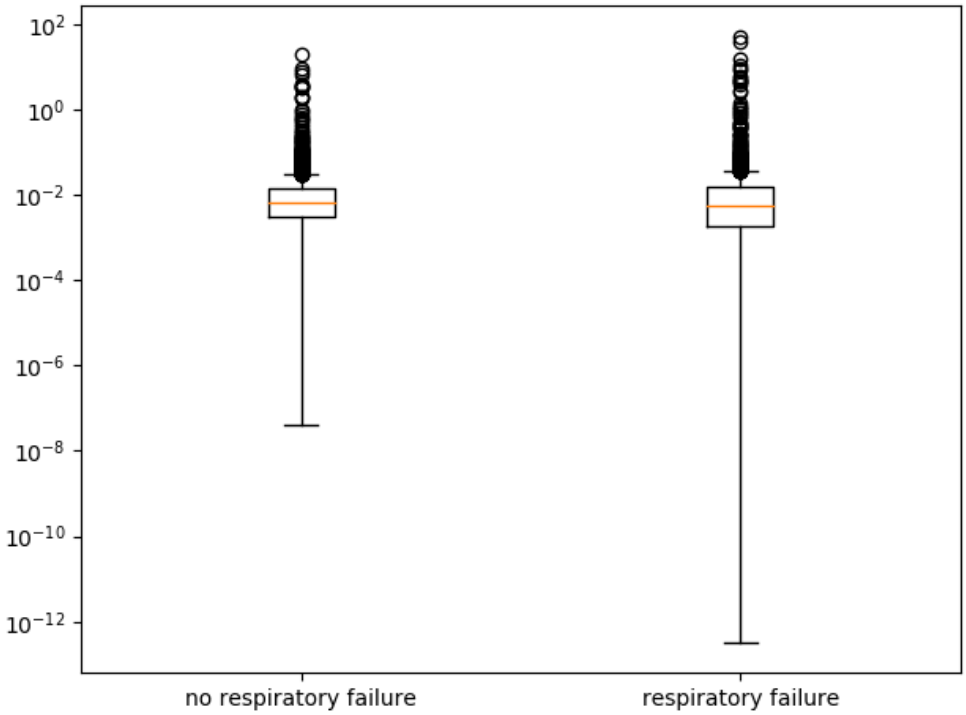

Fig. 4.1: Box plot of the test cross entropy loss for pCO2 data that was fed into the discriminator D. Note that we used a log scale.

4.1 Ventilation Data

4.1.1 pCO2 Data

The test data of the patients who were not suffering from respiratory failure produced an average cross entropy loss (see definition 3 in chapter B of the appendix) of 0.0485 and the test data of the patients who were suffering from respiratory failure an average cross entropy loss of 0.1008. Figure 4.1 shows a box plot for the test dataset. The plot shows that the losses have nearly the same range. We were able to reject the hypothesis $H_0 : \mu_1 = \mu_2$ in favor of $H_a : \mu_1 > \mu_2$ with a p-value of 0.1 (section C.1.1 in chapter C of the appendix). Figure 4.4 shows the ROC curve as well as the AUC.

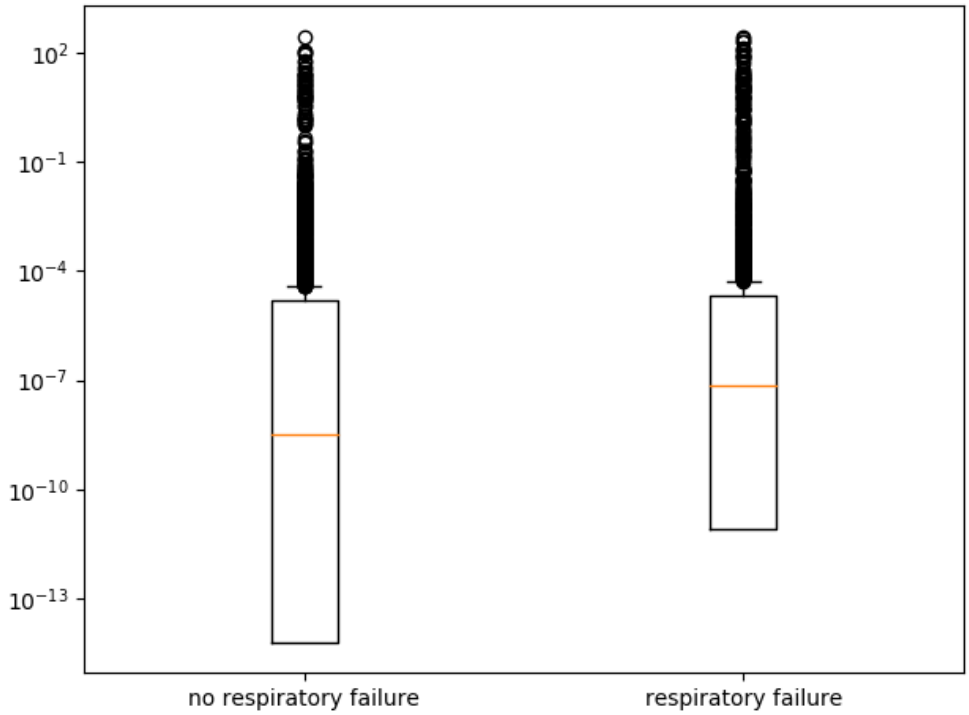

Fig. 4.2: Box plots of the test cross entropy loss for PO2 data that was fed into the discriminator D. Note that we used a log scale.

4.1.2 PO2 Data

The test data of the patients who were not suffering from respiratory failure produced an average cross entropy loss of 0.6691 and the test data of the patients who were suffering from respiratory failure an average cross entropy loss of 1.2531. Figure 4.2 shows a box plot for the test dataset. The plot shows that although the means are varying, the losses have nearly the same range. We were able to reject the hypothesis $H_0 : \mu_1 = \mu_2$ in favor of $H_a : \mu_1 > \mu_2$ with a p-value of 0.1 (section C.1.2 in chapter C of the appendix). Figure 4.4 shows the ROC curve as well as the AUC.

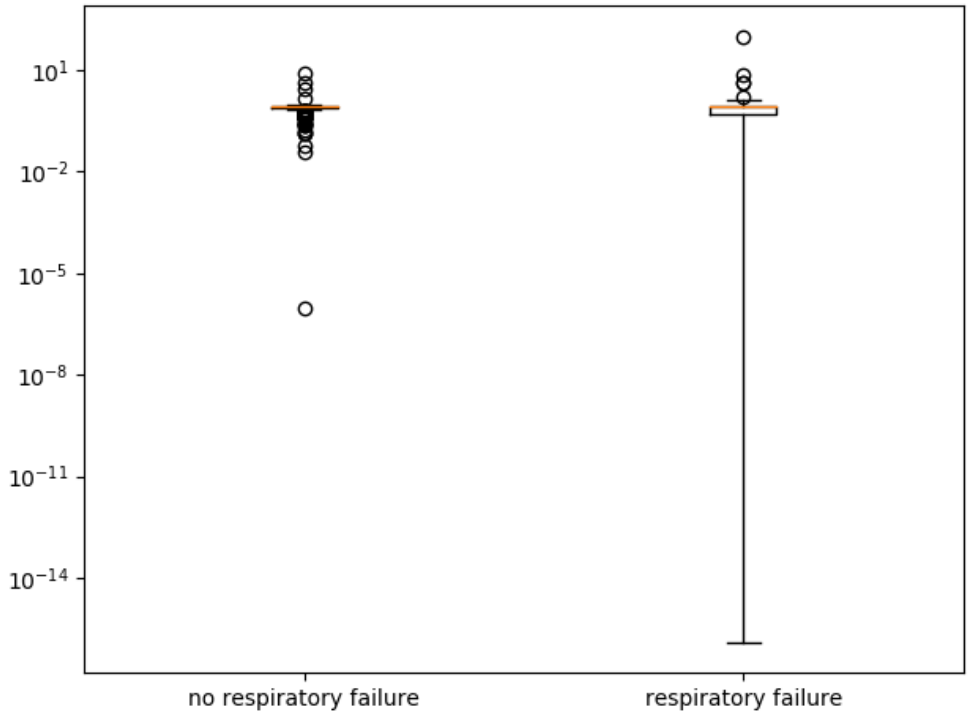

Fig. 4.3: Box plots of the test cross entropy loss for PEEP data that was fed into the discriminator D. Note that we used a log scale.

4.1.3 PEEP Data

The test data of the patients who were not suffering from respiratory failure produced an average cross entropy loss of 0.7925 and the test data of the patients who were suffering from respiratory failure an average cross entropy loss of 0.8984. Figure 4.3 shows a box plot for the test dataset. The plot shows that the losses have approximately the same range. We accepted the hypothesis $H_0 : \mu_1 = \mu_2$ with a p-value of 0.1 (section C.1.3 in chapter C of the appendix). Figure 4.4 shows the ROC curve as well as the AUC.

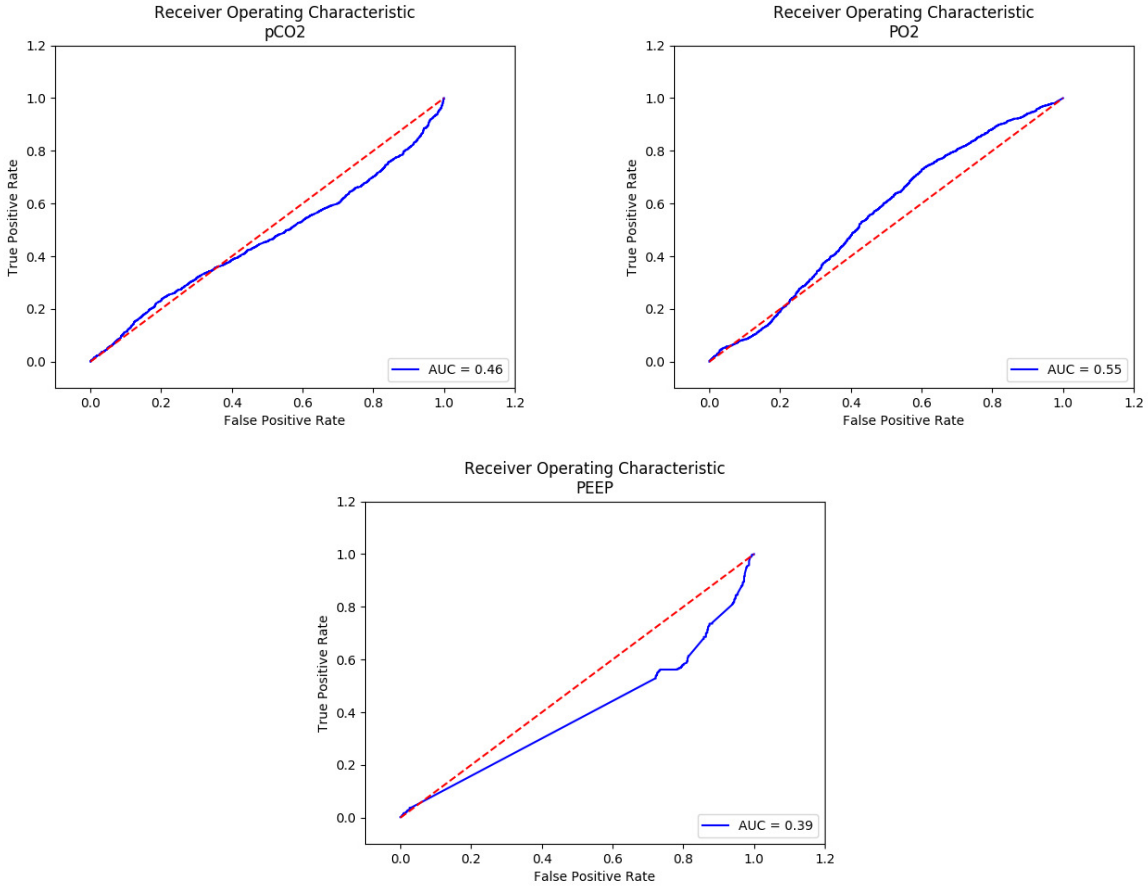

Fig. 4.4: Receiver Operating Characteristic (ROC) curve for ventilation data.

4.2 ECG Data

The test data of normal ECG recordings produced an average cross entropy loss of 0.0021 and the test data of abnormal ECG recordings an average cross entropy loss of 0.0807. Figure 4.5 shows a box plot for the test dataset. We were able to reject the hypothesis $H_0 : \mu_1 = \mu_2$ in favor of $H_a : \mu_1 > \mu_2$ with a p-value of 0.05 (section C.2 in chapter C of the appendix). Figure 4.6 shows the ROC curve as well as the AUC.

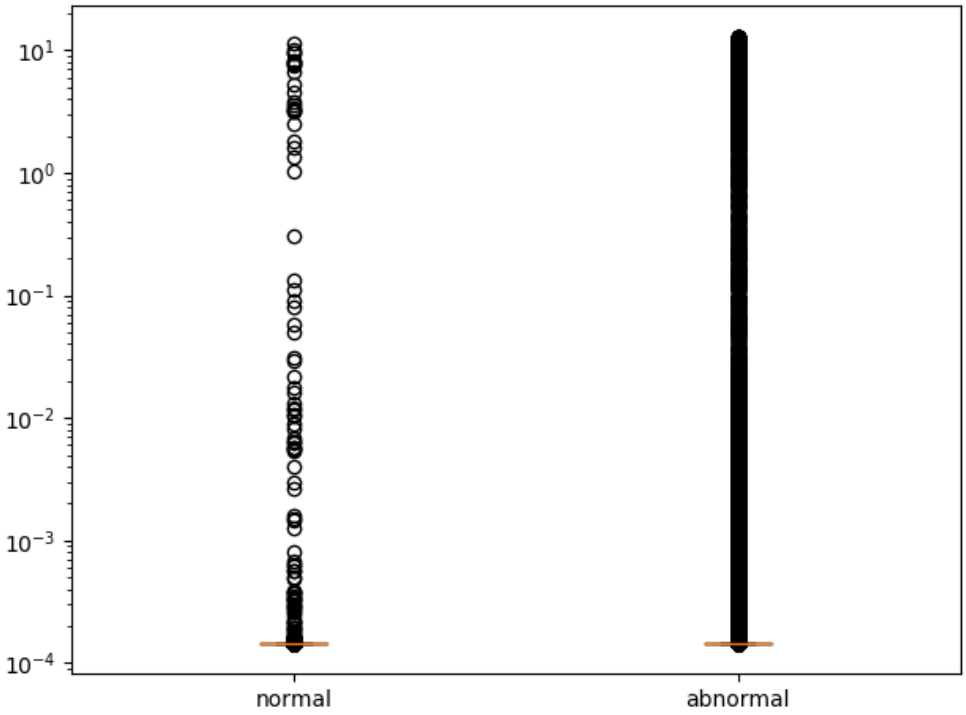

Fig. 4.5: Box plots of the test cross entropy loss for ECG data that was fed into the discriminator D. Note that we used a log scale.

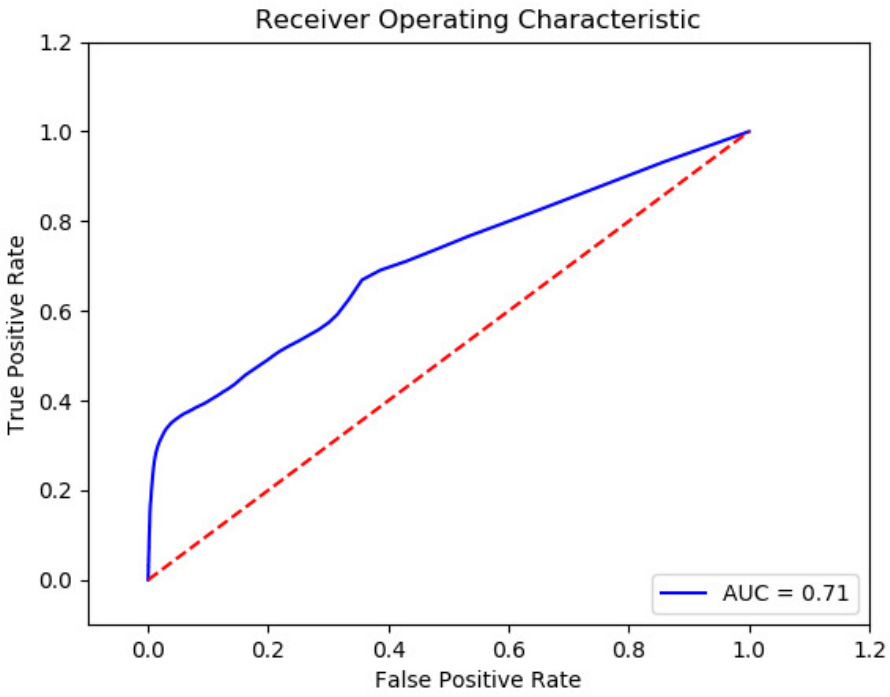

Fig. 4.6: Receiver Operating Characteristic (ROC) curve for ECG data.

Chapter 5
Discussion

5.1 Ventilation Data

Looking at the results in section 4.1, we can observe that the average cross entropy loss was larger for the test data of the patients who were suffering from respiratory failure. This was also verified by means of the t-test with a p-value of 0.1 (section C.1 of the appendix). Only for PEEP data, we could not reject H_0, which could be explained by the significantly less amount of data that was available for PEEP measurements (section 3.5.1). However, as shown by the box plots in section 3.5.1, the losses for the test data of the patients who were suffering from respiratory failure and the test data of the patients who were not suffering from respiratory failure have approximately the same range. This makes it difficult to infer a threshold for the cross entropy loss, where surpassing that threshold could indicate respiratory failure. Furthermore, the ROC curves depicted in figure 4.4 indicate that this method is not suited to serve as a means of medical diagnosis, because the AUC is below 0.7 for all three cases. This means that either *Generative Adversarial Networks* are not capable of distinguishing between measurement data of the two patient groups or the used data does not contain the information that is needed to perform this distinction.

The generalization of the results may be restricted to some extend by the quality as well as the quantity of the data. Due to the relatively small amount of measurement values per patient and the variety in the number of values per patients, we had to make

the assumption that a new value was only recorded in case the values changes (section 3.4). Another limiting factor may also be that the data originates from a single hospital (section 3.3).

5.2 ECG Data

The results in section 4.2 show that the average cross entropy loss was larger for the test dataset containing abnormal ECG recordings. This was also verified by means of the t-test with a p-value of 0.05 (section C.2 of the appendix). The ROC curve depicted in figure 4.6 shows an AUC of 0.71, which indicates that *Generative Adversarial Networks* are capable of distinguishing between normal ECG recordings and abnormal ECG recordings and also may be suited to be used as a means of medical diagnosis.

5.3 Future Work

Due to time constraints, some experiments have been left for the future. Since the introduction of *Generative Adversarial Networks* in 2014, different variations and extensions have been proposed. A prominent example, *Wasserstein GANs* (Arjovsky et al., 2017) [39], addresses the instability of the training procedure of *Generative Adversarial Networks*. During the training procedure of our experiments, one of the two models kept getting ahead of the other model (in training loss). When the gap became to big, usually the other model was not able to recover. *Wasserstein GANs* may reduce this problem. *Recurrent Neural Networks* have proven to be well suited for classification of sequential data [40]. Recently, they have been applied to the classification of ECG recordings [41, 42]. One could implement a generative adversarial network, where the discriminative model D is a recurrent neural network. D would still be trained solely on normal

ECG recordings.

Chapter 6
Conclusion

The objective of this thesis was to evaluate whether *Generative Adversarial Networks* are capable of distinguishing between the data of patients suffering from respiratory failure or cardiac arrhythmia and the data of healthy patients. We were able to show a significant difference between the overall means of the test set containing data of patients with respiratory failure and the test set containing data of patients without respiratory failure. However, the model fell short when it came to the medical diagnosis of individual cases. We attributed this to either the inability of *Generative Adversarial Networks* or to the quality of the used ventilation data.

For our other model, we were also able to show a significant difference between the overall means of the test set containing normal ECG recordings and the test set containing abnormal ECG recordings. Furthermore, the model proved to be suited to be used for medical diagnosis of ECG arrhythmia. This leads us to the overall conclusion that *Generative Adversarial Networks* may be suited for medical diagnosis. This is, however, highly dependent upon the amount and the quality of the available data.

Appendices

Appendix A

Proofs

Proof 1 (Proof of equation 2.16) *This proof is taken from the book Neural Networks and Deep Learning by Michael Nielsen [20].*
Equation 2.15 defines the error $\delta_j^{(l)}$ *in the* j^{th} *neuron in the* l^{th} *layer as* $\delta_j^{(l)} := \frac{\partial C_i}{\partial z_j^{(l)}}$. *For the* j^{th} *neuron in the output layer* L *this equates to:*

$$\delta_j^{(L)} = \frac{\partial C_i}{\partial z_j^{(L)}} \tag{A.1}$$

By applying the Chain rule this can be transformed to:

$$= \sum_k \frac{\partial C_i}{\partial a_k^{(L)}} \frac{\partial a_k^{(L)}}{\partial z_j^{(L)}} \tag{A.2}$$

Where we sum over all of the neurons in the output layer. The output $a_k^{(L)}$ *of the* k^{th} *neuron only depends on the input* $z_j^{(L)}$ *for the* j^{th} *neuron when* $k = j$. *Thus the term* $\frac{\partial a_k^{(L)}}{\partial z_j^{(L)}}$ *equates to* 0 *for every* k *where* $k \neq j$.

$$= \frac{\partial C_i}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \tag{A.3}$$

The latter term is equal to the derivative of the activation function ϕ*:*

$$= \frac{\partial C_i}{\partial a_j^{(L)}} \phi'(z_j^{(L)}) \tag{A.4}$$

□

Proof 2 *Equation 2.11 defines the cost function C_i for a training example $(x^{(i)}, y^{(i)})$ as $C_i = \sum_k \frac{1}{2}(a_k^{(L)} - y_k)^2$. The partial derivative of C_i with respect to $a_j^{(L)}$ can be written as:*

$$\frac{\partial C_i}{\partial a_j^{(L)}} = \frac{\partial(\sum_k \frac{1}{2}(a_k^{(L)} - y_k)^2)}{\partial a_j^{(L)}} \tag{A.5}$$

Every term in the sum where $k \neq j$ equates to 0:

$$= \frac{\partial(\frac{1}{2}(a_j^{(L)} - y_j)^2)}{\partial a_j^{(L)}} \tag{A.6}$$

By taking the derivative we get:

$$= (a_j^{(L)} - y_j) \tag{A.7}$$

□

Proof 3 (Proof of equation 2.19) *This proof is taken from the book Neural Networks and Deep Learning by Michael Nielsen [20].*
Equation 2.15 defines the error $\delta_j^{(l)}$ in the j^{th} neuron in the l^{th} layer as:.

$$\delta_j^{(l)} := \frac{\partial C_i}{\partial z_j^{(l)}} \tag{A.8}$$

By applying the Chain rule this can be transformed to:

$$= \sum_k \frac{\partial C_i}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \tag{A.9}$$

Where we sum over all of the neurons in the next layer $l+1$, which are all influenced by $z_j^{(l)}$. $z_k^{(l+1)}$ can be rewritten as the sum over all of the input neurons in the previous

layer l (see equation 2.4).

$$= \sum_k \frac{\partial C_i}{\partial z_k^{(l+1)}} \frac{\partial (b_k^{(l+1)} + \sum_i w_{ki}^{(l+1)} a_i^{(l)})}{\partial z_j^{(l)}} \tag{A.10}$$

$a_i^{(l)}$ is just the function value of the activation function ϕ for the input $z_i^{(l)}$:

$$= \sum_k \frac{\partial C_i}{\partial z_k^{(l+1)}} \frac{\partial (b_k^{(l+1)} + \sum_i w_{ki}^{(l+1)} \phi(z_i^{(l)}))}{\partial z_j^{(l)}} \tag{A.11}$$

The terms of the sum in the partial derivative $\frac{\partial (b_k^{(l+1)} + \sum_i w_{ki}^{(l+1)} \phi(z_i^{(l)}))}{\partial z_j^{(l)}}$ are equal to 0 whenever $i \neq j$.

$$= \sum_k \frac{\partial C_i}{\partial z_k^{(l+1)}} \frac{\partial (w_{kj}^{(l+1)} \phi(z_j^{(l)}) + b_k^{(l+1)})}{\partial z_j^{(l)}} \tag{A.12}$$

We take the derivative:

$$= \sum_k \frac{\partial C_i}{\partial z_k^{(l+1)}} w_{kj}^{(l+1)} \phi'(z_j^{(l)}) \tag{A.13}$$

$\frac{\partial C_i}{\partial z_k^{(l+1)}}$ is by definition equal to $\delta_k^{(l+1)}$ (see equation 2.15).

$$= \sum_k \delta_k^{(l+1)} w_{kj}^{(l+1)} \phi'(z_j^{(l)}) = \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)} \phi'(z_j^{(l)}) \tag{A.14}$$

□

Proof 4 (Proof of equation 2.21) *The bias unit for the j^{th} neuron in the l^{th} layer only directly influences the input $z_j^{(l)}$ of that particular neuron. By applying the Chain rule we get:*

$$\frac{\partial C_i}{\partial b_j^{(l)}} = \frac{\partial C_i}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} \tag{A.15}$$

$z_k^{(l)}$ can be rewritten as the sum over all of the input neurons in the previous layer $l-1$

(see equation 2.4):

$$= \frac{\partial C_i}{\partial z_j^{(l)}} \frac{\partial (b_j^{(l)} + \sum_k w_{jk}^{(l)} a_k^{(l-1)})}{\partial b_j^{(l)}} \tag{A.16}$$

$\frac{\partial (b_j^{(l)} + \sum_k w_{jk}^{(l)} a_k^{(l-1)})}{\partial b_j^{(l)}}$ *is equal to 1:*

$$= \frac{\partial C_i}{\partial z_j^{(l)}} \tag{A.17}$$

$\frac{\partial C_i}{\partial z_j^{(l)}}$ *is by definition* $\delta_j^{(l)}$ *(see equation 2.15):*

$$= \delta_j^{(l)} \tag{A.18}$$

□

Proof 5 (Proof of equation 2.22) *The weight* $w_{jk}^{(l)}$ *only directly influences the input* $z_j^{(l)}$ *of the* j^{th} *neuron in the* l^{th} *layer. By applying the Chain rule we get:*

$$\frac{\partial C_i}{\partial w_{jk}^{(l)}} = \frac{\partial C_i}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} \tag{A.19}$$

$z_k^{(l)}$ *can be rewritten as the sum over all of the input neurons in the previous layer* $l-1$ *(see equation 2.4):*

$$= \frac{\partial C_i}{\partial z_j^{(l)}} \frac{\partial (b_j^{(l)} + \sum_i w_{ji}^{(l)} a_i^{(l-1)})}{\partial w_{jk}^{(l)}} \tag{A.20}$$

When taking the derivative $\frac{\partial (b_j^{(l)} + \sum_i w_{ji}^{(l)} a_i^{(l-1)})}{\partial w_{jk}^{(l)}}$*, every term in the sum where* $i \neq k$ *vanishes:*

$$= \frac{\partial C_i}{\partial z_j^{(l)}} a_k^{(l-1)} \tag{A.21}$$

$\frac{\partial C_i}{\partial z_j^{(l)}}$ *is by definition* $\delta_j^{(l)}$ *(see equation 2.15):*

$$= \delta_j^{(l)} a_k^{(l-1)} = a_k^{(l-1)} \delta_j^{(l)} \tag{A.22}$$

□

Appendix B
Definitions

Theorem 1 (Hadamard product) *The Hadamard product of two matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ with the same dimensions (not necessarily quadratic) is the component-wise product $A \circ B = [a_{ij}b_{ij}]$, which has the same dimensions as A and B [21].*

Example 1
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{bmatrix}$$

Theorem 2 (Entropy) *The Entropy $H(X)$ of a random variable X with distribution p is a measure of uncertainty. For a discrete variable with n states, it is defined by: $H(X) := -\sum_{k=1}^{n} p(X = k) \log(p(X = k))$ [26].*

Theorem 3 (Cross entropy) *The Cross entropy $H(p, q)$ between two discrete probability distributions p and q is defined as: $H(p, q) = -\sum_{k=1}^{n} p_k \log(q_k)$ [26].*

Theorem 4 (Kullback-Leibler divergence (KL divergence)) *The KL divergence is a measure of dissimilarity of two probability distributions p and q. It is defined as: $KL(p||q) = \sum_{k=1}^{n} p_k \log(\frac{p_k}{q_k})$ [26].*

Appendix C
Statistical Hypothesis Testing

C.1 Ventilation Data

We applied the t-test in order to examine whether test data of patients who were suffering from respiratory failure causes a higher cross entropy loss than test data of patients who were not suffering from respiratory failure, when fed into the discriminator D of a generative adversarial network, that was trained on data of patients who were not suffering from respiratory failure. Subsequently, the subscript 1 of a variable will indicate that the quantity is belonging to the test set of patients who were suffering from respiratory failure and the subscript 2 will indicate that the quantity is belonging to the test set of patients who were not suffering from respiratory failure.

Hypothesis H_0: $\mu_1 = \mu_2$

Alternative hypothesis H_a: $\mu_1 > \mu_2$

t-score:

$$t = \frac{\bar{x}_1 - \bar{x}_2 - 0}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{C.1}$$

Where $\bar{x}$ denotes the sample mean, n the sample size, and s^2 the sample variance. Furthermore, t obeys the t-distribution with $d := n_1 + n_2 - 2$ degrees of freedom. We chose a p-value of 0.1, which specifies the probability that the results occurred by chance. In order to reject H_0, the t-score has to be greater than $t_{0.1,d}$. For 3998 degrees of freedom and a p-value of 0.1, the t-table returns the value 1.282.

C.1.1 pCO2 Data

$$t = \frac{0.1008 - 0.0485 - 0}{\sqrt{\frac{2.303}{2000} + \frac{0.306}{2000}}} = 1.534 \tag{C.2}$$

Due to $t = 1.534 > 1.282$, we can reject H_0 in favor of H_a.

C.1.2 PO2 Data

$$t = \frac{1.2531 - 0.6691 - 0}{\sqrt{\frac{162.526}{2000} + \frac{70.708}{2000}}} = 1.282 \tag{C.3}$$

Due to $t = 1.822 > 1.282$, we can reject H_0 in favor of H_a.

C.1.3 PEEP Data

Due to a smaller amount of PEEP data, we reduced the sizes of the test sets to 500 examples per test set. This gave us 998 degrees of freedom. While still using a p-value of 0.05, the t-table returned a value of 1.646.

$$t = \frac{0.7925 - 0.8984 - 0}{\sqrt{\frac{162.526}{2000} + \frac{70.708}{2000}}} = 0.545 \tag{C.4}$$

Due to $t = 0.545 < 1.282$, we accept H_0.

C.2 ECG Data

We applied the t-test in order to examine whether test data of arrhythmic ECG recordings causes a higher cross entropy loss than test data of normal ECG recordings, when fed into the discriminator D of a generative adversarial network, that was trained only

on data of normal ECG recordings.

We formulated the t-test similar to the tests in section C.1, except we used a p-value of 0.05. For 108096 degrees of freedom and a p-value of 0.05, the t-table returns the value 1.282.

$$t = \frac{0.0021 - 0.0807 - 0}{\sqrt{\frac{0.567}{41457} + \frac{0.014}{66641}}} = 21.27 \tag{C.5}$$

Due to $t = 21.27 > 1.282$, we can reject H_0 in favor of H_a.

Bibliography

[1] J. Ramon, D. Fierens, F. Güiza, G. Meyfroidt, H. Blockeel, M. Bruynooghe, and G. Van Den Berghe. *Mining data from intensive care patients.* Advanced Engineering Informatics, 21 (3), p. 243-256, 2007. 1

[2] H. Wunsch, J. Wagner, M. Herlim, D.H. Chong, A.A. Kramer, and S.D. Halpern. *ICU occupancy and mechanical ventilator use in the United States.* Crit Care Med. 2013 Aug 19; PubMed, 2013. 1

[3] S. Vasilyev, R.N. Schaap, and J.D. Mortensen. *Hospital survival rates of patients with acute respiratory failure in modern respiratory intensive care units.* Chest 1995;107:1083-1088, 1995. 1

[4] S. Ridley and J. Purdie. *Causes of death after critical illness.* Anaesthesia 1992, 47, p. 116-119, 1992. 1

[5] G.A. Miller. *The magical number seven, plus or minus two: some limits on our capacity for processing information.* Psychological Review, 63 (1956), p. 81-97, 1956. 1

[6] J. Ramon. *Predicting Evolution of Critically Ill Patients.* In Proc. of the KDD 2006 workshop on Theory and Practice of Temporal Data Mining, p. 1-3, 2006. 1

[7] A. Cariou, D. Bracco, and A. Combes. *Sudden death in ICU: the Finnish experience.* Intensive Care Medicine 40(12), p. 1960-1962, 2014. 1

[8] O. Lesieur and L. Maxime. *Unexpected Cardiac Arrest in the Intensive Care Unit: State-of the-Art and Perspectives.* Réanimation 26(5), p. 411-424, 2017. 2

[9] B. Goldstein. *Intensive Care Unit ECG Monitoring.* Cardiac Electrophysiology Review 3, p. 308-310, 1997. 2

[10] M. AlGhatrif and J. Lindsay. *A brief review: history to understand fundamentals of electrocardiography.* J Community Hosp Intern Med Perspect 2(1), 2012.

[11] A.Y. Ng and M.I. Jordan. *On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes.* Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic (NIPS'01). MIT Press, Cambridge, MA, USA, 841-848, 2001. 3

[12] Y. Park, J. Ghosh, and M. Shankar. *Perturbed Gibbs Samplers for Generating Large-Scale Privacy-Safe Synthetic Health Data.* In Healthcare Informatics (ICHI), 2013 IEEE International Conference. p. 493-498, 2013. 3

[13] A.E.W. Johnson, T.J. Pollard, L. Shen, L. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L.A. Celi, and R.G. Mark. *MIMIC-III, a freely accessible critical care database.* Scientific Data (2016). DOI: 10.1038/sdata.2016.35. Available at: `http://www.nature.com/articles/sdata201635` 21

[14] A.L. Goldberger, L.A.N. Amaral, L. Glass, J.M. Hausdorff, P.C. Ivanov, R.G. Mark, J.E. Mietus, G.B. Moody, C.K. Peng, and H.E. Stanley. *PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals.* Circulation 101 (23), e215-e220, 2000. 21, 24

[15] V.N. Vapnik. *Statistical Learning Theory.* John Wiley & Sons, 1998. 3

[16] A.D. Dongare, R.R. Kharde, and D. Kachare. *Introduction to Artificial Neural Network.* International Journal of Engineering and Innovative Technology (IJEIT), 2 (1). p. 189-194, 2012. 4

[17] I.J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org` 8

[18] G. Strang. *Calculus.* Massachusetts Institute of Technology: MIT OpenCourseWare, 2005. `https://ocw.mit.edu`. License: Creative Commons BY-NC-SA. 12

[19] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning representations by back-propagating errors.* Nature Vol. 323, p. 533-536, 1986. `https://www.nature.com/articles/323533a0`. 12

[20] M. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015. 5, 12, 47, 48

[21] R.A. Horn. *Topics in Matrix Analysis.* Cambridge University Press. Chapter 5. p. 298-381, 1994. 52

[22] K. Polat and S. Güneş. *Detection of ECG Arrhythmia using a differential expert system approach based on principal component analysis and least square support vector machine.* Applied Mathematics and Computation 186(1), p 898-906, 2007. 17

[23] C. Cortes and V.N. Vapnik. *Support-Vector Networks.* Machine learning 20(3), p. 273-297, 1995. 17

[24] A. Vishwa, M.K. Lal, S. Dixit, and P. Vardwaj. *Clasification Of Arrhythmic ECG Data Using Machine Learning Techniques* [sic]. International Journal of Interactive Multimedia and Artificial Intelligence, 1 (4), p. 68-71, 2011. 17

[25] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks.* ArXiv e-prints. 1406.2661, 2014. 17

[26] K.P. Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press, Cambridge, MA, USA, 2012. 52

[27] *What Is a Ventilator?* - NHLBI, NIH. www.nhlbi.nih.gov. Accessed: 18.05.2018. 20

[28] *Invasive ventilation (IV)* - ResMed. https://www.resmed.com/epn/en/hospital/indications-for-ventilation/treatments/iv--invasive-ventilation--.html. Accessed: 18.05.2018. 20

[29] B. Wedro, D.L. Kulich, and C.P. Davis. *Electrocardiogram (ECG, EKG).* 2017. https://www.emedicinehealth.com/electrocardiogram_ecg/article_em.htm. Accessed: 08.08.2018. 20

[30] A.L. Mora Carpio and J.I. Mora. *Positive End-Expiratory Pressure (PEEP).* In: StatPearls. Treasure Island (FL): StatPearls Publishing, 2018. https://www.ncbi.nlm.nih.gov/books/NBK441904/. 22

[31] Glowm. *A free resource for medical professionals.* ISSN: 1756-2228. https://www.glowm.com/lab_text/item/3. Accessed: 13.06.2018. 24

[32] F. Jager, A. Taddei, G.B. Moody, M. Emdin, G. Antolic, R. Dorn, A. Smrdel, C. Marchesi, and R.G. Mark. *Long-term ST database: a reference for the development and evaluation of automated ischaemia detectors and for the study of the dynamics of myocardial ischaemia.* Medical & Biological Engineering & Computing 41(2):172-183 (2003). 27

[33] G.B. Moody and R.G. Mark. *The impact of the MIT-BIH Arrhythmia Database.* IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001). (PMID: 11446209) 17, 27

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems.* 2015. Software available from tensorflow.org. 31

[35] M. Arjovsky and L. Bottou. *Towards principled methods for training generative adversarial networks.* International Conference on Learning Representations (ICLR 2017), 2017. 32

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research 15, p. 1929-1958, 2014. 33

[37] A.P. Bradley. *The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms.* Pattern Recognition, 30(6), p. 1145-1159, 1997. 34

[38] J.N. Mandrekar. *Receiver operating characteristic curve in diagnostic test assessment.* J Thorac Oncol. 2010;5:1315-6, 2010. 34

[39] M. Arjovsky, S. Chintala, and L. Bottou. *Wasserstein GAN.* arXiv preprint arXiv:1701.07875, 2017. 43

[40] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* Ph. D. thesis, Technical University of Munich, 2008. 43

[41] S. Singh, S.K. Pandey, U. Pawar, and R.R. Janghel. *Classification of ECG Arrhythmia using Recurrent Neural Networks.* Procedia Computer Science 132, p. 1290-1297, 2018. 43

[42] Z. Xiong, M.K. Stiles, and J. Zhao. *Robust ECG signal classification for detection of atrial fibrillation using a novel neural network.* 2017 Computing in Cardiology (CinC), p. 1-4, 2017. 43

Index

Glossary

CPT (Current Procedural Terminology) Medical code set maintained by the American Medical Association.

ECG Electrocardiography.

ICD (International Classification of Diseases) Medical classification list by the World Health Organization (WHO).

ICU Intensive Care Unit.

pCO2 (Partial Pressure of Carbon Dioxide) Indicates the amount of carbon dioxide gas dissolved in the blood.

PEEP (Positive End-Expiratory Pressure) Pressure in the lungs above atmospheric pressure that exists at the end of expiration.

PO2 (Partial Pressure of Oxygen) Indicates the amount of oxygen gas dissolved in the blood.