GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Masterarbeit

Locating Excitation Sites and Spiral Waves from Multichannel-ECG Signals

angefertigt von

Roland Stenger

aus Erlangen

am Max-Planck Institut for Dynamics and Self-Organisation

Bearbeitungszeit: 1. April 2020 bis 15. Juli 2020

Betreuer/in: Baltasar Rüchardt

Erstgutachter/in: Prof. Dr. Ulrich Parlitz

Zweitgutachter/in: Prof. Dr. Alexander Ecker

Contents

CHAPTER 1

Introduction

This thesis is divided into 4 main chapters. Firstly I give a brief introduction, which is intended to give an overview of the two numerical experiments I made and places the experiments in a larger context so that previous work can later be compared to this work. The second chapter is about machine learning as a tool, which is used for the two experiments.

1.1 Subintroduction and so

Subintroduction and so

1.1.1 Subsubintroduction and so

Subsubintroduction and so

1.2 Research context and related work

Easy and so

1.3 Objective of Study

1.4 Thesis structure

This thesis is divided into 4 main chapters. Firstly I give a brief introduction, which is intended to give an overview of the two numerical experiments I made.

The thesis is separated into two parts in which each includes an evaluation of a numerical experiment. Both experiments are based on methods from the field of machine learning, which are summarized in chapter 2.

CHAPTER 2

Deep learning and neural networks

The concepts that fall under the definition of *deep learning* become more and more popular since a wide range of tools are based on it which often achieve state-of-the-art results. Important fields are for example language problems like machine translation [1] or speech recognition [2]. Also for games like chess or Go, tools, based on deep learning are the strongest players [3]. Another example for a breakthrough in recent time is a tool called AlphaFold which „solved“ (according to their definition) the problem of protein folding [4]. It can be said that deep learning is an important topic that has made possible a large number of breakthroughs in a wide range of research areas.

The term deep learning which is a sub field of machine learning refers to the concepts of software development where a computer is not directly programmed to solve a problem, but the instructions how to find a good solution. The approach that falls into the area of deep learning and which is based on this concept is the training of deep neural networks (DNN), a type of artificial neural networks. DNN's are a class of non-linear functions which are characterized by a large computational graph. A typical example of a neural network (regardless if it is deep or not) is the feed-forward network. It consists of the combination (or composition) of several similar functions the so-called layers, where each is build by a combination of a linear function and a non-linearity, the so-called activation function. Each layer provides a new representation of the input which passes through them. With each layer, the representation can be developed in a way, that it finally provides a meaningful representation, with respect to a given problem. This is a very brief description of the functionality, in the following section I will explain the mathematical basics behind it and important variations of the mentioned layers.

2.1 Artificial Neural networks

Artificial neural networks are inspired by biological neural systems. Yet strongly simplified, the terminology is oriented on parts of the brain. A first approach for artificial neural networks was introduced by Warren McCulloch and Walter Pitts 1943 in their paper „A logical calculus of the ideas immanent in nervous activity“ [5], where they explain their idea of a computational model which could represent mathematically how animal brains work on the scale of neurons to perform complex tasks. Their neural network is built from binary units with one or multiple binary inputs. It is able to perform simple binary classifications respectively logical computations. However, a more general description of a neuron is a based on continuous calculations, whose mathematical concepts I explain in the following.

2.1.1 Mathematical computations in artificial neural networks

The smallest functional unit of an artificial neural network is the neuron. As mathematical function f it takes a number of weighted scalars and outputs the sum of them. It is a linear function which quantity is represented by a weight-vector $\mathbf{w}$ and a bias b. The computations of a neuron are

$$y = f_{\text{neuron}}(\mathbf{x}) \equiv \mathbf{w}^T\mathbf{x} + b, \tag{2.1}$$

where $\mathbf{x} = (x_1, ...x_n)^T \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

However, a so-called layer in a feed forward neural network contains a bunch of neurons, the computation for all neurons within the linear part of it thus is written in the form

$$\mathbf{y} = f(\mathbf{x}) \equiv \underline{\underline{\mathbf{w}}}^T\mathbf{x} + \mathbf{b}, \tag{2.2}$$

with the weight-matrix $\underline{\underline{\mathbf{w}}} \in \mathbb{R}^{n\times m}$ and bias-vector $\mathbf{b} \in \mathbb{R}^m$ with the number of neurons m. This computation builds with a so-called activation-function A a single layer in a feed-forward network. Typically, activation-functions are the logistic function

$$\sigma(y) = \frac{1}{1 + \exp^{-y}}, \tag{2.3}$$

or the rectified linear unit (ReLU-function)

$$\mathrm{ReLU}(y) = \max(0, y), \tag{2.4}$$

to mention two of the most frequently used [CITE]. Finally, the computation within a single layer can be written as

$$\mathbf{y} = A \circ f(\mathbf{x}). \tag{2.5}$$

Note that the notation is point-wise, the activation-functions compute every scalar within an input-vector $\mathbf{y}$ independently. In figure (2.1) the analogy to a biological neural network is visualized, where the weight matrix are represented by connections between neurons and a neuron is represented by the nodes (blue circles). In this case a neural network is shown, where three layer are composed.

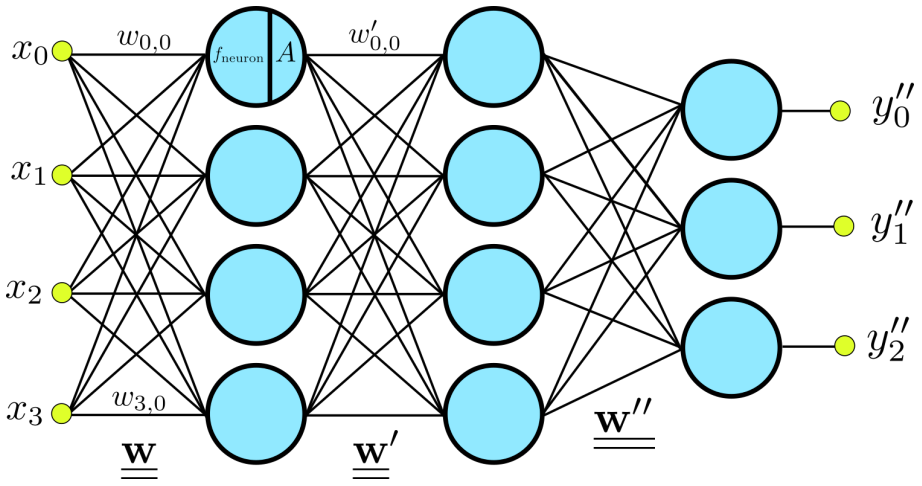

Abb. 2.1: Schematic view of the computation within a feed forward network, considerably the most simple neural network architecture.

The key of a neural network is that the weight matrices and biases are adjusted in a way that the network gives a useful output $\mathbf{y}$ regarding a given problem. These parameter will be adjusted in the training process whose fundamentals I introduce in the following section.

2.1.2 Training of neural networks

The search for the networks optimal parameter adjustment is called training process. The goal of training a neural network is that the network performs good on yet *unseen* data when the performance is measured by a loss-function, which computes the discrepancy between predictions and the *true values*. Predictions are the output of the neural network, previously noted as $\mathbf{y}$ while the *true values* are the aim for what the network should predict, based on a given input. Mostly, the training is a minimization process by applying a distance metric such as mean squared error to quantize the validity of the prediction.

The subject of machine learning can be roughly divided into three areas: Supervised learning, unsupervised learning and reinforcement learning. Here an input is already assigned to a corresponding output, the relationship which the network is supposed to learn. This type of training process is called supervised learning. Since this work contains of two experiments which focus only on supervised learning, when explaining the training process, I only consider this case.

2.1.2.1 Optimization process

Optimizer are iterative algorithms to find the optimal networks parameter which (mostly[1]) minimize a loss function L. With each iteration, a sub set of the training data called batch is passed to the loss function, whose gradient with respect to the networks parameter θ is then used to update these parameter. The gradient calculation is performed by the backpropagation algorithm [6] in the parameter space. There exist a lot different optimizer respectively update-rules, from which I will explain one of the most basic one called *gradient descent*, to have an idea about the general functionality. Every optimizer of this kind is as well based on an update rule including the gradient of a loss function. An update in the *gradient descent* algorithm is computed according to

$$\theta_{\text{new}} = \theta + \eta \cdot \nabla_\theta L(f(X|\theta), y), \tag{2.6}$$

where η is the learning rate, a quantity to adjust how big the network parameters

[1] Dependent on the loss-function, the goal could also be the maximization. Nonetheless every loss-function can be reformulated so that the training process is again a minimization task, without changing the properties of the loss function.

θ can change within an updated. The right choice of this parameter is critical [7]. There are three types of gradient descent methods which differ in the amount of data to be passed through one iteration. If X contains of one *data-point*, the optimizer performs a parameter update with every single training example. In contrast there is the *batch gradient descent* which calculates the gradient on the whole dataset to perform a parameter update The third possibility is a compromise of both, where the optimizer only uses a part of the dataset which however contains more than one single example each, to perform one parameter update.

Probably the most famous optimization-algorithm in machine learning is the Adaptive Moment Estimation optimizer (Adam)[8] which is specifically designed to train neural networks. Firstly introduced in 2014 it showed big performance gains in training speed. In this work I use the same optimizer for every training process.

Overfitting

When training a model on a dataset, we want a loss-function to get smaller during the process according to that dataset, but as well that it can generalize what it learned from the training data on yet unseen data. A problem can appear if the model relies too much on the training data that it even considers secondary patterns like noise, which is a disadvantage for a model which should be able to generalize well. In the example in the left figure of (2.2) the model predicts very well on the known data, but at the same time is not able to capture the most dominant trend which might be handled better by a regression with squared term. Overfitting can appear if the model is too complex or the model was trained for to many iterations. Staying in the example of figure (2.2), the model is too complex, while a regression with squared term would perform better.

Underfitting

The problem of underfitting appears if the model is not capable of processing the complexity of the dataset. Reasons are if the model is not trained enough that did not learn *enough* from the data, or if the model is *too simple*. An example is visualized in figure (2.2, right), the model is not able to capture the main trend as well. This is due to the fact that here a linear regression was chosen as model, which is not able to handle a quadratic trend.

To face these problem, there are a few regularization techniques which I also apply in the neural networks for in this work.

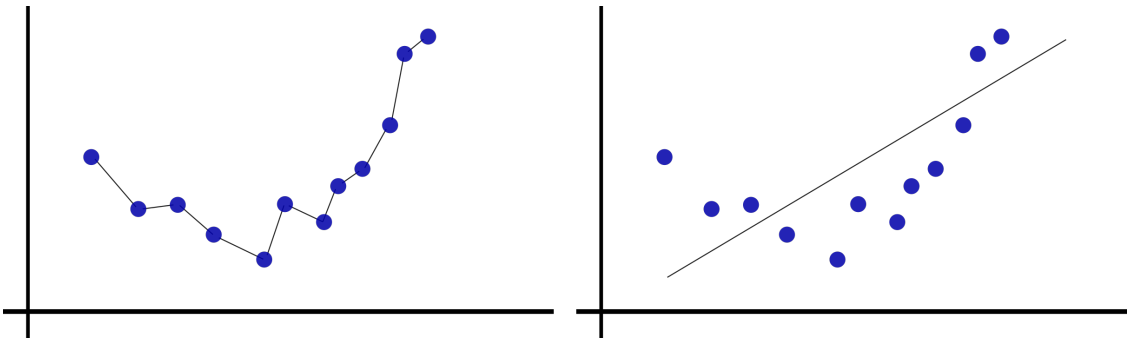

Abb. 2.2: Easy over and under

2.1.3 Regularization against over- and underfitting

The following methods to prevent overfitting and underfitting can be divided into two areas: On the one hand, I present methods that are implemented in the form of additional layers in the network (such as dropout and batch normalization), and on the other hand, there are also rules that only take effect during the training. The difference is that one method co-defines the computational graph and may have trainable parameters, while the other methods are applied during the training process.

2.1.3.1 Dropout

By ignoring a set of neurons a neural network can imitate *sub networks* and is therefore able to have a dynamic architecture. The regularization technique called dropout randomly removes neurons during any execution of the network. Firstly introduced by N. Srivastava et al. [9], dropout helps to prevent overfitting by not allowing the network to have a rigid architecture which may be too oriented to the training data.

2.1.3.2 Batch normalization

Batch normalization can increase the stability of a neural network by normalizing the input of single layers in the neural network. It can increase the training speed with being able to handle bigger learning rates from the optimizer. Deep neural network are sensitive to the choice of initial weights and the training algorithm respectively the choice of the learning rate. Batch normalization increases the robustness with regard to the choice of these parameters [10].

2.1.3.3 Early stopping

While the two previous regularization techniques require to be implemented with the neural network, early stopping is a rule that takes effect during the training process. It interrupts the training process as soon as there is no improvement in the loss for the test set for a longer period, a so-called *plateau*.

2.1.4 Convolutional neural networks

How neural networks possibly have been inspired by biological brains, convolutional neural networks (CNNs) might have its inspiration from the brain's visual cortex. They are commonly designed for tasks in computer vision and have been able to achieve state-of-the-art results with models which includes convolutional operations. Examples *EfficientNet* in image classification [11] or *EfficientPS* in semantic segmentation [12]. Nonetheless, CNNs are not restricted to computer vision tasks, but also successfully used for example in *ContextNet* for speed recognition [13] or in natural language tasks like machine translation as in *ConvS2S* [14]. In this work CNNs are used to process electrocardiography signals (See chapter **??**). However, I firstly focus on visual tasks to explain the computations behind CNNs since it is most common use case. Later in this work, instead of a 2D image as an input, a 1D ECG-signal is used as well, but the mathematical principles are the same, which can easily be transposed to a lower-dimensional case.

2.1.4.1 Kernel-convolution

A convolutional neural network is based on the kernel-convolution. It is a discrete operation where a small matrix W, called kernel or filter, is passed over an image I and transforms it in dependence of the kernel- and image-values. The operation is computed as

$$g_{x,y} = W * I_{x,y} = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} W_{dx,dy} \cdot I_{x+dx,y+dy}, \tag{2.7}$$

where a and b indicate the size of the kernel W. The function is calculated for all x and y on the image on which the filtered image g is based on.

2.1.4.2 Multi-channel convolution

In most images, each pixel is represented by three values which define the color (red, green and blue). The convolutional operation should be able to take these three values (in the following called channels) into account. It is also important that multiple filters can be applied to the image, so that the input image and the output image have multiple channels. A more general description of the kernel-convolution with multi-channel input and output is given by

$$g'_{i,x,y} = W'_i * I_{x,y} = \sum_{j=1}^{N} \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} W_{dx,dy} \cdot I'_{j,x+dx,y+dy}, \tag{2.8}$$

with N as number of channel within the input image (for example $N = 3$ in an rgb-image). Applying (2.8) with every kernel W'_i in $\mathbf{W} := (W_1, ..., W_M)$ gives a multi-channel output $\mathbf{g}'_{x,y} := (g_{1,x,y}, ..., g_{M,x,y})$.

2.1.4.3 Valid- and same- convolution

The equations above don't consider yet how the computation looks like at the edges when $x + dx$ or $y + dy$ exceeds the image boundaries. Two common rules are called *valid* and *same*. *Valid*-convolution calculates only values for the respective point (x, y) as far as it lies in the image, while with *same*-convolution, zeros are added at the edge of the input image, so that the filtered image has the same width and height as the input. The parameter to implement these rules is called padding, it specifies the thickness of the additional margins added to the image with zeros.

2.1.4.4 Strided convolution and max-pooling

In previous equations, the kernel is only shifted with a step-size (stride) of one pixel. However, one can increase the step-size if $\mathbf{g}$ should have smaller spatial dimensions. Furthermore, as consequence of a bigger stride, each pixel in $\mathbf{g}$ is dependent of a bigger sub-field, the so-called *receptive field* on the input image. The size of this field is important for processing spatial relationships within the image [CITE].

While *stride* is a hyperparameter of the kernel-convolution operation, *max-pooling* is a technique that functions as an independent layer in a neural network. It divides the input (here a 2D-image) into subsets like 2×2-fields of spatial neighbouring pixels and outputs only the maximal image value of each. This technique reduces the

spatial dimension of the input as well, and increases the receptive field.

In this work, convolution-layers are implemented within neural networks of both experiments. The first experiment applies a 1D-convolution to process through electrocardiography-signals, while in the second experiment, 2D-convolutions are used to encode spatial correlations and decode a low dimensional representation of images.

The chapter showed how convolutional neural networks process through spatial[2] correlations. In the next chapter I introduce recurrent neural networks, which are designed to process temporal dynamics in sequences.

2.2 Recurrent neural networks

The prominent domain for recurrent neural networks are sequence-to-sequence problems. It is about training neural networks to convert a sequence to another sequence. A famous task in this domain is machine translation where a sequence of words in one language is transformed to another sequence of words of a different language. Other important tasks are for example speech-recognition where an encoded voice recording has to be converted into a sequence of words, or next-frame-prediction, a task to predict the future frames of a video. In each example, recurrent networks were considered as state-of-the-art, at least for a while. [CITE]

A characteristic that is generally seen with sequence-to-sequence (seq2seq) problems is that the sequences are of variable length. For example, a translation-model has to be able to take a sequence with varying length of words to output a translation as a sequence with a varying length as well. Furthermore a strong translation model should be able to translate single words as well as a long sequences of hundreds of words. In this work, in the second numerical experiment (VERWEIS HIER) also faces the characteristic of variable input-output length, where a recurrent neural network is used as well. Therefore, a key feature of machine learning models, designed for seq2seq problems, are their ability to handle sequences of different lengths. In the following, I introduce the smallest functional unit of a recurrent neural network, the *recurrent unit*.

[2]The terminology *spatial* does not necessarily refer to data points that are spatially related. It is to be understood more generally in the sense that data points, such as the pixels of an image, in the form in which they are stored (as a 2D array in this example) also contain information through their position and their neighbors.

2.2.1 The recurrent unit

The recurrent unit is a form of neural network which is, in combination of an iterative approach, able to process through an input sequence with varying length. Recurrent neural networks (RNNs) which are built of one or multiple recurrent units (for example stacked into deep RNNs) can be regarded as a generalization of a neural network. With every iteration a *hidden state*, which has a fixed dimensionality, is evolving to be dependent on every time step before. The *hidden state* can be calculated, regardless of the length of the input sequence, and provides a representation of the sequence.

The following equations are the computations within a very basic recurrent unit which builds the so called Elman network [15]. Given a sequence of inputs $(x_1, x_2, ..., x_T)$, the hidden-state h_t evolves as

$$\begin{aligned} h_t &= f_h(\mathbf{W}_x \cdot x_t + \mathbf{W}_h \cdot h_{t-1} + b_h), \\ y_t &= f_y(\mathbf{W}_y \cdot h_t + b_y). \end{aligned} \tag{2.9}$$

Furthermore it gives an output sequence $(y_0, y_1, ..., .y_T)$ of the same length as the input sequence, which is taken into account in deep recurrent neural networks by stacking multiple RNNs. The functions f_h and f_y are activation functions like tanh or the sigmoid-function.

2.2.1.1 Extensions

Bidirectional RNN

Bidirectional RNNs combine the input sequence and the inversion through time of the same sequence. It can be regarded as an implementation of two independent RNNs which compute through the forward and backward direction of the input, whose output is stacked together. With this form of RNNs the output layer can combine information from the past and future and was firstly introduced by M. Schuster et al. in 1997 [16].

Stacked RNN

RNNs can be stacked by taking the output sequence (y_1, ..., y_T) or the sequence of hidden states (h_1, ..., h_T) as input for a new RNN. Although it is not theoretically clear why stacked RNNs perform in certain tasks better than single-layer RNNs, in practise it provides a higher learning capacity [17, p.51].

2.2.1.2 Training of recurrent neural networks

The computation of the gradient of recurrent neural networks it is not possible by the normal backpropagation algorithm. However, there is a generalization of it called *backpropagation through time* or *BPTT*. The key behind this algorithm is to unfold the network through time into a feed forward network with multiple layers. Therefore the depth depends linear on the length of the input sequence.

During training the gradient tends to vanish or explode, as firstly discovered by Hochreiter in 1991. He described it as follows:
With conventional „Back-Propagation Through Time“ [...], error signals „flowing backwards in time“ tend to either blow up (1) or vanish (2): the temporal evolution of the backpropagated error exponentially depends on the size of the weights. Case (1) may lead to oscillating weights, while in case(2) learning to bridge long time lags takes a prohibitive amount of time, or does not work at all. [18].
Within the learning algorithm gradient descent, which includes backpropagation, these difficulties are theoretically validated with any loss function [19]. Various methods have been proposed to deal with the problem of vanishing and exploding gradient, whereby the most successful is the Long-short-term-memory (LSTM). Hochreiter and Schmidhuber introduced a special kind of RNN in 1997 which is able to prevent the problem of vanishing gradient.

2.2.2 Long-short-term-memory

LSTMs work on the same principle as RNNs by evolving a hidden state through an input sequence, but they use a different function to compute the hidden state which furthermore includes an additional state, the *cell state*. With the computations in a LSTM unit it is provided that the gradient of the cell state can be unchanged through time. It is possible to process long term dependencies in a sequence more efficiently. Furthermore, a LSTM overcomes the problem of a vanishing gradient.

The calculations within a LSTM unit it given by

$$
\begin{aligned}
g_t &= \tanh\left(\mathbf{W}_{cx} \cdot x_t + \mathbf{W}_{ch} \cdot h_{t-1} + b_g\right), \\
i_t &= \sigma\left(\mathbf{W}_{ix} \cdot x_t + \mathbf{W}_{ih} \cdot h_{t-1} + b_i\right), \\
f_t &= \sigma\left(\mathbf{W}_{fx} \cdot x_t + \mathbf{W}_{fh} \cdot h_{t-1} + b_f\right), \\
o_t &= \sigma\left(\mathbf{W}_{ox} \cdot x_t + \mathbf{W}_{oh} \cdot h_{t-1} + b_o\right), \\
C_t &= f_t \circ C_{t-1} + i_t \circ g_t, \\
h_t &= o_t \circ \tanh(C_t),
\end{aligned}
\tag{2.10}
$$

where $\circ$ denotes the Hadamard product, also known as pointwise-multiplication.

2.2.3 Variations and extensions

2.2.3.1 Convolutional LSTM

At least since the performance of convolutional neural networks (CNNs) in the ImageNet competition, it is empirically proven that CNNs are more effective and efficient in certain computer vision tasks like image classification than fully connected neural networks. In the field of sequential learning, computer vision tasks are also formulated, such as video-classification or video-frame interpolation. However, the LSTM as implemented in (2.10) can be regarded as recurrent extension of a fully connected neural network, which is not as efficient in processing spatial correlations as a CNN.

A LSTM with convolutional structure addresses this problem. It is better performing in certain tasks which include spatio-temporal data, as proven for example from Wu et al. for future video synthesis [20]. Therefore, this concept is used in the second numerical experiment of this work, which also deals with the processing of spatio-temporal data.

The equations which build a convolutional LSTM are shown below:

$$
\begin{aligned}
g_t &= \tanh\left(\mathbf{W}'_{cx} * x_t + \mathbf{W}'_{ch} * h_{t-1} + b_g\right), \\
f_t &= \sigma\left(\mathbf{W}'_{fx} * x_t + \mathbf{W}'_{fh} * h_{t-1} + b_f\right), \\
i_t &= \sigma\left(\mathbf{W}'_{ix} * x_t + \mathbf{W}'_{ih} * h_{t-1} + b_i\right), \\
o_t &= \sigma\left(\mathbf{W}'_{ox} * x_t + \mathbf{W}'_{oh} * h_{t-1} + b_o\right), \\
C_t &= f_t \circ C_{t-1} + i_t \circ g_t, \\
h_t &= o_t \circ \tanh(C_t).
\end{aligned}
\tag{2.11}
$$

As introduced in chapter 2.1.4, the symbol * denotes a discrete convolution while the operator $\mathbf{W}'$ are convolutional kernel. The linear operators are replaced by convolutions, that the input of a convolutional LSTM can handle a sequence of spatial data without flattening them to a vector.

2.2.3.2 Spatiotemporal LSTM

As introduced before, a convolutional LSTM is taken into account advantages of convolutions to compute spatial correlation more efficiently. However, in a stacked convolutional LSTM, spatial correlations and temporal dynamics are not equally processed. But these two aspects might be equally important and have to be considered equally significant in a machine learning model for spatio-temporal data. To address this problem, Y. Wang et al. introduce in their paper "PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs"[21] a variation of the LSTM-cell and its integration into a stacked network. They formulate the problem as follows:
[...] spatial representations are encoded layer-by-layer, with hidden states being delivered from bottom to top. However, the memory cells that belong to these [...] layers are mutually independent and updated merely in time domain. Under these circumstances, the bottom layer would totally ignore what had been memorized by the top layer at the previous time step.

The equations for the spatio-temporal LSTM (ST-LSTM) are given by

$$
\begin{aligned}
g_t &= \tanh\left(\mathbf{W}'_{cx} * x_t + \mathbf{W}'_{ch} * h_{t-1} + b_g\right), \\
i_t &= \sigma\left(\mathbf{W}'_{ix} * x_t + \mathbf{W}'_{ih} * h_{t-1} + b_i\right), \\
f_t &= \sigma\left(\mathbf{W}'_{fx} * x_t + \mathbf{W}'_{fh} * h_{t-1} + b_f\right), \\
C_t &= f_t \circ C_{t-1} + i_t \circ g_t, \\
g'_t &= \tanh\left(\mathbf{W}''_{cx} * x_t + \mathbf{W}'_{cm} * M_t^{l-1} + b'_g\right), \\
i'_t &= \sigma\left(\mathbf{W}''_{ix} * x_t + \mathbf{W}'_{im} * M_t^{l-1} + b'_i\right), \\
f'_t &= \sigma\left(\mathbf{W}''_{fx} * x_t + \mathbf{W}'_{fm} * M_t^{l-1} + b'_f\right), \\
M_t^l &= f'_t \circ M_t^{l-1} + i'_t \circ g'_t \\
o_t &= \sigma\left(\mathbf{W}'_{ox} * x_t + \mathbf{W}'_{oh} * h_{t-1} + \mathbf{W}_{om} * M_t^l + b_o\right), \\
h_t &= o_t \circ \tanh(\mathbf{W}_{1\times1} * \left[C_t^l, M_t^l\right]).
\end{aligned}
\tag{2.12}
$$

The idea behind this LSTM-adaptation is visualized in figure (2.4). With each layer, the spatial representation develops within an additional cell-state M with each layer and is then passed to the bottom layer of the next time-step.

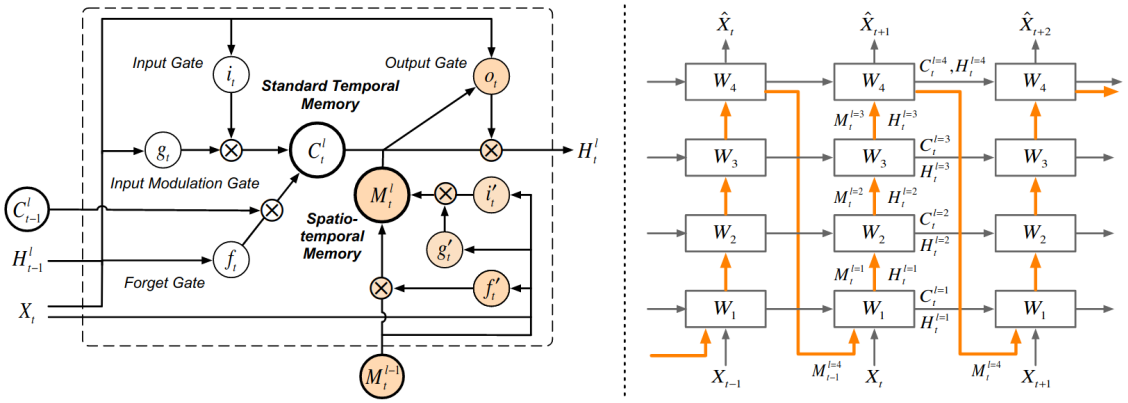

Abb. 2.3: Spatiotemporal LSTM, (left) the computations inside the ST-LSTM-cell, (right) the unfolded cell through time. The orange arrow marks the spatial flow through layer and time-step.

LSTMs are the favored implementation of recurrent networks, it is a standard to overcome general problems like vanishing gradient and learning long-term dependencies. Additionally, convolutional LSTMs or spatio-temporal LSTMs are designed to specifically face problems which occur in deep LSTMs where spatial correlation

and temporal dynamics are not equally processed.

Although recurrent cells have been mentioned so far, it is not yet clear how exactly this iterative process can be used to process a sequence2sequence problem.

2.3 Seq2Seq-models

A seq2seq LSTM-neural network is a type of encoder-decoder model, consisting of two LSTMs, one for encoding the sequence to a *thought vector* of fixed dimension, and the other for generating an output sequence with varying length from the thought vector. Furthermore, each time-step of the input might be processed by an encoder independently, as well as the output sequence steps by a decoder as visualized in figure (2.4).

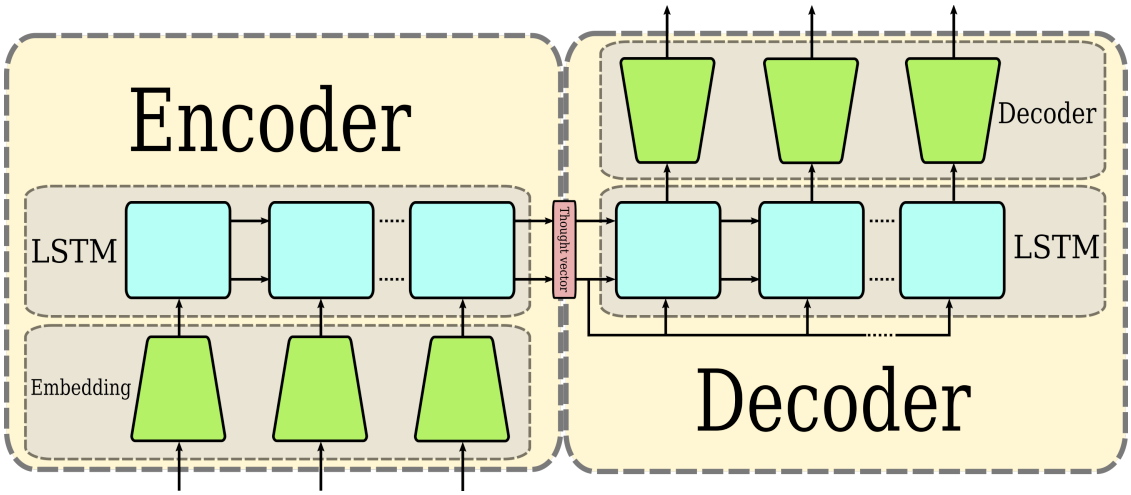

Abb. 2.4: Schematic sequence to sequence model with an encoder-decoder structure.

2.4 Tools

This chapter gives a short overview of the used software and libraries.

2.4.1 Python

Python is a high level, general purpose programming language distributed by the Python Software Foundation ("Python – SF"). Python was first released in 1991. The current Python 3 version,which is used in this thesis, was released in 2008. It runs through an interpreter, thus python code can be run without the need to

compile it. This allows for fast development and testing of code. Due to this and the huge amount of external libraries python is a widely used language in many scientific projects. It is also one of the most used language to develop NNs because it offers many powerful DL libraries, e.g. TensorFlow, Pytorch, Theano, Caffe etc. Furthermore, Python offers an almost limitless amount of other libraries useful for data preprocessing. The following three libraries wereused for this purpose:1.Numpy Numpy is a module that allows for scientific computing in Python. Its most used feature, in the context of this thesis, is the creation, editing and calculation of N-dimensional array objects (“NumPy”).

2.4.2 Keras

2.4.3 PyTorch

PyTorch [22]is an open-source machine learning framework, based on Torch [23], which is the chosen API behind the majority of state-of-the-art models. By current status[3], pytorch is the most popular framework of its kind. Together with keras (the second most used API) around 66% of all papers based on machine learning research have used at least one of these APIs. They support the most commonly used methods such as convolutions, LSTMs in a high abstraction.

2.4.3.1 Automatic differentiation in PyTorch

[3]According to the status in September 2020

CHAPTER 3

The inverse problem of electrocardiography

In broad terms, the inverse problem of electrocardiography (iECG) is about gaining knowledge about the activity of the heart, based on measurements from electrodes on the torso surface [24, p.300] Based on this definition, the experiment described below can also be understood as the processing of the inverse problem of ECG. The aim of this study is to find out to what extent it is possible to differ between several pattern in the heart's activity through ECG measurements.

One of the most important use case of ECG measurements is the classification of the corresponding activity of the heart. [...] A much more difficult task is the complete reconstruction of the cardiac activity or the surface activity using the ECG signals. Mathematically, this can be described as ill-posed, which in our case means that a possible solution is not unique[1]. [...] Therefore, it can be said that the inverse problem of electrocardiography is an important topic in medicine which potential is not yet exhausted.

3.1 Problem formulation

As already briefly explained, this experiment is about the differentiation of activities of the heart, which are characterized by different spatio-temporal excitation patterns. A representation of this activity is provided by the measurements of electrodes at a certain distance from the heart.

In this experiment, the data of the electrical activity of the heart as well as the corresponding ECG signals through the electrodes are simulated (see section 3.5 for

[1] According to the definition of Jacques Hadamard [CITE] a problem is well-posed if (1.) a solution exist, (2.) the solution is unique, and (3.) the solution depends continuously on the data. A problem is ill-posed if it fails to satisfy one or more of these conditions

more details about the simulation). It is a simulation of a pig's heart whose geometric shape comes from a CT-scan. The electrodes are set up at a certain distance within two grids as visualized in figure (3.1). Four electrodes are marked in the graphic from which a virtual reference electrode is calculated whose geometric center lies approximately in the middle of the heart. All electrodes of the two grids have the same reference electrode.

Different excitation patterns are generated by stimulating a previously defined point on the heart surface at a constant frequency, where the position varies in every simulation. This pacing induces a concentric waves which spreads over the entire heart. The aim of this experiment is to predict the position where the pacing events are happening, based on the ECG-signals.

A simulation generates ECG data in the form $X_{\text{ECG}} \in \mathrm{R}^{T \times N}$ where T is the number of discrete time steps and $N = 70$ the number of leads (respectively electrodes, each lead is built with the same virtual electrode). The exact spatio-temporal activity of the heart is not recorded as it is not relevant for the prediction. Only the position of the pacing is stored as a 3-dimensional vector. The selection of the relevant parameters T, N and the conversion of the discrete quantities (time and space) into characteristic quantities are described in chapter 3.5 (simulation) respectively 3.6.1 (data processing).

3.1.1 Related work

A few paper, problems

Since the experiment involves the simulation of electrical propagation in cardial tissue, the biological background is first explained in the following chapter.

3.2 Biological background

This chapter gives an overview of the anatomy of the heart and its smallest functional units, the cardiac muscle cells. The coordinated contraction of these cells leads to a pumping action of the heart. Within a repetitive pumping process, blood is enriched with oxygen within a heart chamber and passed on to the body. In the following I briefly explain how the anatomy of the heart makes this process possible, as well as discuss the electro-chemical properties of cardiac muscle cells, to understand its coordinated contraction through signal propagation which lead to the pumping effect.

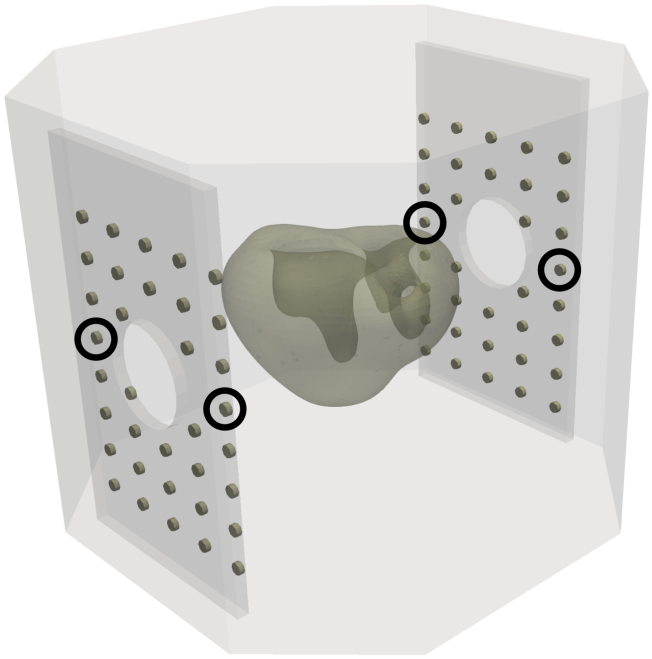

Abb. 3.1: Easy

3.2.0.1 Anatomy of the heart

This section is based on chapter 1 of the book *Herz-Kreislauf* by J. Steffel and T. Lüscher [25]. The heart consists of four chambers, the left and the right atrium and the left and the right ventricle, as visualized in figure (3.2). The right atrium with the right ventricle builds the right half of the heart, which pumps the blood to the lungs for a gas exchange. From there the blood flows towards the left half of the heart (which consists of the left atrium and the left ventricle). The left half of the heart pumps the oxygen-rich blood towards the individual organs. The heart beats between 50 and 80 times a minute in normal condition. The pumping is caused by an electrical impulse, initiated by the sinus node. The pulse (action potential) is a rapid potential change of the membrane potential induced by moving ions through the cell membrane. The pulse propagates from the atria to the ventricles via the atrioventricular node. The electrical signal causes the cells to contract. In addition, these cells are able to transmit the signal to neighboring cells. In the following I explain the details about the signal transmission.

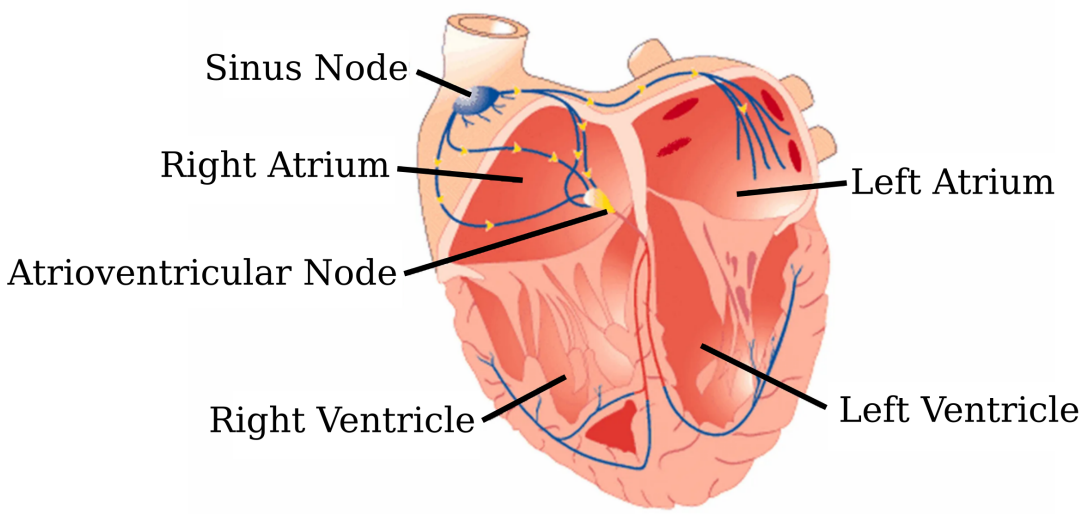

Abb. 3.2: Anatomy of the heart with the four chambers (two atriums and ventricles) including the sinus and atrioventricular node. Copyright 2019 by a-fib.com

3.2.0.2 Electrophysiology of cardiac cells

The action potential causes the heart muscle cells to contract and transmit the signal to other neighboring heart cells. This transmission of stimuli results in a coordinated extraction of the heart in the normal state, which results in a pumping action, the heartbeat.

3.3 Electrocardiography

The electrocardiography is a representation of the heart's electrical activity, measured at a certain distance from the heart. The representation is in the form of potential differences, whereas the potentials are measured with the help of electrodes on the body surface. By subtracting the measurement of one electrode (or virtual electrode) with another as reference, one can gain information about the spatial potential gradient along the vector between both. A pair of electrodes form a so called lead, by subtracting one signal from the other as a reference. A lead can be partially or fully made up of virtual electrodes, where the term virtual is used, as far as if a measurement is build from the average potential from multiple electrodes. A virtual electrode represents the potential at the spatial average of other electrodes of which it is built. A widely known setting is the 12-lead-ECG, where 10 electrodes are used to form 12 leads. The positioning of these is visualized in figure (3.4). The leads

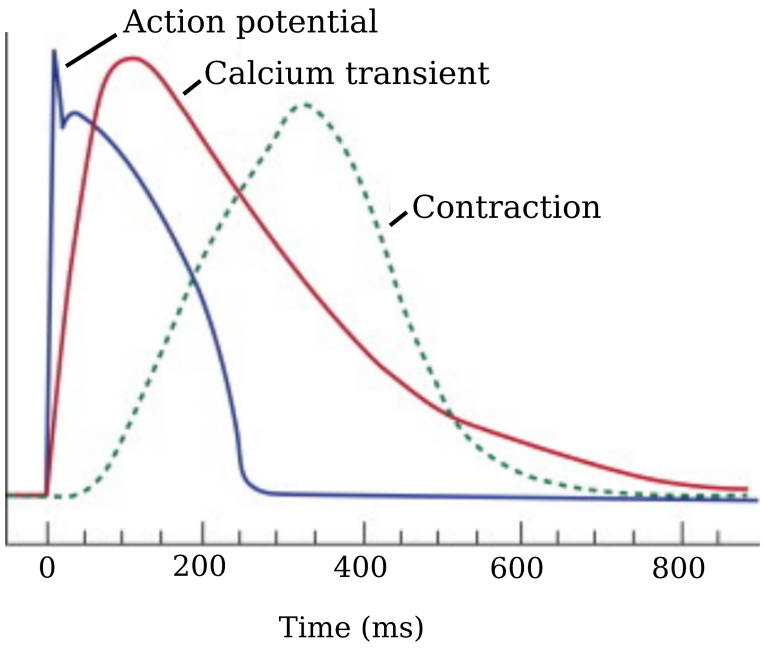

Abb. 3.3: Physiology of the heart. Copyright 2020 by sciencedirect.com

form vectors of which 6 lie on a vertical plane (frontal leads) and 6 on a horizontal plane (transverse leads).

- Frontal leads have in common that one electrode is ideally placed (virtually) inside the heart. With the second electrode, which is marked on the image (3.4) as V1, ...V6 (depending on the lead). The structure makes it possible to measure the potential difference between the inside of the heart and the surface of the body.
- Transverse leads

Here, a virtual electrode is built from **R**, **L** and **F**. It defines pairwise with V1 to V6 the first 6 leads.

The propagation of the electrical signals from the heart to the surface can be modeled as a diffusion process (also simulated in this work as such). This process is topic of the next chapter.

3.3.1 Forward propagation from the heart

The terminology forward and inverse suggest that there is a direction in which the signals propagates. The suggestion here is that the potential differences lead to

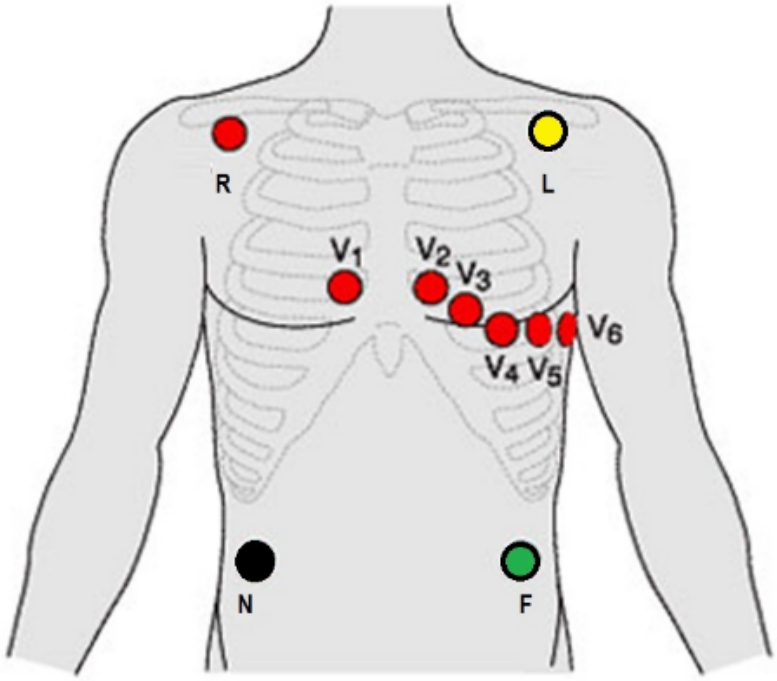

Abb. 3.4: 12-lead ECG, positioning. Copyright 2020 by firstaidforfree.com

3.4 The inverse problem

3.4.1 Related work and limitations

3.5 Simulation of ECG signals

The following is a description of the simulation of a heart-model and its corresponding ECG signals. The electrical activity of the heart is simulated with the FitzHughNaguro model on a pig's heart where the 3D structure is based on a computed tomography scan (CT-scan). Two grids of electrodes are placed at a certain distance from two opposing sides as visualized in figure (3.6). The following data are considered in the numerical experiment:

- The activity of the heart model at each discrete point in the form of a system parameter (see chapter 1000).
- The potential differences of the respective electrodes of the grid, to a virtual electrode at the approximate center of the heart.

The simulation-framework is provided by Baltasar Rüchardt where the generation

of the data can be divided into 4 parts, which I will describe below.

1. Simulation of excitable media (electrical activity of the heart)
Hey

2. Reconstruction of the extracellular potential in the heart
hey

3. Diffusion process in the bath
hey

4. Calculation of the ECG

$$C_m \frac{\partial V_m}{\partial t} + I_{\text{ion}} = \nabla(\sigma_m \nabla V_m) \tag{3.1}$$

applies, where C_m is the membrane capacitance per unit area, and I_{ion} is the ionic current density through the membrane. The parameter σ_m can be regarded as bulk conductivity[2].

The local ion flux is simulated with the FitzHugh-Nagumo model [**?**].

The 3D model of the pig-heart consists of around 32000 connective points to define a finite element mesh. The differential equations are solved with help from the open-source framework DOLFIN [**?**] in Python, the FitzHugh-Nagumo equations are solved by the implementation of the Runge-Kutta solver from scipy [**?**].

The main simulation framework is provided by Baltasar Rüchardt [**?**] and contains four steps:

- augmented monodomain simulation,
- reconstruction of the extracellular potential in the heart,
- diffusion process in the bath,
- calculation of the ECG.

[2]For further derivation and explanation see [**?**].

The potential propagates as a diffusion process through the bath, it can be regarded as a process which blurs the propagating potential. In the last step, because each electrode contains multiple points, the signal is integrated over each of these subsets. Figure (3.6) shows the simulation setup with the heart inside the bath with the included ECG-electrodes, attached on the two panels from two sides.

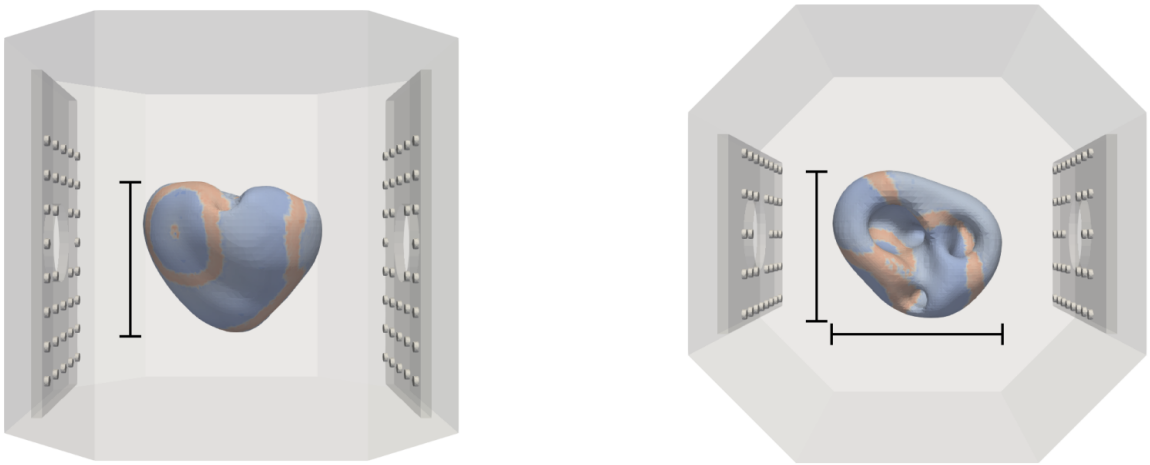

Abb. 3.5: Simulation of a pig heart, visualized with paraview [**?**], the panel on the left and right side on each image contain 64 electrodes for the simulated electrocardiography, as they are placed in the experiment in **??**.

3.6 Machine learning for iECG problem

3.6.1 Data processing

3.6.2 Neural network approach

3.7 Results and validation

In the following I present the performance of the convolutional neural network (c.f. section **??**) which is trained to predict the source of a concentric.

3.7.1 Concentric Wave Localization

With the regularization methods during the training process (c.f. section **??**), the training lasted for 2310 epochs. This took around 58 minutes on a Nvidia K100 GPU. While the heart has an expansion of between around 60 and 80 length units,

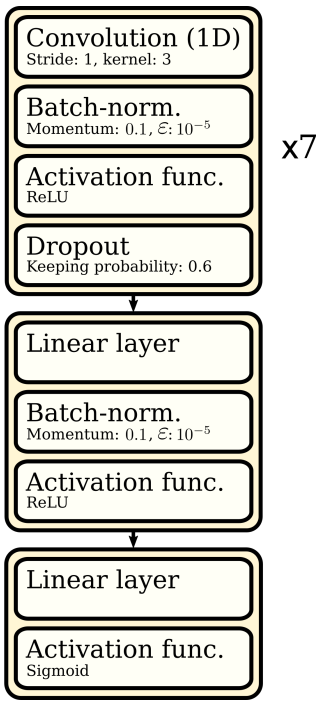

Abb. 3.6

the model is able to predict the source of the concentric wave from the validation set on average with a displacement of 3.11 length units. Imagine a pig's heart which was simulated, and estimating the size of the heart of between 10cm and 20cm in each direction (x, y, z), the average displacement would be approximately between 0.75cm and 1cm. The comparability faces limitations due to the fact that the simulation produces a system without noise (geometrical noise in time and space, uncertainties on the measurements or positioning of the electrodes etc.) Nevertheless, it shows a first attempt that the CNN is able to localize the source of concentric waves (under the prerequisite of perfect conditions) from unknown data, whose displacement is not orders of magnitude worse when comparing with the results from the training-dataset.

training set	test set
1.74	3.11

Tab. 3.1: Average displacement of the prediction from the CNN on the training- and test-dataset.

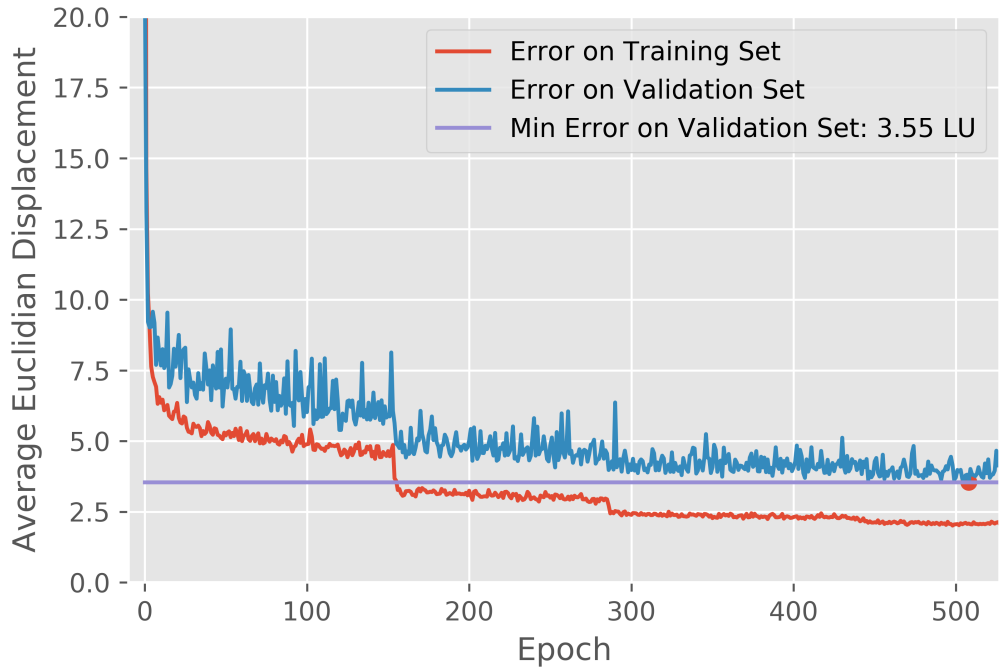

Abb. 3.7: Development of the average euclidean displacement by epoch on the training- and validation set.

3.7.1.1 Limitations of the Model

This experiment is an attempt to show the principle ability to use convolutional neural network to solve a task with a time-series as input. This special posed task of the inverse problem of ECG gave comparably good results. This shows that the neural network is able to process through a temporally random selected section of fixed length, which makes it clear that it recognizes a pattern in the signals regardless of the phase of the concentric wave. The simulated heart contains of around 10000 elements on the surface, which can represent an approximate source of the concentric wave. While having 1024 simulation, having unique starting conditions, around 10% of the available sources are already covered. The properties of a simulation differ only in the location of the pacing events that cause the concentric wave. It is possible that some training and test simulations differ very little from each other, if the locations of the pacing events are very close to each other. In this case it may not be possible to speak of a truly unknown data set. The average minimal distance from a test-simulation to the next training-simulation is around 2.47 LU. It shows that the neural network is trained with pacing positions, which are not far distant

from the test-simulations. Furthermore, the neural networks average displacement of the predictions on the test-dataset is bigger than the average distance to the next training simulation. Nevertheless, the same CNN is able to handle a dataset with additional Gaussian noise. The training on the same dataset with this noise with $\sigma = 0.05$ got displacements as shown in table 3.2:

training set	test set
1.69	3.15

Tab. 3.2: Average displacement of the prediction from the CNN on the training- and test-dataset, additional Gaussian noise with $\sigma = 0.05$ is added to the normalized data (this means that a 5% error is included).

The average displacement on the test-dataset did not increase much (a difference of 0.04LU).

CHAPTER 4

Under-surface prediction

This chapter is about the second numerical experiment. The basis for it is a 3D-simulation of the time evolution of an excitable medium. The question is to what extent a machine learning model respectively a neural network can predict a system quantity inside of the 3D structure which is not *visible* from outside. The input to the neural network and its prediction are values of the same system quantity, where the input is recorded on the surface of the 3D structure over a period of time. Here, *not visible* means that the neural network has no explicit information about the data inside the 3D structure that it is supposed to predict. Only on the basis of the surface it is to extrapolate these. The posing of this problem is motivated by a possible transfer to heart dynamics. It would be a matter of inferring the inner dynamics from the surface dynamics which results from the electrical activity of the heart muscle cells.

In general, one wants to get as much information from the heart's activity as possible with minimally invasive or non-invasive measurement methods to make an intervention to obtain this information as low-risk and cost-effective as possible. While the first experiment was about drawing conclusions about the electrical activity at the surface of the heart from the ECG signals, one prospect of this experiment is to predict the interior electrical activity, based on the heart's surface.

Some experiments have been done by [xx] or [yy]. While these approaches are based on real data and , in this work I will go a step back and want to study the limits of machine learning models that faces up to the task mentioned above, while having *perfect* surface information. *Perfect* means here, unlike in real experiments, that the data does not have any noise[1]. Furthermore, with choosing a cube as com-

[1] If we ignore the numerical error.

paratively simple geometry, I regard this as a base study about the limitations of machine learning models in this kind of problem. This numerical experiments tests two different sequence-to-sequence LSTM architectures on two different parameter sets within the model for excitable media where one builds *concentric waves* and the other *breaking spiral waves*, as visualized in figure [xx]. Details concerning the characteristics of these regimes are explained in section [yy]

How a possible transfer of this experiment to biological data can be realized and to what extent it would build on the results of this study is discussed in chapter xx.

4.1 Methods

Goal is to train a machine-learning model which takes into account the time evolution of the u-value of the Barkley model [26] at the surface, to predict at a certain time point the u-value under the surface. In this chapter I introduce the process of generating training- and validation data and elaborate some characteristics of the simulations.

4.1.1 Model

There are various models to describe the heart dynamic but since this experiments goal is to reconstruct hidden regions under the surface qualitatively, the model does not have to describe the heart dynamic as accurate as possible. In this work the Barkley model is used, which is a system of two coupled differential equations which build a reaction-diffusion system. It is a comparably[2] simple model and consists of two variables u and v which are dependent on the equations

$$\begin{aligned} \frac{\partial u}{\partial t} &= D \cdot \nabla^2 u + \frac{1}{\varepsilon}(1-u)\left(u - \frac{v+b}{a}\right) \\ \frac{\partial v}{\partial t} &= u^\alpha - v. \end{aligned} \tag{4.1}$$

The parameter ϵ controls . A bigger value for a increased the excitation duration while an increasing fraction between b and a, $\frac{b}{a}$, results to a larger excitability

[2]Compared for example with the Hodgkin-Huxley model which consists of three coupled differential equations.

threshold. In this work the value for α is set to 1. The exact values for both regimes are in table (4.1).

4.1.1.1 Simulation

The Barkley-model will be simulated in three dimensions in which the discretised diffusion operator ∇^2 is here described through a 7-point stencil such that

$$\left(\nabla_7^2 u\right)_{i,j,k} = \frac{u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k} + u_{i,j,k-1} + u_{i,j,k+1} - 6u_{i,j,k}}{\Delta s^2}, \quad (4.2)$$

where the indices i,j and k stand for the discretised position within the x-y-z-cube and Δs the spatial discretisation. A simulation step in time will be calculated through the explicit Euler method with

$$\begin{aligned} u(t+1) &= u(t) + \Delta t \frac{\partial u(t)}{\partial t} \\ v(t+1) &= v(t) + \Delta t \frac{\partial v(t)}{\partial t}. \end{aligned} \quad (4.3)$$

that a time-step within the Barkley model is described as

$$\begin{aligned} u(t+1)_{i,j,k} &= u(t)_{i,j,k} + \Delta t \cdot \left(D \cdot \left(\nabla_7^2 u\right)_{i,j,k} + \frac{1}{\varepsilon}(1 - u(t)_{i,j,k}) \left(u(t)_{i,j,k} - \frac{v(t)_{i,j,k} + b}{a} \right) \right) \\ v(t+1)_{i,j,k} &= v(t)_{i,j,k} + \Delta t \cdot \left(u(t)_{i,j,k}^3 - v(t)_{i,j,k} \right) \end{aligned} \quad (4.4)$$

The hyperparameter a, b, ε, Δt and Δs are arbitrary chosen for two regimes, in which one builds concentric waves and the other shows a chaotic behaviour. The exact values are shown in table (4.1). The simulation is implemented in python with orientating on the code for a 2D Barkley simulation by Roland Zimmermann [27].

4.1.1.2 Hyperparameter and simulation size

The parameter set for 'non-chaotic' is adjusted to build concentric waves (as visualized in figure xx), while the chaotic regime produces breaking spiral waves. The

	Non-chaotic	chaotic
a	0.6	1.0
b	0.01	0.15
ϵ	0.02	0.02
Δt	0.01	0.01
Δs	0.1	0.1
α	1	1

Tab. 4.1: Barkley simulation configurations.

parameter adjustment for a and b is based on a paper by Alonso et al. [28] in which they study the parameter-space of the Barkley model and their influence within a 3D-simulation.

These two parameter regimes The parameter are specifically chosen to generate Both simulations are embedded on the cube with 120 unit in each dimension.

4.1.1.3 Characteristic units

To have a more intuitive understanding about time- and spatial sized which are taken into account by the models I apply characteristic dimensions as following:

- Length L_c: The characteristic length L_c is defined by average thickness along the normal of a wave front. Here it is estimated manually by taking the average from 32 measurements $\iota = \overline{(\iota_1, ..., \iota_{32})}$. The length follows to $L_c = \iota \cdot \Delta\text{s}$.

- Velocity V_c: The velocity is measured by counting the amount of discrete length units Δs which are passed per discrete time step Δt. To minimize the error, the distance is measured after 64 time-steps, and the average of 32 of these measurements build the characteristic velocity as $V_c = \nu \cdot \Delta\text{s}/\Delta\text{t}$.

- Time T_c: The characteristic time is applied by the time a wave needs to travel the characteristic length. It follows the time according to $T_c = L_c/V_c \equiv \iota/\nu \cdot \Delta\text{t} =: \tau \cdot \Delta\text{t}$.

The resulting values for ι, ν and τ are shown in table (4.2).

Intuitively, a wave-front which propagates along the normal of the surface in a depth of $\iota \cdot$ds, needs in average τ discrete time steps to be recognisable at the surface.

	1: Non-chaotic	2: chaotic
ι	10.50±1.45	3.15
ν	0.155±0.041	3.15
τ	67.74±20.24	3.15
L_c	1.05±0.15	3.15
V_c	1.55±0.41	3.15
T_c	0.68±0.20	3.15

Tab. 4.2: Characteristic dimensionless units for the two Barkley configurations.

4.1.2 Generating data and preprocessing

The following described procedure of data generation and preprocessing is the same for both regimes. The only difference is the adjusting of hyperparameters according to table (4.1).

4.1.2.1 Data generation

As mentioned before, I consider in the experiment the u value of the Barkley model. Since the cube has an expansion of 120 discretised units in each spatial dimension, a snapshot of the u value is a tensor of order 3 with a dimensionality of (120,120,120)=(depth, height, width). In a single simulation each snapshot is recorded after 16 discrete time-steps from the previous snapshot over a span of 512 steps. Therefore, each simulation saves a tensor with the dimensionality of (32,120,120,120). The recording starts after an initial phase of 2048 time steps. Basis for the preprocessing step, which I explain in the following, are 256 of these simulations for each regime (concentric and chaotic). Here I use X to describe the data the machine learning model gets as input to predict y.

4.1.2.2 Preprocessing

Since a cube has 6 sides, here each simulation produces 6 time-series from the surface. The corresponding y-data is a tensor which saves the u-value for the next lower layers below the respective surface. The maximal depth is set at 32 discrete spatial units. Therefore, from 256 single simulations, one gets $256 \cdot 6 = 1536$ X-y-pairs with $X \in R^{32\times1\times120\times120}$ and $y \in R^{1\times32\times120\times120}$. It is divided into a training-set (1024 pairs) and validation set (512 pairs). Note that the dimensions stand for (time-steps×depth×height×width). The X-data includes only the surface of the

cube, therefore the depth is always 1, while y is the prediction at a certain time-step till a maximal depth of 32. The data within the simulations are computed as float64 (means each single value needs 64 bit of storage with the float data type). Nonetheless it is possible without a great loss on information to convert the data into one-byte-integers (int8) to save storage. Therefore, X and y are rescaled into the range of int8 (-127 to 128).

Within the training process, the data again has to be rescaled into the range from 0 to 1. This is done with the help of a pytorch dataloader, whose functionality is explained in 2.4.3.

4.1.3 C-LSTM and ST-LSTM

In section (2.3) a general design of sequence-to-sequence models with recurrent neural networks was introduced. In this section, I present the exact composition of the models used in this experiment. The two models differ only in the implementation of their recurrent cells, which are the C-LSTM cell (see section 2.2.3.1) and the ST-LSTM cell (see section 2.2.3.2).

The models are based on the same architecture as visualized in figure (2.4). The encoder- (embedding) and decoder-networks (colored green in figure (2.4)) are time distributed, which means that the same network is used separately for each time point of an input sequence. Encoder and decoder are convolutional neural networks whose architecture is visualized in figure (4.1) and (4.2). Some convolutional layers have a stride-parameter of 2, which reduces the spatial dimensionality by the factor of two. Therefore, with an input X with $X \in R^{\text{batch} \times \text{time-steps} \times \text{depth} \times \text{height} \times \text{width}}$, the encoder reduces it to $X_{\text{out}} \in R^{\text{batch} \times \text{time-steps} \times \text{features} \times \text{height}/2 \times \text{width}/2}$. Note, that instead of *depth*, the third dimension represents now the number of channels, which is a hyperparameter of a convolutional layer as introduced in section (2.1.4.2). The value is always set for all convolutional layers within the encoder and decoder to 64.

4.1.4 Optimal hyperparameter search

While the architecture of the two approaches (C-LSTM and ST-LSTM) will be considered as given, I search for the optimal hyperparameter for the training process. These are the loss-function, learning rate, batch size and the optimizer. Table (4.3) shows the values of the hyperparameters that are varied, as well as the ideal parameter combination.

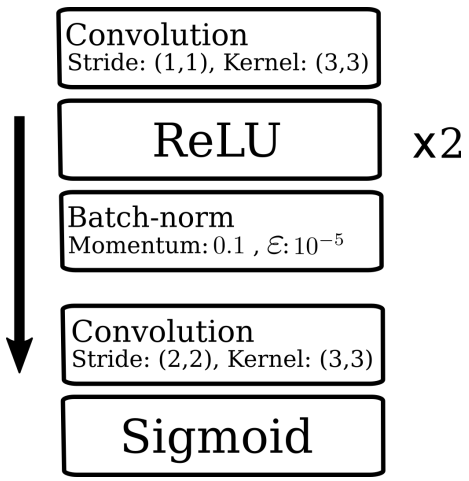

Abb. 4.1: Encoder

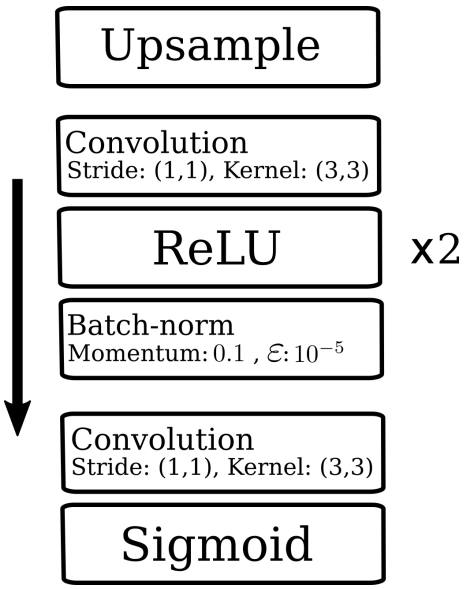

Abb. 4.2: Decoder

Parameter	Values	Best value
Loss function	MSE, MAE	MSE
Learning rate	$\{1, 2, 3, 5\} \cdot 10^{-4}$, $\{1, 2, 3\} \cdot 10^{-3}$	10^{-4}
Batch-size	2,4,6,8	4
Optimizer	SGD[3], Adam	Adam

Tab. 4.3: Characteristic dimensionless units for the two Barkley configurations.

4.2 Specific studies

In this work I consider four studies. Included are the two described models C-LSTM and ST-LSTM to perform tasks within two regimes. All studies have in common, that a machine learning model takes as input a temporal sequence of the u-value from the Barkley model, which is one side of the surface of the 3D simulation with the shape of a cube.

4.2.1 Influence of the input-sequence length

First, I vary the length of the temporal sequence as input. Goal of this study is to find in

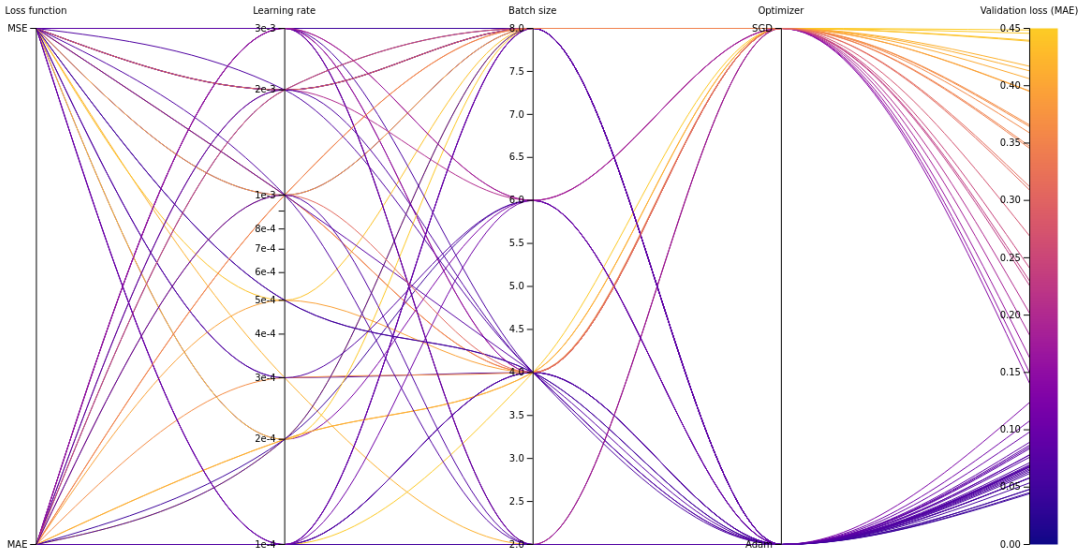

Abb. 4.3: Results of a grid-search with 4 varying hyperparameter. The hyperparameter in this graphic are sorted by its influence to the loss from less important (left) to more important (right) (the importance is sorted by random forest feature importance to calculate how much a hyperparameter choice played a role in the result according to the metric). The parameter-variations and best combination are in table (4.3). The loss is calculated after one epoch with 1024 examples.

4.2.2 Qualitative detection of hidden structures

In the second study I figure out if hidden structures can be deteced by

4.2.3 Comparison C-LSTM, ST-LSTM

4.2.4 Performance of chaotic regime

First, I vary the length of the time interval that the models use as a basis for their prediction. In excitable media, an excitation has an increasing global influence with time passed. This leads to the assumption that excitations below the surface have an influence on the surface dynamics after a certain time passed. The first study is about the question if or rather in which amount this influence can affect the prediction accuracy. The characteristic velocity, which I introduced in chapter 4, plays an important role in this process. Since in both regimes (see table xx) build wave-fronts, the average speed of these wave fronts is used as characteristic velocity.

4.3 Results and validation

Influence of the input-sequence length

The results are separated into three parts. Firstly I show the general ability of detecting hidden structures with a surface-sequence. These results are based on experiments which are performed only with the C-LSTM in regime 1 (see table (4.1) at *non-chaotic* for the exact Barkley-hyperparameter).

The in the second part, I compare the results of C-LSTMs and ST-LSTMs

The plot show curvature which are smoothed with the Savitzy-Golay filter [29] to increase the visibility of the curves. Without the filter they would be very noisy.

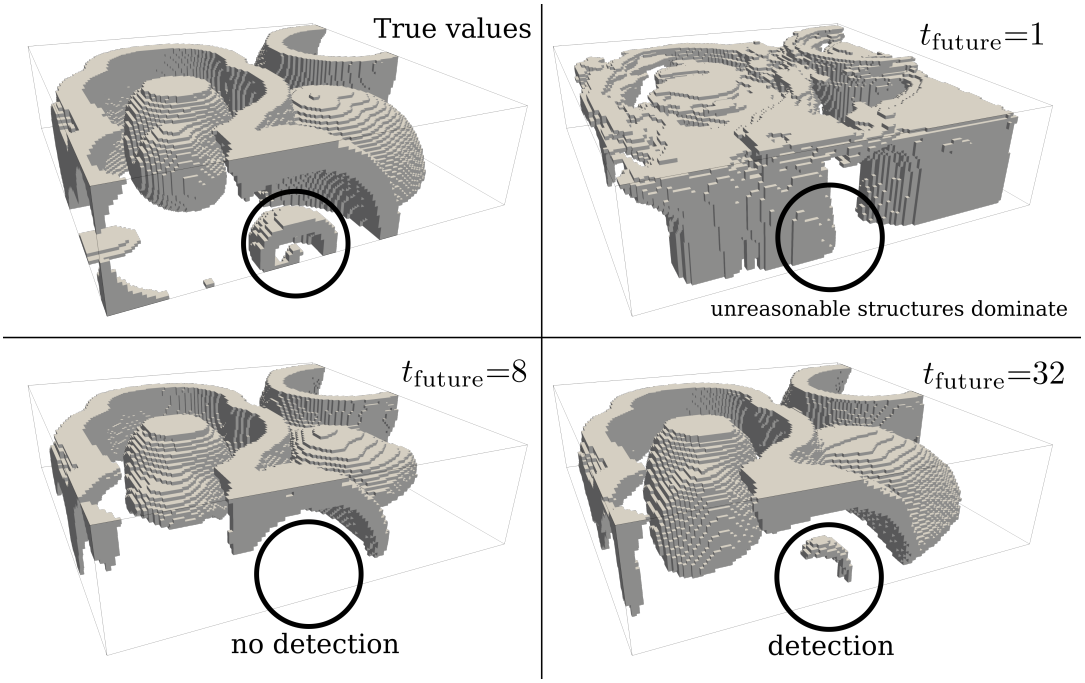

Abb. 4.4: Comparison, a point is visualized as long its value is over 0.5.Comparison, a point is visualized as long its value is over 0.5.Comparison, a point is visualized as long its value is over 0.5.

What are the results What is the research usable for: - A study how good models work in this kind of problem, if it is even possible to detect hidden structures. And the work shows, it definitely is able to detect some of these structures under good circumstances and not in real time because it needs data from the future. Nonetheless some patterns can periodically.

In a periodic dynamic, is it enough to have the surface-data? What is the core of the problem in terms of machine learning? What is proven? How can the results be

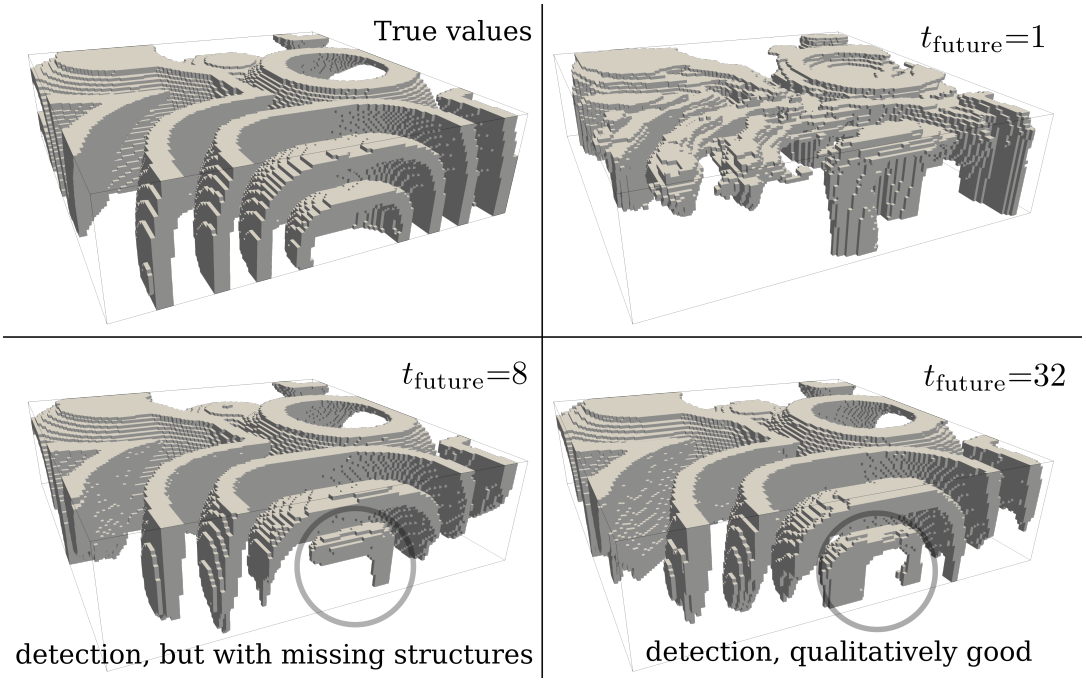

Abb. 4.5: Comparison, a point is visualized as long its value is over 0.5.

useful for future work? - Can the models respectively the problem formulation be generalized, for example for a more complex geometry with the topological description of a doughnut. How can approaches be embedded in the big field of machine learning - shows one approach with is proven in a number of different fields to be able to build a state-of-the-art model or was state-of-the-art in a certain time.

CHAPTER 5

Outlook

Outlook and so

CHAPTER 6

Summary

Literaturverzeichnis

[1] Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. Very Deep Transformers for Neural Machine Translation. *arXiv:2008.07772 [cs]*, October 2020. arXiv: 2008.07772.

[2] Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. December 2015. arXiv: 1512.02595.

[3] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, December 2020.

[4] The AlphaFold team. Alphafold: a solution to a 50-year-old grand challenge in biology, 2020.

[5] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943.

[6] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[7] Yanzhao Wu, Ling Liu, Juhyun Bae, Ka-Ho Chow, Arun Iyengar, Calton Pu, Wenqi Wei, Lei Yu, and Qi Zhang. Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks. *arXiv:1908.06477 [cs, stat]*, October 2019. arXiv: 1908.06477.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[10] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, March 2015. arXiv: 1502.03167.

[11] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [cs, stat]*, September 2020. arXiv: 1905.11946.

[12] Rohit Mohan and Abhinav Valada. EfficientPS: Efficient Panoptic Segmentation. *arXiv:2004.02307 [cs]*, May 2020. arXiv: 2004.02307.

[13] Wei Han, Zhengdong Zhang, Yu Zhang, Jiahui Yu, Chung-Cheng Chiu, James Qin, Anmol Gulati, Ruoming Pang, and Yonghui Wu. ContextNet: Improving Convolutional Neural Networks for Automatic Speech Recognition with Global Context. *arXiv:2005.03191 [cs, eess]*, May 2020. arXiv: 2005.03191.

[14] Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. Classical Structured Prediction Losses for Sequence to Sequence Learning. *arXiv:1711.04956 [cs]*, October 2018. arXiv: 1711.04956.

[15] Jeffrey L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, March 1990.

[16] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, November 1997.

[17] Yoav Goldberg. A Primer on Neural Network Models for Natural Language Processing. *arXiv:1510.00726 [cs]*, October 2015. arXiv: 1510.00726.

[18] Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies. In *A Field Guide to Dynamical Recurrent Networks*. IEEE, 2009.

[19] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994.

[20] Yue Wu, Rongrong Gao, Jaesik Park, and Qifeng Chen. Future Video Synthesis with Object Motion Prediction. *arXiv:2004.00542 [cs]*, April 2020. arXiv: 2004.00542.

[21] Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S Yu. PredRNN: Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs. page 10.

[22] Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[23] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[24] Peter W. et al. Macfarlane. *Herz-Kreislauf: mit 25 Tab.* Springer, 2010.

[25] Jan Steffel, Thomas F. Lüscher, and Corinna Brunckhorst. *Herz-Kreislauf: mit 25 Tab.* Springer-Lehrbuch. Springer Medizin, Heidelberg, 2011. OCLC: 694879718.

[26] D. Barkley. Barkley model. *Scholarpedia*, 3(11):1877, 2008. revision #91030.

[27] P.W.D. Charles. Project title. `https://github.com/charlespwd/project-title`, 2013.

[28] S. Alonso. Taming Winfree Turbulence of Scroll Waves in Excitable Media. *Science*, 299(5613):1722–1725, March 2003.

[29] A. Savitzky and M. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36:1627–1639, 1964.

Erklärung Ich versichere hiermit, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die von mir angegebenen Quellen und Hilfsmittel verwendet habe. Wörtlich oder sinngemäß aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht. Die Richtlinien zur Sicherung der guten wissenschaftlichen Praxis an der Universität Göttingen wurden von mir beachtet. Eine gegebenenfalls eingereichte digitale Version stimmt mit der schriftlichen Fassung überein. Mir ist bewusst, dass bei Verstoß gegen diese Grundsätze die Prüfung mit nicht bestanden bewertet wird.

Göttingen, den 29. Januar 2021

(Roland Stenger)