Ruprecht Karl University of Heidelberg
Institute for Computer Science
Visual Learning Lab

Master Thesis

Heart Arrhythmia Modelling with Invertible Neural Networks

Name:	Mustafa Fuad Rifet Ibrahim
Matriculation number:	3284705
Supervisor:	Prof. Dr. Ullrich Köthe
Date of submission:	December 10, 2020

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht. Die Arbeit ist in gleicher oder vergleichbarer Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Heidelberg, December 10, 2020, ______________________________

Mustafa Fuad Rifet Ibrahim

Abstract

Invertible Neural Networks are a promising family of networks for generative modeling tasks. They have been applied to various scientific models in which they allow to solve the inverse problem of generating input variables for given output variables of the model in a simple and efficient way. This thesis applies Invertible Neural Networks to an electrical signal conduction model of the human heart which involves discrete and sequential data. Various different architectures and data transformations were tested and the results show that Invertible Neural Networks can be successfully applied to this inverse problem case and that especially those networks that utilize recurrent neural networks as subnetworks combined with a suitable transformation of the data work well.

Zusammenfassung

Invertierbare Neuronale Netzwerke sind eine vielversprechende Familie von Netzwerken für generative Modellierung. Diese wurden bereits auf verschiedenste wissenschaftliche Modelle angewandt, bei denen sie es ermöglicht haben das inverse Problem, bei dem bei gegebenen Output Variablen Input Variablen generiert werden sollen, einfach und effizient zu lösen. Diese Thesis wendet die Invertierbaren Neuronalen Netzwerke auf ein Modell der elektrischen Signaltransduktion im menschlichen Herzen an, welches Daten diskreter und sequentieller Natur beinhaltet. Verschiedene Architekturen und Arten der Datentransformation wurden getestet und die Ergebnisse zeigen, dass Invertierbare Neuronale Netzwerke erfolgreich auf dieses inverse Problem angewandt werden können und dass besonders solche Netzwerke, die rekurrente neuronale Netzwerke als Subnetzwerke verwenden zusammen mit einer geeigneten Datentransformation, gut funktionieren.

Contents

Chapter 1

Introduction

The systems considered in the natural sciences are often very complex and non-linear. Scientists usually approximate these systems with simpler models that produce results which can be compared to the real world systems. Whether these models are derived from first principles or attempt to explain phenomena by only borrowing from fundamental theory, they typically map certain input variables that are not easily measured in nature to output variables in a way that is not simply bijective. In this way they attemtpt to provide an explanation of observations in terms of these input variables. However, for this inverse process of finding input variables for given output variables the model has to be used in a smart way to generate possible explanations without needing too much time by searching through the whole solution space. This problem can be avoided in the case where the model implements a bijective mapping, since this allows for the use of the inverse function to quickly generate input variables for given output variables. This is the central idea behind the so called Invertible Neural Networks which are based on the concept of normalizing flows [1][2][3][4][5][6]. This type of statistical model utilizes neural networks to transform the probability distribution of the input data to a standard normal distribution from which one can easily sample and run the network in inverse mode to generate input data. These networks have been applied to a variety of problems including ones that involve sequential data [7].

This thesis is concerned with using these invertible networks in the context of heart arrhythmias by applying them to a model of electrical signal conduction in the human heart developed by Florian Kehlre [8]. This specific application case is interesting because the data considered is partly of discrete and sequential nature. The goal is to apply invertible networks in different ways utilizing different data transformations and network architectures in order to solve the inverse problem for which Kehlre's model was used. This thesis is split up into five main parts. After the introduction, Chapter 2 lays the theoretical groundwork for the rest of the thesis. First, the medical basis is given with a short introduction to the human heart as well as an introduction to Kehlre's model. After that, the mathematical basis for neural networks and specifically the invertible networks mentioned above is presented. Chapter 3 states the objective of this thesis and gives a detailed description of the data used as well as the generation process of that data. It then presents all the approaches that were developed to solve the inverse problem. Chapter 4 explains the training and testing procedure used in this thesis. After that it shows the results that were obtained for all approaches. Finally, Chapter 5 discusses the obtained results and interprets them with respect to model architecture differences as well as data transformations. Chapter 5 ends with an outlook that outlines possible directions for future research on this topic. The code for this thesis will be available on GitHub [9].

Chapter 2

Theoretical Background

This chapter covers the medical basis as well as mathematical theory behind the models presented in this thesis.

2.1 The Human Heart

This section starts with a brief overview of the human heart based largely on [10]. After that, an introduction to a specific model of cardiac signal conduction, developed by Florian Kehrle [8], is given.

2.1.1 Basics

The human heart is an organ with the main job "to collect blood from the tissues of the body and pump it to the lungs and collect blood from the lungs and pump it to all of the tissues of the body" (p. 61 in [10]). The heart consists of different chambers that are separated by valves which maintain a certain blood flow direction (Figure 2.1). These chambers consist of muscle cells, i.e. they can contract and relax to pump out as well as collect blood. The oxygen-poor blood from the tissues of the body is collected in the upper right chamber (*right atrium*), flows through the *tricuspid valve* into the lower right chamber (*right ventricle*) and is pumped through the *pulmonary semilunar valve* to the lungs. The oxygen-rich blood from the lungs is collected in the upper left chamber (*left atrium*), flows through the *bicuspid valve* into the lower left chamber (*left ventricle*) and is pumped through the *aortic semilunar valve* to the tissues of the body.

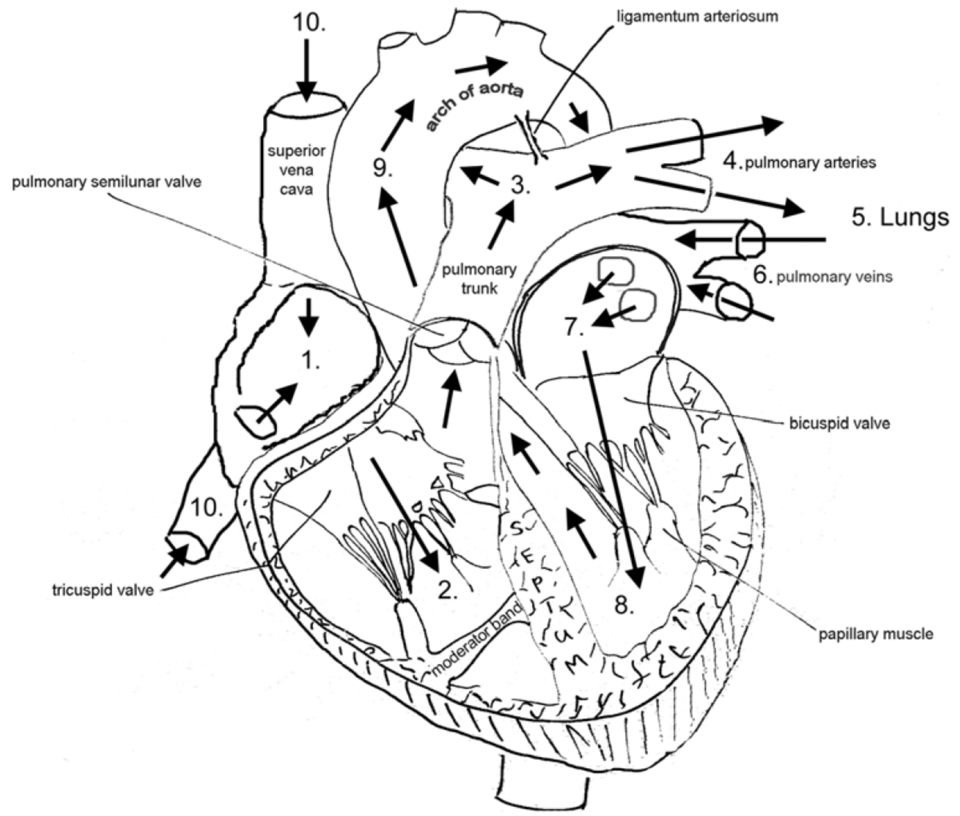

Figure 2.1: Bloodflow within the heart (Fig. 5.8 in [10]). The aortic semilunar valve is not shown.

This process requires a precise and coordinated stimulation of the invovled muscle cells. The electrical conduction system of the heart enables this coordinated control (Figure 2.2). The main component of this system are specialized muscle cells that can spontaneously generate electrical signals and/or conduct them. In a healthy adult heart the initial electrical signal originates from a specific set of those cells located in the right atrium called the *sinoatrial node* (SA node). From there the signal travels through both atria, depolarizing them, and to another set of cells called the *atrioventricular node* (AV node). After that the signal travels through the *bundle of His* and the *Purkinje fibers* leading to the depolarization of the ventricular muscle cells which corresponds to one heart beat.

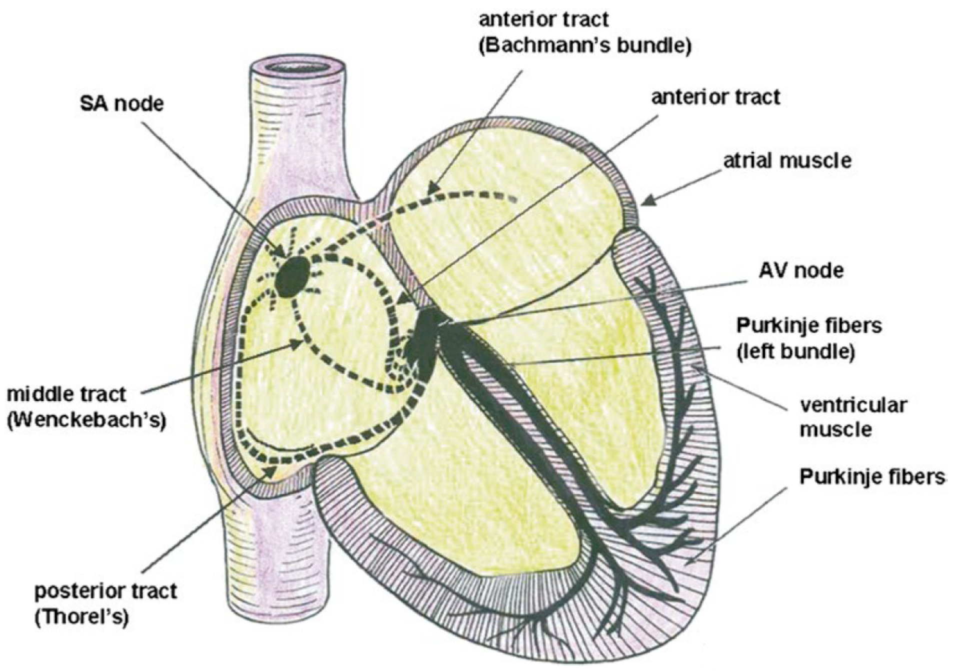

Figure 2.2: Cardiac Signal Conduction (Fig. 13.1 in [10]). The bundle of His is not labeled.

In a healthy human the heart rhythm is regular and the resting heart rate is typically between 60-100 bpm. People suffering from *heart arrhythmia* have a regular or irregular resting heart rate that is too slow (<60 bpm) or too high (>100 bpm). The former is called *bradycardia* and the latter *tachycardia*. The reasons for a given case of heart arrhythmia can be very diverse but typically they can be grouped into one of the following two categories:

(i) Problems with electrical signal initiation

(ii) Problems with electrical signal conduction

An example for (i) would be an atypical signal initiation frequency in the SA node and an example for (ii) would be improper signal conduction through

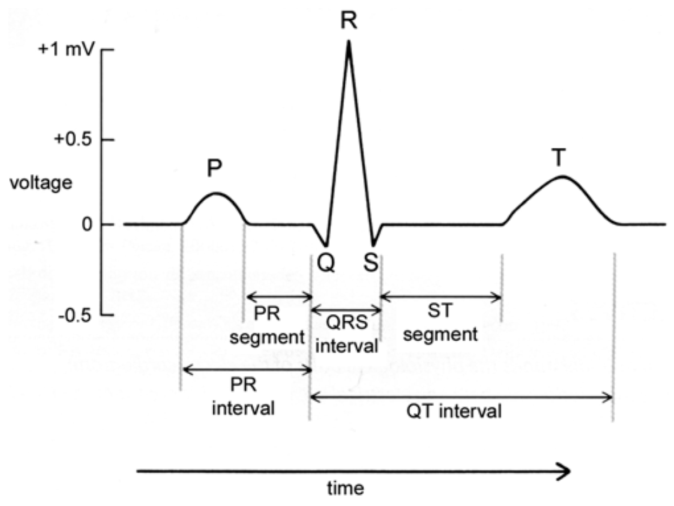

Figure 2.3: Typical waveform of an Electrocardiogram (Fig. 19.5 in [10]). The P wave, QRS complex and T wave correspond to atrial depolarization, ventricular depolarization and ventricular repolarization, respectively. Atrial repolarization is masked by the QRS complex.

the AV node. Kehrle's work focuses on specific tachycardias, namely *atrial fibrillation* (AFib) and *atrial flutter* (AFlut). AFib is characterized by an *irregular atrial rate* of 350-600 bpm and AFlut by a *regular* atrial rate of 220-300 bpm [11]. Atrial rate refers to the number of contractions of the atria per minute as opposed to the *ventricular rate* which refers to the number of contractions of the ventricles per minute. Typically it is the latter one which is referred to when speaking of "heart rate".

Electrocardiograms (ECGs) can reveal information about whether the above described cardiac signal conduction is working properly. An ECG measures the change of electrical activity of the heart. The electrical signal propagating through the heart creates changes in the surrounding electrical field which can be detected using electrodes that are placed on the skin of a person. The result of this measurement is recorded as a diagram, the ECG, which plots voltage versus time and shows waveforms that correspond to the de- and repolarization of the atria and ventricles (Figure 2.3). Particular intervals between specific waves are often referred to by the combination of the respective letters that specify the waves, e.g. the interval between the P- and R-wave is referred to as the PR-interval and the interval between two consecutive R waves as the RR-interval.

2.1.2 Mathematical Modeling

There are many aspects of the heart that can in principle be modeled, such as fluid dynamics of the blood, the muscle contractions, etc. This thesis is concerned with heart arrhythmias and thus uses a particular model of electrical signal conduction to inform the machine learning approach used. This model was developed by Florian Kehrle [8] and is a model of the atrioventricular dynamics of the heart. More precisely, it is concerned with how blocks in the AV node can influence the signal conduction from atria to ventricles and lead to various ventricular signal patterns in the case of a regular atrial rate. The AV node, as mentioned before, is a set of cells through which a signal from the atria has to travel to eventually reach the ventricles and cause a full contraction of the heart. Signals can get blocked when passing through the AV node. This can occur in up to three different locations or levels which is referred to as a *Multilevel AV Block* (MAVB). This blocking can lead to multiple different ratios of incoming signals to conducted signals $m : n$. One such set of signals that is viewed within the context of a certain block ratio is called a block cycle. Figure 2.4 illustrates these concepts and will serve as a helpful intuition for the rest of the thesis. The model in Kehrle's thesis simulates the signal conduction by mapping an input consisting of the following variables:

- 1-3 block pattern lists
- atrial cycle length (AA)
- conduction constant (α)

to a list of RR-intervals. AA represents the time interval between two consecutive atrial excitations (vertical lines at the top of Figure 2.4) in case of a regular atrial rate and α represents the regular signal conduction time in case of a normal sinus rhythm. Both of these constants are defined as integers representing millisecond units. The block pattern lists contain integers $n \in [0, 7]$ that represent block ratios of the form $n+1 : n$ where $n+1$ indicates the number of signals arriving and n is the number of signals passed through.

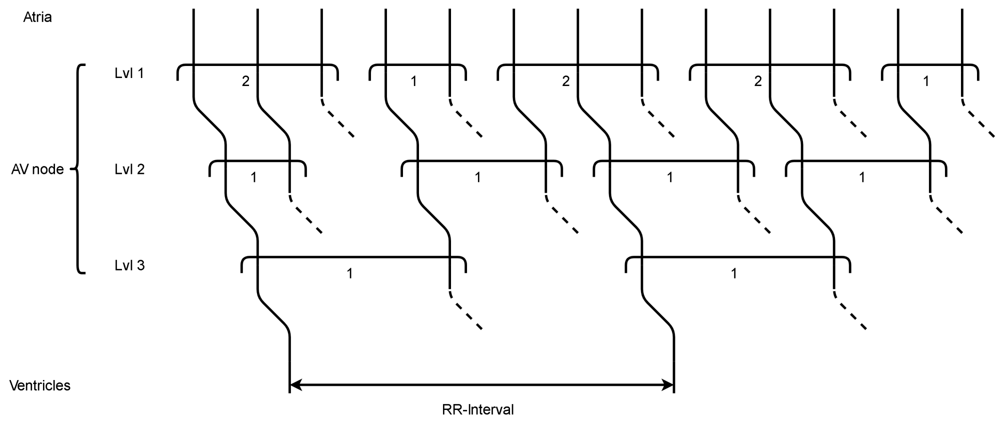

Figure 2.4: Signal diagram. The figure shows a simplified diagram of the signal conduction from atria through the AV node levels to the ventricles (legend on the left). Conducted signals are depicted with continuous lines, whereas blocked signals are shown as dashed lines. Each horizontal round bracket over a group of signals shows one block cycle and the number below a bracket shows the number of conducted signals in that block cycle. In this specific case 11 of the 13 signals arriving at the first level of the AV node were blocked over the three levels of the AV node leading to two signals reaching the ventricles and being registered as R-waves on the ECG. The interval between them is shown with a bidirectional arrow.

It is always the last signal that gets blocked. One exception is 0 which stands for a signal conduction ratio of 1 : 1, i.e. no signal is blocked. The numbers within the lists are ordered chronologically, i.e. they represent time sequences of block ratios. All of the input variables were constrained based on Kehrle's thesis and the Python implementation provided by Felix Bernhardt. AA is constrained to the interval $[188, 400]$ and α to the intveral $[1, AA]$. The sequence of ratios are constrained to fall into one of the five different MAVB types (Figure 2.5). In addition the difference between two successive integers in the block pattern lists must be smaller or equal to 3. This means that a block ratio of $n + 1 : n$ can only change with $n \pm 3$. The number of lists is variable because Bernhardt's Python implementation of Kehrle's model has one function for each MAVB type. MAVB types 1, 2a and 2b take only one list as level 2 is the only one that is variable. MAVB type 2c takes two inputs for level 1 and 2 and MAVB type 3 has the additional level 3 input list. For

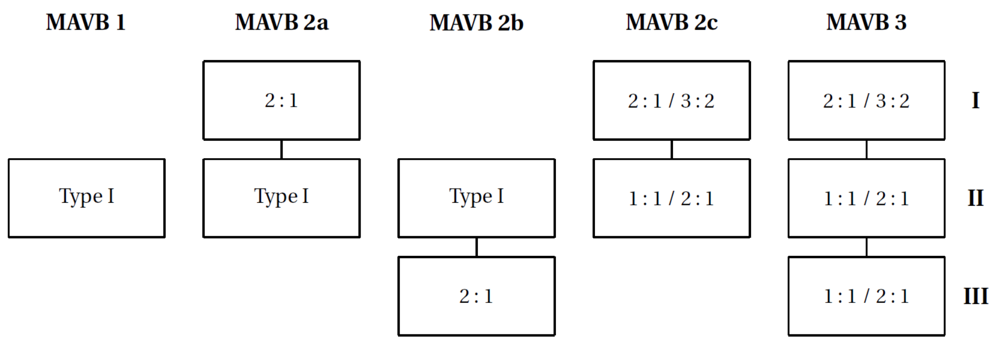

Figure 2.5: The different Multilevel AV Block types (Fig. 4.3 in [8]). The roman numerals on the right represent the three different levels within the AV node where a block can occur. The different block ratios that can occur within one specific level are written in the rectangles where "Type 1" refers to a $n + 1 : n$ block. A misssing rectangle represents no blocking, i.e. a block ratio of $1 : 1$.

this thesis the model will be exclusively used as a way to generate training and testing data for the statistical models. In Kehrle's thesis, the model was developed as part of the software package called *Heidelberg Electrocardiogram Analysis Tool* (HEAT) in which it was used to solve the inverse problem of suggesting a possible explanation for a given ECG measurment (RR-intervals) in terms of the input variables given above. This was used to discriminate between AFib and AFlut (see Subsection 2.1.1 for distinction). Since Kehrle's model, as described above, assumes a regular atrial rate, it can be said that if a given ECG measurement is approximated closely enough with Kehrle's model it probably is caused by AFlut and not AFib. An optimal *cut-off* value was calculated to reach the best possible performance in the discrimination task between AFib and AFlut. The statistical models developed in this thesis will be applied solely to the data generated from Kehrle's model, not real world ECG data, i.e. the data only contains RR-intervals that result from regular atrial rates (AFlut).

2.2 Recurrent Neural Networks

This section gives a brief overview of neural networks in general, based largely on [12], and describes the basic concept of recurrent neural networks as well as two of the most famous and widely used extensions.

2.2.1 Neural Networks

Neural Networks (NNs) are universal function approximators that are composed of linear and non-linear functions [13][14][15]. They consist of *neurons* or *nodes* that are loosely based on their biological counterpart. These neurons have an input as well as an output which is typically calculated by a linear transformation of the input followed by a non-linear function, the so called *activation function*. The neurons are connected to each other in some specific manner, depending on the particular network architecture, by *edges* that represent which neuron's output will be used in a given neuron's input. Furthermore, the neurons are grouped into *layers*, where the first layer is the *input layer* which is simply the input data, the last layer is the *output layer* that has an activation function constraining the output values to a range specific to a given task and the layers in between are called *hidden layers*. A simple example for a NN would be a chain of nodes with no cyclic connections, called a *feed forward* NN. To describe such a network in a mathematical way let f be the NN and $\boldsymbol{x} \in \mathbb{R}^n$ an input vector representing some data. Then the whole network can be written as a function composed of subfunctions representing the results in every layer

$$f(\boldsymbol{x}; \theta) = f_3(f_2(f_1(\boldsymbol{x})))$$

, where in this case we have 3 layers (input layer is not counted) and θ represents all the parameters of the network, i.e. all parameters of all functions involved. In the general case of a feed forward NN with n layers the functions, i.e. the outputs at each layer, are given by

$$f_i(\boldsymbol{h}_{i-1}) = \phi_i\left(W_{h_i h_{i-1}} \boldsymbol{h}_{i-1}\right)$$

with $i = 1, ..., n$, $h_0 = \boldsymbol{x}$ and $h_n = \boldsymbol{y}$. h_i is the i-th *hidden layer*, $W_{h_i h_{i-1}}$ is the *weight matrix* which linearly transforms the output of the previous layer and ϕ is the activation function of the i-th layer. Typical choices for ϕ are the *sigmoid* function, the *tanh* and the rectified linear unit (*ReLU*) which is defined as $max(0, x)$. The neurons mentioned above are the entries of the output vectors in the layers. The computation in a very simple feed forward NN with only two layers is depicted in figure 2.6.

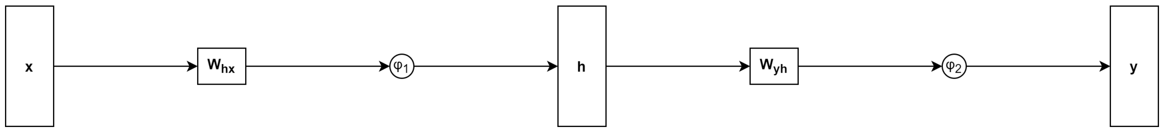

Figure 2.6: Feed forward NN. The circles represent the non-linear activation functions mentioned in the text applied point-wise to the input vector of the given layer. The number of neurons in the layers, i.e. the number of dimensions of $\boldsymbol{x}$, $\boldsymbol{h}$ and $\boldsymbol{y}$, can be arbitrary.

2.2.2 Vanilla Recurrent Neural Networks

Recurrent neural networks (RNNs) are neural networks that are specialized for sequential inputs [16]. They consist of *cells* that read the input sequentially, update their so called *hidden state* and allow for sequential output. Let $\boldsymbol{x}^{(t)}$ be an input vector at time step t. This "time step" is just a positional index in a sequence of inputs and does not necesseraly correspond to time in the real world. The RNN forward pass can then be described by

$$\begin{aligned}\boldsymbol{h}^{(t)} &= f_h(\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}) \\ \boldsymbol{y}^{(t)} &= f_o(\boldsymbol{h}^{(t)})\end{aligned}$$

, where $\boldsymbol{h}^{(t-1)}$ is the hidden state at time step $t-1$ and f_h and f_o are functions that output the new hidden state $\boldsymbol{h}^{(t)}$ and an optional output $\boldsymbol{y}^{(t)}$ at that time step, respectively [17]. The basic (vanilla) version of a RNN cell is characterized by the following equations for the forward pass [16]

$$\begin{aligned}\boldsymbol{h}^{(t)} &= \phi_h\left(W_{hx}\boldsymbol{x}^{(t)} + W_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h\right) \\ \boldsymbol{y}^{(t)} &= \phi_o\left(W_{yh}\boldsymbol{h}^{(t)} + \boldsymbol{b}_y\right)\end{aligned}$$

, where ϕ_h and ϕ_o are element-wise non-linear functions and W_{hh}, W_{hx} and W_{yh} are weight matrices for the linear transformations of the current hidden state $\boldsymbol{h}^{(t-1)}$, the current input $\boldsymbol{x}^{(t)}$ and the computation of the new output $\boldsymbol{y}^{(t)}$ from the new hidden state $\boldsymbol{h}^{(t)}$, respectively. $\boldsymbol{b}_h$ and $\boldsymbol{b}_y$ are bias vectors. The weight matrices are constant across all time steps. An illustration of the computation during one time step of a cell is provided in Figure 2.7.

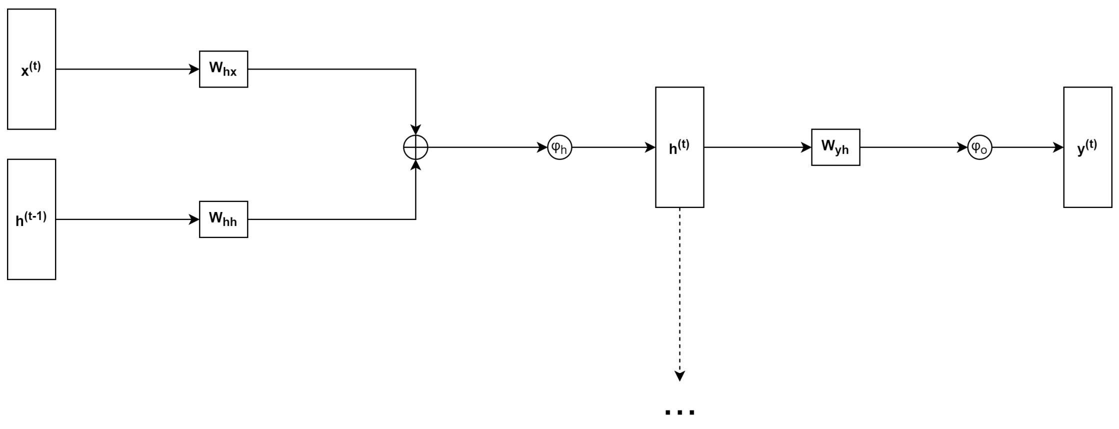

Figure 2.7: Vanilla RNN cell. The circles represent point-wise operations explained in the text and the dashed arrow pointing down represents the fact that the computed new hidden state $\boldsymbol{h}^{(t)}$ will be used in the next time step of the cell. The bias vectors are left out of the figure to avoid clutter.

2.2.3 Long Short-Term Memory Networks and Gated Recurrent Units

Long Short-Term Memory (LSTM) networks are an extension to the basic RNN design described in Subsection 2.2.2. This variant enables the RNN to learn dependencies between sequence elements that are longer i.e. they span more time steps. The difference to vanilla RNNs lies in the particular operations performed in one cell of an LSTM. The forward pass is characterized by the following equations [16]

$$\begin{aligned}
\boldsymbol{g}^{(t)} &= \phi(W_{gx}\boldsymbol{x}^{(t)} + W_{gh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_g) \\
\boldsymbol{i}^{(t)} &= \sigma(W_{ix}\boldsymbol{x}^{(t)} + W_{ih}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i) \\
\boldsymbol{f}^{(t)} &= \sigma(W_{fx}\boldsymbol{x}^{(t)} + W_{fh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f) \\
\boldsymbol{o}^{(t)} &= \sigma(W_{ox}\boldsymbol{x}^{(t)} + W_{oh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o) \\
\boldsymbol{s}^{(t)} &= \boldsymbol{g}^{(t)} \otimes \boldsymbol{i}^{(t)} + \boldsymbol{s}^{(t-1)} \otimes \boldsymbol{f}^{(t)} \\
\boldsymbol{h}^{(t)} &= \phi(\boldsymbol{s}^{(t)}) \otimes \boldsymbol{o}^{(t)}
\end{aligned}$$

These equations introduce the new idea of an *internal state s* which is only altered through linear transformations during a time step. They also describe how the information that was present in the internal state $\boldsymbol{s}^{(t-1)}$ of the memory cell is transformed based on current input information $\boldsymbol{x}^{(t)}$ and the previous hidden state $\boldsymbol{h}^{(t-1)}$. The *forget gate* $\boldsymbol{f}^{(t)}$ controls how much information of the old internal state $\boldsymbol{s}^{(t-1)}$ should be forgotten and the input gate $\boldsymbol{i}^{(t)}$ how much new information $\boldsymbol{g}^{(t)}$ should be incorporated into the old internal state. The output gate $\boldsymbol{o}^{(t)}$ controls the information flow from the updated internal state $\boldsymbol{s}^{(t)}$ to the new hidden state $\boldsymbol{h}^{(t)}$ based on the current input information $\boldsymbol{x}^{(t)}$ and the previous hidden state $\boldsymbol{h}^{(t-1)}$. As before, the $\boldsymbol{b}$ vectors are bias vectors. ϕ is an element-wise tanh function and σ is an element-wise sigmoid function.

There are many variations on this LSTM design. One famous variation is the so called *Gated Recurrent Unit* (GRU). It represents a simplification of the standard LSTM model. The following equations characterize the forward pass [18]

$$\begin{aligned}
\boldsymbol{r}^{(t)} &= \sigma(W_{rx}\boldsymbol{x}^{(t)} + W_{rh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_r) \\
\boldsymbol{z}^{(t)} &= \sigma(W_{zx}\boldsymbol{x}^{(t)} + W_{zh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_z) \\
\tilde{\boldsymbol{h}}^{(t)} &= \phi(W_{\tilde{h}x}\boldsymbol{x}^{(t)} + W(\boldsymbol{r}^{(t)} \otimes \boldsymbol{h}^{(t-1)})) \\
\boldsymbol{h}^{(t)} &= \boldsymbol{z}^{(t)} \otimes \boldsymbol{h}^{(t-1)} + (1 - \boldsymbol{z}^{(t)}) \otimes \tilde{\boldsymbol{h}}^{(t)}
\end{aligned}$$

The forward and input gate got replaced by an *update* gate $\boldsymbol{z}^{(t)}$ and the $\tilde{\boldsymbol{h}}^{(t)}$ represents the current candidate values for the new hidden state $\boldsymbol{h}^{(t)}$. How much information flows from the old hidden state $\boldsymbol{h}^{(t-1)}$ to the candidate values $\tilde{\boldsymbol{h}}^{(t)}$ is controlled by the *reset* gate $\boldsymbol{r}^{(t)}$. Furthermore, there is no distinction being made between internal state and hidden state.

2.3 Flow-based Generative Models

This section starts by giving a brief description of generative modeling with two famous example approaches. Then, the concept of flow-based generative models with a few extensions based on the initial basic design will be presented in greater detail.

2.3.1 Generative Modeling

Generative models learn an approximation of the probability distribution of the training data in order to generate new data that resembles the training data to some degree. Let $\boldsymbol{x}$ be a vector with arbitrary many random variables that represent the training data with the joint probability distribution p. The goal of generative models is to find an approximation $\hat{p}$ of p such that

$$\hat{p}(\boldsymbol{x}; \theta) \approx p(\boldsymbol{x})$$

for any $\boldsymbol{x}$ from the training data. θ represents the parameters of the model. There are several existing generative models each with a different approach. Two very famous approaches are Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN). VAEs consist of an encoder network that maps the input data to a lower dimensional latent space and a decoder network that maps the latent variables back to the input data space [19]. Let $\boldsymbol{z}$ be a vector of the random variables of the latent space. Then

$$\hat{p}(\boldsymbol{z}|\boldsymbol{x}; \phi) \approx p(\boldsymbol{z}|\boldsymbol{x})$$
$$\hat{p}(\boldsymbol{x}|\boldsymbol{z}; \theta) \approx p(\boldsymbol{x}|\boldsymbol{z})$$

describes the approximation that VAEs try to learn, where the encoder is parameterized by ϕ and the decoder is parameterized by θ. The generative model as a whole approximates the joint distribution $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{z}; \theta)p(\boldsymbol{x}|\boldsymbol{z}; \theta)$ with a suitable prior $p(\boldsymbol{z}; \theta)$ e.g. a Gaussian. VAEs are trained by maximizing the so called *evidence lower bound* (ELBO) on the marginal likelihood of the data which in practice often corresponds to minimizing the negative version:

$$-D_{KL}(\hat{p}(\boldsymbol{z}|\boldsymbol{x}; \phi)||p(\boldsymbol{z}; \theta)) + \mathbb{E}_{\hat{p}(\boldsymbol{z}|\boldsymbol{x}; \phi)}\Big[\log \hat{p}(\boldsymbol{x}|\boldsymbol{z}; \theta)\Big]$$

, where D_{KL} is the Kullback–Leibler divergence. GANs on the other hand consist of a generator network G that generates new data and a discriminator network D that discriminates between real and fake data [20]. More precisely, $G(\boldsymbol{z}; \theta_g)$ maps input noise variables $\boldsymbol{z}$ with pre-defined prior p_z to the data space and $D(\boldsymbol{x}, \theta_d)$ maps that generated data to a scalar that represents the probability that that $\boldsymbol{x}$ is true, i.e. comes from the true probability distribution of the data p rather than from the probability distribution of the generated data p_g. The network D is trained to correctly discriminate between real and fake data, i.e. to maximize the probability of assigning the right label to real and fake data while the network G is trained to minimize $\log(1 - D(G(\boldsymbol{z})))$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p(x)} [\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})} [\log(1 - D(G(\boldsymbol{z})))]$$

Flow-based generative models are yet another approach to approximating the probability distribution of data and have been gaining popularity in the last few years with more research exploring their potential applications. The following subsections provide a detailed description of that approach.

2.3.2 Normalizing Flow

Normalizing flow constitutes the core mathematical concept underlying flow-based generative models. Let $X_0 \in \mathbb{R}^d$ be a random variable and $f_i : \mathbb{R}^d \to \mathbb{R}^d, \ i = 1, .., k$ a sequence of invertible smooth transformations. Furthermore, let $p(X_0) = p_0$ be the probability distribution of X_0. Then, the successive application of the transformations f_i to X_0

$$X_k = f_k \circ ... \circ f_2 \circ f_1(X_0)$$

yields a sequence of random variables $X_0, ..., X_k$, called *flow*, and thus a sequence of associated probability distributions $p_0, ..., p_k$, called *normalizing flow*[2]. The change-of-variables formula describes how the probability distributions are related to each other:

$$p_i(X_i) = p_{i+1}(X_{i+1}) \left| \det \frac{\partial f_{i+1}}{\partial X_i} \right|$$

, where $\det \frac{\partial f_{i+1}}{\partial X_i}$ is the Jacobian determinant. This allows for the transformation of a simple, well understood probability distribution to an arbitrarily complex one and of course the other way around as well due to the invertibility of the transformation sequence. It follows that for the whole transformation sequence we have

$$p_0(X_0) = p_k(X_k) \prod_{i=0}^{k} \left| \det \frac{\partial f_{i+1}}{\partial X_i} \right|$$
$$\log(p_0(X_0)) = \log(p_k(X_k)) + \sum_{i=0}^{k} \log\left(\left| \det \frac{\partial f_{i+1}}{\partial X_i} \right| \right)$$

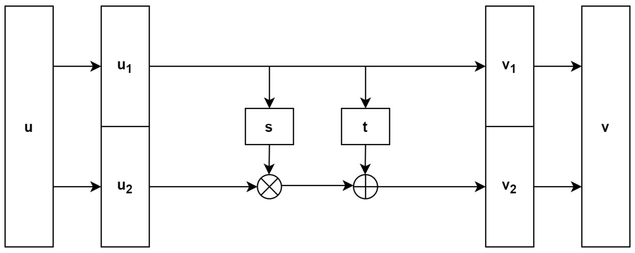

Figure 2.8: Affine coupling block. s and t represent arbitrary neural networks (called subnetworks) and the circles represent point-wise operations explained in the text.

Flow-based generative models utilize this fact by learning an invertible transformation from data space to a latent space with a probability distribution from which one can easily draw samples and generate new data by running the transformations backwards. The choice for the tranformations f_i is very important since this dictates both how tractable the Jacobian determinant is and how efficient the calculation of the transformation sequence is. In order to simplify the calculations one can choose transformations based on how the associated Jacobian matrix is structured. For example the determinant of a triangular matrix can be easily computed by multiplying its diagonal elements. This is the core idea behind flow-based models such as *real NVP* (rNVP) [3]. rNVP uses an affine transformation as the basic building block of the model (Figure 2.8). Consider $\boldsymbol{u} \in \mathbb{R}^D$ as an input vector that is split into two parts $\boldsymbol{u}_1 \in \mathbb{R}^d$ and $\boldsymbol{u}_2 \in \mathbb{R}^{D-d}$. Then the transformation is given by

$$\begin{aligned} \boldsymbol{v}_1 &= \boldsymbol{u}_1 \\ \boldsymbol{v}_2 &= \boldsymbol{u}_2 \otimes \exp(s(\boldsymbol{u}_1)) + t(\boldsymbol{u}_1) \end{aligned}$$

, where $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ constitute the two parts of the output vector. This transformation can easily be inverted

$$\begin{aligned} \boldsymbol{u}_1 &= \boldsymbol{v}_1 \\ \boldsymbol{u}_2 &= (\boldsymbol{v}_2 - t(\boldsymbol{u}_1)) \oslash \exp(s(\boldsymbol{u}_1)) \end{aligned}$$

and has the following Jacobian

$$\begin{pmatrix} I_d & 0 \\ \frac{\partial \boldsymbol{v}_1}{\partial \boldsymbol{u}_1} & \operatorname{diag}(\exp(s(\boldsymbol{u}_1))) \end{pmatrix}$$

The determinant is thus efficiently computed as the product over the entries of the vector $\exp(s(\boldsymbol{u}_1))$. Furthermore, since the inverse does not require the inversion of the s and t networks, they can be arbitrarily complex. In rNVP multiple affine coupling blocks are used in an alternating fashion to ensure that input dimensions that remained unaltered through one coupling block are altered in the next coupling block. The full model involves more intricate design choices regarding the splitting of the input vector and the overall structure of the network which are not further discussed here. A more recent variation on this flow design is called *Glow* [5]. In Glow the coupling block is very similar to the affine coupling block discussed above except for the fact that only a single neural network is used to jointly compute the coefficients for the affine transformation:

$$\begin{aligned}\boldsymbol{v}_1 &= \boldsymbol{u}_1 \\ \boldsymbol{v}_2 &= \boldsymbol{u}_2 \otimes \exp(q_s(\boldsymbol{u}_1)) + q_t(\boldsymbol{u}_1)\end{aligned}$$

, where q_s and q_t represent the respective parts of the network output corresponding to the scaling and shifting coefficients. This reduces the computational cost and can speed up training of the model. Furthermore Glow uses invertible 1x1 convolutions to permute the data between the coupling blocks and similar to rNVP a multi-scale architecture is used which will not be further discussed here. The 1x1 convolutions can be helpful with data that has a 2D or 3D layout, such as images, but is of no relevance for the medical data with which this thesis is concerned.

2.3.3 Invertible Neural Networks for Inverse Problems

From this subsection onward the term *Invertible Neural Network* (INN) will be used according to the definiton given in [6]:

(i) The mapping from inputs to outputs is bijective, i.e. its inverse exists.

(ii) Both forward and inverse mapping are efficiently computable.

(iii) Both mappings have a tractable Jacobian, which allows explicit computation of posterior probabilities

In other words they are flow-based generative models that use specific transformations, such as those in rNVP or Glow, to achieve the mentioned criteria. In their paper Ardizzone et al [6] present a way of using INNs to solve

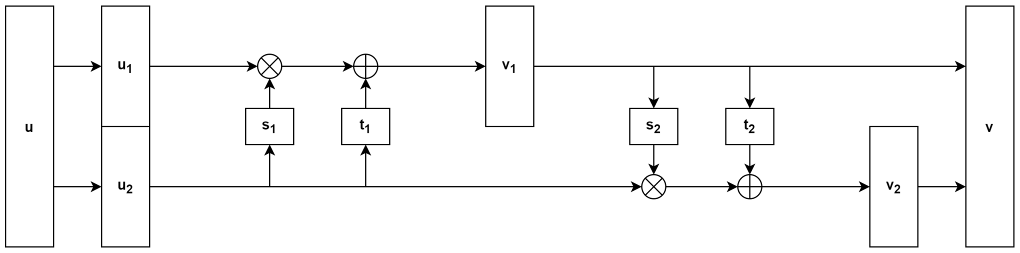

Figure 2.9: INN affine coupling block. s_1, s_2 and t_1, t_2 represent arbitrary neural networks and the circles represent point-wise operations explained in the text.

inverse problems in the natural and life sciences. More precisely, given $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{y} \in \mathbb{R}^M$ which represent quantities of some kind of natural process for which there is an existing model s that approximates the non-injective forward mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$, they aim to model the full posterior $p(\boldsymbol{x}|\boldsymbol{y})$ of the ambiguous backward mapping from $\boldsymbol{y}$ to $\boldsymbol{x}$. This can be understood as a conditional generative modeling problem. The network uses coupling blocks that represent a fusion of two complementary affine coupling blocks (Figure 2.9) as they were discussed in Subsection 2.3.2 together with randomized but fixed permutations that shuffle the vector content between coupling blocks around. The transformation function of the coupling blocks is given by

$$\begin{aligned}\boldsymbol{v}_1 &= \boldsymbol{u}_1 \otimes \exp(s_1(\boldsymbol{u}_2)) + t_1(\boldsymbol{u}_2)\\ \boldsymbol{v}_2 &= \boldsymbol{u}_2 \otimes \exp(s_2(\boldsymbol{v}_1)) + t_2(\boldsymbol{v}_1)\end{aligned}$$

with the inverse

$$\begin{aligned}\boldsymbol{u}_2 &= (\boldsymbol{v}_2 - t_2(\boldsymbol{v}_1)) \oslash \exp(s_2(\boldsymbol{v}_1))\\ \boldsymbol{u}_1 &= (\boldsymbol{v}_1 - t_1(\boldsymbol{u}_2)) \oslash \exp(s_1(\boldsymbol{u}_2))\end{aligned}$$

The network approximates and learns the forward and backward mapping jointly

$$\begin{aligned}[\boldsymbol{y}, \boldsymbol{z}] &= f(\boldsymbol{x}; \theta)\\ \boldsymbol{x} &= g(\boldsymbol{y}, \boldsymbol{z}; \theta)\end{aligned}$$

, where $\boldsymbol{z} \in \mathbb{R}^K$ is a latent variable that is supposed to capture information lost during the forward mapping. This latent variable is chosen to follow a probability distribution, e.g. Gaussian, from which one can sample easily in order to generate data through the backward mapping. The network thus outputs the condition variable $\boldsymbol{y}$ and the latent variable $\boldsymbol{z}$. The dimensions of input and output to the network have to match for the bijectivity to hold and so input or output are zero padded if it is required. They train the network using three losses

$$\begin{aligned} \mathcal{L}_y(\boldsymbol{y}, f_y(\boldsymbol{x}; \theta)) \\ \mathcal{L}_z(q(\boldsymbol{y}, \boldsymbol{z}), p(\boldsymbol{y})p(\boldsymbol{z})) \\ \mathcal{L}_x(p(\boldsymbol{x}), q(\boldsymbol{x})) \end{aligned}$$

, where $\mathcal{L}_y$ is a supervised loss comparing the y-part of the network forward output to the true $\boldsymbol{y}$ from the known model $s(\boldsymbol{x}) = \boldsymbol{y}$ (e.g. L2-loss), $\mathcal{L}_z$ is a divergence measure (e.g. Maximum Mean Discrepancy) that measures the mismatch between the approximated joint distribution $q(\boldsymbol{y}, \boldsymbol{z})$ of the network and the product of the probability distributions of the known model outputs $p(\boldsymbol{y})$ and a prior on the latent variable $p(\boldsymbol{z})$. $\mathcal{L}_x$ is similar to $\mathcal{L}_z$ but compares the probability distribution of the generated data $q(\boldsymbol{x})$ to the prior of the real data $p(\boldsymbol{x})$.

2.3.4 Conditional Invertible Neural Networks

Conditional Invertible Neural Networks (cINNs) are an extension of the INN design described in Subsection 2.3.3. They extend the affine coupling block design of INNs as in figure 2.9 by introducing a conditional input into the subnetworks (Figure 2.10). The transformation function is given by [21]

$$\begin{aligned} \boldsymbol{v}_1 &= \boldsymbol{u}_1 \otimes \exp(s_1(\boldsymbol{u}_2, \boldsymbol{c})) + t_1(\boldsymbol{u}_2, \boldsymbol{c}) \\ \boldsymbol{v}_2 &= \boldsymbol{u}_2 \otimes \exp(s_2(\boldsymbol{v}_1, \boldsymbol{c})) + t_2(\boldsymbol{v}_1, \boldsymbol{c}) \end{aligned}$$

with the inverse

$$\begin{aligned} \boldsymbol{u}_2 &= (\boldsymbol{v}_2 - t_2(\boldsymbol{v}_1, \boldsymbol{c})) \oslash \exp(s_2(\boldsymbol{v}_1, \boldsymbol{c})) \\ \boldsymbol{u}_1 &= (\boldsymbol{v}_1 - t_1(\boldsymbol{u}_2, \boldsymbol{c})) \oslash \exp(s_1(\boldsymbol{u}_2, \boldsymbol{c})) \end{aligned}$$

The conditional input $\boldsymbol{c}$ can either be fed directly into the subnetworks or be pre-processed through for example a neural network. This network can also be jointly trained with the cINN in order to optimize the feature extraction performed by that pre-processing network in the context of the

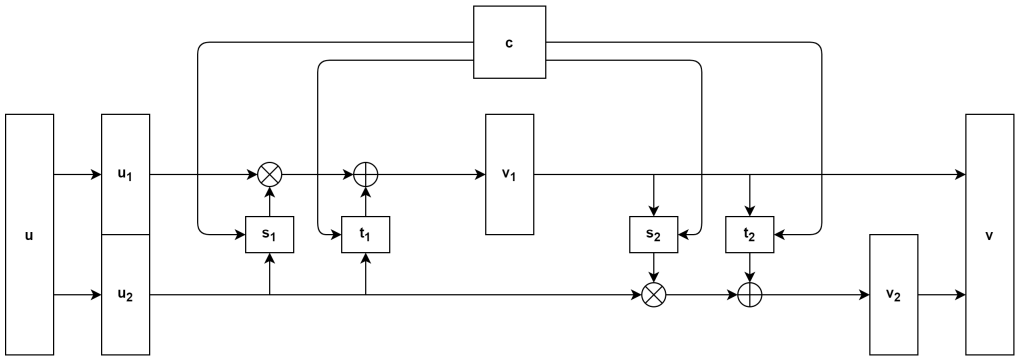

Figure 2.10: cINN affine coupling block. s_1, s_2 and t_1, t_2 represent arbitrary neural networks, the circles represent point-wise operations explained in the text and $\boldsymbol{c}$ represents the conditional variable.

specific generative modeling task at hand. That means that this type of INN only outputs the latent variable $\boldsymbol{z}$ as opposed to the INN in Subsection 2.3.3 which outputs both condition and latent variable. This design makes the input and output of the cINN simpler because it alleviates the need to choose the size of the latent variable and the padding in order to match the input and output dimensions. It also allows for simpler training through maximum likelihood estimation of the model parameters $p(\theta; \boldsymbol{x}, \boldsymbol{c}) \propto p_X(\boldsymbol{x}; \boldsymbol{c}, \theta) \cdot p_\theta(\theta)$ by minimizing the loss

$$\mathcal{L} = \mathbb{E}_i \left[-\log(p_X(\boldsymbol{x}_i; \boldsymbol{c}_i, \theta)) \right] - \log(p_\theta(\theta))$$

The invertible transformation described above transforms the probability distribution of input data X to the probability distribution $p_Z(\boldsymbol{z})$ on latent space Z

$$p_X(\boldsymbol{x}; \boldsymbol{c}, \theta) = p_Z(f(\boldsymbol{x}; \boldsymbol{c}, \theta)) \left| \det \left(\frac{\partial f}{\partial x} \right) \right|$$

By inserting this equation into the likelihood function and assuming a standard normal distribution for $p_Z(\boldsymbol{z})$ and a Gaussian prior for the weights θ with $\frac{1}{2\sigma_\theta^2} \equiv \tau$ one obtains

$$\mathcal{L} = \mathbb{E}_i \left[\frac{||f(\boldsymbol{x}_i; \boldsymbol{c}_i, \theta)||_2^2}{2} - \log|J_i| \right] + \tau ||\theta||_2^2$$

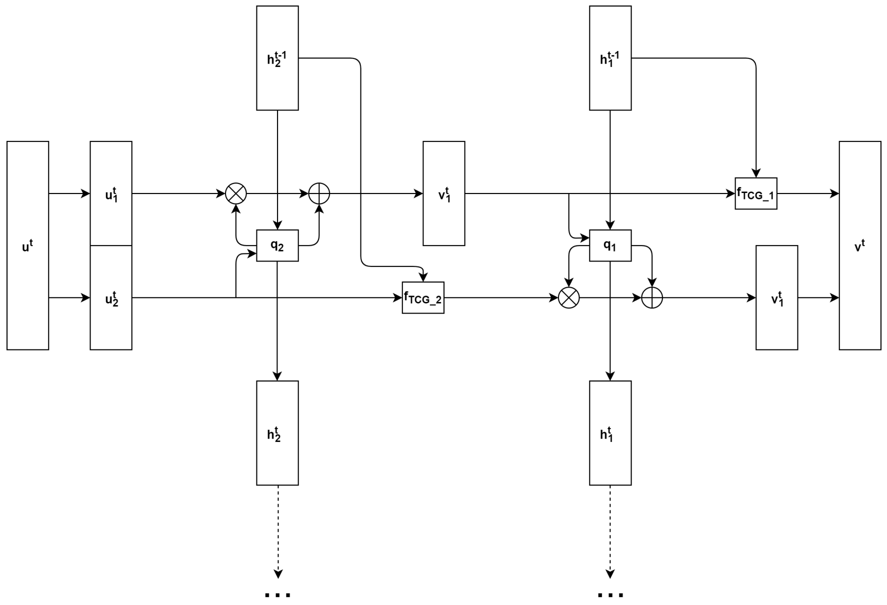

Figure 2.11: CRow coupling block. q_1 and q_2 represent arbitrary RNNs with hidden state h_1 and h_2, respectively. The circles represent point-wise operations explained in the text and the dashed arrows pointing down represent the fact that the computed new hidden states h_1^t and h_2^t will be used in the next time step.

with i indexing the i-th training sample. In their paper they used the cINN to generate images and it was found that this architecture allows for the generation of diverse and sharp images and does not experience mode collapse.

2.3.5 Conditional Recurrent Flow

Conditional Recurrent Flow (CRow) [7] represents the fusion of RNNs with INNs as they were described in Subsection 2.3.3. The aim here is to utilize the generative power of INNs in the context of sequential data, in this case neuroimaging data. This is achieved by using RNNs, in this case a GRU, as the subnetworks of a coupling block and by the inclusion of a temporal gate f_{TCG} that regulates information flow based on past information (Figure 2.11). The following equations describe the forward pass of one coupling block at one time step

$$\begin{aligned}
\boldsymbol{v}_1^t &= f_{TCG_1}\left(\boldsymbol{u}_1^t \otimes \exp(q_{s_2}(\boldsymbol{u}_2^t, \boldsymbol{h}_2^{t-1})) + q_{t_2}(\boldsymbol{u}_2^t, \boldsymbol{h}_2^{t-1}), \boldsymbol{h}_1^{t-1}\right) \\
\boldsymbol{v}_2^t &= f_{TCG_2}\left(\boldsymbol{u}_2^t, \boldsymbol{h}_2^{t-1}\right) \otimes \exp(q_{s_1}(\boldsymbol{v}_1^t, \boldsymbol{h}_1^{t-1})) + q_{t_1}(\boldsymbol{v}_1^t, \boldsymbol{h}_1^{t-1})
\end{aligned}$$

with the inverse

$$\begin{aligned}
\boldsymbol{u}_2^t &= f_{TCG_2}^{-1}\left((\boldsymbol{v}_2^t - q_{t_1}(\boldsymbol{v}_1^t, \boldsymbol{h}_1^{t-1})) \oslash \exp(q_{s_1}(\boldsymbol{v}_1^t, \boldsymbol{h}_1^{t-1})), \boldsymbol{h}_2^{t-1}\right) \\
\boldsymbol{u}_1^t &= \left(f_{TCG_1}^{-1}\left(\boldsymbol{v}_1^t, \boldsymbol{h}_1^{t-1}\right) - q_{t_2}(\boldsymbol{u}_2^t, \boldsymbol{h}_2^{t-1})\right) \oslash \exp(q_{s_2}(\boldsymbol{u}_2^t, \boldsymbol{h}_2^{t-1}))
\end{aligned}$$

, where

$$\begin{aligned}
f_{TCG}(\alpha^t, \boldsymbol{h}^{t-1}) &= \alpha^t \otimes \mathrm{cgate}(\boldsymbol{h}^{t-1}) \\
f_{TCG}^{-1}(\alpha^t, \boldsymbol{h}^{t-1}) &= \alpha^t \oslash \mathrm{cgate}(\boldsymbol{h}^{t-1})
\end{aligned}$$

and cgate being any learnable function with a sigmoid at the end. The subnetworks q_1 and q_2 are RNNs and like with Glow they output both scaling $\boldsymbol{s}$ and shifting $\boldsymbol{t}$ coefficients of the affine transformation. The respective part of the network output is indicated by a subscript. $\boldsymbol{h}^t$ represents the hidden state of a RNN at time step t. They train the network using the same three losses $\mathcal{L}_y$, $\mathcal{L}_z$ and $\mathcal{L}_x$ discussed in Subsection 2.3.3.

Chapter 3

Methodology

This chapter states the central objective of this thesis. After that it describes the data generation process and ends with a detailed description of the different models developed and implemented in this thesis.

3.1 Objective

The objective of this thesis is to use INNs to construct a statistical model that allows for sampling of input variables for the aforementioned atrioventricular dynamics model of Florian Kehrle for a given list of RR-intervals. The idea is to not only find the best possible solution but also to find multiple, possibly quite different, alternative solutions if they exist. Multiple different approaches will be tested against each other to see which approach works best. In the following Kehrle's model will be referred to as the *forward model* because it maps the aforementioned input variables to RR-intervals and the statistical models developed in this thesis attempt to solve the inverse problem stated above. Furthermore, this keeps the terminology in line with the terminology of the original INN paper [6].

3.2 Data

All data used for training and testing the models is created under the constraints mentioned in Subsection 2.1.2. The following steps outline how the input variables for the forward model as well as the corresponding RR-intervals are generated:

1. initialize $AA \in [188, 400]$ and $\alpha \in [1, AA]$

2. initialize the desired number of R-waves $n\_Rwaves \in [6, 25]$

3. randomly choose MAVB type (1, 2a, 2b, 2c or 3)

4. under the constraints of the chosen MAVB type construct a random array of integers for Lvl 3, Lvl 2 and Lvl 1 in that order, such that the number of conducted signals at Level 3 is equal to the desired amount of R-waves and no block ratio entry is unused

5. pass the block patterns, AA and α through the forward model to get the corresponding RR-intervals and use $n\_Rwaves - 1$ of those intervals for the conditional variable

The constraint on the number of R-waves comes from Kehrle's thesis in which he constrained the ECGs that were considered for the above mentioned inverse problem to a sector of 6 to 25 R-waves. All approaches use this generated data as the basis. The fourth step of the procedure above is the most complex and important for the results of the thesis because the process of generating the block pattern lists influences the probability distribution of the data and so should be explained further. The reason for generating the data with three lists, i.e. block ratios for all levels, regardless of the particular MAVB type, is to have a unified basis of data for all approaches. This ensures that all approaches work with a ground base of data that has the same probability distribution which makes the approaches more comparable. This also makes it easier to use the data for approaches that require more complicated transformations (see Section 3.3). The rest of this section explains the block pattern list generation in more detail.

For the sake of brevity, only the part of the algorithm for MAVB type 1 will be explained. For further details see the pseudocode for the remaining MAVB types in Appendix A. The central idea behind the algorithm, regardless of MAVB type, is that level 3 has to conduct enough signals so that the desired amount of R-waves can be reached or in other words that the desired number of RR-intervals is produced when using the generated block pattern example with the forward model. Level 2 has in turn to provide enough signals so that level 3 can fulfill the above requirement which means that it might have to conduct more signals than level 3 in the case when signals get blocked in level 3. The same holds between level 1 and level 2. Note that in the cases of the MAVB type 1, 2a and 2b it can happen that the forward model produces more RR-intervals than desired because as mentioned in Subsection 2.1.2 the forward model implementation only takes one list as an input for those MAVB types. Given this one list it produces the maximum amount of RR-intervals which results when assuming enough atrial signals so that all $n+1:n$ blocks receive $n+1$ signals. This however is not a problem because step 5 of the data generation procedure above ensures that for the conditional variable only the intended amount of RR-intervals is used. The remaining differences in the respective algorithms for the different MAVB types are caused by the specific block ratio constraints of each type which leads to different integer ranges that can be used to fill the level lists as well as the amount of conditions for a given level that can lead to the fulfillment of the above mentioned requirements.

The part of the algorithm for the MAVB type 1 (Algorithm 3.1) fills the level 1 and level 3 lists purely with 0s (1:1 blocks), since there are no blocks on those levels for this MAVB type. Level 2 is filled with block ratios 2:1 to

Algorithm 3.1: Generate MAVB 1 Block Pattern

```
  Input: $n\_Rwaves$
  Output: The block pattern lists: $lvl\_1$, $lvl\_2$, $lvl\_3$
1 Initialize empty block pattern lists $lvl\_1$, $lvl\_2$, $lvl\_3$
2 fill $lvl\_3$ with 0s such that $len(lvl\_3) = n\_Rwaves$
3 randomly fill $lvl\_2$ with $n \in [1, 7]$ such that
    $sum(lvl\_2) \geq n\_Rwaves$ and $sum(lvl\_2) - lvl\_2$[-1] $< n\_Rwaves$
4 if $sum(lvl\_2) = n\_Rwaves$ then
5     fill $lvl\_1$ with 0s such that $len(lvl\_1) = n\_Rwaves + len(lvl\_2)$
        or $len(lvl\_1) = n\_Rwaves + len(lvl\_2) - 1$
6 if $sum(lvl\_2) > n\_Rwaves$ then
7     fill $lvl\_1$ with 0s such that
        $len(lvl\_1) = n\_Rwaves + len(lvl\_2) - 1$
```

8:7 such that $sum(lvl\_2) \geq n\_Rwaves$. In order to constrain the size of the level 2 block pattern list so that it doesn't become arbitrarily large, the algorithm requires that without the block ratio entry added last in level 2, the sum over level 2 becomes smaller than n_Rwaves. This is also done to make sure that no block receives no signals, i.e. is "unused" as mentionoed in the procedure steps above. The sum over the entries of level 2 is allowed to go above n_Rwaves to include all possible cases of the MAVB types that contain a variable $n+1:n$ block at level 1 for a given desired amount of R-waves. This does not pose a problem because level 1 is set to have just the right amount of 0s, i.e. conducted signals, depending on level 2. In case level 2 can maximally conduct exactly n_Rwaves signals, level 1 can either conduct that amount of signals plus the signals that get blocked in level 2 (one blocked signal per entry in level 2) or omit the last signal because it gets blocked in the last block of level 2 anyway so the number of R-waves does not change. In the case that level 2 can maximally conduct more than n_Rwaves signals, level 1 can only be set to conduct $n\_Rwaves + len(lvl\_2) - 1$ signals, i.e. the desired amount of R-waves plus one extra signal for every block in level 2 except for the last block because this extra signal would be passed through and not get blocked.

3.3 Models

This section explains the seven different approaches that are used in this thesis in an attempt to achieve the above mentioned objective. These approaches will be first split up according to the data transformation of the forward model variables that they require and second according to their INN architecture. The following paragraphs will be titled with an abbreviation for the approach used in order to have a convenient way to referr to them, escpecially when discussing the experiments and results in Chapter 4.

3.3.1 Block Pattern Approaches

The following approaches use the block pattern lists of the input variables directly and only encode or transform them in a way so as to make the variables suitable for neural network training and the particular architecture used.

bp_cINN

The first approach uses a cINN as described in Subsection 2.3.4. For this the entries in the block pattern lists are encoded as one-hot arrays with length 8 and concatenated over all three levels along one dimension to form one long 1-D tensor as an input to the cINN. The two constants AA and α are appended at the end. The total length of the tensor is constrained to 1402. This is can be derived by thinking about the case which would need the most block ratio entries. Consider the maximum amount of R-waves which is 25. Then, in order for level 3 to have the maximum amount of blocks while conducting 25 R-waves it needs 25 2:1 blocks. Level 2 can now maximally conduct 50 signals which can be achieved with 50 2:1 blocks. The same thought is applied to the first level which leads to 100 2:1 blocks in level 1. This results in the maximum amount of blocks of 175 over all three levels which when encoded as one-hot arrays with length 8 results in a tensor with length 1400. The conditional variable for the cINN contains the RR-intervals for a given set of forward model input variables. This condition tensor has length 24 because of the maximum R-wave limit mentioned above. Figure 3.1 shows a graphical illustration. Sampling from the cINN is very simple since the dimensions of the latent variable tensor can be set to be exactly the same as those of the input tensor and the entries are sampled from a standard normal distribution.

bp_cINN_multi

The second approach is very similar to the first one. The difference is that multiple cINNs are used, each one designed and trained specifically for one MAVB type. More precisely, each cINN exploits the specific constraints of each MAVB type which allows for a reduction of the size of the input tensor compared to the bp_cINN approach. For example, MAVB 1 has blocks only on level 2 and those blocks range from a 2:1 block to a 8:7 block which means that only the second level has to be considered and the one-hot encoding can be reduced to a tensor of length 7. The block pattern structure with the maximum block across all levels in this case consists only of 25 2:1 blocks in level 2 which results in an input tensor size of 175 plus the two constants AA and α. The idea is that forcing a specific MAVB type from the networks might result in more diversity when sampling from all networks for a given RR-interval list. Sampling from the cINNs in this case is as simple as with the bp_cINN approach but of course it has to be done for five separate networks.

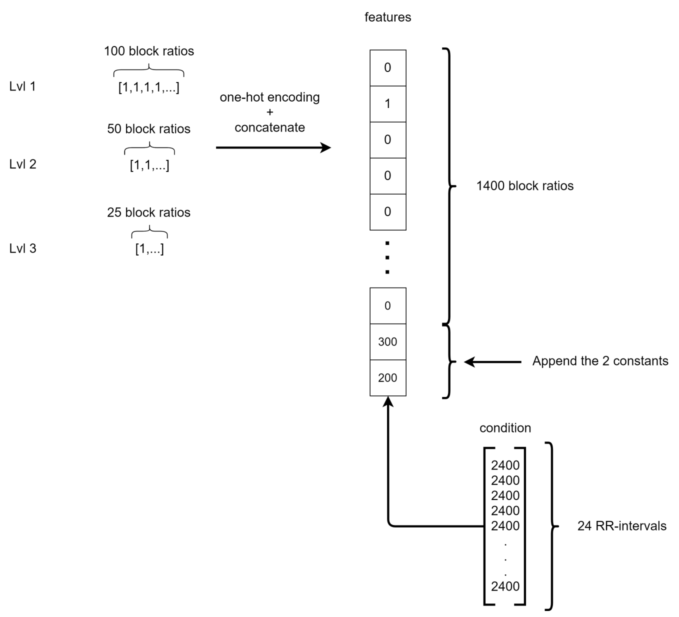

Figure 3.1: Data transformation for bp_cINN. The figure shows how a sample with the maximum number of block ratios would be transformed into the input tensor for the bp_cINN approach. For every other sample, that is all samples that have less ratios in the block pattern lists, unused entries in the 1-D input tensor are left as zeros. At the very bottom one can see the conditional variable with the RR-intervals that is appended to the input tensor only right before passing the input tensor to the fully connected subnetworks. The values in the conditional variable were crated by using the block pattern lists in the figure as well as $AA = 300$ and $\alpha = 200$ as inputs to the forward model.

bp_rcINN

The third approach differs from the previous two in the INN architecture. The INN used here is a combination of the cINN architecture (see Subsection 2.3.4) and the CRow architecture (see Subsection 2.3.5). In other words, it is a cINN that uses recurrent subnetworks (Figure 3.2). This network structure will be referred to as a *rcINN*. The forward pass through one coupling block

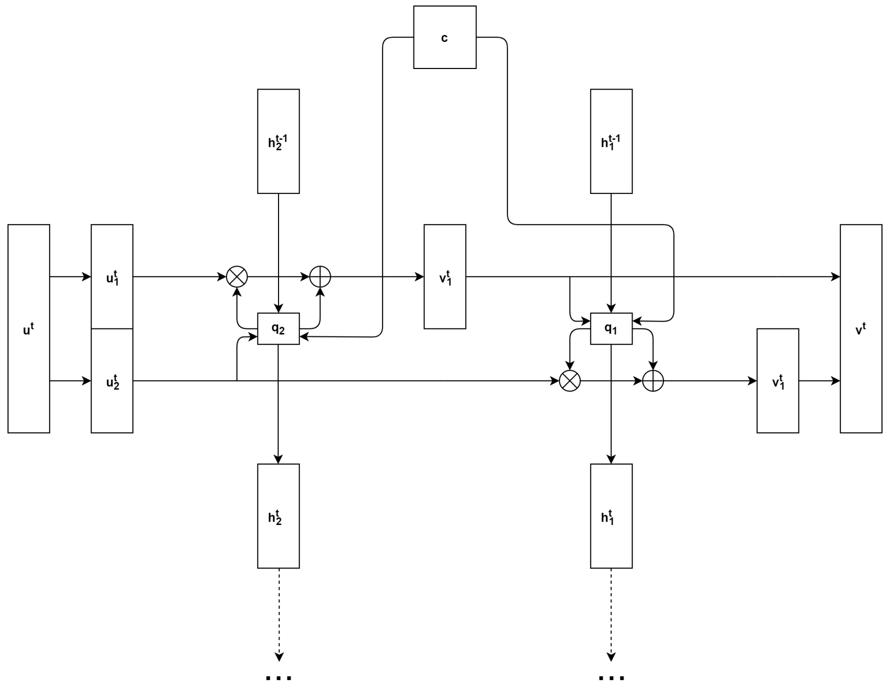

Figure 3.2: rcINN coupling block. u and v are the input and output vectors of the coupling block, respectively. c represents the conditional variable. q_1 and q_2 represent arbitrary RNNs with hidden state h_1 and h_2, respectively. The circles with an "x" and a "+" represent point-wise multiplication and addition respectively. The dashed arrows pointing down represent the fact that the computed new hidden states h_1^t and h_2^t will be used in the next time step.

at one time step can be written as

$$\boldsymbol{v}_1^t = \boldsymbol{u}_1^t \otimes \exp(q_{s_2}(\boldsymbol{u}_2^t, \boldsymbol{c}, \boldsymbol{h}_2^{t-1})) + q_{t_2}(\boldsymbol{u}_2^t, \boldsymbol{c}, \boldsymbol{h}_2^{t-1})$$
$$\boldsymbol{v}_2^t = \boldsymbol{u}_2^t \otimes \exp(q_{s_1}(\boldsymbol{v}_1^t, \boldsymbol{c}, \boldsymbol{h}_1^{t-1})) + q_{t_1}(\boldsymbol{v}_1^t, \boldsymbol{c}, \boldsymbol{h}_1^{t-1})$$

with the inverse

$$\boldsymbol{u}_2^t = (\boldsymbol{v}_2^t - q_{t_1}(\boldsymbol{v}_1^t, \boldsymbol{c}, \boldsymbol{h}_1^{t-1})) \oslash \exp(q_{s_1}(\boldsymbol{v}_1^t, \boldsymbol{c}, \boldsymbol{h}_1^{t-1}))$$
$$\boldsymbol{u}_1^t = (\boldsymbol{v}_1^t - q_{t_2}(\boldsymbol{u}_2^t, \boldsymbol{c}, \boldsymbol{h}_2^{t-1})) \oslash \exp(q_{s_2}(\boldsymbol{u}_2^t, \boldsymbol{c}, \boldsymbol{h}_2^{t-1}))$$

The idea is to take advantage of the fact that the block pattern part of the input variables is of sequential nature and RNNs are particularly suited for such data. Just as in the previous two approaches, this approach utilizes the same one-hot encoding of the block ratios but requires an additional transformation of the block pattern lists in order to make the input tensor suitable for a RNN. More precisely, one time step of the transformed input corresponds to one atrial signal and the features of the input tensor are the block ratio entries of the block cycles to which the considered signal belongs. As long as there is no new signal passing through a given level of the AV node, the block ratio entry in the input tensor does not change for the next time step. Expressed in terms of the visual intuition of Figure 2.4, following the vertical lines from the atrial level to the ventrical level and using the number of conducted signals in the block cycle through which the vertical line passes results in the transformed input tensor. Whenever a vertical line ends in a dashed line, the last block ratio entry of the level under the dashed line is taken as the current entry in the input tensor. A visual example is shown in Figure 3.3. The transformation ends with the last signal that passes through to the ventricles and ignores blocked ones after that. After this transformation the block ratio entries are encoded with a one-hot encoding like in the previous approaches and all the time steps are concatenated along the sequence dimension resulting in a 2D tensor with shape ($24 \times$ sequence length). The two constants AA and α are extended across the sequence dimension, i.e. they are constant along the sequence dimension, and appended forming a final input tensor with shape ($26 \times$ sequence length). The conditional variable has to be transformed as well, since it has to be appended to the 2D input tensor. For this, two approaches were developed of which the second one will be explained in the next paragraph (see bp_rcINN_matching). The first option is to simply extend the tensor of RR-intervals along the sequence dimension, i.e. the conditional variable is constant for every time step (see left-hand side of Figure 3.4). This means that the RNN subnetwork sees all the RR-interval information at every time step without establishing any previous connection between the block ratio entries and specific single RR-intervals that they are responsible for. However, the advantage of this approach is that it is simpler and less resource intensive as the second variant. Sampling from the rcINN is not as simple as with the cINN approaches described above. The latent variable tensor has an additional sequence dimension whose length has to be set before sampling the tensor entries from a standard normal distribution and backwards passing through the rcINN. To avoid simply having to choose randomly or brute forcing every possibility, a separate cINN is used for this purpose to provide reasonable suggestions for the sequence length for a given

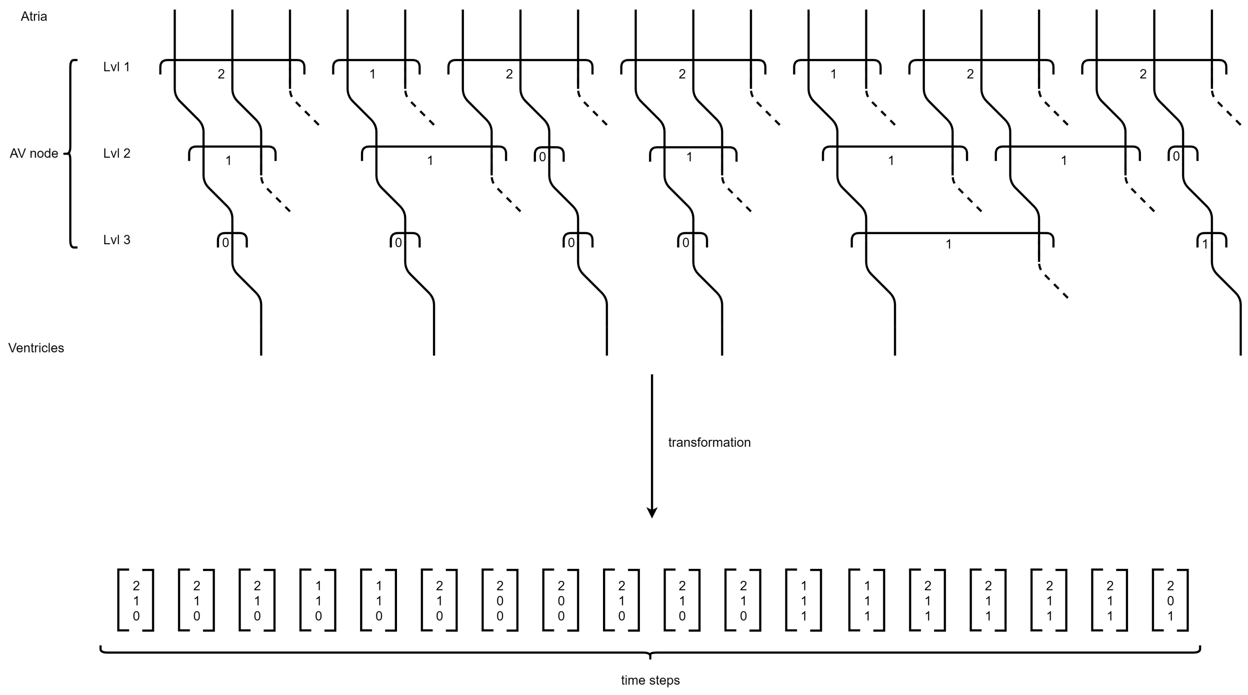

Figure 3.3: First part of the data transformation for bp_rcINN and bp_rcINN_matching. The figure shows an example MAVB type 3 block pattern that is transformed to a series of time steps following the transformation described in the text.

RR-interval list. This sequence cINN is trained with a one-hot encoding of the sequence length of block pattern samples and uses RR-intervals as the conditional variable. When sampling from the rcINN this cINN is used first to sample sequence length suggestions and then the rcINN is run backwards with the suggested sequence lentgths.

bp_rcINN_matching

The last approach that uses the block patterns directly is very similar to the bp_rcINN approach. The only differnce is how the conditional variable is transformed. In this approach the RR-intervals in the conditional variable are matched to the time steps that contain the signals that are responsible

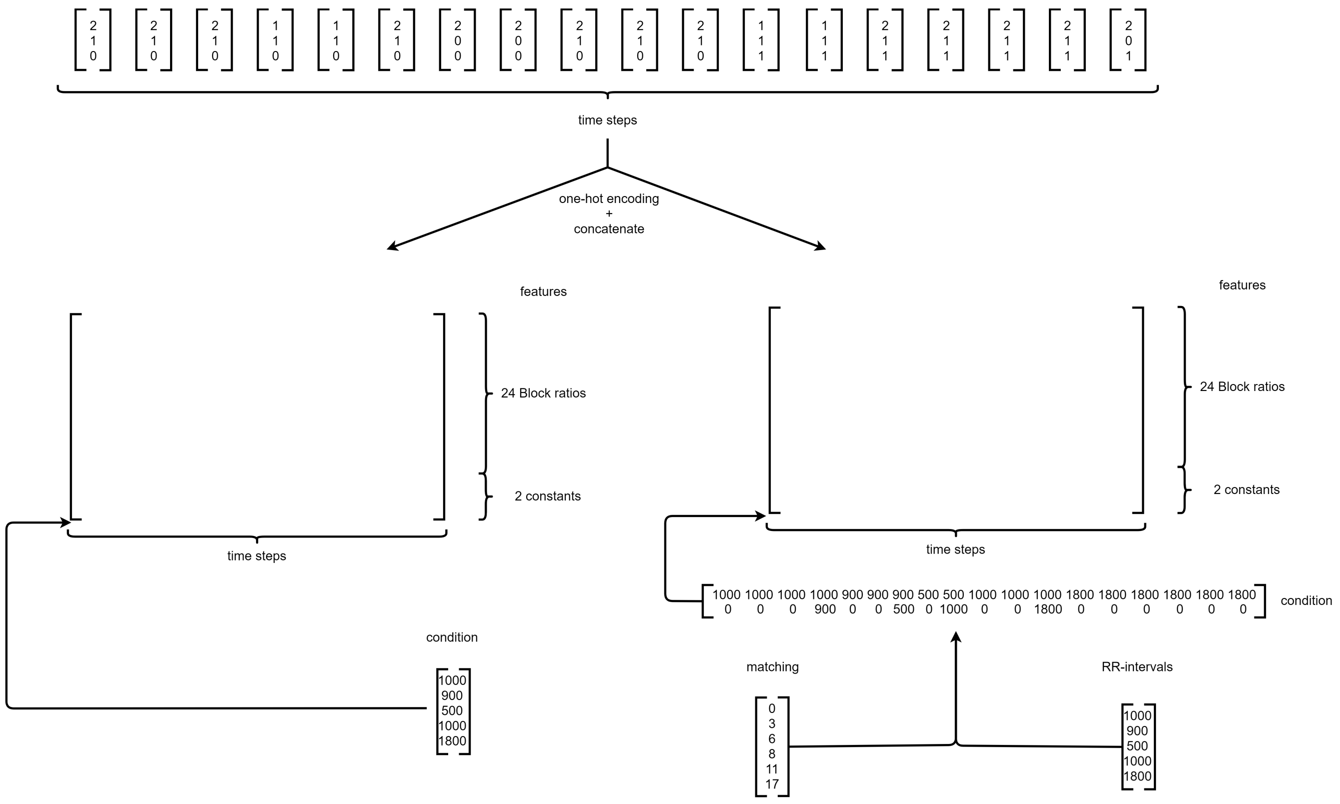

Figure 3.4: Second part of the data transformation for bp_rcINN and bp_rcINN_matching. In this part the final input tensor is generated and the transformation of the conditional variable is shown. The left hand side shows the bp_rcINN variant of the transformation of the conditional variable while the right-hand side shows the bp_rcINN_matching variant. The RR-interval values were created with the forward model by using the block pattern lists of the example in this figure and Figure 3.3 as well as constants $AA = 300$ and $\alpha = 200$. The condition tensors are only appended right before passing the input tensor to the RNN subnetworks. Details regarding the transformations are in the text.

for the given RR-interval. This is done by keeping track of when the block ratio of the third level changes during the transformation of the input data because whenever it does this indicates a signal being passed through the third level to the ventricles due to the fact that the first signal in a block cycle is never blocked. With this information the time steps that contain the signals that form the start and end of an interval are known and thus all time steps from start to end can be matched to the corresponding RR-interval. Note that with this transformation there is an overlap of matched RR-intervals at the time steps that contain both the signal that represents the end of the previous RR-interval and the start of the next RR-interval. This is the reason why the conditional variable is transformed to a tensor of shape (2 $\times$ sequence length) to allow for the matching of up to two RR-intervals to a given time step. In the cases where there are two RR-intervals to be matched to a given time step, the first RR-interval in chronological order is put in the first entry while the second RR-interval is put in the second entry. In the cases where there is only one RR-interval to be matched to a given time step, the first entry is used for that and the second is left as a zero. The right-hand side of Figure 3.4 shows a visualization of this transformation. Sampling from this rcINN is more complicated than it is the case with the bp_rcINN approach. The reason for that is that a given RR-interval list now has to be matched to the time steps of the latent variable. Just like with the sequence length problem this is solved by using a separate cINN that can suggest a matching based on a RR-interval list. This matching cINN is trained with one-hot input tensors that represent the number of intervals to which each individual RR-interval of the conditional variable has to be matched. Since the RR-intervals are always in chronological order this amount of intervals to which a given RR-interval has to be matched can be easily converted to a time step index and thus used for matching a given RR-interval list to the latent variable when sampling from the rcINN. After using the sequence length cINN to generate sequence length suggestions, the matching cINN is used to generate matching suggestions. These suggestions are then paired with fitting sequence lengths and the resulting pairs of sequence length and matching are then used to sample the corresponding latent variables and generate the corresponding transformed conditional variables.

3.3.2 Signal Approaches

The approaches in this subsection require a transformation of the block pattern lists to a matrix that represents the signal diagram used in Figure 2.4. This matrix contains 1s and 0s that indicate a signal passing through and no signal passing through, respectively. Three arrays are generated for the three block pattern lists and then concatenated into a 2D tensor of shape

$$\begin{array}{c} [2,1,2,2,1,2,2] \\ [1,1,0,1,1,1,0] \\ [0,0,0,0,1,1] \end{array} \longrightarrow \begin{bmatrix} 1&1&0&1&0&1&1&0&1&1&0&1&0&1&1&0&1&1 \\ 1&0&0&1&0&0&1&0&1&0&0&1&0&0&1&0&0&1 \\ 1&0&0&1&0&0&1&0&1&0&0&1&0&0&0&0&0&1 \end{bmatrix}$$

Figure 3.5: Block pattern to signal transformation. The figure shows the transformation of an example block pattern to a signal matrix. The transformation details are in the text.

($3 \times$ sequence_length). The sequence length is determined by the last signal passing through the first layer. For a given block ratio $n + 1 : n$, n 1s are inserted into the corresponding array followed by a 0, except for the case of a 1:1 ratio in which case just a 1 is inserted. To align the different levels and ensure that there is never a 1 in level 2 or 3 when there is no 1 in the level above, a 0 is inserted in level 2 whenever there is a 0 in level 1 and the same thing is done between level 2 and level 3. Figure 3.5 shows the transformation of the same block pattern that was used in Figure 3.3 and Figure 3.4. As mentioned before, MAVB type 1, 2a and 2b have a variable block ratio $n + 1 : n$ in level 2 which in some cases leads to the sum over level 2 being higher than the required amount of conducted signals for the next level (see Section 3.2). Using the above described signal transformation would result in too many 1s in level 2 in such cases. That is why a "correction" algorithm is used before transforming a given block pattern. This algorithm simply reduces the block at the end of the level 2 list in case the sum over level 2 exceeds the corresponding condition for the respective MAVB type (see Algorithm A.6 in Appendix A fore more details). The transformation from block patterns to signals is not bijective. Multiple block patterns can lead to the same signal matrix. For example this is the case for the above mentioned MAVB type 1, 2a and 2b because of the variable $n + 1 : n$ block in level 2 (Figure 3.6). This problem of ambiguity when transforming back from signals to block patterns is solved by transforming a given signal matrix to every possible block pattern structure to which it can map under the constraints of the MAVB types. Incorporating the constraints into the transformation process makes the amount of block patterns that can be generated from a signal matrix managable and thus using a more complex method like a separate network was avoided.

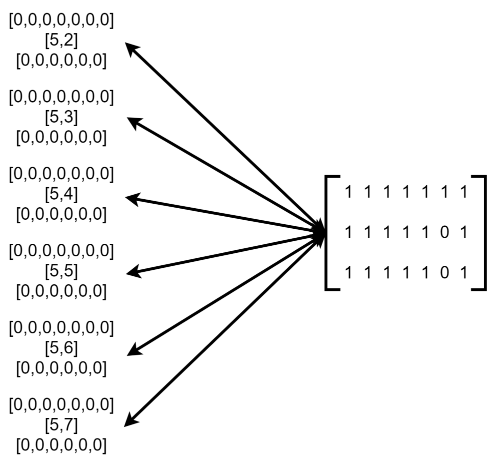

Figure 3.6: Ambiguity in the transformation between block patterns and signal matrices. In this example a MAVB type 1 block pattern is used and the number of R-waves is 6.

signal_cINN

The first approach that utilizes the signal matrix is similar to the bp_cINN approach. A cINN is used for which the rows of the signal matrix are first filled up with 0s to a length of 200 and then concatenated along the sequence dimension to form a 1D tensor with length 600. The length comes from the fact that the maximum R-wave constraint mentioned previously together with the block structure with the maximum block across all levels leads to a maximum of 200 signals at level 1 and since the other two arrays are aligned with level 1 they also have a maximum length of 200 which results in a 1D tensor with length 600 when concatenated (Figure 3.7). The constants AA and α are simply appended at the end. Sampling from the cINN is just as simple as with the bp_cINN approach.

signal_rcINN

The second signal approach uses a rcINN to utilize the sequential nature of the signal matrix, similar to the bp_rcINN approach. Since the signal

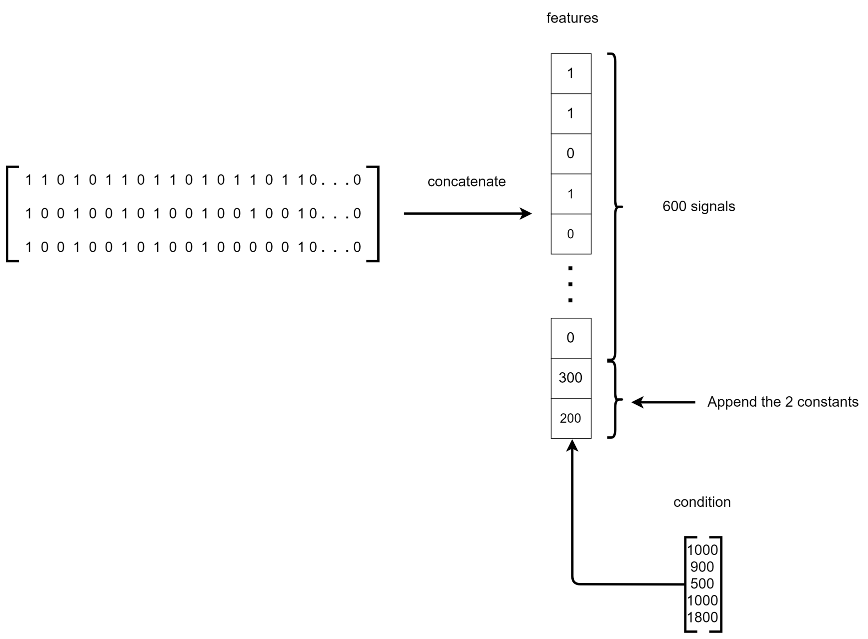

Figure 3.7: Data transformation for signal_cINN. The figure shows the transformation of the signal matrix from Figure 3.5 to the input tensor for the signal_cINN approach. The block pattern lists from Figure 3.5 and constants $AA = 300$ and $\alpha = 200$ were used to generate the RR-intervals of the conditional variable. The conditional variable is appended right before passing the input tensor to the fully connected subnetworks.

matrix is already perfectly suitable as an input tensor for a RNN, no further transformation is needed aside from attaching the two constants AA and α at every time step. The conditional variable with all the RR-intervals is appended at every time step as well. Sampling from the rcINN presents the same challenge of having to choose the sequence length dimension of the latent variable as with the bp_rcINN approach and this is solved in exactly the same manner with a separate sequence cINN for the sequence length suggestion.

signal_rcINN_matching

The last signal approach is very similar to the signal_rcINN approach and differs only in the transformation of the conditional variable. Just like with the bp_rcINN_matching approach the RR-intervals of the conditional variable are matched to the corresponding time steps. In this case however the identification of the time steps at which a signal passes through level 3 and is thus recognized as a R-wave is simpler since this requires only checking the position of 1s in the last row of the signal matrix. The RR-intervals are then matched to the corresponding time steps just as in the bp_rcINN_matching approach. The same challenge presents itself when sampling from the rcINN and is again solved by using two cINNs, one for the sequence length suggestion and one for the matching suggestion. Aside from this, the approach is exactly the same as the signal_rcINN approach.

Chapter 4

Experiments

This chapter gives a detailed description of the training and testing procedure and then presents the results.

4.1 Setup

This section describes the overall training and testing procedure as well as what parameters were used during training and testing and what metrics were considered.

4.1.1 Training

All INNs (see Appendix B for specific subnetwork architecture) are implemented with the Framework for Easily Invertible Architectures (FrEIA) [22] and Pytorch [23]. The rcINN approaches all use LSTMs as their subnetworks. 5 GLOW coupling blocks are used for all models because of the results for the Gaussian mixture model shown in Figure 3 of the Bayesflow paper [24]. The clamp parameter is set to 2.0 for all models because of the recommendation in the README.rst of the FrEIA github repository [22]. The FrEIA code was changed slightly in order to allow for the usage of RNNs with the API of Pytorch in the specific case of Glow coupling blocks. All networks including the additional networks used for sampling from the recurrent approaches are trained with an online-learning approach. This was inspired by the training procedure in the Bayesflow paper [24]. At every epoch a big batch of 102400 data samples is generated with the procedure described in Section 3.2 and then smaller batches of size 512 are passed through the networks and gradient descent is used to update the model parameters. For the recurrent approaches the generated data is grouped into batches of equal sequence length and then when possible batches of 512 are used to feed through the networks, otherwise the entire batch of smaller size is used. Gaussian noise is added to one-hot encodings of block ratio entries as well as all signal matrices in order to transform the discrete data to continuous data where the theory on normalizing flows presented in Subsection 2.3.2 is applicable to. Furthermore, all entries containing the constants AA and α and the RR-intervals of the conditional variable are normalized by subtracting their respective mean and dividing by their respective standard deviation. These means and standard deviations are approximated over 1000000 samples and used for all approaches. The online-training eliminates the problem of overfitting, since every epoch presents completely new examples to the networks, and the added time for an epoch due to the generation of data is managable because the forward model is fast enough. The AdamW optimizer from the Pytorch library is used with a starting learning rate of 10^{-3} and an exponential decay rate of 0.95. The networks were trained for 100 epochs. No extensive hyperparameter tuning was done. All training runs took approximately 1-3 hours on a desktop PC with a NVIDIA© RTX 2070 Super and a AMD© Ryzen 9 3900X.

4.1.2 Testing

To test the performance of the approaches 1000 samples for each MAVB type are generated according to the data generation procedure in Section 3.2. This data is used for all approaches. For the bp_cINN and signal_cINN approach 5000 latent variable samples were used for inference. For the bp_rcINN and signal_rcINN approach 1000 latent variable samples for the sequence cINN are used to geneare sequence lengths suggestions. The top 10 most frequently suggested lengths are then used for the inference with the rcINN where 500 latent variable samples are used per suggsted sequence length. For the bp_rcINN_matching and signal_rcINN_matching approach the same is done as far as the sequence lengths cINN goes. 5000 latent variable samples are used for the matching cINN and then every suggestion is paired with the top 10 suggested sequence lengths if possible. Finally 50 latent variable samples per pair of matching and sequence length are used and up to the first 10 pairs for a given sequence length are used for inference. For the bp_cINN_multi approach 1000 latent variable samples are used per network. These numbers were chosen to make the results more comparable. Every sample from a given INN is transformed back from the corresponding input tensor form to the block pattern list form and constants AA and α are extracted and then both block pattern as well as constants are checked for validity, i.e. the sample has to obey the constraints used in generating the data (Section 3.2 and Subsection 2.1.2) and thus be a valid input for the forward model. The metrics chosen for the testing procedure are used to test both performance as well as diversity of the solutions. In the following f stands for the forward model and x^* are the input variables used to generate the RR-interval list $f(x^*)$ that is used with the given network to generate x. The *mean absolute error* (MAE) is used to measure the difference between two RR-interval lists:

$$\mathrm{MAE} = \frac{1}{n} \cdot \sum_{i=1}^{n} |f(x)_i - f(x^*)_i|$$

For the sake of brevity, the RR-intervals for which at least one valid solution could be sampled are referred to as *solved* RR-intervals. For the following metrics always the x that resulted in the closest RR-intervals among the valid suggestions during inference is used and all measurements are done separately for each MAVB type of $f(x^*)$.

- Mean of MAE over all solved RR-intervals (Err) - to assess how well the solution suggested by the network x can approximate the true RR-interval list $f(x^*)$ regardless of similarity of x and x^*.
- Mean of MAE over the solved RR-intervals, where x has the same MAVB type as x^* (Err_{same}) - to assess how well the solution suggested by the network x can approximate the true RR-interval list $f(x^*)$ using the same MAVB type as x^*.
- Mean of MAE over the solved RR-intervals, where x has a different MAVB type as x^* (Err_{alt}) - to assess how well the solution suggested by the network x can approximate the true RR-interval list $f(x^*)$ using a different MAVB type as x^*. This is used as an indication to how well alternative solutions perform, i.e. the "usefullness" of the diversity of the solutions.

The difference in MAVB type is used as an indication for an alternative solution because it is easy to check and resembles a significant difference in the solution, i.e. most of the time it also means different constants AA and α.

This way little differernces in the constants AA and α as well as very similar block patterns with the same MAVB type are not counted as alternative solutions. In addition to the above described performance oriented metrics, the three following additional metrics are measured to see how the diversity of the solutions differs between the appraoches:

- Mean of the ratio of alternative MAVB type solutions to same MAVB type solutions over all solved RR-intervals ($ratio_{alt}$) - to assess in what proportion alternative solutions are utilized
- Means of the ratios of the different MAVB types of the alternative MAVB type solutions over the solved RR-intervals, where x has a different MAVB type as x^* ($ratio_{alt\_types}$) - to assess what kind of alternative MAVB types are preferred in what situations.
- Ratios of the different MAVB types with regards to how often they were the top alternative solution over the solved RR-intervals, where x has a different MAVB type as x^* ($ratio_{top\_alt\_types}$) - to assess which alternative MAVB types were the most useful in explaining the given RR-interval list.

4.2 Results

First, the performance oriented measures Err, Err_{same} and Err_{alt} are shown in Table 4.1. Table 4.2, 4.3 and 4.4 show rankings of the approaches based on the results in Table 4.1. The bp_rcINN_matching approach is the best performing approach across all metrics followed by the bp_cINN_multi approach. Both of them are also able to solve the vast majority of the RR-intervals used for inference. On the other hand the signal_rcINN approach is the worst peforming approach across all metrics and has the lowest amount of solved RR-intervals across all MAVB types. The following patterns can be seen in the rankings:

- Block pattern approaches are generally higher ranked compared to the signal approaches.
- Recurrent block pattern approaches are generally higher ranked than the non-recurrent block pattern approaches. This effect is not as clear with the signal approaches.

- Matching variants of the recurrent approaches are generally higher ranked than their non-matching counterparts.

The results for the diversity metrics $ratio_{alt}$, $ratio_{alt\_types}$ and $ratio_{top\_alt\_types}$ are shown in Table 4.5 - 4.11. A few patterns can be seen for the $ratio_{top\_alt\_types}$ metric:

- MAVB type 2c and 3 are always among the most frequent top alternative solutions for each other. This effect can be seen especially clearly in the two recurrent matching approaches.
- For all the block pattern approaches MAVB type 2a and 2b are always among the most frequent top alternative solutions for each other.
- For all the block pattern approaches MAVB type 2a is always among the most frequent top alternative solutions for MAVB type 1.
- For all the signal approaches MAVB type 3 is always among the most frequent top alternative solutions for all the other MAVB types.

The $ratio_{alt\_types}$ results also show some patterns:

- The recurrent block pattern approaches have MAVB type 2c and 3 solutions more often among the most frequent suggested solutions compared to the non-recurrent block pattern approaches.
- The signal approaches also have MAVB type 2c and 3 solutions more often among the most frequent suggested solutions compared to the non-recurrent block pattern approaches.
- The bp_cINN approach always has MAVB type 2a as the most frequent suggested solution for other MAVB types.

Looking at the $ratio_{alt}$ results one can see the following:

- The recurrent block pattern approaches suggest fewer alternative MAVB type solutions for MAVB type 3 than their non-recurrent counterparts.
- All the signal approaches suggest fewer alternative MAVB type solutions for MAVB type 3 than the non-recurrent block pattern approaches.
- For all approaches it is the case that more than at least half of the suggested solutions for MAVB type 2b and 2c are alternative solutions .

- For all approaches it is the case that less than a third of the suggested solutions for MAVB type 1 are alternative solutions.

Taking the top performing approaches as far as Err_{alt} goes and looking at what their common top alternative solutions are (bold values for $ratio_{top\_alt\_types}$ in Table 4.5 - 4.11) gives the following pattern per MAVB type of $x*$:

- MAVB type 1: 2a, 2c, 3
- MAVB type 2a: 2b, 2c, 3
- MAVB type 2b: 2a, 2c, 3
- MAVB type 2c: 3
- MAVB type 3: 2c

Appendix C shows a few interesting examples for good and bad alternative solutions generated with the bp_rcINN_matching approach.

	MAVB type				
	1	2a	2b	2c	3
bp_cINN					
Err	45 ± 26 (955)	**56 ± 43** (971)	73 ± 39 (969)	185 ± 66 (936)	259 ± 100 (517)
Err_{same}	46 ± 28 (919)	**56 ± 40** (965)	**76 ± 35** (311)	175 ± 65 (378)	248 ± 111 (279)
Err_{alt}	71 ± 34 (381)	115 ± 86 (543)	73 ± 39 (969)	185 ± 66 (936)	259 ± 95 (494)
bp_cINN_multi					
Err	**17 ± 18** (1000)	**28 ± 30** (1000)	**46 ± 31** (1000)	**78 ± 34** (999)	202 ± 122 (747)
Err_{same}	**18 ± 20** (998)	**29 ± 31** (1000)	**52 ± 39** (909)	**78 ± 33** (995)	242 ± 107 (380)
Err_{alt}	**49 ± 23** (745)	**55 ± 49** (996)	**52 ± 29** (1000)	167 ± 66 (942)	206 ± 139 (683)
bp_rcINN					
Err	**27 ± 19** (1000)	**47 ± 36** (1000)	**57 ± 32** (1000)	139 ± 58 (1000)	270 ± 121 (998)
Err_{same}	**29 ± 23** (990)	**49 ± 38** (995)	**60 ± 32** (999)	144 ± 60 (999)	274 ± 120 (998)
Err_{alt}	64 ± 36 (531)	**55 ± 38** (1000)	66 ± 38 (1000)	139 ± 58 (1000)	262 ± 117 (743)
bp_rcINN_matching					
Err	**22 ± 16** (1000)	**35 ± 27** (999)	**37 ± 24** (999)	**50 ± 33** (996)	**100 ± 67** (898)
Err_{same}	**27 ± 21** (986)	**46 ± 39** (967)	**58 ± 30** (992)	**52 ± 35** (990)	**100 ± 67** (898)
Err_{alt}	**34 ± 19** (435)	**43 ± 27** (999)	**38 ± 25** (999)	**50 ± 33** (996)	**128 ± 76** (544)
signal_cINN					
Err	43 ± 29 (912)	79 ± 63 (1000)	92 ± 58 (1000)	173 ± 63 (999)	297 ± 138 (814)
Err_{same}	41 ± 29 (833)	**60 ± 44** (711)	**79 ± 43** (162)	199 ± 75 (912)	300 ± 136 (814)
Err_{alt}	68 ± 36 (481)	99 ± 68 (1000)	92 ± 58 (1000)	173 ± 63 (999)	268 ± 101 (565)
signal_rcINN					
Err	58 ± 42 (811)	162 ± 91 (527)	160 ± 84 (499)	213 ± 94 (519)	348 ± 153 (354)
Err_{same}	47 ± 27 (707)	87 ± 72 (20)	90 ± 46 (166)	210 ± 87 (283)	356 ± 156 (348)
Err_{alt}	106 ± 40 (438)	162 ± 90 (527)	176 ± 76 (497)	213 ± 94 (519)	335 ± 153 (185)
signal_rcINN_matching					
Err	**28 ± 20** (984)	64 ± 52 (782)	79 ± 54 (767)	115 ± 58 (684)	**151 ± 87** (375)
Err_{same}	**29 ± 19** (978)	**48 ± 38** (433)	**59 ± 31** (9)	128 ± 64 (546)	**151 ± 87** (374)
Err_{alt}	**50 ± 30** (362)	81 ± 54 (782)	79 ± 54 (767)	115 ± 58 (684)	**161 ± 72** (283)

Table 4.1: Performance results. All measurements are given in milliseconds with an absolute uncertainty of 1 standard deviation and the numbers in paranthesis give the number of solved RR-intervals either regardless of MAVB type, with the same MAVB type as x^* or a different MAVB type. All measurements are rounded to full milliseconds. For each metric and MAVB type the best performances across the approaches that are within 1 standard deviation from each other are printed in bold font.

	Err
1	bp_rcINN_matching
2	bp_cINN_multi
3	bp_rcINN
4	signal_rcINN_matching
5	bp_cINN
6	signal_cINN, signal_rcINN

Table 4.2: Performance rankings for *Err*. Ranking of the approaches for *Err* based on for how many MAVB types a given approach was among the top performing approaches.

	Err_{same}
1	bp_rcINN_matching
2	bp_cINN_multi, signal_rcINN_matching (!)
3	bp_rcINN
4	bp_cINN, signal_cINN
5	signal_rcINN

Table 4.3: Performance rankings for Err_{same}. Ranking of the approaches for Err_{same} based on for how many MAVB types a given approach was among the top performing approaches. The "(!)" next to the signal_rcINN_matching approach is to remind the reader that the approach solved only 9 of the 1000 RR-intervals with the same MAVB type as x^* and so its ranking spot should not be viewed too positively.

	Err_{alt}
1	bp_rcINN_matching
2	bp_cINN_multi
3	signal_rcINN_matching
4	bp_rcINN
5	signal_cINN, signal_rcINN, bp_cINN

Table 4.4: Performance rankings for Err_{alt}. Ranking of the approaches for Err_{alt} based on for how many MAVB types a given approach was among the top performing approaches.

bp_cINN	MAVB type				
	1	2a	2b	2c	3
$ratio_{alt}$	0.219 ± 0.326	0.144 ± 0.176	0.979 ± 0.042	0.958 ± 0.070	0.840 ± 0.246
$ratio_{alt\_types}$					
1	–	$\mathbf{0.202 \pm 0.339}$	0.026 ± 0.086	0.055 ± 0.160	0.003 ± 0.031
2a	$\mathbf{0.860 \pm 0.239}$	–	$\mathbf{0.886 \pm 0.154}$	$\mathbf{0.863 \pm 0.197}$	$\mathbf{0.860 \pm 0.213}$
2b	0.031 ± 0.088	$\mathbf{0.187 \pm 0.265}$	–	0.020 ± 0.053	0.018 ± 0.050
2c	0.054 ± 0.114	$\mathbf{0.291 \pm 0.196}$	0.043 ± 0.065	–	$\mathbf{0.118 \pm 0.208}$
3	0.054 ± 0.114	$\mathbf{0.320 \pm 0.218}$	0.046 ± 0.071	0.061 ± 0.118	–
$ratio_{top\_alt\_types}$					
1	–	**0.169**	0.020	0.066	0.002
2a	**0.848**	–	**0.966**	**0.699**	**0.670**
2b	0.052	**0.481**	–	0.025	0.026
2c	0.092	**0.315**	0.009	–	**0.302**
3	0.008	0.035	0.005	**0.210**	–

Table 4.5: Diversity results for bp_cINN. The ratios for $ratio_{alt}$ and $ratio_{alt\_types}$ are given with an uncertainty of 1 standard deviation. All ratios shown are rounded to three decimal places and thus might not add up to 1. Ratios that are at least 0.100 are printed in bold font.

bp_cINN_multi	MAVB type				
	1	2a	2b	2c	3
$ratio_{alt}$	0.195 ± 0.266	0.299 ± 0.148	0.904 ± 0.073	0.522 ± 0.252	0.854 ± 0.293
$ratio_{alt\_types}$					
1	–	$\mathbf{0.105 \pm 0.197}$	0.048 ± 0.107	$\mathbf{0.163 \pm 0.268}$	0.006 ± 0.043
2a	$\mathbf{0.215 \pm 0.313}$	–	$\mathbf{0.713 \pm 0.171}$	$\mathbf{0.717 \pm 0.314}$	$\mathbf{0.541 \pm 0.347}$
2b	0.038 ± 0.083	$\mathbf{0.321 \pm 0.176}$	–	$\mathbf{0.116 \pm 0.177}$	$\mathbf{0.101 \pm 0.179}$
2c	$\mathbf{0.746 \pm 0.351}$	$\mathbf{0.560 \pm 0.220}$	$\mathbf{0.234 \pm 0.143}$	–	$\mathbf{0.352 \pm 0.308}$
3	0.002 ± 0.007	0.014 ± 0.027	0.006 ± 0.010	0.030 ± 0.107	–
$ratio_{top\_alt\_types}$					
1	–	0.093	0.057	**0.193**	0.001
2a	**0.192**	–	**0.782**	**0.555**	**0.224**
2b	0.060	**0.763**	–	**0.128**	0.035
2c	**0.748**	**0.142**	**0.161**	–	**0.739**
3	0	0.002	0	**0.123**	–

Table 4.6: Diversity results for bp_cINN_multi. The ratios for $ratio_{alt}$ and $ratio_{alt\_types}$ are given with an uncertainty of 1 standard deviation. All ratios shown are rounded to three decimal places and thus might not add up to 1. Ratios that are at least 0.100 are printed in bold font.

bp_rcINN	MAVB type				
	1	2a	2b	2c	3
ratio$_{alt}$	0.236 ± 0.341	0.881 ± 0.036	0.888 ± 0.048	0.666 ± 0.057	0.293 ± 0.254
ratio$_{alt\_types}$					
1	–	0.022 ± 0.080	0.014 ± 0.058	0.038 ± 0.133	0.002 ± 0.031
2a	0.085 ± 0.087	–	$\mathbf{0.133 \pm 0.039}$	$\mathbf{0.164 \pm 0.071}$	$\mathbf{0.218 \pm 0.199}$
2b	0.019 ± 0.025	$\mathbf{0.117 \pm 0.050}$	–	$\mathbf{0.156 \pm 0.083}$	$\mathbf{0.170 \pm 0.141}$
2c	$\mathbf{0.432 \pm 0.071}$	$\mathbf{0.379 \pm 0.057}$	$\mathbf{0.380 \pm 0.044}$	–	$\mathbf{0.610 \pm 0.181}$
3	$\mathbf{0.465 \pm 0.078}$	$\mathbf{0.482 \pm 0.081}$	$\mathbf{0.474 \pm 0.066}$	$\mathbf{0.642 \pm 0.114}$	–
ratio$_{top\_alt\_types}$					
1	–	0.037	0.030	0.028	0.001
2a	**0.337**	–	**0.895**	0.044	**0.164**
2b	**0.111**	**0.887**	–	**0.108**	**0.133**
2c	**0.339**	0.027	0.026	–	**0.701**
3	**0.213**	0.049	0.049	**0.820**	–

Table 4.7: Diversity results for bp_rcINN. The ratios for *ratio*$_{alt}$ and *ratio*$_{alt\_types}$ are given with an uncertainty of 1 standard deviation. All ratios shown are rounded to three decimal places and thus might not add up to 1. Ratios that are at least 0.100 are printed in bold font.

bp_rcINN_matching	MAVB type				
	1	2a	2b	2c	3
ratio$_{alt}$	0.194 ± 0.312	0.842 ± 0.130	0.943 ± 0.032	0.670 ± 0.098	0.128 ± 0.157
ratio$_{alt\_types}$					
1	–	0.036 ± 0.125	0.023 ± 0.086	0.043 ± 0.144	0.003 ± 0.033
2a	0.084 ± 0.106	–	$\mathbf{0.172 \pm 0.141}$	0.040 ± 0.056	0.019 ± 0.045
2b	0.013 ± 0.018	0.072 ± 0.040	–	0.023 ± 0.025	0.042 ± 0.110
2c	$\mathbf{0.437 \pm 0.061}$	$\mathbf{0.374 \pm 0.079}$	$\mathbf{0.343 \pm 0.067}$	–	$\mathbf{0.936 \pm 0.124}$
3	$\mathbf{0.466 \pm 0.061}$	$\mathbf{0.518 \pm 0.098}$	$\mathbf{0.462 \pm 0.114}$	$\mathbf{0.894 \pm 0.155}$	–
ratio$_{top\_alt\_types}$					
1	–	0.050	0.018	0.004	0
2a	**0.163**	–	**0.386**	0	0
2b	0.030	**0.393**	–	0	0.013
2c	**0.451**	**0.212**	**0.260**	–	**0.987**
3	**0.356**	**0.344**	**0.335**	**0.996**	–

Table 4.8: Diversity results for bp_rcINN_matching. The ratios for *ratio*$_{alt}$ and *ratio*$_{alt\_types}$ are given with an uncertainty of 1 standard deviation. All ratios shown are rounded to three decimal places and thus might not add up to 1. Ratios that are at least 0.100 are printed in bold font.

signal_cINN	MAVB type				
	1	2a	2b	2c	3
$ratio_{alt}$	0.300 ± 0.390	0.923 ± 0.081	0.997 ± 0.011	0.895 ± 0.063	0.130 ± 0.131
$ratio_{alt\_types}$					
1	–	0.012 ± 0.055	0.007 ± 0.038	0.022 ± 0.096	0.002 ± 0.021
2a	0.068 ± 0.090	–	0.080 ± 0.084	0.085 ± 0.097	$\mathbf{0.371 \pm 0.232}$
2b	0.002 ± 0.008	0.04 ± 0.015	–	0.003 ± 0.011	0.006 ± 0.020
2c	$\mathbf{0.201 \pm 0.100}$	$\mathbf{0.122 \pm 0.072}$	$\mathbf{0.107 \pm 0.058}$	–	$\mathbf{0.621 \pm 0.239}$
3	$\mathbf{0.729 \pm 0.148}$	$\mathbf{0.862 \pm 0.100}$	$\mathbf{0.806 \pm 0.140}$	$\mathbf{0.890 \pm 0.137}$	–
$ratio_{top\_alt\_types}$					
1	–	0.019	0.021	0.037	0.002
2a	0.094	–	**0.536**	**0.116**	**0.322**
2b	0.006	0.045	–	0.003	0.002
2c	0.058	**0.105**	0.038	–	**0.674**
3	**0.842**	**0.831**	**0.405**	**0.844**	–

Table 4.9: Diversity results for signal_cINN. The ratios for $ratio_{alt}$ and $ratio_{alt\_types}$ are given with an uncertainty of 1 standard deviation. All ratios shown are rounded to three decimal places and thus might not add up to 1. Ratios that are at least 0.100 are printed in bold font.

signal_rcINN	MAVB type				
	1	2a	2b	2c	3
$ratio_{alt}$	0.274 ± 0.375	0.999 ± 0.005	0.939 ± 0.121	0.924 ± 0.092	0.178 ± 0.223
$ratio_{alt\_types}$					
1	–	0.022 ± 0.098	0.016 ± 0.068	0.043 ± 0.150	0.017 ± 0.104
2a	0.002 ± 0.011	–	0.001 ± 0.009	0.001 ± 0.007	0.005 ± 0.027
2b	0.006 ± 0.026	0.072 ± 0.145	–	0.069 ± 0.134	$\mathbf{0.788 \pm 0.328}$
2c	$\mathbf{0.238 \pm 0.110}$	0.067 ± 0.087	0.069 ± 0.085	–	$\mathbf{0.190 \pm 0.303}$
3	$\mathbf{0.754 \pm 0.114}$	$\mathbf{0.839 \pm 0.191}$	$\mathbf{0.913 \pm 0.123}$	$\mathbf{0.886 \pm 0.191}$	–
$ratio_{top\_alt\_types}$					
1	–	0.057	0.048	0.060	0.016
2a	0.005	–	0.026	0.002	0.005
2b	0.034	**0.292**	–	0.083	**0.741**
2c	0.041	0.002	0.012	–	**0.238**
3	**0.920**	**0.649**	**0.913**	**0.855**	–

Table 4.10: Diversity results for signal_rcINN. The ratios for $ratio_{alt}$ and $ratio_{alt\_types}$ are given with an uncertainty of 1 standard deviation. All ratios shown are rounded to three decimal places and thus might not add up to 1. Ratios that are at least 0.100 are printed in bold font.

signal_rcINN_matching	MAVB type				
	1	2a	2b	2c	3
$ratio_{alt}$	0.060 ± 0.146	0.941 ± 0.083	1.000 ± 0.001	0.840 ± 0.109	0.183 ± 0.140
$ratio_{alt\_types}$					
1	–	$\mathbf{0.171 \pm 0.337}$	$\mathbf{0.159 \pm 0.326}$	$\mathbf{0.195 \pm 0.354}$	0.022 ± 0.124
2a	0.022 ± 0.040	–	0.053 ± 0.075	0.025 ± 0.052	$\mathbf{0.125 \pm 0.190}$
2b	0 ± 0	0 ± 0.001	–	0 ± 0	0 ± 0.001
2c	$\mathbf{0.264 \pm 0.087}$	$\mathbf{0.182 \pm 0.114}$	$\mathbf{0.176 \pm 0.108}$	–	$\mathbf{0.853 \pm 0.217}$
3	$\mathbf{0.714 \pm 0.096}$	$\mathbf{0.646 \pm 0.282}$	$\mathbf{0.612 \pm 0.266}$	$\mathbf{0.781 \pm 0.348}$	–
$ratio_{top\_alt\_types}$					
1	–	**0.107**	**0.116**	**0.145**	0.004
2a	0.044	–	**0.249**	0.009	0.025
2b	0	0.003	–	0	0
2c	0.055	0.088	0.043	–	**0.972**
3	**0.901**	**0.802**	**0.592**	**0.846**	–

Table 4.11: Diversity results for signal_rcINN_matching. The ratios for $ratio_{alt}$ and $ratio_{alt\_types}$ are given with an uncertainty of 1 standard deviation. All ratios shown are rounded to three decimal places and thus might not add up to 1. Ratios that are at least 0.100 are printed in bold font.

Chapter 5

Discussion and Outlook

5.1 Discussion of the Results

This section discusses the results shown in Chapter 4 split up into performance metrics and diversity metrics.

5.1.1 Performance

The performance results overall show that INNs are indeed applicable to input variables that are of discrete nature such as the block pattern lists. Using Gaussian noise to transform the inputs to a continuous space seems to be sufficient in order for the INNs to learn from the data and suggest reasonable input variables for given conditional variables. Another interesting thing to note is that there is a noticeable performance drop for all approaches when comparing MAVB type 2c to MAVB type 1, 2a, 2b and when comparing MAVB type 3 to MAVB type 2c. This is in line with the amount of block pattern lists the forward model requires for each MAVB type. For MAVB type 1, 2a, 2b the forward model only needs one list, whereas for MAVB type 2c and 3 it needs 2 and 3 lists, respectively. These latter two MAVB types might be more complex and harder to learn. Despite the fact that the block patterns as well as the RR-intervals are of sequential nature the bp_cINN_multi approach performed very well relative to the other approaches, reaching second place across all performance metrics considered. This shows that even a cINN that takes no advantage of the sequential nature of the input and conditional data is able to learn the data distribtuion to some degree. For the block pattern approaches the recurrent models performed overall better than their non-recurrent counterparts, suggesting that using recurrent subnetworks can indeed have a positive effect when dealing with sequential data. Taking advantage of not only the sequential nature of the input data (block pattern lists) but also of the sequential nature of the conditional variable (RR-intervals) by matching the intervals specifically to their corresponding time steps of the input data resulted in another increase in performance for both the recurrent block pattern approach as well as the signal approach. In fact, these two approaches were the best performing approaches for the complicated MAVB type 3. Overall, the block pattern approaches performed better than the signal approaches which is mainly due to the recurrent block pattern approaches outperforming the recurrent signal approaches. It is not clear as to why that was the case. Finally, as a rough absolute measure of performance the cut-off value of Kehrle's thesis, mentioned in Subsection 2.1.2, should be considered. This cut-off value was based on the root mean squared error (RMSE) between the predicted signal time points of the forward model and the ventricular signal time points of the ECG. Kehrle kindly provided the calculated MAE values for comparison but the fact that signal differences were measured instead of interval differences like in this thesis makes it impossible to directly compare the results here with the cut-off value. However, the cut-off value can be used as a rough measure to gauge whether the mean of the MAE achieved by the approaches is at least in the right order of magnitude. The cut-off value varied depending on the number of RR-intervals from 23 ms to 32 ms. Again, the expected result is an MAE below the cut-off value, which in a discrimination scenario like the one considered in Kehrle's thesis would correspond to diagnosing the underlying medical condition as AFlut which for the considered data is always correct. Considering this, the best approach, the bp_rcINN_matching approach, has a mean MAE in the right order of magnitude for all MAVB types except for MAVB type 3 (see Table 4.1). This is the case even when adding one standard deviation of the mean on top. This further shows that INNs do have the potential to perform decently on this type of data but of course more research is required before reaching any definitive conclusions on their actual usefulness with real world ECG data.

5.1.2 Diversity

The results of the diversity oriented metrics show a possible hint at the underlying similarity between the different MAVB types. Taking a closer look at the results for $ratio_{top\_alt\_types}$ reveals some interesting patterns. One obvious pattern is that MAVB type 2c and 3 are often the top alternative solution for each other. The use of MAVB type 3 to explain MAVB type 2c can be explained by the fact that every MAVB type 2c block pattern is also a MAVB type 3 block pattern where level 3 only has 1:1 blocks. The use of MAVB type 2c to explain MAVB type 3 might be due to the similarity in the block ratio constraints. MAVB type 2a is often among the top alternative solutions for MAVB type 1. This does make sense because the only difference between those two types is that for a given atrial rate only every second signal arrives at level 2 for MAVB type 2a compared to type 1. This might be compensable by an adjustment of the constants AA and α for some cases. Similar logic can be applied to the relationship between MAVB type 1 and 2b or MAVB type 2a and 2b. Indeed, when looking at the top alternative solutions of the best performing approaches, as far as $ratio_{top\_alt\_types}$ goes, 2a and 2b are used for each other and again 2a is seen among the top alternative solutions for type 1. Taking a look at the results of $ratio_{alt}$ and $ratio_{alt\_types}$ one can see that different approaches prefer to use different MAVB types in their alternative solutions. How well this preference is aligned with the underlying similarity of the data mentioned above might have something to do with the differences in performance between the approaches. For example the bp_rcINN_matching approach seems to have understood the fundamental similarity between MAVB type 2c and 3 because it predominantly suggests these two types as solutions for each other. The same can be seen in the signal_rcINN_matching approach as well as the signal_cINN approach. However, this alone can not be taken as an indication of good performance as the signal_cINN approach did not perform well for MAVB type 2c and 3 as far as the alternative solutions go. The signal approaches overall seem to use predominantly alternative solutions for MAVB type 2a, 2b and 2c and more precisely they tend to use MAVB type 3. A similar fixation on a specific MAVB type can be seen in the bp_cINN approach which preferrs MAVB type 2a solutions. This might deviate from the underlying similarity of the MAVB types too much to the point where it negatively affects the performance. Finally, the bp_cINN_multi approach which consists of multiple networks trained specifically for each MAVB type for the purposes of diversity, seems not have a significantly bigger ratio of alternative solutions to same type solutions compared to other approaches. Overall it seems like the MAVB types have similarities with some MAVB types more than others and that this can be seen to some degree in the preferences of the more successful models.

5.2 Outlook

Since the results showed that recurrent subnetworks do have a positive effect on performance it is probably a good idea to pursue the strategy of using recurrent subnetworks for future research. Additionally, since the bp_rcINN_matching approach was the overall best performing approach it might be worth investigating more ways of taking advantage of the sequential nature of both block pattern lists (input variable) as well as RR-intervals (conditional variable). One way to do this is by using the Bayesflow method [24]. With this approach a so called *summary network* is used to pre-process the conditional variable and summarize the important information which is then used in an INN. The summary network is designed to be particularly suitable for the type of data for which the Bayesflow method is used. In the context of the data used in this thesis, the natural choice for the summary network would be an RNN such as an LSTM because the RR-intervals are of sequential nature. This combined with an rcINN utilizing the strengths of recurrent subnetworks could lead to better results than were achieved with the bp_rcINN_matching approach. Another idea is to use the concept of the bp_cINN_multi approach for a recurrent approach, i.e. using multiple networks trained specifically on one MAVB type. Just as it was beneficial in the cINN case it might lead to better results in the rcINN cases. Besides that, further research into possibly other ways of transforming the block patterns might yield better results. Of course all the approaches tested in this thesis might benefit from an extensive hyperparameter optimization as well as more training time. Finally, future research can also explore the usefulness of the INN models in the discrimination between AFlut and AFib with real world ECG data similar to the validation done in Kehrle's thesis (see Chapter 5 of [8]).

Appendix A

Algorithms

Algorithm A.1: Generate MAVB 2a Block Pattern

```
   Input: $n\_Rwaves$
   Output: The block pattern lists: $lvl\_1$, $lvl\_2$, $lvl\_3$
1  Initialize empty block pattern lists $lvl\_1$, $lvl\_2$, $lvl\_3$
2  fill $lvl\_3$ with 0s such that $len(lvl\_3) = n\_Rwaves$
3  randomly fill $lvl\_2$ with $n \in [1, 7]$ such that
     $sum(lvl\_2) \geq n\_Rwaves$ and $sum(lvl\_2) - lvl\_2$[-1] $< n\_Rwaves$
4  if $sum(lvl\_2) = n\_Rwaves$ then
5      fill $lvl\_1$ with 1s such that $len(lvl\_1) = n\_Rwaves + len(lvl\_2)$
         or $len(lvl\_1) = n\_Rwaves + len(lvl\_2) - 1$
6  if $sum(lvl\_2) > n\_Rwaves$ then
7      fill $lvl\_1$ with 1s such that
         $len(lvl\_1) = n\_Rwaves + len(lvl\_2) - 1$
```

The algorithm for 2b is first shown in the correct version (Algorithm A.2) that leads to no block unused in any case. The implemented version for this thesis below (Algorithm A.3) has a slight mistake in the case where $sum(lvl\_2) = 2 \cdot n\_Rwaves$. In this case when level 2 has a 2:1 block at the end, i.e. a 1 as the last entry of the list, the option to fill level 1 with 0s such that $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$ would lead to that last 2:1 block being unused, i.e. there is no incoming signal for it. That is why there has to be a distinction made between the case where there is a 1 at the end of level 2 lvl_2[-1] $= 1$ and where there is not lvl_2[-1] > 1. The former case has to leave out the option mentioned above that could lead to an unused block. This however should have no effect on any of the results because even in the case where a network suggests a result with an additional 1 at the end of level 2 this would first not lead to any error because the forward model only takes level 2 as an input and assumes maximum atrial signals and second would only lead to one additional RR-interval when used with the forward model which will be ignored when comparing with the conditional variable that was used to generate the network suggestion because the conditional variable is truncated according to the desired amount of R-waves in the fifth data generation procedure step of Section 3.2. Furthermore, it has neither an effect on the transformation of the block patterns to a sequence suitable for training with a RNN in the block pattern approaches since that procedure stops with the last signal that is conducted through level 3 nor does it have an effect on the transformation to a signal matrix because the correction algorithm before the transformation ignores the extra entry at the end (see A.6).

Algorithm A.2: Generate MAVB 2b Block Pattern

```
   Input: $n\_Rwaves$
   Output: The block pattern lists: $lvl\_1$, $lvl\_2$, $lvl\_3$
1  Initialize empty block pattern lists $lvl\_1$, $lvl\_2$, $lvl\_3$
2  fill $lvl\_3$ with 1s such that $len(lvl\_3) = n\_Rwaves$
3  randomly fill $lvl\_2$ with $n \in [1, 7]$ such that
     $sum(lvl\_2) \geq 2 \cdot n\_Rwaves - 1$ and
     $sum(lvl\_2) - lvl\_2$[-1] $\leq 2 \cdot n\_Rwaves - 1$
4  if $sum(lvl\_2) = 2 \cdot n\_Rwaves - 1$ then
5      fill $lvl\_1$ with 0s such that
         $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2)$ or
         $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$
6  if $sum(lvl\_2) = 2 \cdot n\_Rwaves$ then
7      if $lvl\_2$[-1] $= 1$ then
8          fill $lvl\_1$ with 0s such that
             $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2)$ or
             $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2) - 1$
9      if $lvl\_2$[-1] $> 1$ then
10         fill $lvl\_1$ with 0s such that
             $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2)$ or
             $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2) - 1$ or
             $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$
11 if $sum(lvl\_2) > 2 \cdot n\_Rwaves$ then
12     if $sum(lvl\_2) - lvl\_2$[-1] $= 2 \cdot n\_Rwaves - 1$ then
13         fill $lvl\_1$ with 0s such that
             $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2)$
14     if $sum(lvl\_2) - lvl\_2$[-1] $< 2 \cdot n\_Rwaves - 1$ then
15         fill $lvl\_1$ with 0s such that
             $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$ or
             $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2) - 1$
```

Algorithm A.3: Generate MAVB 2b Block Pattern (Implemented Version)

```
   Input: $n\_Rwaves$
   Output: The block pattern lists: $lvl\_1$, $lvl\_2$, $lvl\_3$
 1 Initialize empty block pattern lists $lvl\_1$, $lvl\_2$, $lvl\_3$
 2 fill $lvl\_3$ with 1s such that $len(lvl\_3) = n\_Rwaves$
 3 randomly fill $lvl\_2$ with $n \in [1, 7]$ such that
     $sum(lvl\_2) \geq 2 \cdot n\_Rwaves - 1$ and
     $sum(lvl\_2) - lvl\_2[-1] \leq 2 \cdot n\_Rwaves - 1$
 4 if $sum(lvl\_2) = 2 \cdot n\_Rwaves - 1$ then
 5     fill $lvl\_1$ with 0s such that
         $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2)$ or
         $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$
 6 if $sum(lvl\_2) = 2 \cdot n\_Rwaves$ then
 7     fill $lvl\_1$ with 0s such that
         $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2)$ or
         $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2) - 1$ or
         $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$
 8 if $sum(lvl\_2) > 2 \cdot n\_Rwaves$ then
 9     if $sum(lvl\_2) - lvl\_2[-1] = 2 \cdot n\_Rwaves - 1$ then
10         fill $lvl\_1$ with 0s such that
             $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2)$
11     if $sum(lvl\_2) - lvl\_2[-1] < 2 \cdot n\_Rwaves - 1$ then
12         fill $lvl\_1$ with 0s such that
             $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$ or
             $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2) - 1$
```

Algorithm A.4: Generate MAVB 2c Block Pattern

```
  Input: $n\_Rwaves$
  Output: The block pattern lists: $lvl\_1$, $lvl\_2$, $lvl\_3$
1 Initialize empty block pattern lists $lvl\_1$, $lvl\_2$, $lvl\_3$
2 fill $lvl\_3$ with 0s such that $len(lvl\_3) = n\_Rwaves$
3 randomly fill $lvl\_2$ with $n \in [0, 1]$ such that $len(lvl\_2) = n\_Rwaves$
4 if $lvl\_2$[-1] = 0 then
5     randomly fill $lvl\_1$ with $n \in [1, 2]$ such that
        $sum(lvl\_1) = n\_Rwaves + sum(lvl\_2)$
6 if $lvl\_2$[-1] = 1 then
7     randomly fill $lvl\_1$ with $n \in [1, 2]$ such that
        $sum(lvl\_1) = n\_Rwaves + sum(lvl\_2)$ or
        $sum(lvl\_1) = n\_Rwaves + sum(lvl\_2) - 1$
```

Algorithm A.5: Generate MAVB 3 Block Pattern

```
   Input: $n\_Rwaves$
   Output: The block pattern lists: $lvl\_1$, $lvl\_2$, $lvl\_3$
1  Initialize empty block pattern lists $lvl\_1$, $lvl\_2$, $lvl\_3$
2  randomly fill $lvl\_3$ with $n \in [0, 1]$ such that $len(lvl\_3) = n\_Rwaves$
3  if $lvl\_3$[-1] = 0 then
4      randomly fill $lvl\_2$ with $n \in [0, 1]$ such that
         $len(lvl\_2) = n\_Rwaves + sum(lvl\_3)$
5  if $lvl\_3$[-1] = 1 then
6      randomly fill $lvl\_2$ with $n \in [0, 1]$ such that
         $len(lvl\_2) = n\_Rwaves + sum(lvl\_3)$ or
         $len(lvl\_2) = n\_Rwaves + sum(lvl\_3) - 1$
7  if $lvl\_2$[-1] = 0 then
8      randomly fill $lvl\_2$ with $n \in [1, 2]$ such that
         $sum(lvl\_1) = len(lvl\_2) + sum(lvl\_2)$
9  if $lvl\_3$[-1] = 1 then
10     randomly fill $lvl\_2$ with $n \in [1, 2]$ such that
         $sum(lvl\_1) = len(lvl\_2) + sum(lvl\_2)$ or
         $sum(lvl\_1) = len(lvl\_2) + sum(lvl\_2) - 1$
```

Algorithm A.6: Block Pattern Correction

```
Input: $lvl\_1$, $lvl\_2$, $lvl\_3$, $bp\_type$, $n\_Rwaves$
Output: The corrected block pattern lists: $lvl\_1$, $lvl\_2$, $lvl\_3$
1  if $bp\_type = 1$ or $bp\_type = 2a$ then
2      subtract from last entry of $lvl\_2$ such that
         $sum(lvl\_2) = n\_Rwaves$
3  if $bp\_type = 2b$ then
4      if $sum(lvl\_2) = 2 \cdot n\_Rwaves$ and
         $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$ then
5          subtract 1 from last entry of $lvl\_2$
6      if $sum(lvl\_2) > 2 \cdot n\_Rwaves$ then
7          if $sum(lvl\_2) - lvl\_2[-1] < 2 \cdot n\_Rwaves - 1$ then
8              if $len(lvl\_1) = 2 \cdot n\_Rwaves + len(lvl\_2) - 1$ then
9                  subtract from last entry of $lvl\_2$ such that
                     $sum(lvl\_2) = 2 \cdot n\_Rwaves$
10             if $len(lvl\_1) = 2 \cdot n\_Rwaves - 1 + len(lvl\_2) - 1$ then
11                 subtract from last entry of $lvl\_2$ such that
                     $sum(lvl\_2) = 2 \cdot n\_Rwaves - 1$
12         if $sum(lvl\_2) - lvl\_2[-1] = 2 \cdot n\_Rwaves - 1$ then
13             subtract from last entry of $lvl\_2$ such that
                 $sum(lvl\_2) = 2 \cdot n\_Rwaves$
```

Appendix B

Models

Subnetwork Architecture of All Models

- bp_cINN: feed forward NN with 2 hidden layers of size 2048
- bp_rcINN: LSTM with 2 layers and hidden size of 64
- bp_rcINN_matching: LSTM with 2 layers and hidden size of 64
- sequence cINN for bp_rcINN and bp_rcINN_matching: feed forward NN with 2 hidden layers of size 512
- matching cINN for bp_rcINN and bp_rcINN_matching: feed forward NN with 2 hidden layers of size 512
- bp_cINN_multi 1: feed forward NN with 2 hidden layers of size 512
- bp_cINN_multi 2a: feed forward NN with 2 hidden layers of size 512
- bp_cINN_multi 2b: feed forward NN with 2 hidden layers of size 1024
- bp_cINN_multi 2c: feed forward NN with 2 hidden layers of size 512
- bp_cINN_multi 3: feed forward NN with 2 hidden layers of size 1024
- signal_cINN: feed forward NN with 2 hidden layers of size 1024
- signal_rcINN: LSTM with 2 layers and hidden size of 64
- signal_rcINN_matching: LSTM with 2 layers and hidden size of 32

- sequence cINN for signal_rcINN and signal_rcINN_matching: feed forward NN with 2 hidden layers of size 512
- matching cINN for signal_rcINN and signal_rcINN_matching: feed forward NN with 2 hidden layers of size 512

Appendix C

Solutions

The following shows some example alternative solutions generated with the bp_rcINN_matching approach. These were picked to show both how good and how bad the solutions can be. Surrounded by equal signs is the RR-interval list that the forward model generates with the input variables given below the RR-interval list. Below that, surrounded by dashed lines, is the RR-interval list that the forward model generates with the alternative solution of the network given below the RR-interval list. The last value within the dashed lines is the MAE between the two RR-interval lists.

```
=====================================================
RR: [456.5 456.5 456.5 456.5 456.5 456.5 405.  456.5 456.5 456.5 456.5 456.5]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [7, 7],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 1
AA: 393
alpha: 12
=====================================================
--------------------------
RR: [456.53216235 456.53216235 456.53216235 456.53216235 456.53216235
 456.53216235 452.2474556  456.53216235 456.53216235 456.53216235
 456.53216235 456.53216235]
Bp: [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [7, 7],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2a
AA: 199.46502685546875
alpha: 53.31740188598633

MAE: 3.966770119137209
--------------------------
```

```
=====================================================
RR: [426.5 426.5 746.  746.  639.  426.5 426.5 639. ]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [3, 1, 1, 3, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 1
AA: 373
alpha: 266
=====================================================
--------------------------
RR: [425.22537231 425.22537231 743.99357605 744.29522705 637.83805847
 425.22537231 425.22537231 637.83805847]
Bp: [[1, 1, 1, 1, 2, 1, 2, 1, 1, 2], [0, 0, 1, 1, 1, 0, 0, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 3
AA: 212.61268615722656
alpha: 106.155517578125

MAE: 1.3916988372802734
--------------------------

=====================================================
RR: [566.         319.         385.33333333 385.33333333 385.33333333
 319.         349.2        349.2        349.2        349.2
 349.2        319.        ]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [2, 4, 6, 6],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 1
AA: 295
alpha: 24
=====================================================
--------------------------
RR: [567.42919922 263.75711823 303.67208099 378.28613281 378.28613281
 378.28613281 378.28613281 378.28613281 378.28613281 378.28613281
 378.28613281 263.75711823]
Bp: [[1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2],[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 189.14306640625
alpha: 74.61405181884766

MAE: 34.365617752075195
--------------------------

=====================================================
RR: [550.         440.         403.33333333 403.33333333 403.33333333
 440.         440.         440.         660.         440.
 550.         660.         440.         440.         440.
 440.         403.33333333 403.33333333 403.33333333 440.
 374.         374.         374.        ]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[2, 4, 3, 1, 2, 1, 3, 4, 6],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 1
AA: 330
alpha: 110
=====================================================
-------------------------
RR: [377.94812012 377.94812012 377.94812012 377.94812012 377.94812012
 377.94812012 621.66760635 323.20269394 377.94812012 377.94812012
 377.94812012 755.89624023 755.89624023 621.66760635 566.92218018
 323.20269394 377.94812012 377.94812012 377.94812012 377.94812012
 566.92218018 377.94812012 377.94812012]
Bp: [[1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 188.97406005859375
alpha: 54.745426177978516

MAE: 102.74510192871094
-------------------------

=====================================================
RR: [596.5 596.5 695.  596.5 596.5 695. ]
Bp: [[1, 1, 1, 1, 1, 1, 1, 1, 1], [3, 3, 5], [0, 0, 0, 0, 0, 0, 0]]
MAVB: 2a
AA: 236
alpha: 223
=====================================================
-------------------------
RR: [594.94107056 594.94107056 680.70560455 594.94107056 594.94107056
 707.49022675]
Bp: [[1, 2, 2, 1, 2, 2, 2, 1], [1, 1, 1, 1, 1, 1, 1], [0, 0, 0, 0, 0, 0, 1]]
MAVB: 3
AA: 198.31369018554688
alpha: 85.76453399658203

MAE: 5.503389994303386
```

```
--------------------------

======================================================
RR: [519. 519. 519. 519. 519.]
Bp: [[1, 1, 1, 1, 1, 1, 1], [6], [0, 0, 0, 0, 0, 0]]
MAVB: 2a
AA: 234
alpha: 213
======================================================
--------------------------
RR: [518.15823364 532.7605896  518.15823364 532.7605896  518.15823364]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0], [2, 2, 2], [0, 0, 0, 0, 0, 0]]
MAVB: 1
AA: 350.3062744140625
alpha: 182.45431518554688

MAE: 6.009295654296784
--------------------------

======================================================
RR: [920.5 920.5 920.5 920.5 842.  887.2 887.2 887.2 887.2 887.2 842.  920.5
 920.5 920.5]
Bp: [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[5, 6, 5], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2a
AA: 377
alpha: 88
======================================================
--------------------------
RR: [1087.33940125  937.10853577  867.62054443  867.62054443  867.62054443
  867.62054443  867.62054443  867.62054443  867.62054443  867.62054443
  867.62054443  867.62054443  867.62054443  867.62054443]
Bp: [[1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 289.20684814453125
alpha: 219.71885681152344

MAE: 42.64168439592634
--------------------------

======================================================
RR: [ 846.75  846.75  846.75  846.75  861.    985.5   985.5  1416.    861.
  893.    893.    893.    861.    893.    893.    893.    861.    985.5
  985.5   861.    893.    893.    893.    861.  ]
Bp: [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[5, 3, 1, 4, 4, 3, 4, 6], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2a
AA: 354
alpha: 153
=======================================================
--------------------------
RR: [1073.87063599  838.54101563  951.24337769  807.88420105  807.88420105
 1073.87063599 1247.8866272  1073.87063599 1073.87063599  838.54101563
  951.24337769  838.54101562  951.24337769  838.54101562  951.24337769
  838.54101562  951.24337769  807.88420105  807.88420105  920.58656311
  777.22738647  777.22738647  920.58656311  777.22738647]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[2, 4, 5, 2, 1, 2, 2, 4, 4, 4, 4, 5, 7, 7],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
MAVB: 2b
AA: 357.9568786621094
alpha: 174.0159912109375

MAE: 108.75391133626208
--------------------------

=======================================================
RR: [ 675.   675.   675.  1000.8  675.6  675.6 1000.5  675.   675.  1000.5
  675.   675.   675.  1000.5  675.   675. ]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[7, 6, 7, 7, 7],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
MAVB: 2b
AA: 336
alpha: 327
=======================================================
--------------------------
RR: [ 671.88336182  671.88336182  671.88336182 1007.82504272  671.88336182
  671.88336182 1007.82504272  671.88336182  671.88336182 1007.82504272
  671.88336182  671.88336182  671.88336182 1007.82504272  671.88336182
  671.88336182]
Bp: [[1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1],
[0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 335.9416809082031
alpha: 188.1779327392578

MAE: 4.22498931884769
--------------------------

=====================================================
RR: [453.33333333 453.33333333 453.33333333 472.         464.        ]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [7, 6], [1, 1, 1, 1, 1, 1]]
MAVB: 2b
AA: 200
alpha: 40
=====================================================
--------------------------
RR: [454.81648382 454.81648382 454.81648382 454.81648382 454.81648382]
Bp: [[1, 1, 1, 1, 1, 1], [7], [0, 0, 0, 0, 0, 0]]
MAVB: 2a
AA: 200.53167724609375
alpha: 78.5445785522461

MAE: 6.1632967631022435
--------------------------

=====================================================
RR: [ 722.66666667  722.66666667  722.66666667  976.5         737.
  976.5         737.          737.          983.66666667  751.33333333
 1388.         1041.          998.          998.          737.        ]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[7, 5, 5, 4, 1, 1, 2, 3, 5],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
MAVB: 2b
AA: 347
alpha: 261
=====================================================
--------------------------
RR: [ 696.25775146  696.25775146  696.25775146 1044.3866272   696.25775146
 1044.3866272   696.25775146  696.25775146 1044.3866272   696.25775146
 1392.51550293 1044.3866272  1044.3866272  1044.3866272   696.25775146]
Bp: [[1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 2, 2, 2, 1],
[0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
```

```
AA: 348.1288757324219
alpha: 171.31442260742188

MAE: 39.62932807074661
-------------------------

=====================================================
RR: [1200.   911.   911.  1089.  1089.  1022.  1033.5  911.  1033.5  888.8
  888.8 1022.4  948.  1274.  1052.   948.  1052.   948.  1089.  1089.
  911.   911.  1052.   948. ]
Bp: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[2, 5, 3, 3, 5, 6, 4, 1, 4, 4, 3, 5, 4],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
MAVB: 2b
AA: 400
alpha: 178
=====================================================
-------------------------
RR: [ 916.82272339 1418.927948    916.82272339 1418.927948    916.82272339
  938.66965485  938.66965485  938.66965485  938.66965485  916.82272339
  906.6524353   906.6524353   906.6524353   906.6524353   906.6524353
  916.82272339  885.30762227  885.30762227  885.30762227  885.30762227
  885.30762227  885.30762227  916.82272339  885.30762227]
Bp: [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 2, 2, 5, 6, 7, 7],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2a
AA: 389.2917785644531
alpha: 138.23916625976562

MAE: 134.6900199042425
-------------------------

=====================================================
RR: [425. 397. 425. 548. 822. 822. 822. 397. 973. 822. 822. 548.]
Bp: [[2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1],
[0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 274
```

```
alpha: 123
=====================================================
--------------------------
RR: [420.5271759  399.62565613 420.5271759  546.76855469 820.15283203
 820.15283203 820.15283203 399.62565613 967.29573059 820.15283203
 820.15283203 546.76855469]
Bp: [[2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 1],
[0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 3
AA: 273.38427734375
alpha: 126.24137878417969

MAE: 2.6333300272623696
--------------------------

=====================================================
RR: [ 514.  936.  514.  936.  514.  580. 1160. 1160.  936.  870.  514. 1160.
  870.  356.  514.  870.]
Bp: [[2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 2, 2, 2],
[0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 290
alpha: 66
=====================================================
--------------------------
RR: [ 385.2829895   892.15411377  506.87112427  892.15411377  594.76940918
  594.76940918  980.05239868 1101.64053345  892.15411377  892.15411377
  594.76940918 1189.53881836  892.15411377  385.2829895   506.87112427
  892.15411377]
Bp: [[1, 2, 2, 2, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 2, 2],
[0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
MAVB: 3
AA: 297.38470458984375
alpha: 87.89828491210938

MAE: 50.90074157714844
--------------------------

=====================================================
RR: [1493. 1143.  412. 1493.  412. 1493. 1174. 1524.  762.  731. 1143.  412.
 1493.  412.  731. 1174. 1143. 1493.  412.  762. 1143. 1493.  412.]
Bp: [[1, 1, 2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1],
[1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 381
alpha: 350
=====================================================
--------------------------
RR: [1495.16003418 1121.37002563 1121.37002563 1121.37002563  453.07236481
 1415.87767792 1200.6523819  1415.87767792  747.58001709  747.58001709
 1121.37002563  453.07236481 1415.87767792  453.07236481  668.29766083
 1121.37002563 1121.37002563 1121.37002563  453.07236481  668.29766083
 1121.37002563 1495.16003418  747.58001709]
Bp: [[1, 1, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1],
[1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 3
AA: 373.7900085449219
alpha: 79.28235626220703

MAE: 111.8431920590608
--------------------------

=====================================================
RR: [600. 600. 600. 600. 746. 600. 254.]
Bp: [[1, 2, 2, 2, 2, 1, 2, 2], [0, 0, 0, 0, 1, 1, 1, 1, 0, 0],
[1, 1, 0, 0, 0, 0, 0, 0]]
MAVB: 3
AA: 200
alpha: 146
=====================================================
--------------------------
RR: [595.359375   595.359375   595.359375   595.359375   730.90405273
 595.359375   261.36157227]
Bp: [[1, 2, 2, 2, 2, 1, 2, 2], [1, 1, 1, 1, 1, 1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 198.453125
alpha: 135.544677734375

MAE: 6.52294921875
--------------------------

=====================================================
RR: [1428.  476.  714. 1428.  952. 1666.  819.  371.  952.  952.  343.  371.]
Bp: [[1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2],
[1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1],
[1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0]]
MAVB: 3
AA: 238
alpha: 105
=====================================================
--------------------------
RR: [1292.6204834   646.3102417   646.3102417  1292.6204834   969.46536255
 1292.6204834   646.3102417   461.33535767  969.46536255  969.46536255
  508.13000488  461.33535767]
Bp: [[1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2],
[1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 323.1551208496094
alpha: 138.18023681640625

MAE: 121.0854263305664
--------------------------

=====================================================
RR: [ 391. 1044.  391.  470. 1252.  470. 2009.  965.  574. 1148. 1044. 1826.
  861. 1148.  470. 1826.]
Bp: [[2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 2, 2, 1],
[0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0],
[0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0]]
MAVB: 3
AA: 287
alpha: 183
=====================================================
--------------------------
RR: [ 569.31663513  720.09729004  510.82929993  569.31663513 1440.19458008
  510.82929993 1289.41392517 1440.19458008 1080.14593506 1230.92658997
  569.31663513  720.09729004 1440.19458008  720.09729004 1230.92658997
 1289.41392517]
Bp: [[2, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1],
[0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
MAVB: 2c
AA: 360.04864501953125
alpha: 150.78065490722656

MAE: 413.71489810943604
--------------------------
```

Bibliography

[1] O. Rippel and R. Prescott Adams, “High-Dimensional Probability Estimation with Deep Density Models,” *arXiv e-prints*, p. arXiv:1302.5125, Feb. 2013.

[2] D. Jimenez Rezende and S. Mohamed, “Variational Inference with Normalizing Flows,” *arXiv e-prints*, p. arXiv:1505.05770, May 2015.

[3] L. Dinh, J. Sohl-Dickstein, and S. Bengio, “Density estimation using Real NVP,” *arXiv e-prints*, p. arXiv:1605.08803, May 2016.

[4] L. Dinh, D. Krueger, and Y. Bengio, “NICE: Non-linear Independent Components Estimation,” *arXiv e-prints*, p. arXiv:1410.8516, Oct. 2014.

[5] D. P. Kingma and P. Dhariwal, “Glow: Generative Flow with Invertible 1x1 Convolutions,” *arXiv e-prints*, p. arXiv:1807.03039, July 2018.

[6] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe, “Analyzing Inverse Problems with Invertible Neural Networks,” *arXiv e-prints*, p. arXiv:1808.04730, Aug. 2018.

[7] S. J. Hwang, Z. Tao, W. H. Kim, and V. Singh, “Conditional Recurrent Flow: Conditional Generation of Longitudinal Samples with Applications to Neuroimaging,” *arXiv e-prints*, p. arXiv:1811.09897, Nov. 2018.

[8] F. Kehrle, *Inverse Simulation for Cardiac Arrhythmia*. PhD thesis, Otto von Guericke University Magdeburg, 2018.

[9] “Heart-arrhythmia-modelling-with-invertible-neural-networks.” `https://github.com/MFRIbrahim/Heart-Arrhythmia-Modelling-with-Invertible-Neural-Networks`.

[10] T. G. Laske, M. Shrivastav, and P. A. Iaizzo, *Handbook of Cardiac Anatomy, Physiology, and Devices.* Springer International Publishing Switzerland, 3rd ed., 2015.

[11] H.-C. Pape, A. Kurtz, and S. Silbernagl, *Physiologie.* Georg Thieme Verlag KG, 9th ed., 2019.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] G. Cybenko, "Approximation by superposition of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[14] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[15] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The Expressive Power of Neural Networks: A View from the Width," *arXiv e-prints*, p. arXiv:1709.02540, Sept. 2017.

[16] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A Critical Review of Recurrent Neural Networks for Sequence Learning," *arXiv e-prints*, p. arXiv:1506.00019, May 2015.

[17] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to Construct Deep Recurrent Neural Networks," *arXiv e-prints*, p. arXiv:1312.6026, Dec. 2013.

[18] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv e-prints*, p. arXiv:1406.1078, June 2014.

[19] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv e-prints*, p. arXiv:1312.6114, Dec. 2013.

[20] 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*, p. arXiv:1406.2661, June 2014.

[21] L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, "Guided Image Generation with Conditional Invertible Neural Networks," *arXiv e-prints*, p. arXiv:1907.02392, July 2019.

[22] "Framework for easily invertible architectures." `https://github.com/VLL-HD/FrEIA`, Version: November 17, 2020.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[24] S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Köthe, "BayesFlow: Learning complex stochastic models with invertible neural networks," *arXiv e-prints, IEEE Transactions on Neural Networks and Learning Systems (in press)*, p. arXiv:2003.06281, Mar. 2020.