Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Jakob Benjamin Huhle

born in Munich (Germany)

2024

Reproduction of AdEx dynamics on neuromorphic hardware through data embedding and simulation-based inference

This Bachelor Thesis has been carried out by Jakob Benjamin Huhle at the
Kirchhoff-Institute for Physics in Heidelberg
under the supervision of
Dr. Johannes Schemmel

Abstract

When attempting to replicate a specific observation in modeling, identifying the appropriate parameterization for the system is often challenging. In this thesis, we conducted experiments on the neuromorphic hardware BrainScaleS-2. The objective of this study was to identify suitable hardware parameters that enable the successful emulation of a given voltage trace on the hardware. Previously, the SNPE algorithm has proven to be suitable for handling noisy observational data. To this end, we employed different data embedding techniques to compress the original observed data, particularly focusing on autoencoders. The pre-trained encoder is designed to transform the observed data into a lower-dimensional representation, which is then fed into the SNPE algorithm. The algorithm is tasked with approximating the posterior distributions of the model parameters, which are subsequently analyzed and compared. The methodology was first tested for simulations and then for emulations on hardware. Inference was performed on two, four, and eight hardware parameters. Results have shown that inference is successful for up to four dimensions, while the outcomes for eight dimensions seem promising but require further investigation.

Zusammenfassung

Bei dem Versuch, eine spezifische Beobachtung im Modellieren zu replizieren, ist es oft eine Herausforderung, die geeignete Parametrisierung für das System zu bestimmen. In dieser Arbeit führten wir Experimente auf der neuromorphen Hardware BrainScaleS-2 durch. Das Ziel dieser Untersuchung bestand darin, geeignete Hardwareparameter zu identifizieren, mit denen eine zuvor gegebene Spannungszeitreihe erfolgreich auf der Hardware emuliert werden kann. Wie zuvor gezeigt wurde, kann der SNPE-Algorithmus mit rauschbehafteten Beobachtungsdaten umgehen. Wir wendeten mehrere Methoden an, um die Dimensionalität unserer beobachteten Daten zu reduzieren. Dabei legten wir einen besonderen Fokus auf Autoencoder. Der vortrainierte Encoder ist darauf konzipiert, die beobachteten Daten in eine niederdimensionale Darstellung zu komprimieren, die dann dem SNPE-Algorithmus übergeben wird. Der Algorithmus hat dann die Aufgabe, die Wahrscheinlichkeitsverteilung der Modellparameter zu approximieren. Die resultierenden Verteilungen wurden anschließend analysiert und verglichen. Die Methodologie wurde zunächst für Simulationen und anschließend für Emulationen auf der Hardware getestet. Die Inferenz wurde für zwei, vier und acht Hardwareparameter durchgeführt. Unsere Ergebnisse zeigten, dass die Inferenz bis zu vier Dimensionen erfolgreich ist, während die Resultate für acht Dimensionen vielversprechend erscheinen, jedoch weiterer Untersuchungen bedürfen.

Contents

1. Introduction

> "The first and simplest emotion which we discover in the human mind, is curiosity."
>
> – Edmund Burke

Curiosity of the human mind about itself has been present in numerous cultures throughout history. Even the ancient Egyptians engaged in early forms of neuroscience research (Martín-Araguz et al., 2002). However, they did lack the adequate means to gain knowledge about the brain, leading to inaccurate assumptions. Since then, humanity has come a long way. Today, we even build computer architectures which aim to replicate and mimic the human brain. Those architectures are called "neuromophic hardware".

One major advantage of this kind of hardware is its ability to emulate the dynamics observed in biological nerve cells, or neurons. If the hardware can accurately replicate the physical properties, such as the membrane voltage, of biological neurons under different conditions, this indicates that it has been successfully designed in alignment with biological principles and effectively incorporates key aspects of brain functions. Furthermore, emulations on neuromorphic hardware are often faster and more energy-efficient than traditional simulations (Hasler and Marr, 2013). By accurately emulating neural systems, researchers can study neuronal behavior in various scenarios without needing to run resource-intensive simulations. This also reduces the need for time-consuming lab experiments, making it a powerful tool for neuroscience research.

In this thesis, the central objective is: given a target voltage trace, we want to find the most probable hardware parameters with which that trace can be replicated on the BrainScaleS-2 system (Pehle et al., 2022).

Therefore, we leverage the sequential neural posterior estimation (SNPE) algorithm (Greenberg, Nonnenmacher, and Macke, 2019). The SNPE algorithm aims to approximate the posterior distribution of the parameters by performing multiple emulations on the hardware itself. In neuroscience, the SNPE algorithm has primarily been applied to simulations of mathematical models of neural systems (Lueckmann, Pedro J Goncalves, et al., 2017). Furthermore, it has also been utilized on the BrainScaleS-2 system before, although for a different research question (Kaiser et al., 2023). Thus, this thesis presents the first approach to leveraging the SNPE algorithm for extracting possible hardware parameters from complete membrane voltage traces on BrainScaleS-2.

We aim to infer two, four, and eight parameters simultaneously and evaluate the algorithm's performance. Access to the posterior allows us to quantify the confidence in parameter estimations and unveil correlations, which might provide valuable insights into the properties of the hardware implementation. To perform parameter inference, it is often essential to compress the data before passing it to the SNPE algorithm, as fewer input data can improve inference results (Pedro J Goncalves et al., 2020). Thus, we explore various methods of data embedding which can learn a low-dimensional representation of the time series data. In this regard, we place a strong focus on autoencoders.

As a proof of concept, we first conduct our investigations on simulations before moving on to emulations using the hardware. This stepwise approach ensures that our inference pipeline is functioning correctly before transitioning to experiments on the BrainScaleS-2 system.

2. Background

2.1. Biological neuron

Neurons, or nerve cells, are the fundamental building blocks of biological computing systems, such as the brain. The human brain consists of approximately 100 billion neurons, which form a highly intricate neural network with trillions of connections between them (Gulati, 2015). A neuron's anatomy can be seen in figure 1. Generally speaking, it consists of dendrites, the soma, or cell body, and an axon. Through dendrites, the neuron receives information from other neurons, while it can send its own information through the axon and axon terminals to neighboring nerve cells. The cell body contains the nucleus with its genetic material and cellular organelles (Martin, 2021).

Communication between different cells occurs through so-called action potentials (see figure 1). These action potentials are significant local changes in the membrane voltage, caused by the exchange of ions between the outside (extracellular space) and the inside (intracellular space) of the cell. Under resting conditions, the membrane voltage remains at a constant resting potential of around $-70\,\mathrm{mV}$. An action potential results from transient changes, primarily due to the exchange of sodium (Na^+) and potassium (K^+) ions. With sufficient stimulation, the voltage can reach a critical threshold. At this point, the depolarization phase begins, where the membrane voltage rapidly shifts from around $-70\,\mathrm{mV}$ to $40\,\mathrm{mV}$. In the second phase, called repolarization, the voltage difference returns toward the resting potential. However, it can dip below the resting potential before rising back up to it. This phase is known as the refractory period. On the other hand, if the stimulus fails to raise the voltage above the critical threshold, no action potential will be triggered (Mark W Barnett, 2007).

Through the dendrites, a neuron receives information of action potentials from neighboring cells. These inputs are integrated, and if the cumulative change in membrane voltage reaches a threshold of approximately $-55\,\mathrm{mV}$, the neuron generates its own action potential. The depolarization of the membrane during the action potential creates local currents that depolarize adjacent sections of the axon membrane. In this way, the action potential propagates down the axon. At the end of the axon, at the axon terminals, the electrical pulse can be converted into the release of chemical neurotransmitters and transmitted via synapses[1] to the next neuron (Mark W Barnett, 2007).

[1] A synapse is the junction between two neurons, where the transfer of information occurs, typically through neurotransmitters that bridge the gap between the neurons.

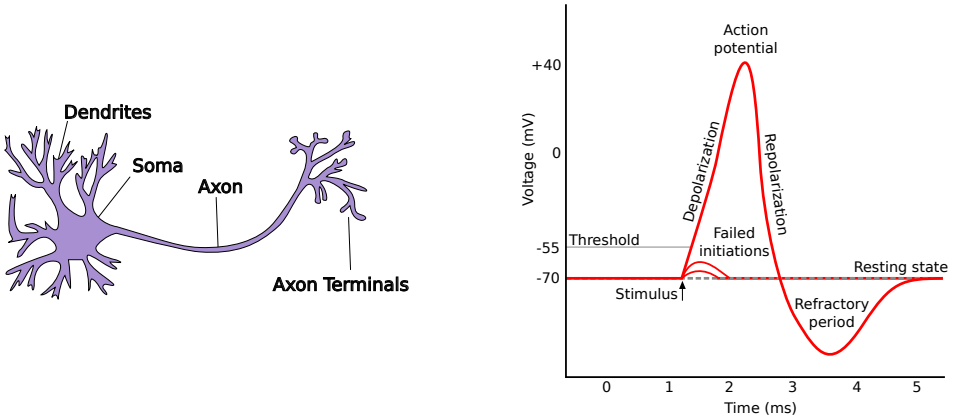


Figure 1: **Left:** generalized anatomy of a nerve cell; **Right:** Different phases of an action potential. If the membrane voltage reaches a threshold of about $-55\,\mathrm{mV}$, an action potential will be created. The voltage will then rapidly rise in the depolarization phase. After reaching the maximum value of about $40\,\mathrm{mV}$, it will decrease again in the repolarization phase. Lastly, it can further decrease in the refractory period, before reaching the value of the resting potential again. Source: adjusted from Iberri, 2007; Jarosz, 2009.

Through this process, signals can be exchanged, facilitating interaction between neurons. The form of an action potential is mostly identical. Therefore, much information is encoded in the sequence of multiple action potentials, forming a spike train or firing pattern (Bialek et al., 1991; Campo, 2020). Knowing about a neuron's spiking activity can teach us much about how a neuron processes and transmits information. Spiking patterns are linked to behavior and cognitive functions. Analyzing these patterns can provide insights into how specific neural circuits contribute to sensory perception, motor control, decision-making, and other cognitive processes (Shinomoto et al., 2009).

2.2. AdEx

Neuronal modeling holds profound importance in the realm of the neurosciences. It is the process of representing a biological neuron through a mathematical structure that incorporates its biophysical and geometrical characteristics. This structure is referred to as the neuron's model. The aims of developing accurate models include, for instance, estimating the biophysical parameters of real neurons or shedding light on how the information-processing properties of real neurons might function (Windhorst and Johansson, 1999).

Numerous models have been developed to capture a neuron's behavior under different circumstances. These models include, for instance, the Leaky

Integrate-and-Fire model, Hodgkin-Huxley models, and more complex biophysical conductance-based models (Gerstner and Kistler, 2002). A good model should preferably be simple in order to be computationally efficient to simulate and interpret. On the other hand, it should still be sophisticated enough to model diverse behaviors and types of neurons to closely approximate biological reality.

A suitable model that meets both of these criteria is the adaptive exponential integrate-and-fire (AdEx) model (Brette and Gerstner, 2005). While remaining relatively simple with only two differential equations and a reset condition, it is capable of modeling diverse neuronal firing patterns (Naud et al., 2008). Additionally, it has been shown that it can closely match direct measurements in cortical neurons (Jolivet et al., 2008).

The AdEx model is an advancement of the leaky integrate-and-fire model. The key differences lie in an additional exponential term and an adaptation variable. The exponential term allows the model to effectively handle fast input signals, while the adaptation is crucial for modeling the neuron's response to sustained stimuli (Fourcaud-Trocmé et al., 2003; Izhikevich, 2003). Adaptation refers to the neuron's ability to reduce or increase its firing rate over time, even when the input remains constant, mimicking real neural behavior. These adjustments make the model more biologically realistic.

The model consists of the following two differential equations:

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = -g_{\mathrm{L}}(V - E_L) + g_{\mathrm{L}}\Delta_{\mathrm{T}} \exp\left(\frac{V - V_{\mathrm{T}}}{\Delta_{\mathrm{T}}}\right) + I - w, \tag{1}$$

and

$$\tau_w \frac{\mathrm{d}w}{\mathrm{d}t} = a(V - E_{\mathrm{L}}) - w. \tag{2}$$

In addition, a reset condition ensures the updating of the relevant variables after a spike has occurred:

If $V > V_{\mathrm{th}}$, then

$$\begin{aligned} V &\to V_{\mathrm{r}} \\ w &\to w_{\mathrm{r}} = w + b. \end{aligned} \tag{3}$$

The model thus simulates the evolution of the membrane potential V and the adaptation current w over time when a constant current I is injected. Param-

eters of the model are the total capacitance C, the total leak conductance g_{L}, the effective rest potential E_{L}, the threshold slope factor Δ_{T} and the effective threshold potential V_{T}. These parameters are also known as scaling parameters because they affect the scaling of the time axis as well as the stretch and offset of the state variables. We can express C and g_{L} in terms of the time scale $\tau_m = \frac{C}{g_{\mathrm{L}}}$. After applying additional scaling, the two equations (1) and (2) can be simplified into a system of dimensionless variables and only four parameters (J. Touboul, 2008). These four parameters are called the bifurcation parameters and are directly proportional to the conductance a, the time constant τ_w, the spike-triggered adaptation b, and the reset potential V_{r}. By modifying these parameters, different firing patterns can emerge, which may, for example, vary in the inter-spike intervals[2] or in the form of the reset after a spike (Naud et al., 2008).

If the injected current I is strong enough to drive the membrane potential V above V_{T}, the potential diverges to infinity in finite time. In numerical simulations, the integration is halted when the membrane potential reaches the threshold V_{th} and the potential is reset to a value V_{r}.

Equation (2) describes the evolution of the adaptation current w. It incorporates both spike-triggered adaptation, achieved through the reset $w \to w + b$, and a linear coupling of the effective reset potential E_{L} with the voltage, mediated by the parameter a. Hence, the larger the adaptation variable b, the more difficult it becomes for the neuron to spike over time, even if the injected stimulus remains unchanged. The parameter a with a positive value may model ionic channels that tend to hyperpolarize the membrane. In contrast, a depolarizing effect occurs for negative values of a.

2.3. Analog neuromorphic hardware: the BrainScaleS-2 system

Brain-inspired computing has been a promising approach to explore alternative computer architectures beyond the standard von Neumann architecture (Indiveri et al., 2011). The general goal of constructing neuromorphic hardware consists of taking the biological brain as a model for computing. Compared to classical approaches, it can excel in several areas, such as high-speed modeling of large-scale neural systems and machine learning-inspired training of spiking neural networks (Billaudelle et al., 2022). Furthermore, it can offer more energy-efficient AI solutions (Mueller, 2014).

One realization of such neuromorphic hardware is the BrainScaleS-2 system (Pehle et al., 2022). This computing platform aims to replicate the biological

[2] An inter-spike interval is the time interval between two subsequent spikes.

brain by constructing a network of in-silicon neurons interconnected through plastic synapses. Instead of representing biological processes by changing a multitude of bits, the temporal evolution of physical quantities, such as current and voltage, directly corresponds to neuronal dynamics, enabling the analog hardware to emulate spiking neural networks. On the BrainScaleS-2 hardware, emulation operates at a 1000-fold accelerated time scale compared to the biological time domain.

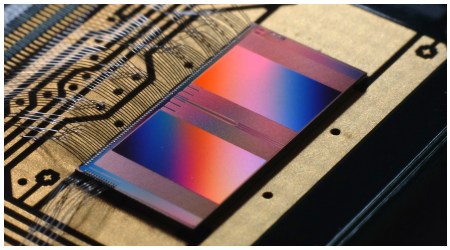
The heart of this system is the HICANN chip (see figure 2). The chip consists of four quadrants, each containing 128 neurons and a synaptic crossbar with 256 rows and 128 columns. In total, the chip hosts 512 neuron circuits. Digital processors are located at both the top and bottom of the chip, capable of reading and writing the digital state of their respective halves of the synaptic crossbar. Through field-programmable gate arrays (FPGAs), the chip can be configured in real time from a host computer. This allows experiments designed in software to be easily executed on the hardware. Observables like the emulated membrane voltage can be accessed via the FPGA on the host side. BrainScaleS-2 extensions for common frameworks like *PyNN* (Davison et al., 2009) and PyTorch (Paszke et al., 2019) exist, simplifying experiment design.

The system is designed to faithfully emulate the AdEx equations (see chapter 2.2). However, some deviations exist. For instance, the adaptation current w of equation (1) is implemented as a voltage on the hardware instead of as a current. For more details, see Billaudelle et al., 2022.

An on-chip Digital-to-Analog Converter provides 24 analog parameters for each neuron circuit, enabling precise tuning of experiments. It has a 10 bit[3] resolution, allowing most parameters to be configured with 1024 distinct values. All potentials and conductances from equation (1) can be controlled by these 24 parameters, allowing for the creation of rich neuronal dynamics.

Analog neuromorphic systems always suffer from temporal noise, fixed-pattern parameter deviations, and divergence from the original model equations. Variations in manufacturing conditions and materials used in the production impact those discrepancies. Calibration is employed to mitigate these disadvantages. Nevertheless, it cannot fully compensate for them, and variations in different neuron circuits may occur even on the same chip.

[3] Parameters are adjustable from 0 to 1022.



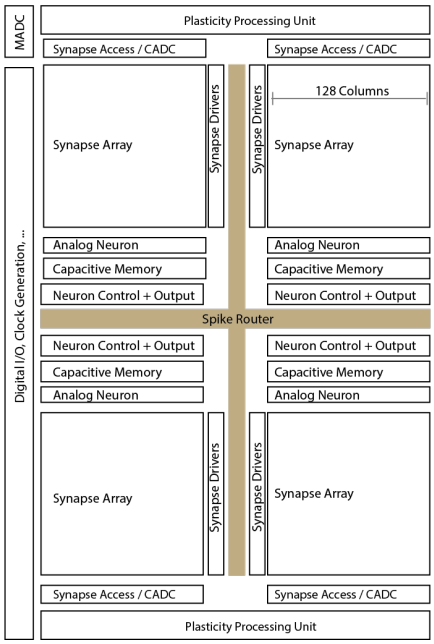


Figure 2: The HICANN chip. **Left:** photograph of the chip. **Right:** the schematic floorplan of the chip: two processor cores with access to the synaptic crossbar array are located on the top and bottom. In the middle, the 512 neuron circuits and analog parameter storage are arranged. Events generated by the neurons and external events are routed to the synapse drivers and to/from the digital I/O located on the left edge of the chip with the help of the event router. Source: Pehle et al., 2022.

2.4. Simulation-based inference

In our world, most observations, such as temperature readings or recordings of seismic and acoustic sound waves, are relatively easy to obtain. A common interest is understanding the pattern of creation for specific observations. However, determining the underlying cause of those observations can be a challenging task. In science, this problem is referred to as an "inverse problem". It is assumed that there exists a known mapping $T : \Theta \to X$ between the parameter space Θ and the observation space X that models a physical law or device. Unlike a direct problem, which involves computing an effect $T(\theta \in \Theta)$, the inverse problem involves identifying the specific cause θ responsible for creating the observed result $x \in X$ (Richter, 2020). In the context of this thesis, the observation x would be a voltage trace from a simulated or in-silicon neuron. The mapping T would involve simulating the AdEx equations or emulating them on the BrainScaleS-2 hardware using the relevant set of model or hardware parameters, which represent the cause θ.

Multiple algorithms have been developed to solve inverse problems computationally, e.g. evolutionary algorithms or particle swarm algorithms (Van Geit, De Schutter, and Achard, 2008). With the advancements in deep learning in recent years, simulation-based inference algorithms have become a prominent candidate for solving these types of problems (Cranmer, Brehmer, and Louppe, 2020). Given an observation x, these algorithms can be used to obtain the posterior distribution $p(\theta|x)$ to find the most probable parameters that could have generated this observation. A key component in these algorithms is the use of a simulator. The simulator takes a vector of parameters θ as input, calculates a series of internal states or latent variables $z_i \sim p_i(z_i|\theta, z_{<i})$ and outputs a data vector $x \sim p(x|\theta, z)$ (Cranmer, Brehmer, and Louppe, 2020).

Normally, Bayesian inference is leveraged to calculate the posterior:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{\int dz\ p(x|\theta')p(\theta')}. \tag{4}$$

However, it can be too expensive or even impossible to compute the likelihood $p(x|\theta)$ in some cases. The likelihood could be obtained by integrating over all possible trajectories through the latent space of the simulator

$$p(x|\theta) = \int dz\ p(x, z|\theta). \tag{5}$$

For a sequential data generation procedure, the joint probability density $p(x|\theta)$

of the data x and the latent variables z can be rewritten into

$$p(x, z|\theta) = p(x|\theta, z) \prod_i p_i(z_i|\theta, z_{<i}). \tag{6}$$

Thus, because of the immense number of possible latent variables z, it is impossible to compute the integral in equation (5) for simulators with large latent spaces. Instead, some simulation-based inference algorithms aim to approximate the posterior directly, bypassing the need for the likelihood, which leads to their classification as likelihood-free algorithms.

2.4.1. SNPE algorithm

One type of likelihood-free algorithm is the SNPE algorithm (Greenberg, Nonnenmacher, and Macke, 2019). It utilizes active learning, where the model itself selects the data it needs most to improve its performance. Given an observation $\hat{x}$, the SNPE algorithm allows for finding an approximation of the posterior distribution $p(\theta|\hat{x})$. Ideally, the most probable parameters θ of that posterior can reproduce the observation $\hat{x}$, when fed into the simulator. A target observation $\hat{x}$, a prior $p(\theta)$ and a model or simulator are taken as inputs. Similar to standard parameter search methods, the algorithm utilizes simulations. However, instead of filtering out specific simulations, it uses all of them to train artificial neural networks for conditional density estimation (Papamakarios, Pavlakou, and Murray, 2017). In the first step, the algorithm samples random parameters from the prior $\theta' \sim p(\theta)$. The simulator then uses these parameters to generate observations x', which is similar to sampling from the likelihood $x' \sim p(x|\theta)$. Next, a neural density estimator (NDE) is trained to learn the mapping between the sampled parameters and the generated observations, thereby approximating the posterior distribution. The NDE is a flexible set of probability distributions parameterized by a neural network, typically trained by minimizing the negative log-likelihood of the previously drawn samples. At this stage, one obtains a posterior distribution of the parameters for any observation x.

If one is only interested in a specific observation $\hat{x}$, the inference procedure can be repeated several times, making the algorithm sequential. In this case, the approximated posterior from the previous round is used as the proposal prior for the next round, which enhances the results. However, the posterior is then no longer amortized, meaning it cannot be reused for another observation x. If one wants to obtain the posterior for a new observation x, the entire procedure must be repeated from the beginning for that new observation.

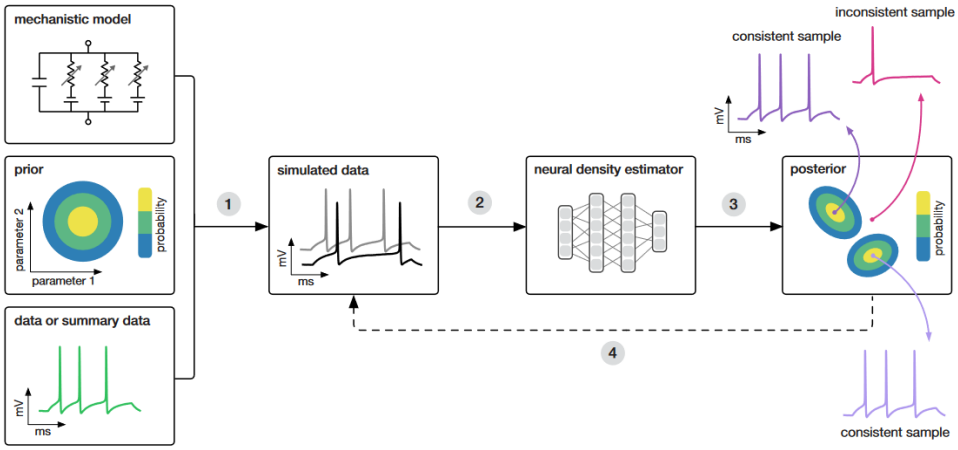


Figure 3: The functionality of the SNPE algorithm. The algorithm takes a mechanistic model, a prior for the model parameters, and the (embedded) data as inputs. SNPE then (1) draws random samples from the prior and performs simulation with those parameters; (2) trains a neural density estimator which should learn the correlation of the data and the model's parameters; (3) can approximate the posterior; (4) uses an initial estimate of the posterior as the new prior in subsequent inference rounds. Source: Pedro J Goncalves et al., 2020.

The functionality of the SNPE algorithm is displayed in figure 3.

Truncated Sequential Neural Posterior Estimation A potential improvement method for the parameter inference involves drawing samples from a truncated posterior (Deistler, Pedro J. Goncalves, and Macke, 2022). During the training of the NDE, there is a chance that the learned posterior approximation may extend beyond the support of the given prior. Consequently, many samples drawn from the posterior during the sampling step may fall outside the prior boundaries and must be rejected. This can considerably prolong the sampling process. To address this issue, one can choose to sample from a truncated posterior, where samples drawn from the prior are rejected if they do not lie within the support of the posterior. This approach ensures that the resulting proposals are proportional to the prior, allowing the neural network to be trained with maximum likelihood in each round. The functioning of the truncated SNPE algorithm is displayed in figure 4.

Posterior analysis After obtaining the approximated posterior, various techniques can be applied to assess its quality. One example are posterior predictive checkss (PPCs) (Berkhof, Mechelen, and Hoijtink, 2000). PPCs are a common safety check to verify that the approximated posterior is not poor. However, they do not confirm that the posterior is necessarily good, as it

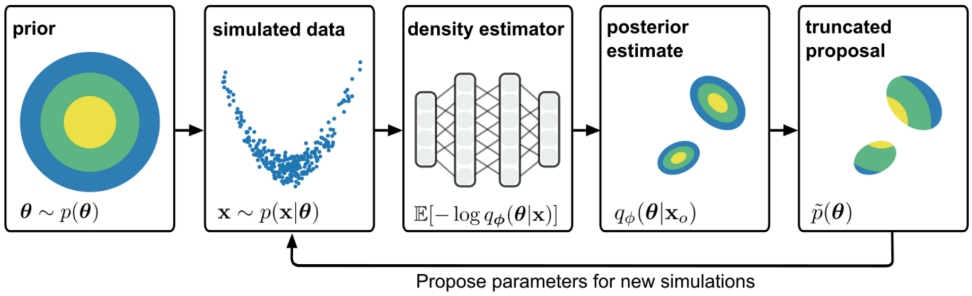


Figure 4: Truncated Sequential Neural Posterior Estimation. The algorithm begins by sampling from the prior distribution, running the simulator, and training a neural density estimator to approximate the posterior. In the standard SNPE algorithm, parameters are sampled from the approximate posterior but are rejected if they fall outside the support of the prior. Here, proposals are truncated versions of the prior and the neural density estimator can be trained with maximum likelihood in each round. Source: Deistler, Pedro J. Goncalves, and Macke, 2022.

can still be overconfident[4]. A PPC compares the data x_{gen}, generated using posterior samples $\theta_{\text{posterior}}$, with the observed data $\hat{x}$. If the inference was successful, the simulated data x_{gen} should resemble the observed data $\hat{x}$. PPCs help identify if any bias was introduced during inference and whether x_{gen} systematically differs from $\hat{x}$.

With the obtained posterior, one can not only infer the best parameters responsible for the specific observations but also detect correlations between different parameters. This allows us to analyze whether certain parameters can vary or need to be finely tuned, as compensation mechanisms may exist between them (Marder and Taylor, 2011). Pearson correlation coefficients can be calculated to quantify the strength and direction of linear relationships between parameters (Benesty et al., 2009). Additionally, conditional distributions can be extracted to further investigate parameter correlations and dependencies.

Another method for conducting posterior diagnostics is sensitivity analysis. Sensitivity analysis is a method used to determine which parameters significantly influence the behavior of a system. These parameters require precise tuning. By examining the posterior distribution, we can assess how uncertain or sensitive the model is to specific parameters, gaining deeper insights into the underlying dynamics of the system. This is done by making use of the *Active Subspace* (Constantine, Dow, and Q. Wang, 2014). In essence, we aim to identify directions in the parameter space where the posterior probability

[4]This means the posterior might underestimate uncertainty, resulting in predictions that appear more certain than they actually are.

exhibits significant changes. This is accomplished by computing the matrix

$$M = \mathbb{E}_{p(\theta|\hat{x})}\left[\nabla_\theta p(\theta|\hat{x})^T \nabla_\theta p(\theta|\hat{x})\right] \tag{7}$$

with the posterior $p(\theta|\hat{x})$ and then performing eigendecomposition. Strong eigenvalues resulting from this process indicate that the gradient of the posterior density is large along the corresponding eigenvector directions.

2.4.2. Data embedding

Time series voltage traces from in-silicon neurons serve as our observations for the parameter inference procedure. Hence, the NDE is supposed to learn the relationship between the model parameters and the resulting time series data. When dealing with such high-dimensional data (i.e. voltage traces with many datapoints) in deep learning, dimensionality reduction techniques are often applied to reduce computational costs. This approach offers several advantages. Firstly, a large amount of data increases the model's complexity, often requiring more layers, which in turn increases training time as more parameters need to be learned. Furthermore, high-dimensional data frequently contains redundant or irrelevant information, referred to as noise. By reducing the data to its most relevant features in a lower-dimensional space, noise is less likely to confuse the model. Additionally, dimensionality reduction helps reduce overfitting[5] and improves generalization, as the model focuses on essential features rather than learning noisy details.

This step is pivotal in striking the right balance between reducing information redundancy while retaining the relevant features essential for effective parameter inference. It ensures that the encoded features capture the crucial aspects of the data most useful for density estimation in simulation-based inference. In computational neuroscience, handcrafted summary statistics are often applied, extracting features such as the spike shape or the firing frequency, among others (Pedro J Goncalves et al., 2020). However, there is a risk that important characteristics of the data may be overlooked with such predefined features. Additionally, designing and implementing relevant features for new or different types of data can be time-consuming and labor-intensive.

Many algorithms and techniques have been developed to compress time series data. Typical choices are making use of different kinds of principle component

[5] The model learns the training data too well, capturing not only the underlying patterns but also the noise and irrelevant details, leading to poor generalization to new, unseen data.

analysis, singular value decomposition, maximum value unfolding or fourier or wavelet transforms, just to name a few (Ashraf et al., 2023; Oliveira et al., 2023; Chiarot and Silvestri, 2023). Another approach is to use deep learning methods for reducing the data size. We aim to have a dimensionality reduction technique that is highly flexible and can be applied to various types of time series data without having to adjust many parameters. Thus, in this thesis, we focused on unsupervised deep learning techniques, which are responsible for extracting the most relevant features of the data and encoding them into a low-dimensional feature space. For our purposes, autoencoders were implemented to encode the data.

Autoencoder An autoencoder (P. Li, Pei, and J. Li, 2023) is an unsupervised[6] deep learning model that learns a lower-dimensional representation of data given a large number of samples. Generally speaking, it consists of two main components: an encoder and a decoder (see figure 5). Both the encoder and the decoder are composed of neural networks. The specific architecture can vary significantly depending on the particular task. The encoder is responsible for learning the key features of the input data, thereby reducing the dimensionality of the data into a lower-dimensional space, which is referred to as the latent space. The decoder, on the other hand, is tasked with reconstructing the original signal solely from this latent space representation. Ideally, the output of the autoencoder should be as close as possible to the original input data.

To achieve this, the autoencoder is typically trained by minimizing the Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2\,, \tag{8}$$

with x_i being the datapoint at index i of the original time series and $\hat{x}_i$ the datapoint at index i of the reconstructed time series. The MSE thus measures the mean squared deviation between the original data and the reconstructed data, indicating how well the autoencoder reproduces the input data.

Autoencoders are a common method used to tackle a multitude of problems. They are applied in tasks such as classification (Zheng et al., 2016), anomaly detection (Guo et al., 2018; Chen et al., 2020), forecasting (Sagheer and Kotb, 2019), and synthetic data generation (Wan, Zhang, and He, 2017). Depending

[6] Unsupervised learning refers to a type of machine learning where the model learns from unlabeled data, without explicit guidance on the correct output.

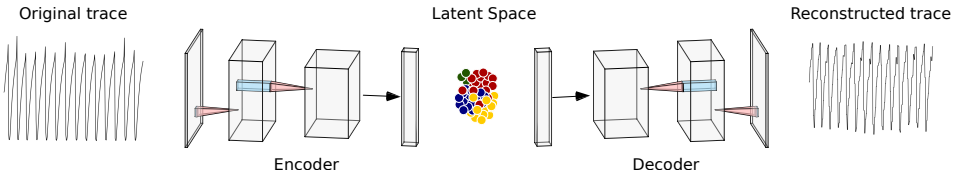


Figure 5: Structure and functionality of an autoencoder. Data is fed into the encoder and thus broken down into the most relevant features in the latent space. The decoder tries to reconstruct the original data solely from the latent space representation. The MSE loss is often used to evaluate its performance.

on the specific problem at hand, their design, training, and architecture can vary greatly.

In this thesis, both a convolutional autoencoder and a combination of a convolutional and a recurrent autoencoder were implemented. As the names suggest, convolutional and recurrent layers are used in the encoder and decoder parts of the architecture.

A convolutional layer performs matrix multiplication which uses filters on the input data. The inputs are convolved with the learnable kernels and added with biases to generate the output features. The following equation describes a single 1-dimensional convolutional layer for the j^{th} output feature X_j^l in layer l (F. Wang et al., 2019):

$$X_j^l = f\left(\sum_{i \in M_j} X_i^{l-1} * W_{ij}^l + b_j^l\right). \tag{9}$$

W_{ij}^l resembles the weight that connects X_j^l and X_i^{l-1}. M_j is the connection between X_j^l and the output features of the previous layer. The convolution operation, which generates the high-level representations of the input, is represented by $*$. Lastly, b_j^l is the corresponding bias. The resulting output is then passed into an activation function $f(\cdot)$, which, among other things, introduces non-linearity and controls the output range. Small kernels are able to extract local and more detailed features while bigger kernels can extract more holistic features (Schmidhuber, 2015).

Another prominent layer architecture for time series data, where the ordering of the datapoints is relevant, are recurrent layers. One example of a recurrent layer is the long short-term memory (LSTM) layer (Hochreiter and Schmidhuber, 1997). The typical structure of a LSTM block is displayed in figure 6.

The LSTM contains so called gates which are separate neural networks. They can learn the important information in the sequential data and thus regulate the flow of information. The three gates are the input gate I_t, the output gate O_t and the forget gate F_t. In addition, two states exist, the long term cell state c_t and the short term cell state h_t. The gates decide which information is allowed in the cells states and which information can be forgotten. Hence, the cell states are carefully regulated by the gates. An LSTM block can be described by the following equations (Dasan and Panneerselvam, 2021):

$$\begin{aligned} I_t &= \sigma(W_{xI}x_t + W_{hI}h_{t-1} + W_{cI}c_{t-1} + B_I) \\ F_t &= \sigma(W_{xF}x_t + W_{hF}h_{t-1} + W_{cF}c_{t-1} + B_F) \\ c_t &= F_t c_{t-1} + I_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + B_c) \\ O_t &= \sigma(W_{xO}x_t + W_{hO}h_{t-1} + W_{cO}c_t + B_O) \\ h_t &= O_t \tanh(c_t). \end{aligned} \tag{10}$$

Here, W_{xI}, W_{xF}, W_{xO} and W_{xc} represent the weight matrices of the different gates and long-term cell state to the input x_t. W_{hI}, W_{hF}, W_{hc}, and W_{hO} represent matrices of weights from the input gate, forget gate, long-term cell state, and output gate to the intermediate output h_{t-1}. Similarly, W_{cI}, W_{cF}, and W_{cO} represent matrices of weights from the input gate, forget gate, and output gate to the cell state c_t. B are the bias vectors, and $\sigma(\cdot)$ represents the sigmoid function, defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{11}$$

To compare different designs of autoencoders, several metrics can be calculated to quantify their performance (Dasan and Panneerselvam, 2021; Yildirim, Tan, and Acharya, 2018).

The compression CR determines how strong the model can compress the data. It is calculated from the size of the original data $\mathrm{S_{ori}}$ and the size of the encoded data $\mathrm{S_{enc}}$:

$$\mathrm{CR} = \frac{\mathrm{S_{ori}}}{\mathrm{S_{enc}}}. \tag{12}$$

A bigger value of CR hence indicates a stronger compression.

The variance between the output predicted by the model and the actual output can be measured by the root mean squared error (RMSE) which is just taking

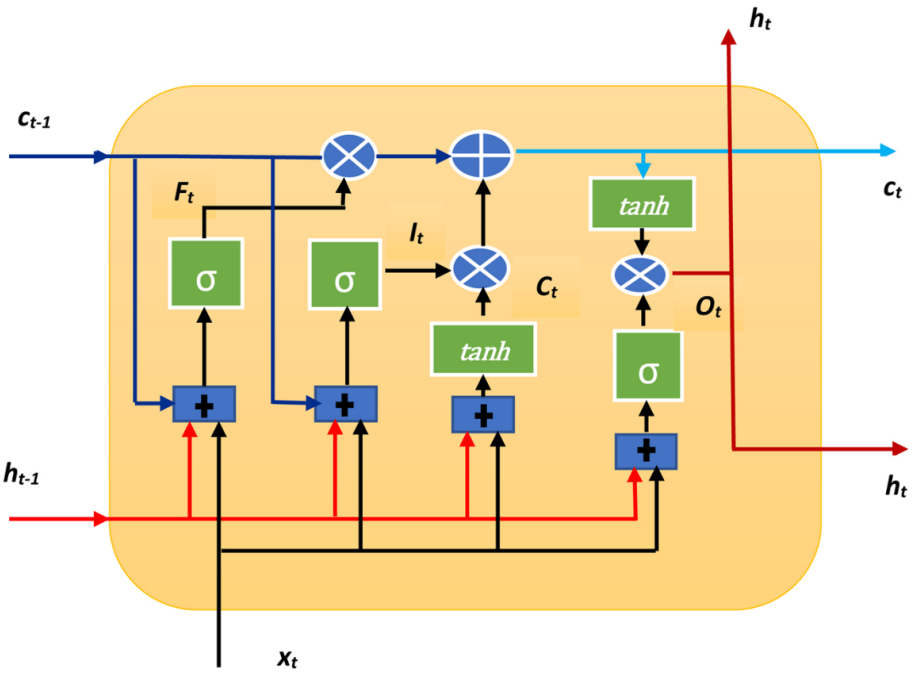


Figure 6: Structure of an LSTM block with the long-term state c_t, the short-term state h_t, the input gate I_t, the forget gate F_t and the output gate O_t. Source: Dasan and Panneerselvam, 2021.

the square root of the MSE:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}. \tag{13}$$

Another often used metric is the percentage root mean square difference (PRD):

$$\text{PRD} = \frac{\sqrt{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}}{\sqrt{\sum_{i=1}^{N} x_i^2}} \times 100. \tag{14}$$

The PRD determines the quality of the reconstructed data and should be as low as possible for a high compression quality. The normalized PRD (PRDN) is hence given by subtracting the mean $\overline{x}$ of the original trace in the denominator:

$$\text{PRDN} = \frac{\sqrt{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}}{\sqrt{\sum_{i=1}^{N}(x_i - \overline{x})^2}} \times 100. \tag{15}$$

Finally, the Quality-Score (QS) determines the effectiveness of compression

algorithms. It is the ratio of the CR and the PRD:

$$\mathrm{QS} = \frac{\mathrm{CR}}{\mathrm{PRD}}. \tag{16}$$

Furthermore, the autoencoder can additionally serve as a denoising mechanism. Given that the broader objective is to emulate biological traces on the BrainScaleS-2 hardware, it is important to note that measurements of biological neurons typically contain some level of noise. The autoencoder can be trained to ignore this noise. Assuming the noise is Gaussian-distributed, the idea behind training a denoising autoencoder is to synthetically add Gaussian noise to the voltage traces in the training dataset. These noisy traces are then passed through the model during training. However, the loss is computed based on the reconstruction of the clean signal, not the artificially created noisy signal. In this way, the autoencoder learns features from the noisy traces that enable it to reconstruct the original clean signal. To assess the model's denoising performance, we examine the difference in the signal-to-noise ratio $\mathrm{SNR}_{\mathrm{imp}}$ between the input signal and the output signal of the autoencoder:

$$\mathrm{SNR}_{\mathrm{imp}} = \mathrm{SNR}_{\mathrm{out}} - \mathrm{SNR}_{\mathrm{in}}. \tag{17}$$

Here, $\mathrm{SNR}_{\mathrm{in}}$ and $\mathrm{SNR}_{\mathrm{out}}$ are defined by:

$$\mathrm{SNR}_{\mathrm{in}} = 10 \times \log_{10}\left(\frac{\sum_{i=1}^{N} x_i^2}{\sum_{i=1}^{N}(\tilde{x}_i - x_i)^2}\right) \tag{18}$$

and

$$\mathrm{SNR}_{\mathrm{out}} = 10 \times \log_{10}\left(\frac{\sum_{i=1}^{N} x_i^2}{\sum_{i=1}^{N}(\hat{x}_i - x_i)^2}\right). \tag{19}$$

In these equations, x_i represents the value of sampling point i in the original signal, $\tilde{x}_i$ the value of sampling point i in the noisy signal, and $\hat{x}_i$ the value of sampling point i in the denoised time series. N is the length of the whole signal (Chiang et al., 2019).

3. Methods

3.1. Experiment setup

The experiments were conducted in two phases. First, the entire inference pipeline was tested using numerical simulations as a proof of concept. Following this, the inference was implemented with emulation on the BrainScaleS-2 hardware.

Simulation The simulations involved integrating the AdEx equations (1) and (2). A step current was injected into the simulated neuron, causing it to emit spikes. As a result, the distinct spike patterns that emerged differed only due to variations in the model parameters.

The simulated *runtime* of 100 ms was fixed throughout all experiments in this thesis, as was the *time offset* of 0.1 ms. Before each simulation, a waiting period equal to the *time offset* is applied before the constant current is set. Only then will the experiment run for the duration of the *runtime*. Similarly, after the *runtime*, the current will be set to zero, and another period of *time offset* will be waited. This is done to ignore any initial transients at the beginning or end when setting the parameters. A simulation timestep of 0.01 ms was consistently employed throughout all simulations, which resembles the time interval in which each variable gets updated.

The simulation was implemented in Python using the Brian framework (Stimberg, Brette, and Goodman, 2019). Brian is a Python package with a C++ backend for faster computation.

Emulation The emulations, on the other hand, were conducted by mimicking the AdEx equations on the BrainScaleS-2 hardware. The procedure was similar to that of the simulations, with an external constant current applied to induce spiking. The *runtime* was set to 1 ms on hardware, which corresponds to 1000 ms in biological time due to the acceleration factor. Additionally, the *time offset* was set to 0.3 ms, which converts to 300 ms in biological time.

Certain hardware parameters were set to fixed values and not altered in the course of this thesis. The specific settings can be seen in table 7 in the appendix.

It is crucial to conduct all experiments on the same neuron circuit, as dif-

ferent circuits may exhibit varying properties due to production differences[7]. Consequently, if an experiment is performed on another circuit with the same parameters, the results could vary significantly. Furthermore, the same calibration was loaded before each experiment to ensure consistency.

The emulation was coded with the BrainScaleS-2 extension of PyNN (Davison et al., 2009).

3.2. Datasets

In order to train and assess the autoencoder models, training, validation and test data is needed. Thus, we began by constructing appropriate datasets.

We opted for a relatively large dataset size of 200,000 voltage traces. This size can be reduced depending on the specific task at hand. However, if the number of variable parameters increases, it is essential to keep the curse of dimensionality[8] in mind when selecting an appropriate dataset size (Aremu, Hyland-Wood, and McAree, 2020).

We automated the creation of datasets, which can now be generated with a variety of different settings. Datasets can be built using simulation or emulation. However, parallelization for the trace generation is only supported for datasets created through simulations, as the same neuron circuit must be maintained on the hardware (see chapter 3.1). The creation process is generated in chunks of 20 samples and then saved in an HDF5 file to avoid memory overflow. In addition, the resulting data can already be interpolated[9] to fewer datapoints at this stage to drastically reduce storage space.

Parameters for the resulting voltage traces can be configured. Furthermore, the ranges of parameters that should vary in the voltage traces of the dataset can be specified. The program then draws random samples from a uniform distribution, constrained by the specified limits, and simulates or emulates the voltage traces for these selected parameters. This process yields a dataset of voltage traces that differ solely based on the specific parameters chosen within the given ranges. The fixed values were selected to ensure that well-defined spiking patterns emerged.

[7] We used wafer 72 FPGA 0 on the BrainScaleS-2 hardware for all our experiments. At the time of these experiments the setup was equipped with chip number 82.

[8] In high-dimensional spaces, data points become sparse, meaning that the available data may not cover the space adequately and thus making it difficult for the model to learn relationships effectively.

[9] Estimating unknown values that fall between known data points using mathematical functions.

We built three datasets with emulated data and one with simulated data. Table 1 and table 2 display the chosen parameters and parameter ranges for the created datasets. Note that not all hardware parameters are relatable to model parameters from the AdEx equations. For more information see Billaudelle et al., 2022.

Table 1: Parameter settings for the datasets consisting of emulated voltage traces on the BrainScaleS-2 hardware. The symbol ✓ means the parameter was varied between the boundaries 0 to 1022 in discrete values with arbitrary units.

parameter	fixed value	$2D$	$4D$	$8D$
$E_{\mathrm{l}}^{\mathrm{adapt}}$	500	✗	✗	✗
V_{ref}	492	✗	✗	✗
I	618	✗	✗	✗
$\mathrm{g_L}$	68	✗	✗	✗
C	63	✗	✗	✗
g_{reset}	742	✗	✗	✗
a	160	✓	✓	✓
τ_{w}	107	✓	✓	✓
b	509	✗	✓	✓
$\mathrm{V_r}$	636	✗	✓	✓
Δ_{T}	560	✗	✗	✓
$\mathrm{V_T}$	774	✗	✗	✓
$\mathrm{E_L}$	836	✗	✗	✓
V_{th}	516	✗	✗	✓

Table 2: Parameter settings for the single dataset consisting of simulated voltage traces.

parameter	fixed value	boundaries
C (pF)	100	-
Δ_τ (mV)	2	-
E_{L} (mV)	-70	-
V_{T} (mV)	-50	-
g_{L} (nS)	10	-
V_{r} (mV)	-	-70 to -50
a (nS)	-	30 to 1000
b (pA)	-	0 to 200
τ_{w} (ms)	-	30 to 800

3.3. Data embedding

We implemented two different autoencoder models to compress our time series data. One model is a fully convolutional autoencoder (CONV-AE), while the

other includes a single LSTM layer as the final layer in the encoder, which we call a convolutional-LSTM autoencoder (CONV-LSTM-AE). The key idea in that model is that the LSTM layer should account for the temporal sequence of features learned by the preceding convolutional layers. We based our work on the models proposed in Yildirim, Tan, and Acharya, 2018 and Dasan and Panneerselvam, 2021, adjusting the architectures to meet our specific requirements, such as modifying the number of layers, layer dimensions, bottleneck[10] size, and activation functions. The final model architectures are provided in tables 8 and 9 in the appendix.

The core principle behind the data reduction in the encoder is the use of max-pooling operations after the convolutional layers. Max-pooling is a down-sampling technique that reduces the spatial dimensions of the input by selecting the maximum value from a set of neighboring values, thereby preserving important features while reducing data size. In contrast, the decoder uses up-sampling operations, which duplicate the values from the previous layer to expand the data size back to its original dimensions. The subsequent convolutional layers adjust these values appropriately.

Batch normalization was used to stabilize and accelerate training by normalizing the activations within each mini-batch, reducing internal covariate shift. The CONV-AE employed ReLU activation functions after each convolutional layer, except for the last layer where a sigmoid activation function was used to normalize the output to a range from 0 to 1. The CONV-LSTM-AE, on the other hand, utilized tanh activation functions, with a sigmoid activation on the final layer as well.

Dropout was used in the CONV-LSTM-AE to prevent overfitting. Dropout randomly deactivates a certain percentage of neurons during training, which helps prevent the model from becoming too reliant on specific neurons and encourages it to generalize better to unseen data.

The dataset was split into training, validation, and test sets with a ratio of 8:1:1. Before feeding the data into the model, several transformations were applied. If necessary, the traces were cut down to 10,000 data points. Additionally, the traces were interpolated to further reduce the data to 1024 datapoints. It should be noted that excessive interpolation will destroy the structure of our data. Furthermore, the traces were normalized as well. Multiple methods for normalization exist (Lima and Souza, 2023), and we opted for min-max-scaling:

[10] The bottleneck is the last layer of the encoder which typically the smallest dimension.

$$x_i' = \frac{x_i - \min(T)}{\max(T) - \min(T)}. \tag{20}$$

Here, x_i' resembles the normalized data point of the time series T at position i. We used the global maximum and minimum of the attainable voltage values, which were $-70\,\mathrm{mV}$ and $0\,\mathrm{mV}$ for simulations, and $0\,\mathrm{MADC}$ and $1022\,\mathrm{MADC}$ for emulations. Afterwards, Gaussian noise could be added to the traces to train a denoising autoencoder. We conducted training without added noise, as well as with a signal-to-noise ratio (SNR) of $20\,\mathrm{dB}$.

After preprocessing the data, the training was conducted on a GPU, as convolutions are significantly faster on such hardware, greatly accelerating the process and reducing training time. We used a batch size of 32, and a seed of 213 was set to ensure the results could be reproduced. Furthermore, we chose the mean squared error as a loss function (see equation 8). The training was conducted for a maximum of 150 epochs.

For certain training runs where the model's performance plateaued, likely due to being stuck in a local minimum of the loss landscape, a weighted loss function was applied. A selected number of data points were randomly sampled from the trace, and their reconstruction errors were multiplied by a predetermined weight. Essentially, this method applied a random weighted mask to the reconstruction error, encouraging the model to escape the local minimum. Additionally, to help the model learn the peaks in the voltage traces more effectively, points at peaks could be randomly selected. A sliding window approach was implemented for this purpose: the window, with a defined size, moved across the trace and identified the minimum and maximum values (i.e. the peaks and troughs). These points could then be weighted further using an additional weight value.

We employed the Adam optimizer to adjust the model weights (Kingma and Ba, 2014). However, we opted for a learning rate of 10^{-4} which is smaller than the default value, as our models exhibited heightened sensitivity to larger learning rates. Consequently, we implemented a learning rate schedule for the training process (see figure 7). It is a common practice to incorporate a learning rate warm-up at the beginning of training (Kalra and Barkeshli, 2024). During the warm-up phase, the learning rate starts from a small value and increases linearly every 100 batches until the base learning rate is reached. This approach helps to stabilize training by preventing large weight updates early on, which can lead to instability. Given that our dataset is relatively large, we limited the warm-up period to the first 2000 training batches. After the warm-up, the learning rate remained constant until the onset of the decay

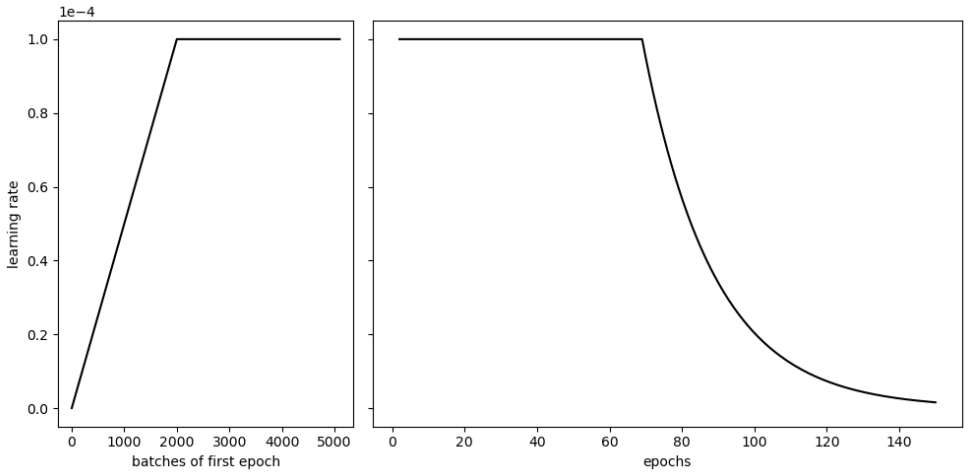


Figure 7: Learning rate schedule during the training. Note the different x scales for the right and left panel.

schedule, which we set to begin at epoch 70. We opted for an exponential decay of the learning rate after each epoch (Z. Li and Arora, 2019). A decaying learning rate in the later stages of training helps the model converge more effectively by allowing for finer updates to the weights, thereby enhancing performance as it approaches the optimal solution.

After each training epoch, the loss was calculated and averaged over the entire validation dataset to assess hyperparameter choices and evaluate the model's ability to generalize and perform on data which it was not trained on. To streamline the training process, an early stopping mechanism was implemented, which takes a *patience* and a minimum *threshold* as inputs. If the validation error does not improve over a specified number of *patience* epochs beyond the *threshold*, training will be terminated. We set a *patience* of 20 epochs and the *threshold* to 0, as we were comfortable with even minor improvements in performance on the validation set. We assumed that the model's performance would not increase further after the validation error failed to improve for 20 consecutive epochs. This mechanism conserves unnecessary training time when no significant improvements in results are anticipated.

After the training was completed, the test loss was calculated over the entire test dataset to quantify the model's performance on new, unseen data. Additionally, the other metrics from chapter 2.4.2 were also computed for further evaluation.

The creation and training of the models were conducted using the *PyTorch* 2.4.0. framework (Paszke et al., 2019).

3.4. Simulation-based Inference

The central question of this thesis was whether the SNPE algorithm is able to infer the correct hardware parameters for a given observation. To test this, an observation was created by emulating a specific voltage trace. Thus, knowing the correct parameters, we could asses the algorithm's performance. To conduct the inference, we utilized the algorithm developed by Greenberg, Nonnenmacher, and Macke, 2019, which is implemented in the Python package *sbi* (Tejero-Canteroe et al., 2020).

The generated voltage traces were interpolated down to 1024 values and then normalized as well. Afterward, the traces were compressed using different embedding techniques, including passing them through the encoder of a pretrained autoencoder, compression via wavelet transform (Weeks and Bayoumi, 2002) (appendix), or a fully connected neural network (FCNN) simultaneously trained with the NDE of the SNPE algorithm.

The FCNN consisted of 5 layers with dimensions *[800, 400, 300, 100, 30]*. Similarly, we experimented with retraining the pretrained encoder of the autoencoder simultaneously with the NDE of the SNPE algorithm. This transfer learning approach aimed to fine-tune the encoder's model weights further, seeking to obtain even more optimal summary features for parameter inference. Additionally, we tested feeding the traces directly into the SNPE algorithm without any additional compression techniques, using only the interpolated 1024 values.

For each inference round, 1000 samples were drawn. However, in the first round of the SNPE algorithm, more samples can be drawn than in subsequent rounds, as this initial round requires a broad exploration of the entire parameter space to capture the full range of possible values. A total of 20 rounds of inference were performed to enable investigation of the posterior distribution at different stages and to analyze the effects of additional rounds on the inference process.

We used a masked autoregressive flow (MAF) as our NDE (Papamakarios, Pavlakou, and Murray, 2017). A MAF transforms a normal distribution into other probability distributions. In our case, we used five transformations, with each transformation consisting of two blocks, each containing 50 hidden units. These transformations are chained together sequentially. For further details, see Papamakarios, Nalisnick, et al., 2021.

The training of the NDE and embedding networks was executed on a GPU, as this significantly accelerated the training process. A seed value of 42 was set

to ensure the reproducibility of our results. Inference was performed using the same parameter settings as outlined in table 1 and table 2. As a result, the inference pipeline was tasked with inferring two, four, and eight parameters simultaneously.

We sampled from a truncated posterior in an example where eight parameters needed to be inferred, as standard sampling was too slow to achieve sufficient rounds of inference. Additionally, an analysis of the approximated posteriors has been conducted for a selection of the results.

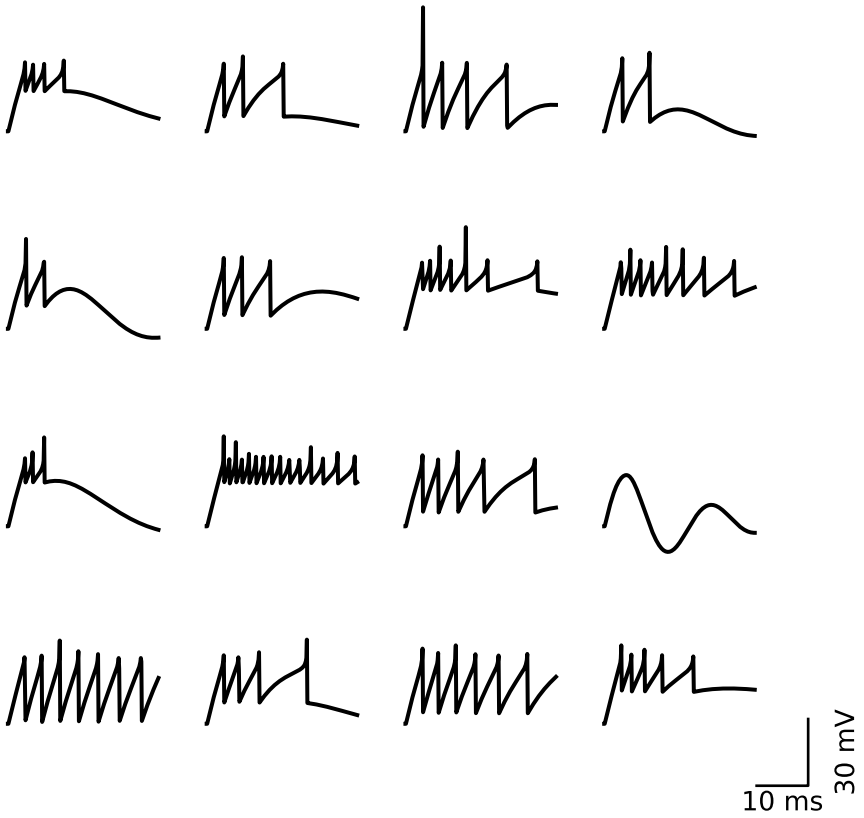


Figure 8: Voltage traces of the simulated dataset with parameter setting from table 2. The traces were interpolated down to 1024 datapoints and only the first 300 datapoints are displayed for visualization purposes.

4. Results

4.1. Datasets

Samples of the simulated dataset with parameter setting from table 2 are displayed in figure 8, whereas samples from the 4D dataset with hardware parameters from table 1 are displayed in figure 9. Samples form the other emulated datasets can be seen in the appendix in figure 38 and 39.

4.2. Data embedding

We trained both models, introduced in chapter 3.3, using the same training routine and compared their performance. Figure 10 shows the losses on the

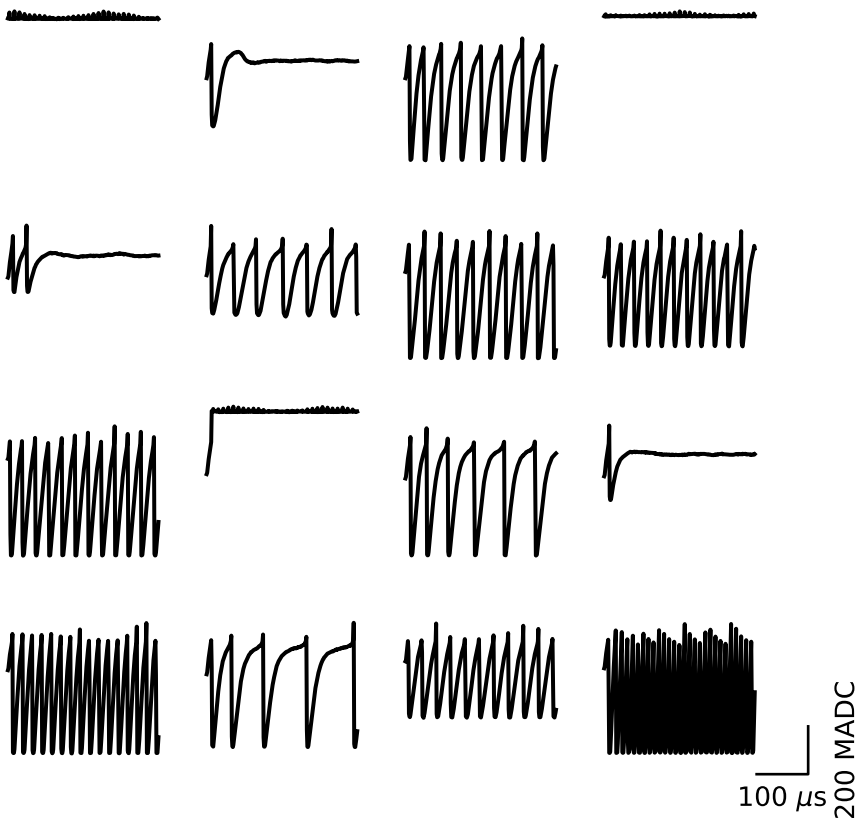


Figure 9: Voltage traces of the 4D dataset with parameter setting from table 1. The traces were interpolated down to 1024 datapoints and only the first 300 datapoints are displayed for visualization purposes.

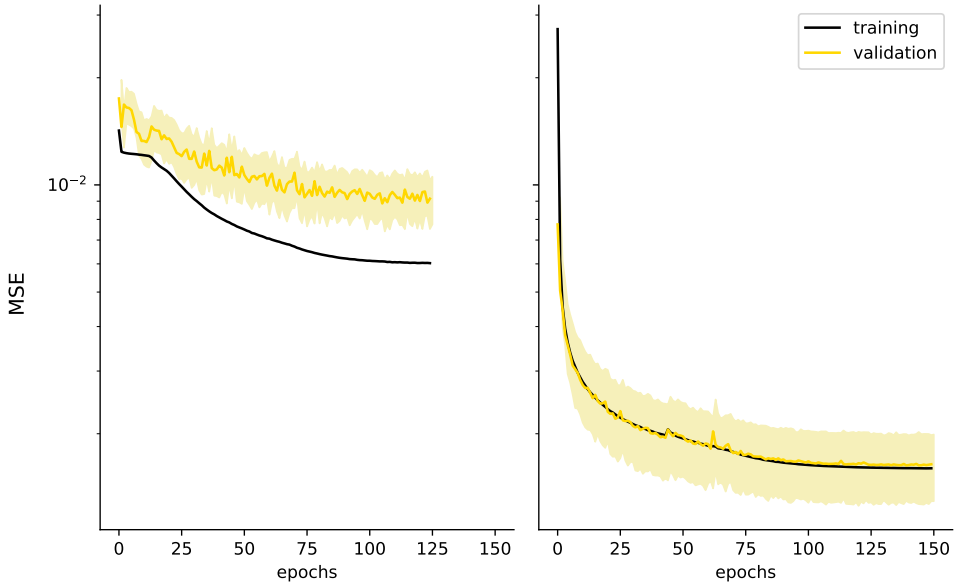


Figure 10: Comparison between the average training and validation losses of two autoencoder models. One standard deviation of the validation loss is shown, too. **Left:** CONV-LSTM-AE; **Right:** CONV-AE; the training for both models was conducted on the 4D emulation dataset with parameter settings of table 1.

training and validation sets during training. It quickly becomes apparent that the CONV-AE outperformed the CONV-LSTM-AE in both training and validation losses. In fact, the training of the CONV-LSTM-AE was terminated at epoch 126 due to the early stopping mechanism, as the validation loss has not improved over the last 20 epochs, even though the training loss continued to decrease slightly. This indicates that the CONV-LSTM-AE is much more prone to overfitting. This is also why we implemented dropout with a probability of 0.5, whereas we did not use any dropout in the CONV-AE.

Reconstructions and their corresponding true traces are shown in figure 11. It is clearly visible that the CONV-AE performs better than the CONV-LSTM-AE in reconstructing the original voltage traces.

Knowing that we clearly have a superior candidate among our two models, we then needed to identify and adjust the optimal hyperparameters[11], such as learning rate, layer dimensions, number of layers and activation functions. The choice of the correct hyperparameters is crucial for the model's performance. Therefore, we conducted extensive testing. A selection of the results is presented below.

[11] Hyperparameters refer to model settings that are determined before training and cannot be learned during the process.

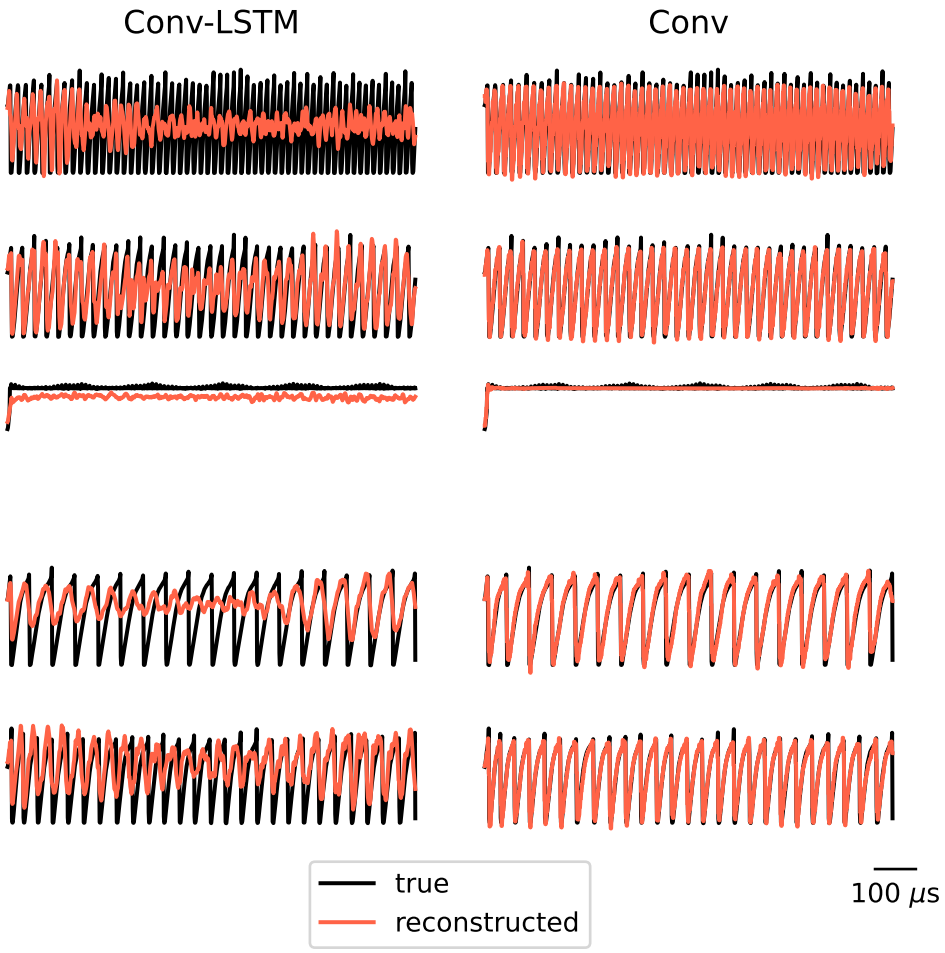


Figure 11: Reconstructed traces after a forward pass through the autoencoder models are shown along with their corresponding true traces from the test set. **Left:** CONV-LSTM-AE; **Right:** CONV-AE. The model parameters learned during training, as displayed in Figure 10, were used.

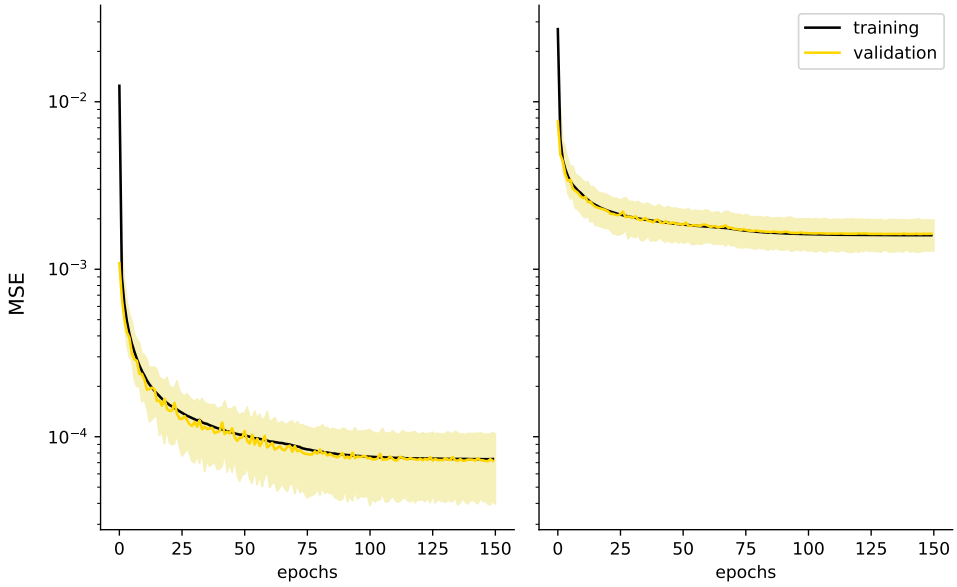


Figure 12: Comparison between the average training and validation losses of two autoencoder models. One standard deviation of the validation loss is shown, too. **Left:** CONV-AE with latent space dimension of 64; **Right:** CONV-AE with latent space dimension of 32; the training for both models was conducted on the 4D emulation dataset with parameter settings of table 1.

Obviously, the size of the bottleneck has a significant impact on the quality of the reconstruction. If the latent space dimension is too small, the autoencoder cannot learn all the necessary features that define the trace. On the other hand, if the latent space is too large, the autoencoder loses the effect of compression. Additionally, it is more prone to overfitting, as it can capture more irrelevant features that are specific to the training data. Figure 12 shows the difference in loss for latent space sizes of 64 and 32 of the CONV-AE. While it is evident that the model with the larger latent space performs better, we chose to use the smaller model with a latent space dimension of 32. This decision prioritizes achieving higher compression over better reconstruction quality, as the reconstructions of the model with the smaller latent space dimension were already quite satisfactory. Our goal was to capture only the most essential features of the trace for later use in the SNPE algorithm.

However, one should always keep the complexity of the dataset in mind when constructing a suitable model architecture. A higher-dimensional dataset, characterized by more variable parameters, likely has more features that need to be learned. Thus, it might be necessary to increase the bottleneck size of the autoencoder when training on such a dataset. Figure 41 in the appendix demonstrates this problem. The loss curves of the same model architecture on a 2D and 8D dataset, as shown in table 1, are presented. The loss for the

smaller-dimensional dataset is lower since the 32-dimensional latent space is able to capture the most relevant features. However, a latent space dimension of 32 might not capture all relevant features for the 8D dataset, resulting in a higher loss.

During the testing process, we encountered a performance issue with the autoencoder on datasets where individual samples did not vary significantly from each other. These datasets had small parameter ranges for the varying parameters. As a result, the loss got stuck and remained constant over the training epochs while reconstructions of samples were poor. To address this, we implemented the option of training with a randomized weighted loss, as described in chapter 3.3. This forced the model to explore different environments of the parameter space, thereby escaping the local minima in the loss landscape. However, when the training data varied significantly, as seen in the 4D emulation dataset of table 1, the training with weighted loss performed slightly worse than without the additional weights (see figure 40 in the appendix). Therefore, we chose to train without those additional weights for all presented datasets in chapter 3.2.

Since the denoising ability of autoencoders can be beneficial for inferring hardware parameters from real biological neuron traces in the long run, we tested that as well. To do this, we introduced artificial Gaussian noise with a SNR of 20 dB to the voltage traces. The loss curve during training is displayed in Figure 13. Compared to the training run without artificially added noise, both the training and validation losses are higher, which was to be expected. Figure 14 demonstrates the functionality of the denoising autoencoder, which successfully reduces most of the noise.

Finally, we calculated the test metrics described in Chapter 2.4.2 to further quantify the performance of our models (see Table 3). The compression ratio (CR) of the CONV-LSTM-AE is nearly half that of the CONV-AE, as the CONV-LSTM-AE has a latent space size of 60, while the CONV-AE has a latent space of size 32. The percentage root mean square difference (PRD) and quality score (QS) also strongly favors the latter model. Furthermore, the CONV-LSTM-AE exhibits higher variance in the reconstructions, which can be measured by the root mean square error (RMSE). As expected, the mean squared error ($\mathrm{MSE}_{\mathrm{test}}$) calculated on the test set is also significantly smaller for the CONV-AE.

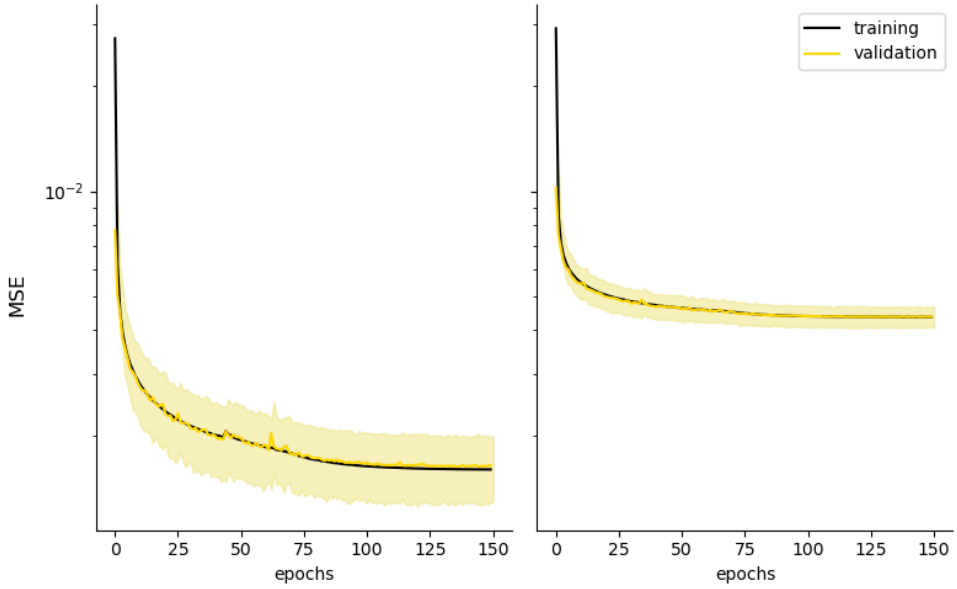


Figure 13: Comparison between the average training and validation losses of two autoencoder models. One standard deviation of the validation loss is shown, too. **Left:** CONV-AE without added noise during training; **Right:** CONV-AE with added noise during training resulting in a SNR of 20 dB. The training for both models was conducted on the 4D emulation dataset with parameter settings of table 1.

Table 3: Test metrics of the two proposed models. Calculations were performed on the test set.

Metric	CONV-LSTM-AE	CONV-AE	denoising CONV-AE
CR	17.067	32.000	32.000
PRD %	20.434	7.928	8.384
PRDN %	352.300	54.884	80.802
QS	0.835	4.035	3.816
RMSE	0.003	0.001	0.001
MSE_{test}	0.0098	0.0016	0.0017
SNR_{imp} dB	-	-	4.685

As a result of our experiments, we selected the CONV-AE as our model to compress the voltage time series from 1024 datapoints to just 32, before feeding them into the SNPE algorithm.

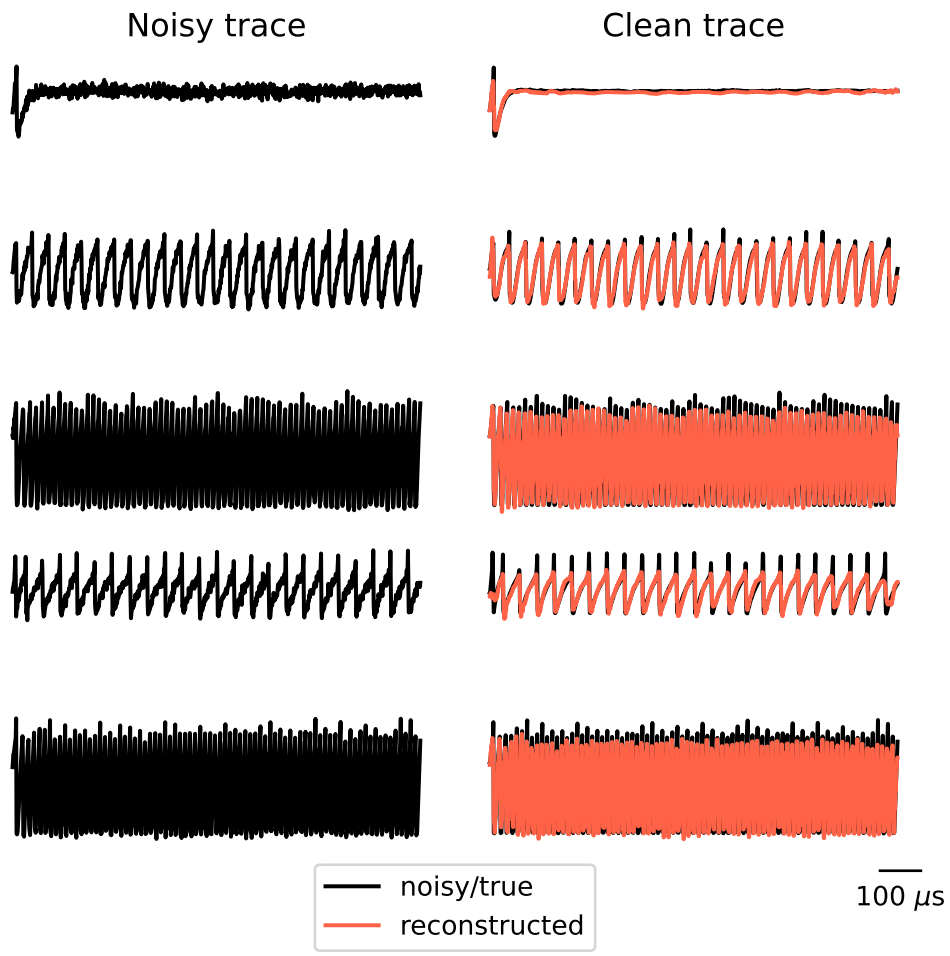


Figure 14: Effect of the denoising autoencoder. **Left:** voltage traces with artificially added noise which were fed into the autoencoder; **Right:** clean original traces and denoised corresponding reconstructions from the autoencoder.

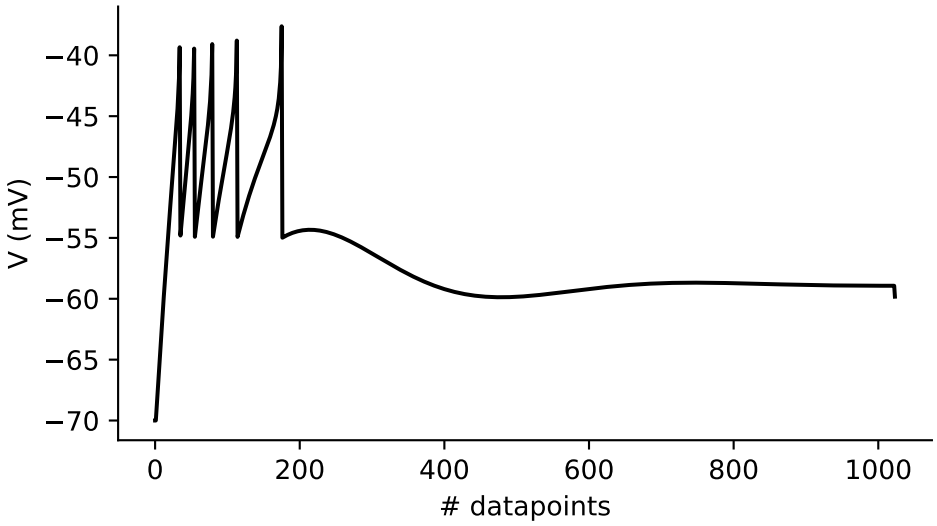


Figure 15: Simulated observation for which accurate model parameters need to be inferred. The search parameters for this trace are shown in table 4.

4.3. Simulation-based inference

To validate our inference pipeline, we first conducted inference on simulated data to ensure the accuracy and robustness of the method before applying it to infer hardware parameters. This initial step allowed us to identify any potential issues and fine-tune the model, ensuring that it performs optimally on emulated data.

4.3.1. SNPE on simulated data

We implemented multiple methods for embedding the data, including the encoder from a pretrained autoencoder, FCNNs, wavelet transforms, and feeding raw data directly into the SNPE algorithm without embedding. Each of these methods was initially tested on simulated data as a proof of concept. The observation plots shown here illustrate the results.

The starting point was the observation displayed in Figure 15, and we aimed to infer the accurate model parameters of the AdEx equations 1 and 2 that can replicate this trace. The algorithm was aware of all model parameters except for four. The four unknown parameters of that observation are presented in Table 4.

Table 4: Parameter settings of the simulated observation. Other model values are equivalent to those in table 2.

Parameter	**Value**
a (nS)	80
b (pA)	80
V_{r} (mV)	-55
τ_{w} (ms)	50

First, a pretrained encoder is used to embed the data. The CONV-AE for this purpose was trained on the dataset with the parameter settings specified in table 2. The results of the inference across the entire parameter range are displayed in figure 16, while figure 17 provides a zoomed-in view for more detail. It can be observed that the inferred parameters closely match the target values, demonstrating the effectiveness of our method.

In order to assess whether the posterior is a meaningful approximation, samples were drawn from it and plotted alongside the original observation in figure 18. As expected, some of the drawn samples deviate slightly more from the original observation, while others show a very close alignment with the original voltage trace.

Similar results were obtained for other embedding techniques. Samples drawn from the approximated posteriors of the FCNN are shown in figure 19, while those from the wavelet embedding are displayed in figure 42. Even directly inputting the interpolated dataset of 1,024 points into the SNPE algorithm yielded promising results, as shown in figure 20. These consistently satisfying results confirm that our implementation of the inference pipeline is correct and functioning as expected.

Interestingly, depending on the specific type of data embedding technique used, the shape of the resulting approximated posterior can vary. The shape of the posterior can provide insights into correlations between parameters. However, it appears that the specific type of embedding technique employed can influence the results in this regard.

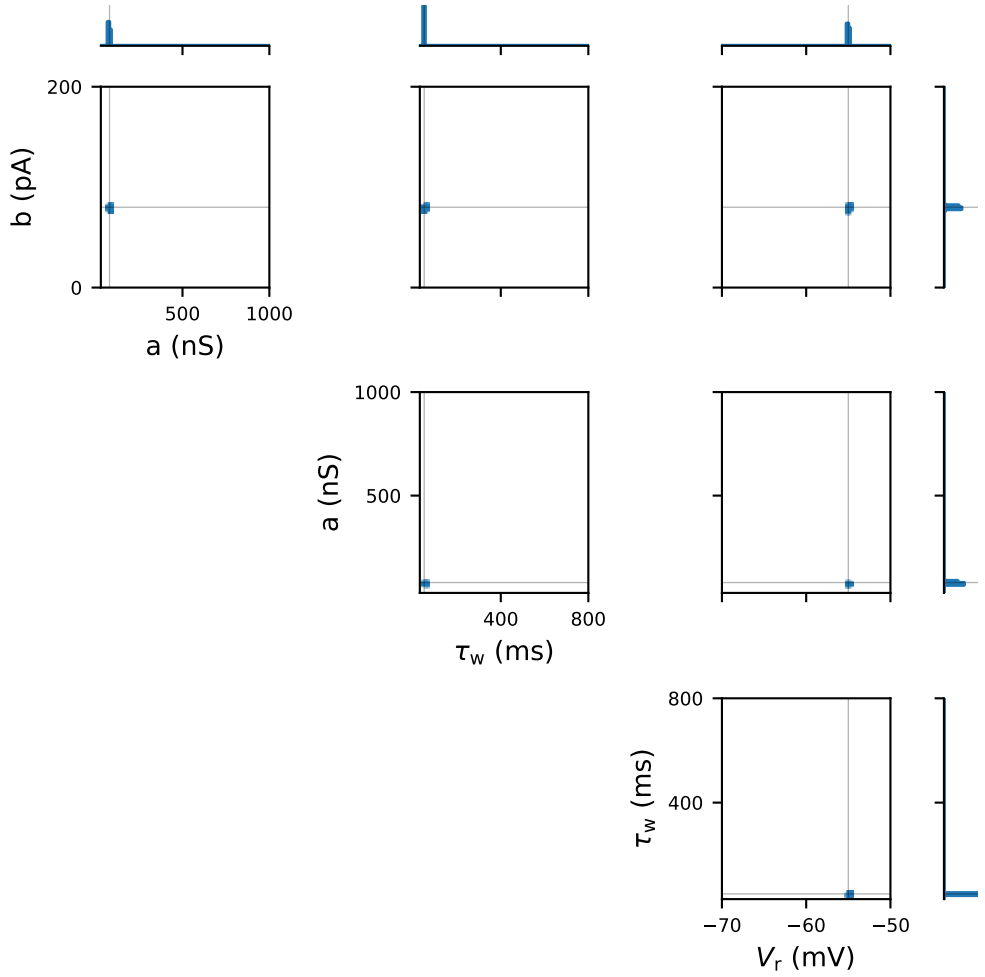


Figure 16: Pair plot of 1000 drawn samples from the approximated posterior. The true parameters are located at the intersection of the grey lines and are set according to the values in table 4. The encoder of a pretrained CONV-AE was used as the data embedding technique. 20 rounds of inference with 1000 simulations each were performed. A zoomed-in version is displayed in figure 17.

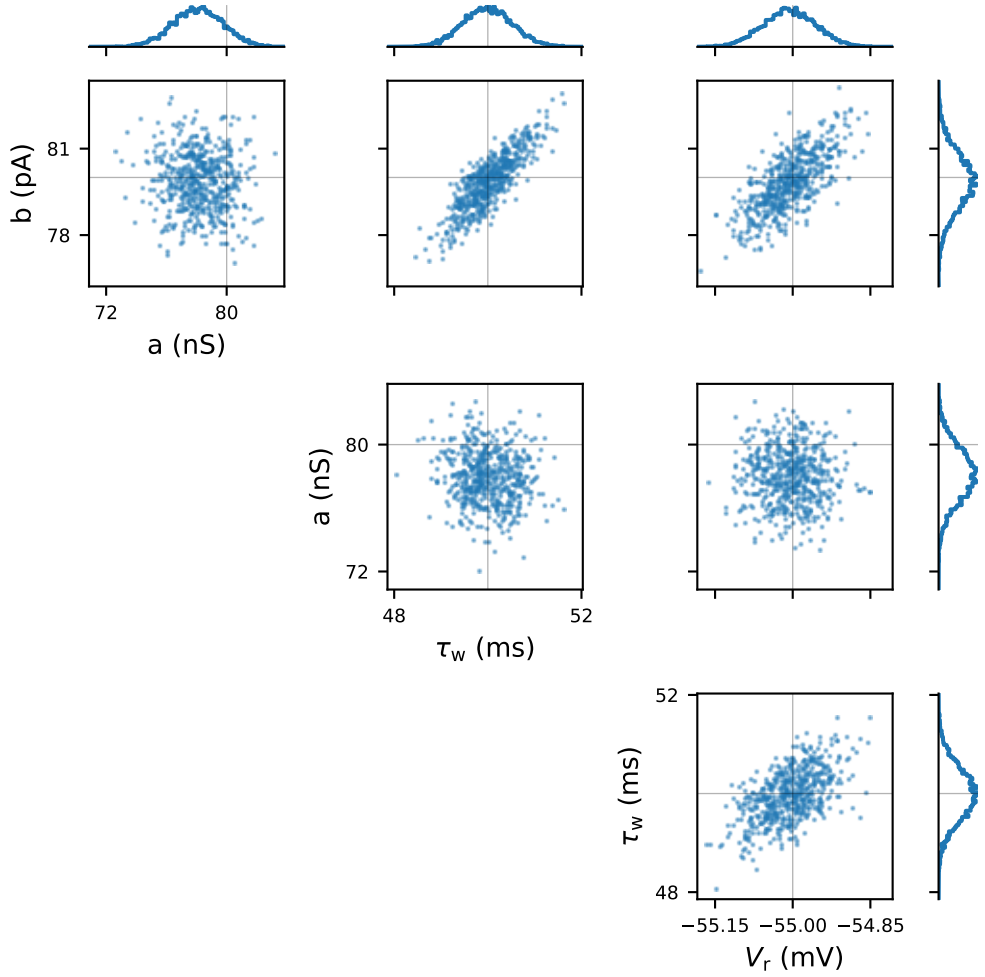


Figure 17: Pair plot of 1000 drawn samples from the approximated posterior. The true parameters are located at the intersection of the grey lines and are set according to the values in table 4. The encoder of a pretrained CONV-AE was used as the data embedding technique. 20 rounds of inference with 1000 simulations each were performed.

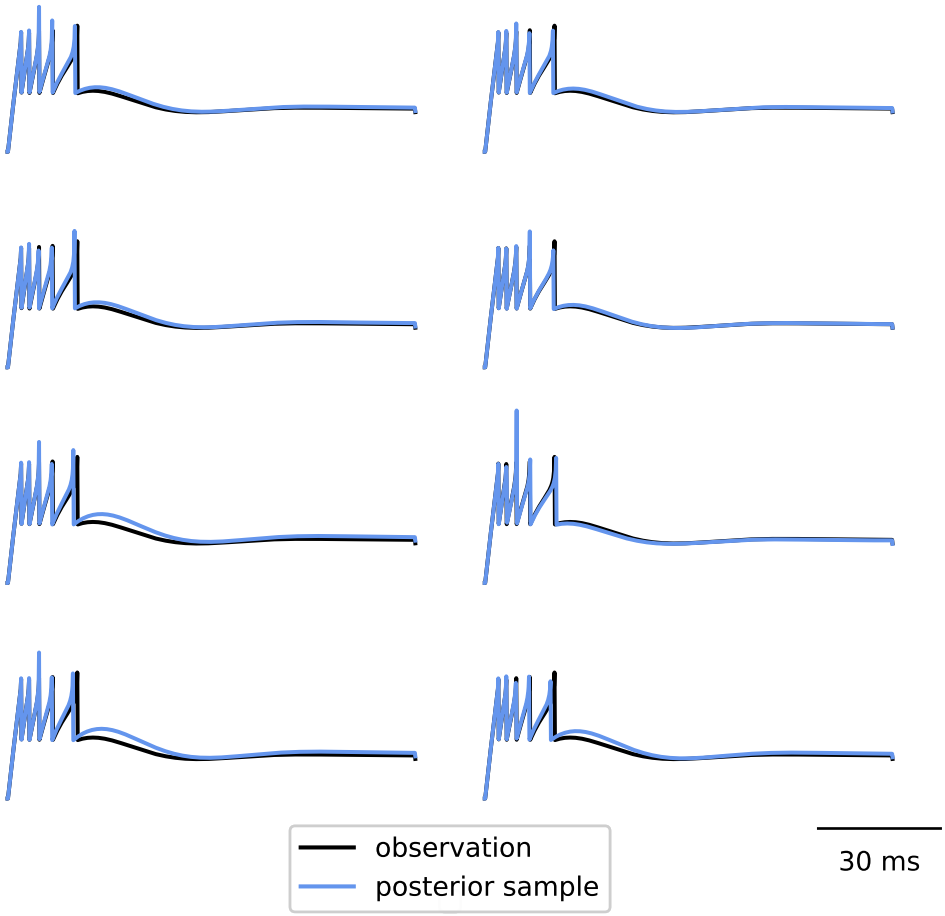


Figure 18: Comparison of the original observation with simulations of drawn posterior samples of the posterior displayed in the figure 17.

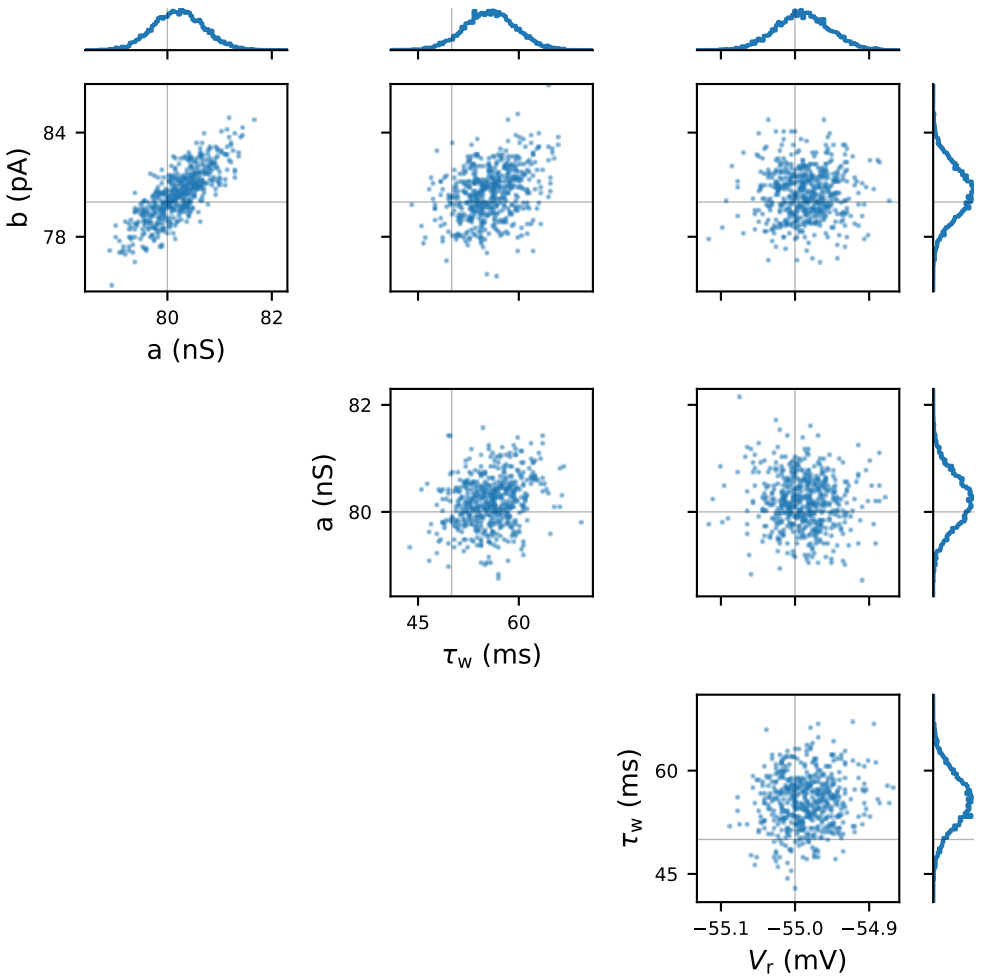


Figure 19: Pair plot of 1000 drawn samples from the approximated posterior. The true parameters are located at the intersection of the grey lines and are set according to the values in table 4. A FCNN simultaneously trained with the NDE of the SNPE algorithm was used as the data embedding technique. 20 rounds of inference with 1000 simulations each were performed.

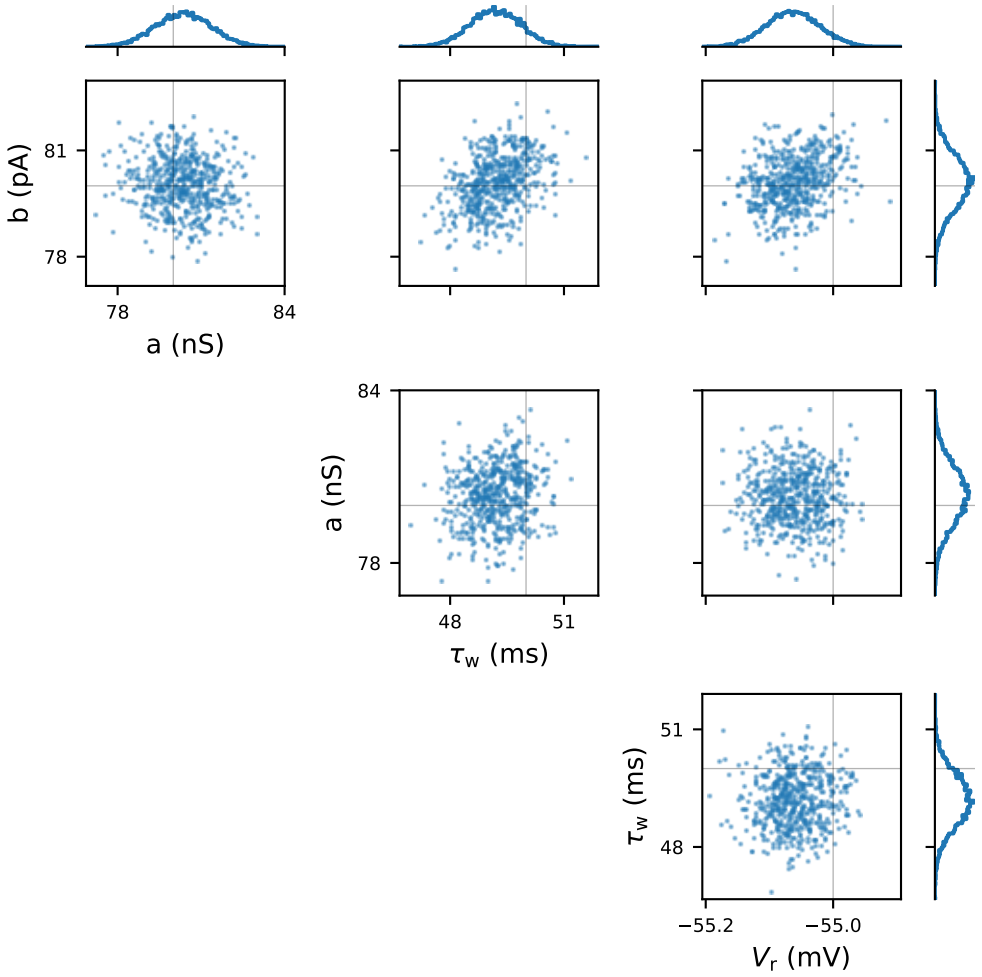


Figure 20: Pair plot of 1000 drawn samples from the approximated posterior. The true parameters are located at the intersection of the grey lines and are set according to the values in table 4. No additional data embedding was used at all. 20 rounds of inference with 1000 simulations each were performed.

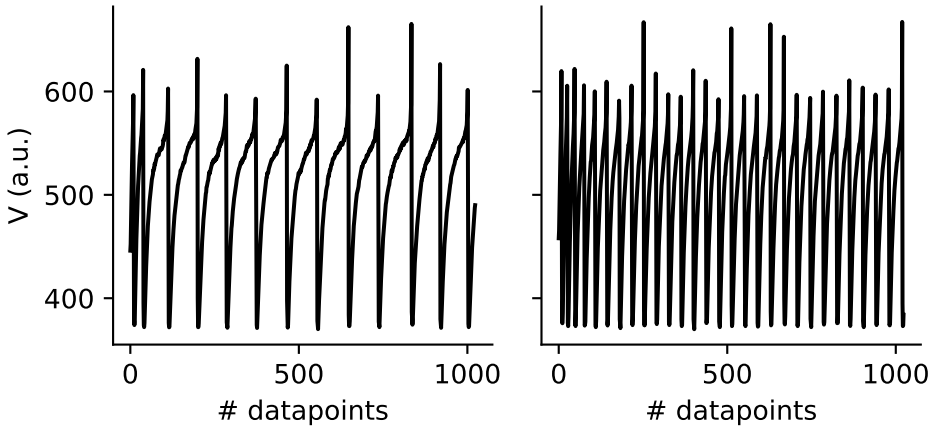


Figure 21: The observed voltage traces to whom accurate parameters should be inferred. **Left:** slow spiking observation $\hat{x}_{\text{slow}}$; **Right:** fast spiking observation $\hat{x}_{\text{fast}}$. Both traces only differ between each other in their a and τ_{w} values which are shown in table 5. The other parameters are the same as the fixed parameters displayed in table 1.

4.3.2. SNPE on emulated data

Since our inference tests were successful on simulated data, we then conducted tests on hardware emulations. To achieve this, two observations with different parameter settings shown in table 5 were created and are displayed in figure 21. We were interested in whether the specific type of observation would impact the inference results. Thus, both traces were fed into our inference pipeline.

Table 5: Differing parameter values of observations $\hat{x}_{\text{slow}}$ and $\hat{x}_{\text{fast}}$.

parameters	$\hat{x}_{\text{slow}}$	$\hat{x}_{\text{fast}}$
a	350	160
τ_{w}	200	107

2D inference We began by attempting to infer two parameters, specifically a and τ_{w}, from the slower spiking observation $\hat{x}_{\text{slow}}$. To achieve this, we utilized the encoder of the pretrained CONV-AE as our embedding technique. We did not perform further training of the encoder at this point. Samples drawn from the two-dimensional approximated posterior are plotted in figure 22. It can be observed that the general region of the true parameters could be found within the parameter space. However, most of the posterior samples are systematically shifted and do not accurately reflect the true location in the parameter space.

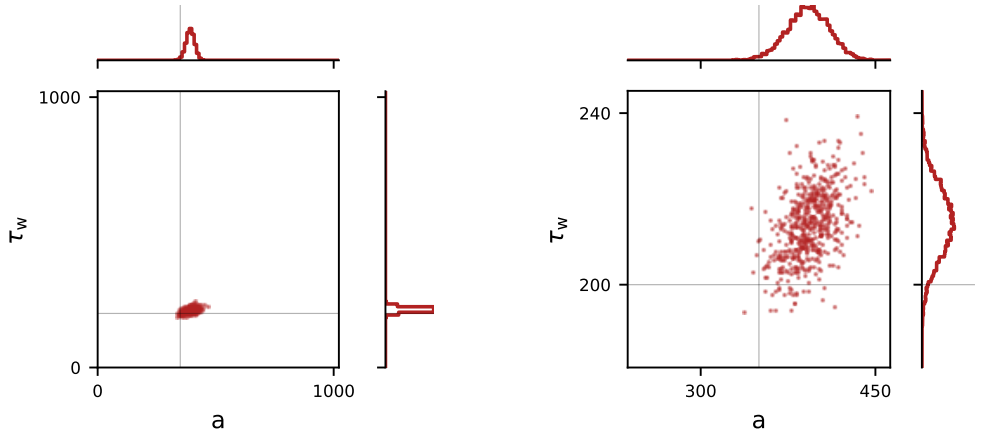


Figure 22: Samples drawn from the 2D posterior for the target observation $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE without additional retraining was used as the embedding technique. 20 inference rounds with 1000 simulations each were performed. **Left:** view of the whole parameter range; **Right:** zoomed-in view of the plotted samples.

Naturally, we were interested in how the inference procedure could possibly be improved. Therefore, we chose to retrain the encoder, this time simultaneously with the NDE of the SNPE algorithm. We loaded the previously learned weights, hypothesizing that they would be further fine-tuned through the additional training. The rationale for beginning the training with the previously learned weights as initialization, rather than using random initialization, was to potentially reduce the overall training time, as the encoder already had a solid starting point. The result of the inference with this embedding technique is presented in figure 43. However, no significant improvement in the approximated posterior was observed. Nevertheless, when plotting the log-probability of the posterior over multiple rounds, we noticed that training converged faster when the encoder was trained alongside the SNPE algorithm compared to the scenario without this training (see figures 23 and 24). In fact, with the additional training of the encoder, the posterior becomes narrower and more defined in earlier rounds. We conclude that without additional training, the SNPE algorithm requires more rounds to achieve the same results. By training the embedding network concurrently, we need fewer rounds, as the inference process becomes more effective due to the extraction of better features that facilitate inference. However, ultimately approximated posterior is still quite similar.

We want to use this relatively simple 2D example to demonstrate a posterior diagnostics technique by examining the sensitivity of the posterior to parameter changes. As described in chapter 2.4.1, when performing sensitivity analysis,

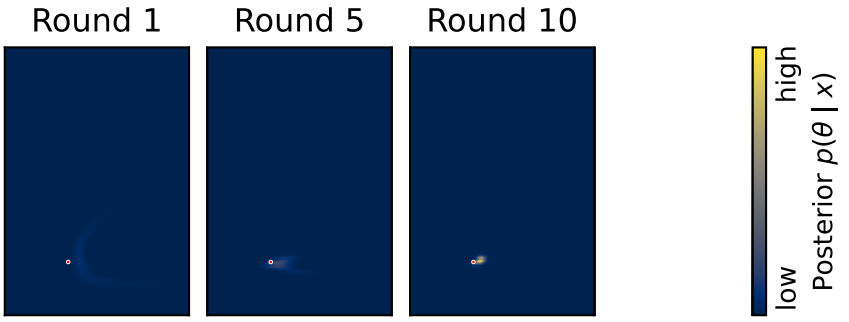


Figure 23: Log-probability of the approximated posterior displayed in figure 22 over different inference rounds. The red dot resembles the parameters of the target observation $\hat{x}_{\text{slow}}$. The encoder of the CONV-AE without additional retraining was used as the data embedding technique.

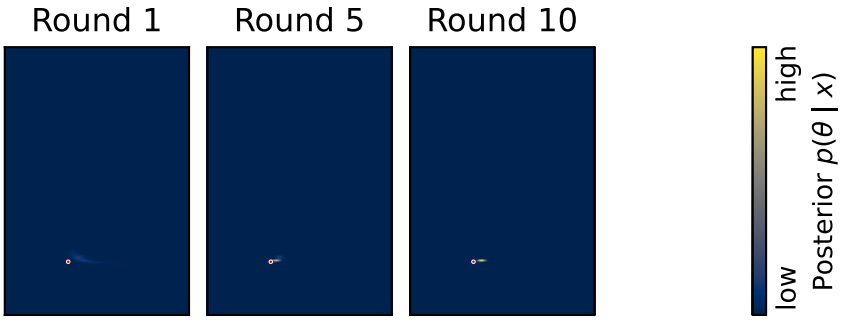


Figure 24: Log-probability of the approximated posterior displayed in figure 43 over different inference rounds. The red dot resembles the parameters of the target observation $\hat{x}_{\text{slow}}$. For the data embedding, the encoder of the CONV-AE was simultaneously retrained with the NDE of the SNPE algorithm.

Table 6: Eigenvalues and eigenvectors from the posterior sensitivity analysis of the approximated posterior displayed in figure 22.

Eigenvalues	**Eigenvectors**
$\lambda_1 = 0.0003$	$\mathbf{v_1} = \begin{bmatrix} -0.9688 \\ -0.2480 \end{bmatrix}$
$\lambda_2 = 0.0019$	$\mathbf{v_2} = \begin{bmatrix} -0.2480 \\ +0.9688 \end{bmatrix}$

one leverages the *Active Subspace* method (Constantine, Dow, and Q. Wang, 2014). Fundamentally, one calculates the gradient of the posterior density with respect to the parameters. This process yields two eigenvalues and their corresponding eigenvectors. A strong eigenvalue indicates that the gradient of the posterior density is large. This means that the posterior is more sensitive to changes in the parameters along the corresponding eigenvector. By calculating these for our approximated posterior displayed in figure 22, we obtain the results shown in table 6.

The eigenvalue λ_2 is significantly larger than λ_1, leading us to conclude that we obtain more varied results when simulating parameters which were varied along the direction of $\mathbf{v}_2$ compared to varying the parameters along $\mathbf{v}_1$. This result was also anticipated from the posterior plot in figure 22. Thus, this example is quite trivial. However, this example serves as a proof of concept and can be applied to high-dimensional posteriors, where it is challenging to develop intuition about the sensitivity of the posterior through pair plots. Additionally, this method allows us to explore variations in specific features of the embedded data rather than examining all inputs at once.

4D inference In the next step, the goal was to infer four parameters simultaneously. The algorithm was supposed to find the parameters a, b, τ_w and V_r of observation $\hat{x}_\mathrm{fast}$, using the encoder of our CONV-AE as the embedding technique without additional retraining. The posterior samples drawn from the approximated posterior are shown in figure 25, covering the entire possible parameter range for the 4D dataset described in table 1.

The true parameters could not be determined precisely. The approximated posterior exhibits a high level of inaccuracy. However, the two-dimensional posterior view does not always have to follow a Gaussian distribution and be point-like. It remains unclear whether the algorithm has learned a poor approximation of the true posterior or if the parameters exhibit correlations and compensation mechanisms. It is still possible that the same trace could be

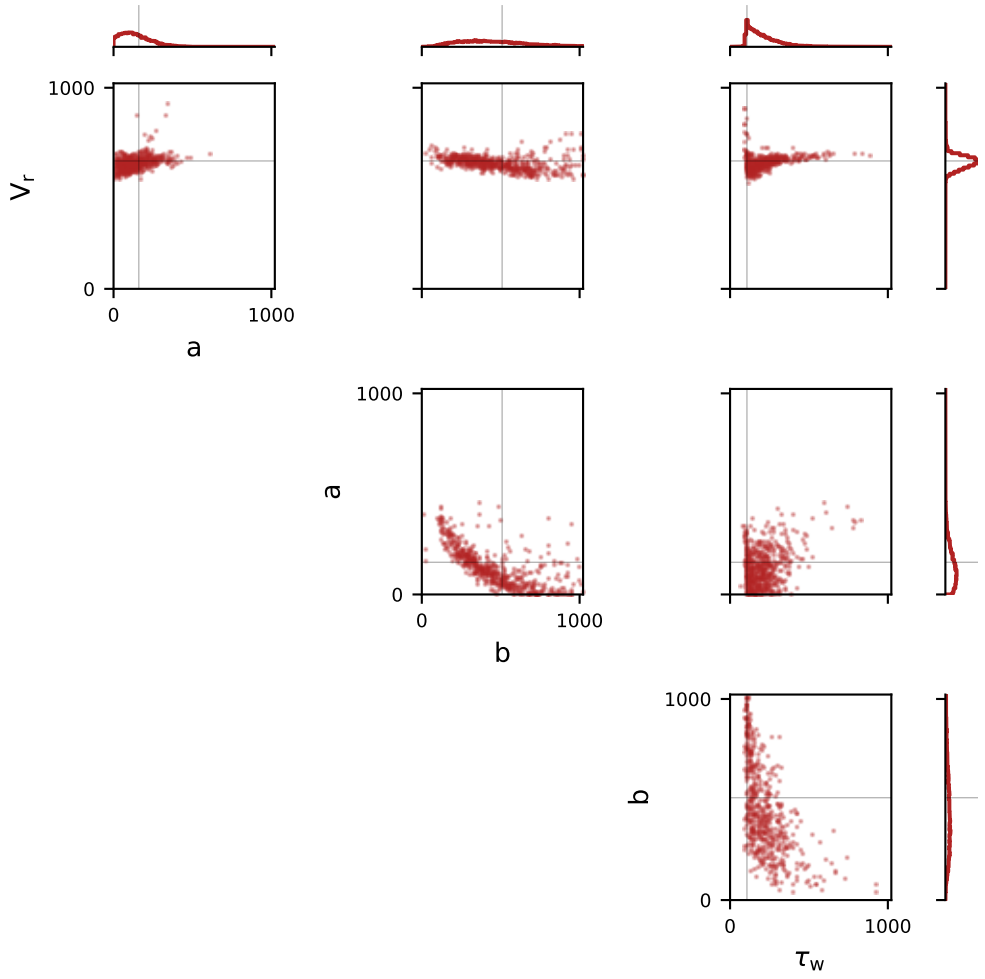


Figure 25: Samples drawn from the 4D posterior for the target observation $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE without additional retraining was used as the embedding technique. 20 inference rounds with 1000 simulations each were performed.

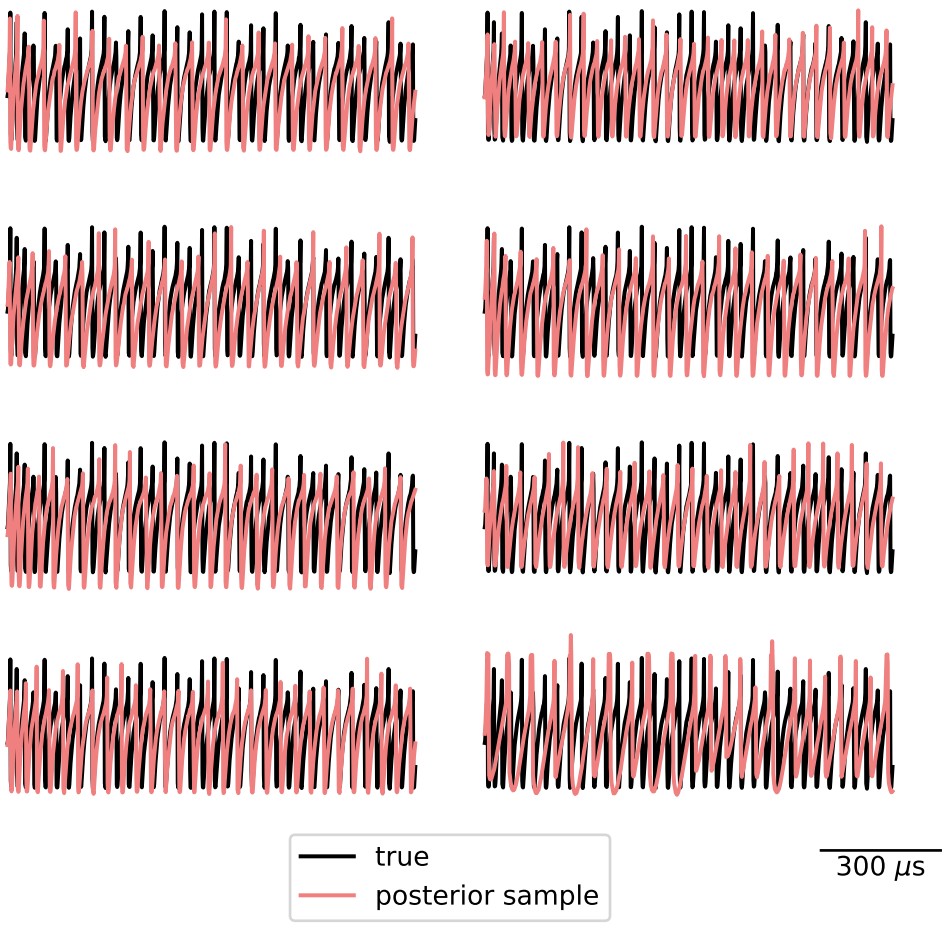


Figure 26: Generated voltage traces with drawn parameters from the posterior displayed in figure 25. $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5 served as the target observation which is shown in black.

generated by a different combination of parameters. To investigate this, we generated voltage traces with randomly drawn posterior samples (see figure 26). We observe that the overall structure is mostly similar to the original observation. Nevertheless, systematic shifts and variations in inter-spike intervals and voltages are notably present, leading to inaccurate results.

Next, we tried feeding the data directly into the SNPE algorithm without any embedding (see figure 27). This approach could decently identify the values of V_r, a and τ_w, but the algorithm struggled to infer the value of b precisely.

In order to investigate why the inference on hardware performs worse than on simulated data, we performed multiple emulations with the exact same parameters and compared the results. Figure 28 displays several emulations of the observation $\hat{x}_{\text{slow}}$.

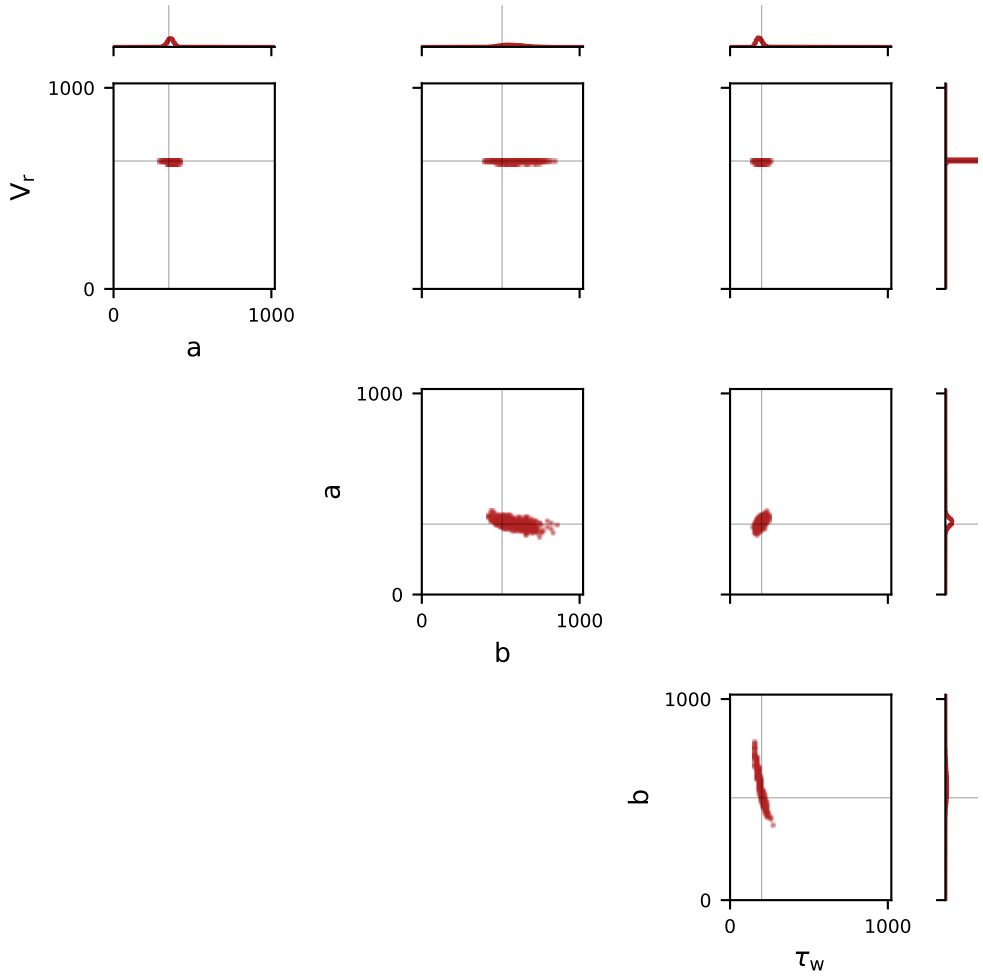


Figure 27: Samples drawn from the 4D posterior for the target observation $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5. No additional data embedding technique was leveraged. 20 inference rounds with 1000 simulations each were performed.

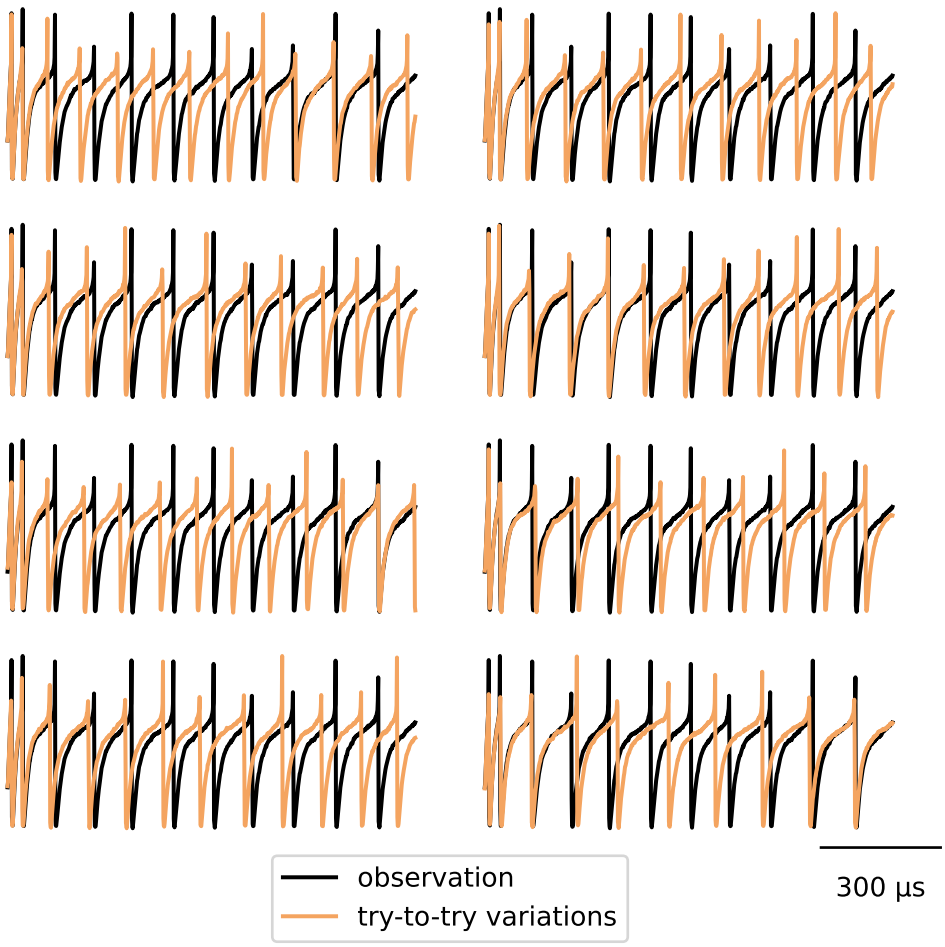


Figure 28: Voltage traces emulated with the same set of hardware parameters. The black trace is the same throughout the whole figure and serves as a comparison. The chosen parameters were those of the observation $\hat{x}_{\text{slow}}$ (see tables 1 and 5).

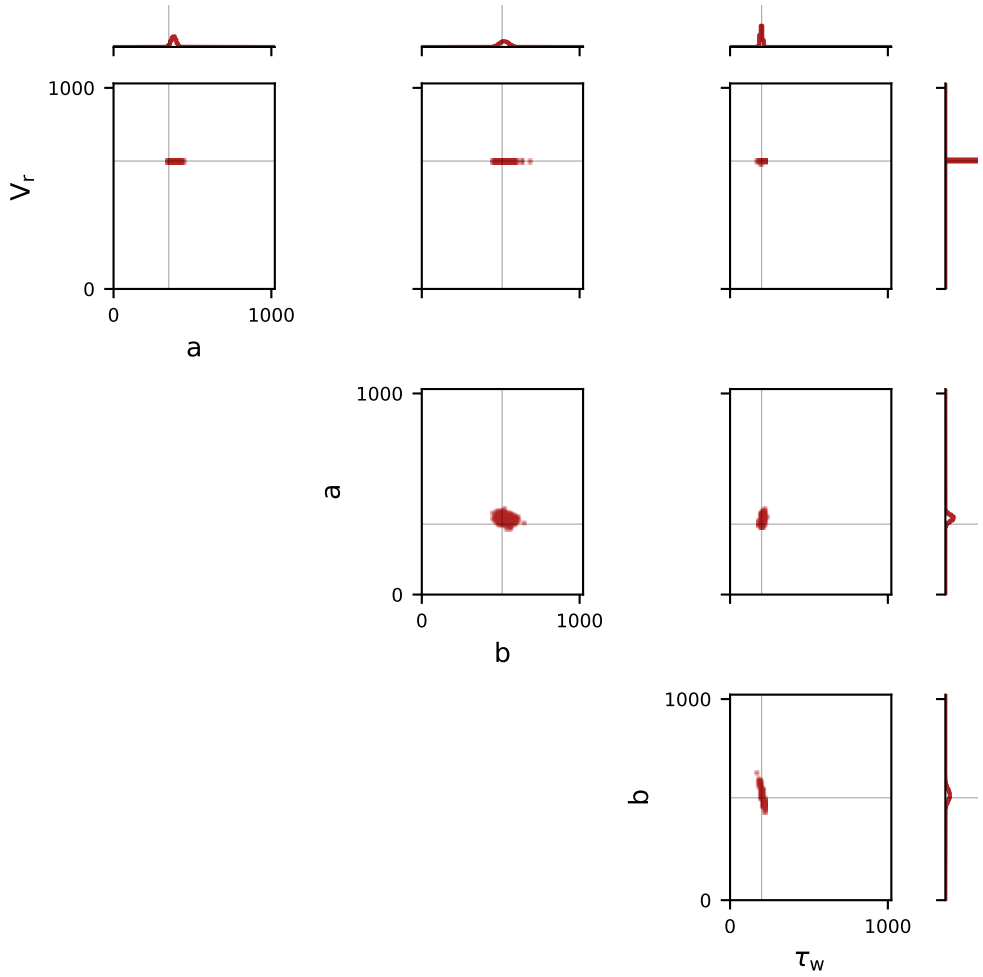


Figure 29: Samples drawn from the 4D posterior for the target observation $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE with additional concurrent retraining with the NDE of the SNPE algorithm was used as the embedding technique. 20 inference rounds with 1000 simulations each were performed.

The results show significant variations among the traces, even though the exact same parameters were used for the emulation. This indicates that hardware emulations are not deterministic and always have some inherent noise that distorts the results. Obviously, this presents a major challenge for the SNPE algorithm, as traces, even with the correct parameters, do not automatically resemble the original observation.

By training the encoder of the CONV-AE simultaneously with the NDE, we obtained the results displayed in figure 29. Compared to the results without any embedding displayed in figure 27, the posterior is now more precise and fine-tuned.

To improve our results further, we expanded the latent space of our encoder

from 32 to 64 features. Additionally, we increased the complexity of the NDE in the SNPE algorithm by raising the number of hidden features from 50 to 100 and the number of transformations from 5 to 10. Samples drawn after 20 rounds of inference from the resulting posterior are shown in figure 30. Additionally, samples from the first inference round are plotted, too, highlighting the advantage of the sequential inference in the SNPE algorithm. The posterior becomes significantly more refined after multiple rounds of inference. The obtained results are already quite accurate. Emulations of drawn posterior samples are displayed in figure 31. Except for similar deviations observed in the try-to-try variations in figure 28, the overall structure remains very similar between the different traces and the original observation.

To further investigate the obtained posterior, we plotted a zoomed-in view of the plotted samples in figure 30 (see figure 32). We observe that the approximated posterior still misses the true observation with a slight shift in the parameter space. This may indicate that the posterior is too overconfident.

It seems that some parameters exhibit strong correlations between each other. In particular, parameter b and τ_{w} show a strong negative correlation. According to the 2D pairplot, an increase in τ_{w} can still produce a similar trace if b decreases simultaneously. To further investigate this possible correlation, we calculated the Pearson correlation coefficients, which are visualized in figure 33.

As expected, the Pearson coefficients indicate a strong negative correlation between the parameters b and τ_{w}. Additionally, there appears to be a relatively strong positive correlation between τ_{w} and a as well.

Often, correlations between two parameters are not readily apparent from a multidimensional pairplot. In a typical pairplot, we observe the relationships between all parameters in a multidimensional space, meaning that many parameters can vary simultaneously, which can obscure specific correlations between just two parameters. Thus, it is not necessarily evident from the marginal distributions if the parameters are constrained to a narrow region in the parameter space. To address this, we can examine 2D slices of the posterior by keeping all but two parameters constant if we chose a certain condition for the constant parameters. This approach allows us to investigate what data the remaining two parameters can produce, potentially revealing correlations that we did not observe previously, where deviations in one parameter can be compensated by changes in another. Figure 34 illustrates such a conditional pairplot, where the original observation $\mathrm{obs}_{\mathrm{slow}}$ was chosen as the condition. Here, we again observe the anti-correlation between τ_{w} and b, as well as the correlation between τ_{w} and a.

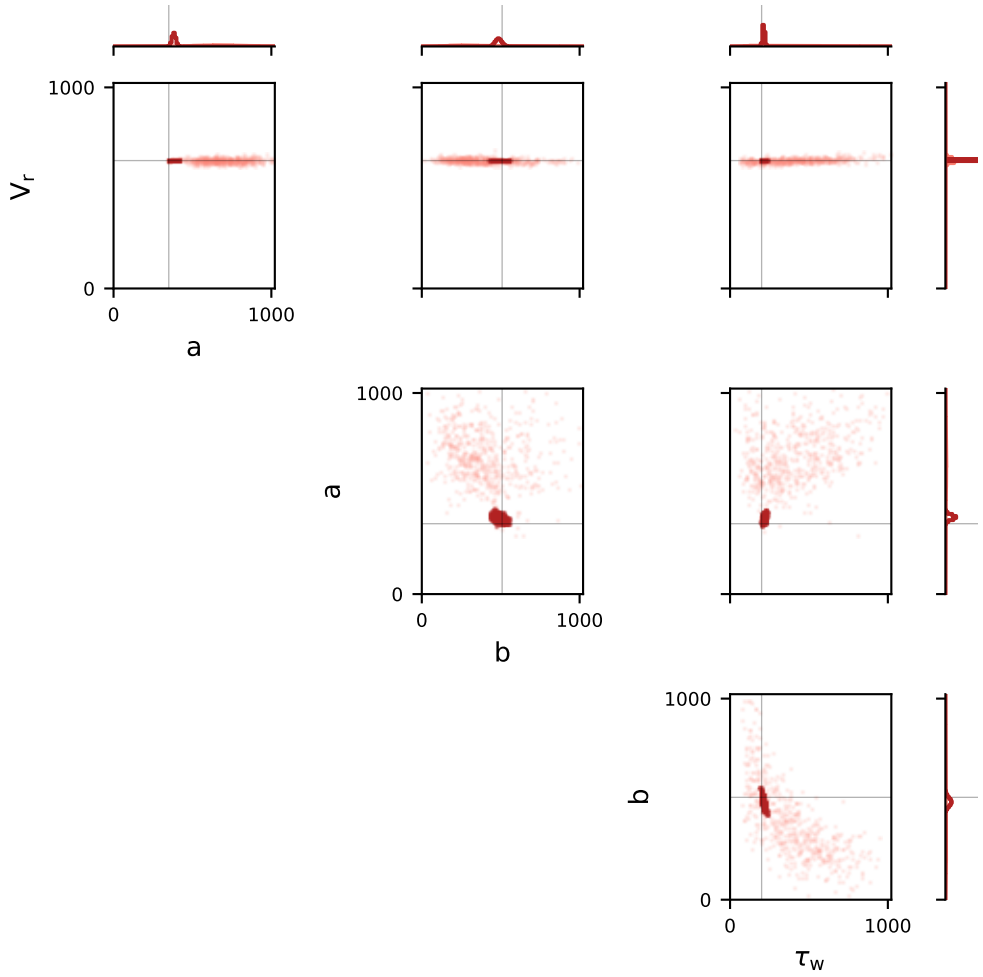


Figure 30: Samples drawn from the 4D posterior for the target observation $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE with a bigger latent space size of 64 and additional concurrent retraining with a the NDE of the SNPE algorithm was used as the embedding technique. The number of hidden features of the NDE was raised from from 50 to 100 and the number of transformations from 5 to 10. 20 inference rounds with 1000 simulations each were performed. The deep red samples represent the posterior after round 20, while the light red points illustrate the posterior from round 1.

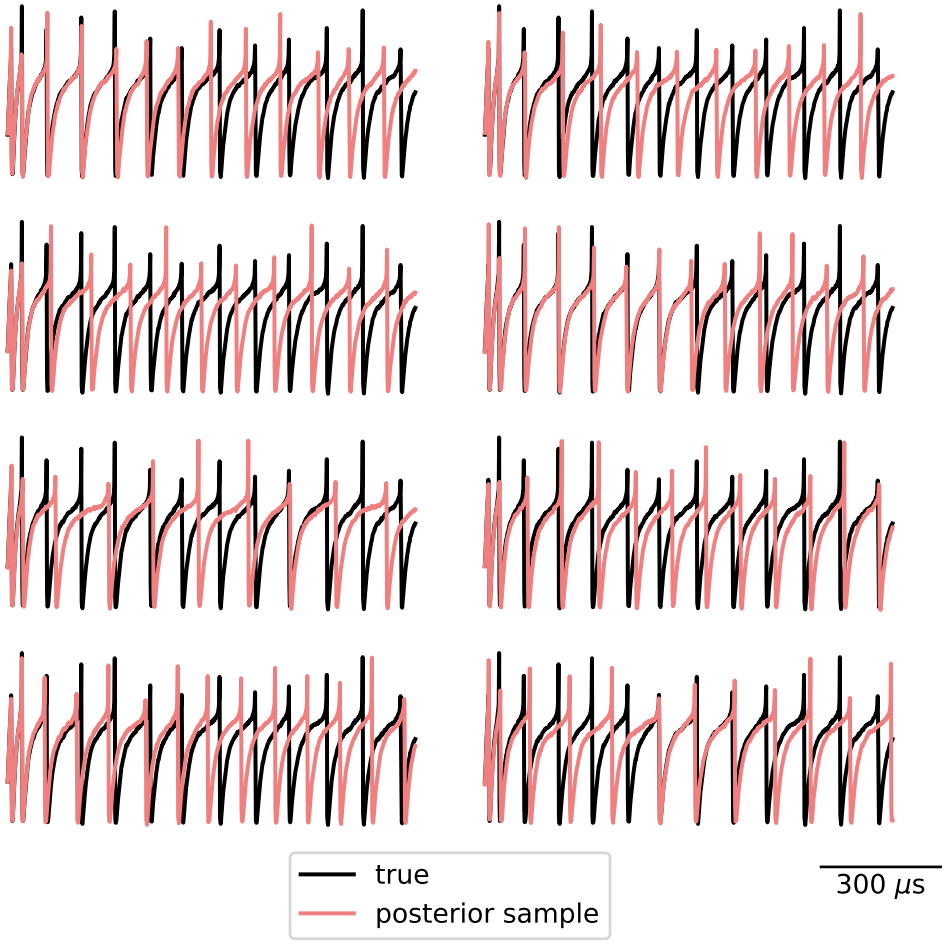


Figure 31: Generated voltage traces with drawn parameters from the posterior displayed in figure 30. $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5 served as the target observation which is shown in black.

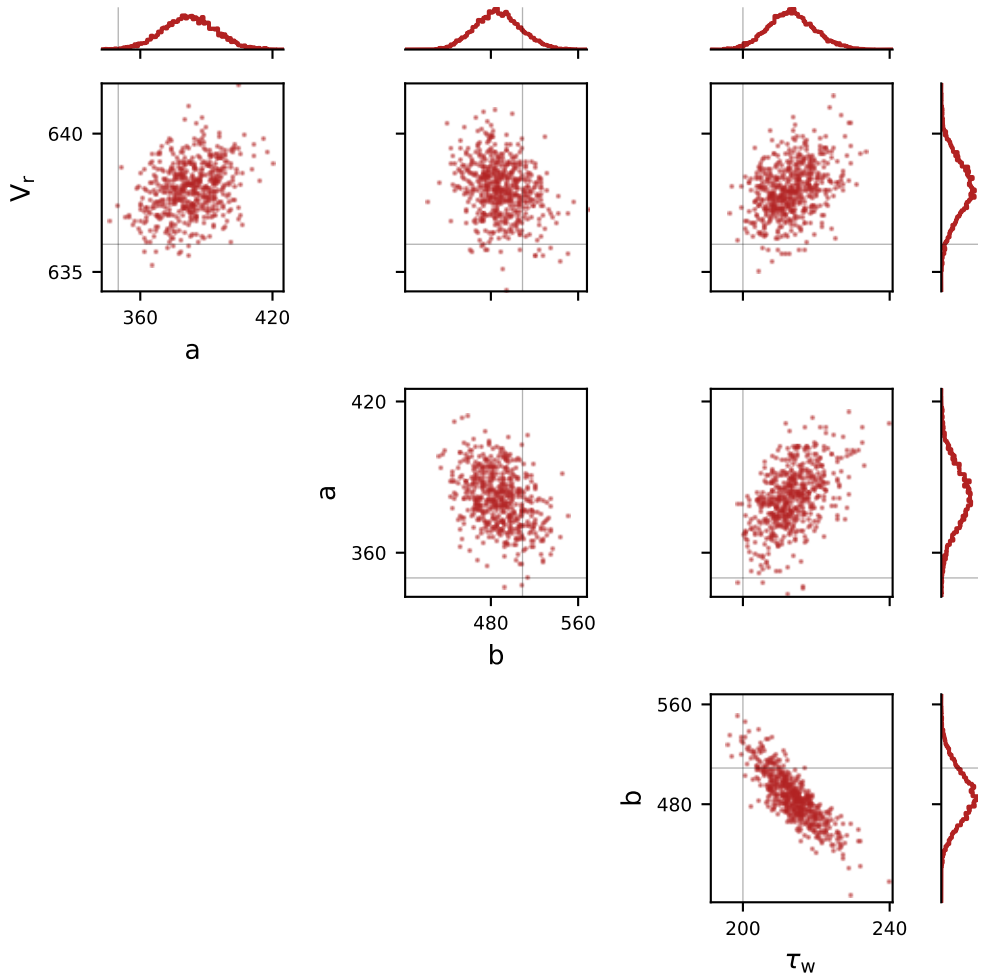


Figure 32: Zoomed-in view of the pairplot in figure 30.

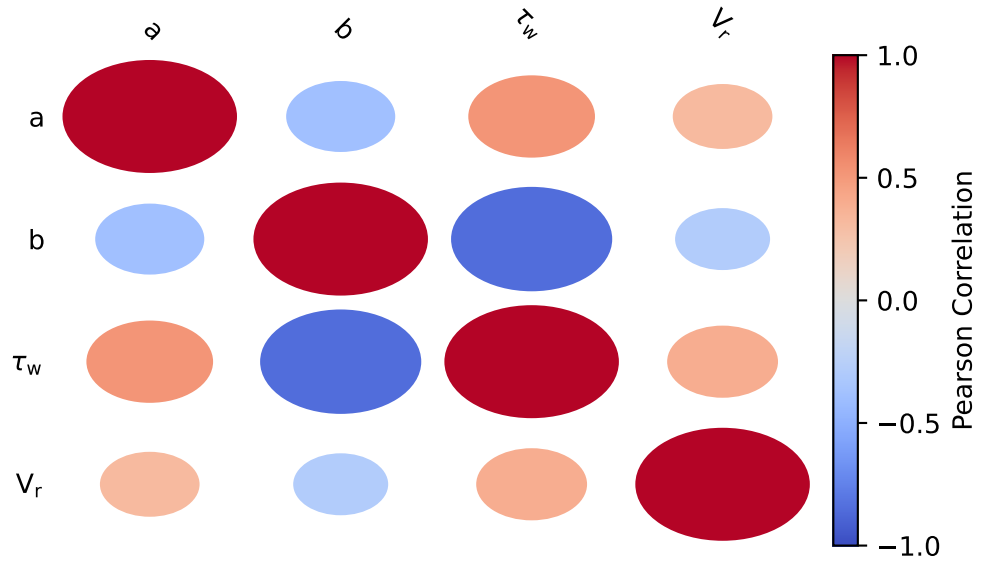


Figure 33: Visualization of the Pearson correlation coefficients of the parameters inferred through the posterior displayed in figures 30 and 32.

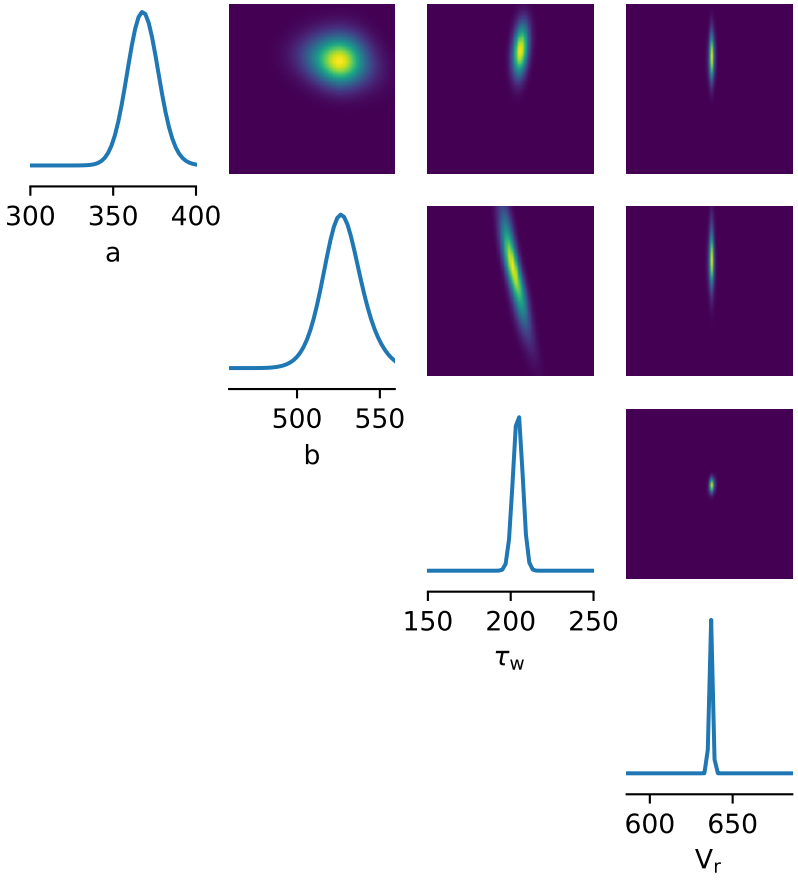


Figure 34: Conditional pairplot of the posterior displayed in figures 30 and 32. The original observation $\hat{x}_{\text{slow}}$ was chosen as the condition.

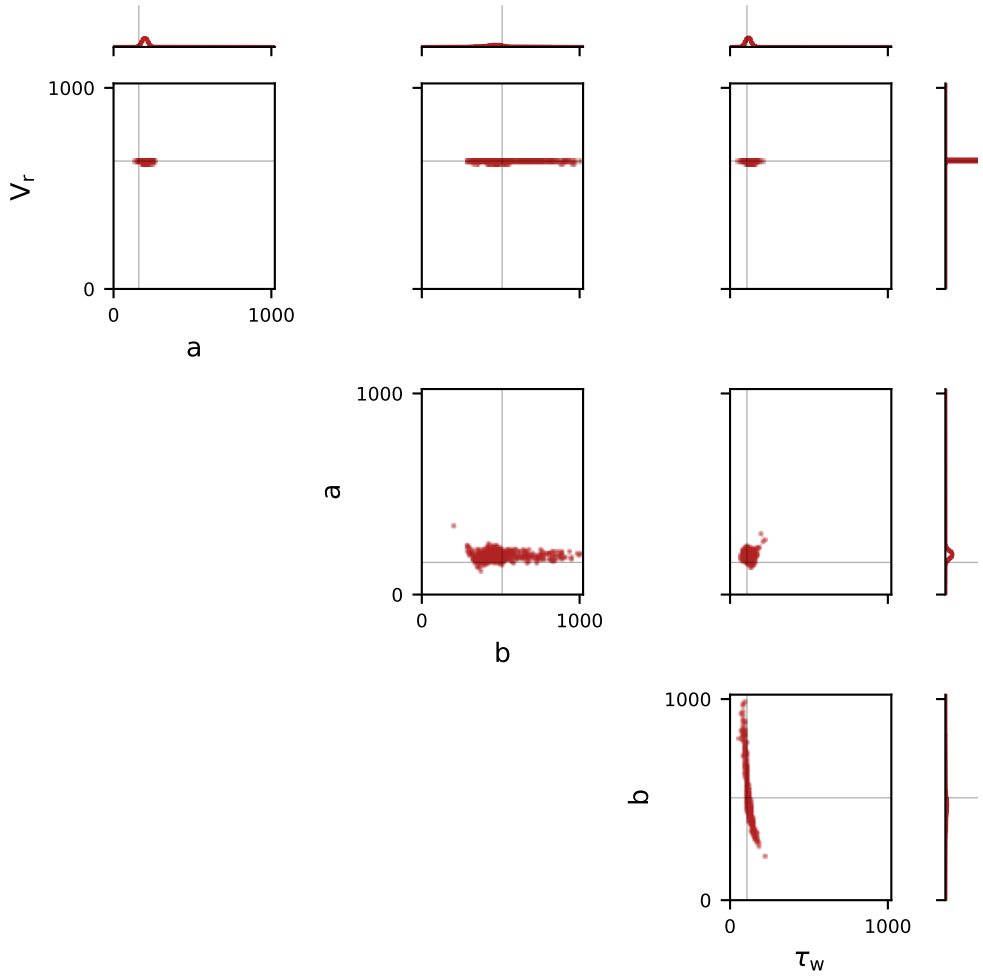


Figure 35: Samples drawn from the 4D posterior for the target observation $\hat{x}_{\text{fast}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE with additional concurrent retraining with the NDE of the SNPE algorithm was used as the embedding technique. 20 inference rounds with 1000 simulations each were performed.

Lastly, we also demonstrate the inference results under the same condition as in figure 29, but now with the observation $\hat{x}_{\text{fast}}$, which is displayed in figure 35. The results differ, indicating that the quality of inference depends on the specific observation which is used. Some observations may be easier for the algorithm to infer parameters from than others.

8D inference Since we achieved quite satisfying results on the 4D parameter inference problem, we extended our pipeline to an eight-dimensional problem, where additional parameters, namely Δ_T, V_T, E_L, and V_T, needed to be inferred. Figure 36 shows samples drawn from the approximated posterior, using the encoder of the CONV-AE with additional retraining as the data embedding technique. The target observation was $\hat{x}_{\text{fast}}$. It is evident that

the parameters V_T, V_r, and V_{th} were inferred quite accurately, whereas the algorithm struggled to pinpoint precise regions in the parameter space for the remaining hardware parameters.

Additionally, we performed inference on the slow observation $\hat{x}_{\text{slow}}$ (see figure 37). In this case, we used the encoder of the CONV-AE with a larger latent size of 64 instead of 32. The encoder was also retrained in conjunction with the NDE. The results in the marginal distributions were similar to the previous approximated posterior shown in figure 36, although the drawn samples exhibited slight variations.

Furthermore, we attempted the problem with a more complex NDE featuring 100 hidden features and 10 transformations. The number of simulations was also increased. 4000 simulations were performed in the first round, while 2000 simulations were carried out in each subsequent round. In this case, we utilized samples from a truncated posterior since drawing samples from the full posterior became too time-consuming. However, the results were not promising, as the algorithm failed to identify a refined region within the parameter space, leading to a very broad posterior (see figure 44).

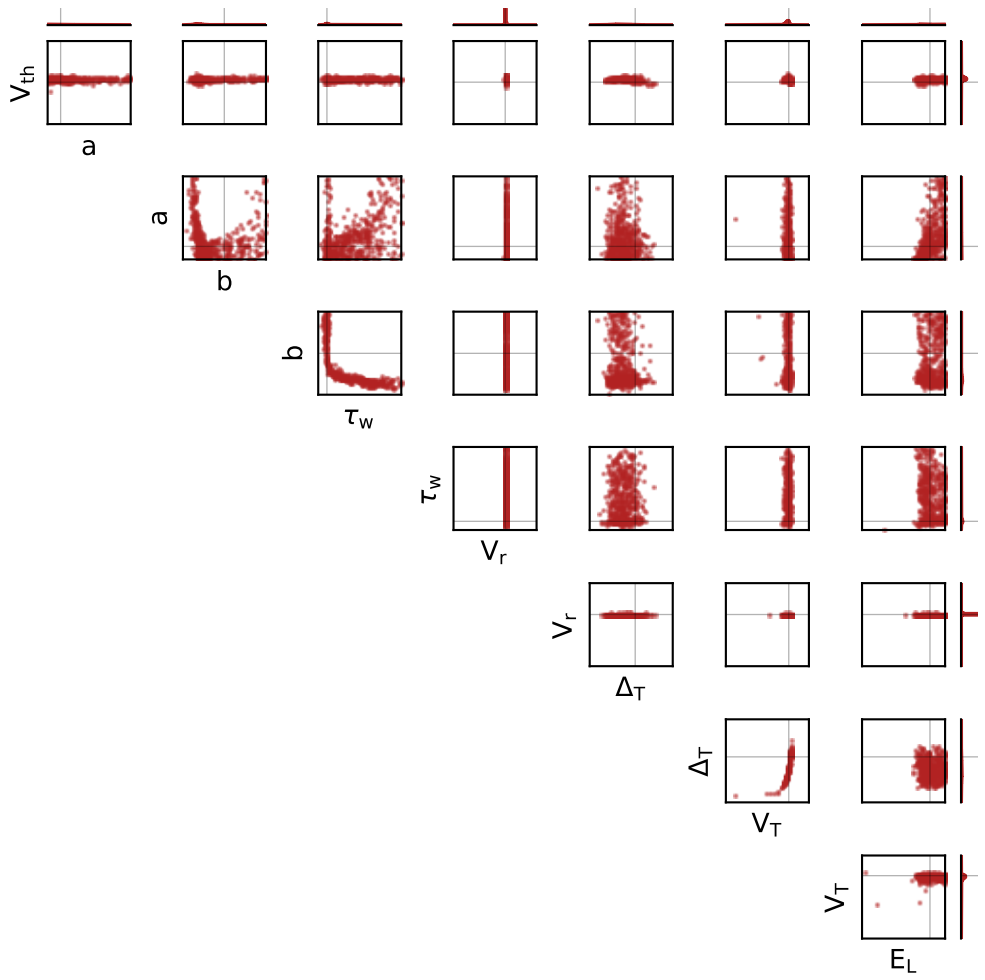


Figure 36: Samples drawn from the 8D posterior for the target observation $\hat{x}_{\text{fast}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE with additional concurrent retraining with the NDE of the SNPE algorithm was used as the embedding technique. 20 inference rounds were performed. 3000 simulations were conducted in the first round while 1000 simulations were carried out in each subsequent round. The plot displays the whole possible parameter range from 0 to 1022 for each parameter.

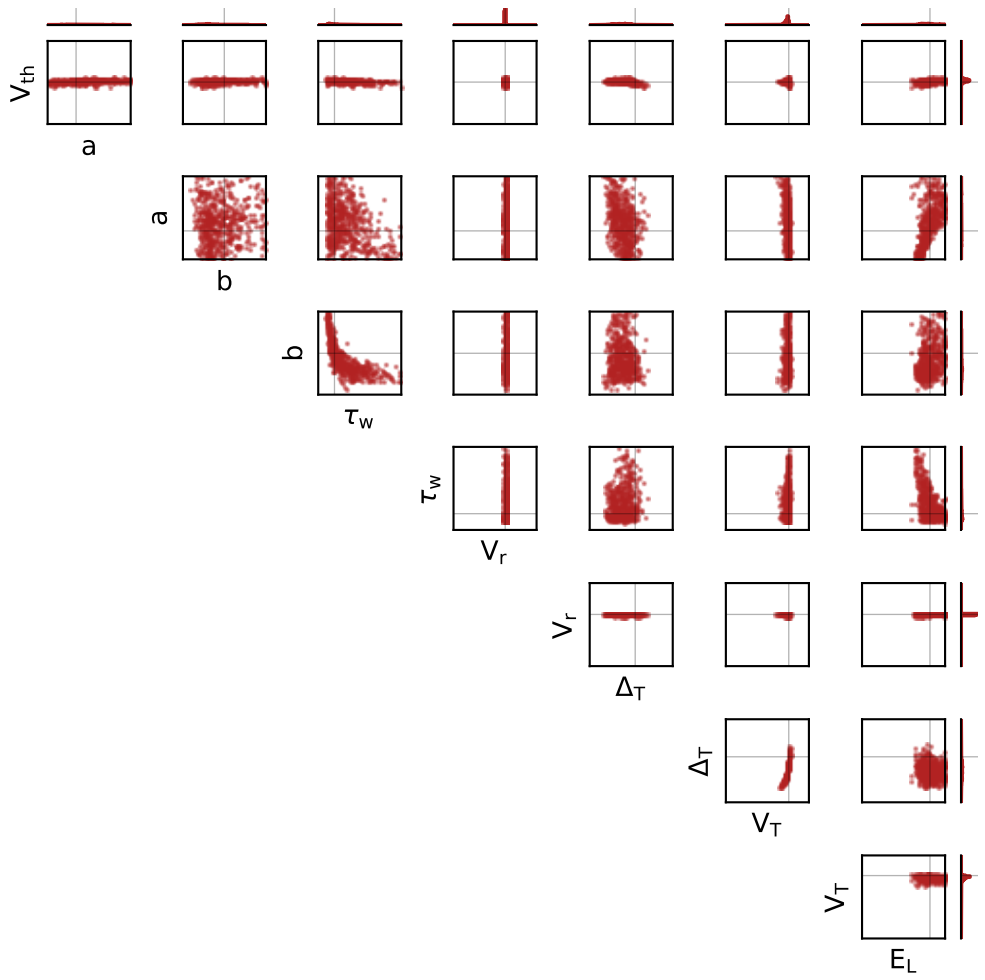


Figure 37: Samples drawn from the 8D posterior for the target observation $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE with a latent space size of 64 and additional concurrent retraining with the NDE of the SNPE algorithm was used as the embedding technique. 20 inference rounds were performed. 3000 simulations were conducted in the first round while 1000 simulations were carried out in each subsequent round. The plot displays the whole possible parameter range from 0 to 1022 for each parameter.

5. Summary and Discussion

The central question in this thesis was whether we can correctly infer multiple hardware parameters of emulated voltage traces on the BrainScaleS-2 system using the sequential neural posterior estimation (SNPE) algorithm. Therefore, we implemented different embedding methods that learn summary statistics of the data to ensure that the SNPE algorithm only receives a compressed version of the time series data. This helps reduce computational cost and improve inference accuracy, as only the important features of the data are considered. The embedding of data is especially important for longer traces with many data points. In this context, we had a primary focus on autoencoders.

5.1. Datasets

In order to train and test our autoencoder models, we needed sufficient and accurate data. We developed an efficient data generation and storage process. Using this, one simulated and three emulated datasets with varying numbers of variable parameters were created (see tables 2 and 1).

In our emulated datasets, we obtained a great diversity of different traces (see figure 9). The dataset size of 200,000 traces could have been smaller for our purposes. However, one should keep the curse of dimensionality in mind when determining the dataset size. As the number of variable parameters increases, exponentially more samples are needed, as data points become sparse in the higher-dimensional space. Therefore, larger datasets are necessary if one intends to train autoencoders on data with more variable parameters. Additionally, the parameter ranges of the datasets have a significant impact on the effective training of the autoencoder, as the data might be too similar or overly variable, depending on the ranges.

In this thesis, we did not delve deeper into other properties of our datasets. For example, certain types of traces may be underrepresented, affecting the ratios between different classes. It is possible that a specific slow-spiking trace with a distinct spike reset shape occurs only in a narrow region of the parameter space, resulting in a limited number of samples. This can lead to the autoencoder not learning these types of traces effectively because their limited presence in the dataset does not significantly influence the loss. Consequently, if this particular trace is selected during inference, the autoencoder may struggle to encode it accurately. In our case, the autoencoders encountered the most difficulty when reconstructing very fast-spiking traces, which may have been rare in the datasets, as the adaptation variables were varied,

often leading to predominantly slow-spiking traces.

5.2. Data embedding

Using autoencoders for low-dimensional representation learning was the primary method investigated in this thesis. Therefore, we implemented two models, the convolutional autoencoder (CONV-AE) and the convolutional-LSTM autoencoder (CONV-LSTM-AE), whose architectures are displayed in tables 8 and 9. Furthermore, we developed a carefully designed training routine that includes several steps, ranging from data preprocessing and learning rate schedules to weighted losses. However, the latter is only useful for datasets with very similar traces.

When we compared the two models against each other, it was evident that the CONV-LSTM-AE performed quite poorly in the loss curves as well as in the reconstruction quality (see figure 10 and figure 11). The training was even prematurely terminated due to a lack of improvement in the validation loss. This indicated that the model was very prone to overfitting, as the validation error stagnated while the training error still decreased. Even a dropout probability of 0.5 could not mitigate the issues observed. One reason for this might be that the CONV-LSTM-AE is a much larger model with significantly more learnable parameters than the CONV-AE (see tables 8 and 9). A larger model has many more weights that can be adjusted, which can lead to the model capturing unimportant features. Additionally, the training loss of the CONV-LSTM-AE was systematically higher than that of the CONV-AE, indicating that this model was generally not well-suited for the task or inadequately trained and thus struggled to learn the data effectively. In addition, it remains uncertain why the validation error of that model was already higher than the training error in the first epoch. Normally, the training error is much higher in the first epoch due to random initialization, as the weights have no prior knowledge of the data and quickly decrease after around 200 batches. Thus, it is also possible that there was a systematic error in the implementation of the CONV-LSTM-AE that could not be identified.

The CONV-AE, on the other hand, performed quite well in the reconstruction traces and test metrics (see figure 11 and table 3). We investigated the impact of different latent sizes for this model. If the latent size is too small, it can not learn all of the important features. If it is too large, it may lead to overfitting, longer training times, and a loss of compression ability. Therefore, it is essential to find the optimal trade-off between compression ratio and reconstruction quality. However, as shown in figure 41, the specific model's

performance heavily depends on the given dataset. If the dataset has more complexity, it is likely that more features need to be learned and a bigger latent size is thus necessary.

Finally, we also investigated the denoising ability of the CONV-AE. To do this, we trained the model by artificially adding noise while calculating the loss with respect to the clean trace. Figure 14 displays the denoising performance on randomly chosen samples from the test set with a signal-to-noise ratio (SNR) of 20 dB, which worked quite well. However, the validation and test loss were higher compared to training without noise (see figure 13 and table 3).

In general, the peaks of a voltage trace are challenging to learn as can be seen in figures 11 or 14. That is because they often vary due to the irregular sampling of voltage values on the hardware or the fixed simulation time grid in simulations. According to theory, these peaks should consistently reach a fixed threshold before resetting. However, this threshold is not always aligned with the simulation time grid or is not always sampled, leading to values captured before or after the peak that do not accurately reflect the true peak. Additionally, peaks often get truncated during the interpolation process. As a result, the height of the peaks exhibits a random nature, making it difficult, if not impossible, for the autoencoder to learn effectively.

It should be noted that the loss from the autoencoder trained on the simulations dataset was smaller than the losses when it was trained on the emulation datasets. This is because the adaptation was quite strong in the simulations dataset, resulting in most traces exhibiting spiking primarily at the beginning and then stabilizing. Consequently, the majority of traces do not differ significantly, making it easier for the model to learn effectively.

5.3. Simulation-based inference

We first tested the entire inference procedure on simulated data, yielding promising results for embeddings with the encoder of the CONV-AE, the fully connected neural network (FCNN), as well as with wavelet transforms. Furthermore, feeding in the raw 1024 data points after interpolation also worked surprisingly well (see chapter 4.3.1). However, we noticed that the shape of the resulting posterior in the 2D pairplots varies depending on the embedding technique which is used. For instance, there is a correlation between the variables b and V_r in the posterior when using the encoder of the CONV-AE as an embedding (see figure 17). In contrast, the posterior shape of these parameters with the FCNN as an embedding appears completely uncorrelated (see figure 19). This is an interesting discovery. Different embedding techniques may

either neglect certain features of the data, thereby obscuring potential correlations, or they might impose those correlations themselves. Additionally, it is also possible that the SNPE algorithm exhibits variability in different runs, leading to different posterior shapes. This would need to be investigated by conducting multiple inference runs under the same conditions and comparing the resulting posteriors.

However, when running inference on emulated data, we encountered several challenges. The CONV-AE did not perform as expected, unlike in the simulations (see figure 25).

A reason for this is that emulations on the hardware are not entirely deterministic. As shown in figure 28, even emulations with the same set of parameters can vary significantly in their inter-spike intervals. This is due to the intrinsic noise present in the hardware. At this point, we can only hypothesize about the types of noise responsible for these deviations. For instance, it is possible that the digitally set parameters are not precisely translated onto the hardware, but rather that some variance is present. Thus, the final hardware parameters might deviate slightly from the intended values. Additionally, other noise sources, such as varying source voltage, could also contribute to these variations. To address this issue, it would be necessary to systematically investigate the sources of the noise and explore potential mitigation strategies to make hardware emulations more accurate. On top of that, we also did not select the target trace to be stable by optimizing the dynamic ranges.

It is also important to note that the adaptive exponential integrate-and-fire (AdEx) parameters do not directly correspond to the hardware parameters. While the hardware parameters are aligned with the AdEx parameters, variations due to the specific implementation of the hardware are indeed possible. Furthermore, the ranges of the hardware parameters do not automatically match the ranges from the AdEx parameters. If one wishes to achieve the exact same conditions, it would be essential to determine which regions of the hardware parameter space correspond to which regions in the AdEx parameter space (similar to what was done in Billaudelle et al., 2022). It could also be the case that we are operating within a parameter space defined by the hardware parameters where inference using simulations would also perform poorly.

Since we often observed a slight systematic shift of the posterior from the true observations (for instance, see figure 32), it is possible that our target observation $\hat{x}_{\text{slow}}$ corresponds to a trace that occurs less frequently due to try-to-try variations, making it resemble more traces with slightly shifted parameters. Hence, our approximated posteriors might be overconfident, too.

Unperturbed by this setback, we then attempted to retrain the encoder of the CONV-AE concurrently with the neural density estimator (NDE) of the SNPE algorithm, which worked quite well (see figure 29). The algorithm was able to identify a narrow region in the parameter space near the true observation. It should be noted that only because the posterior is broad and not point-like at the observation that it represents a poor approximation. It is also possible that there is high variance in the parameter choices or that correlation and compensation mechanisms exist between the parameters. Conversely, the approximated posterior can be overconfident, meaning it is too narrow and neglects variances in the parameters that still produce a relatively similar voltage trace. In our case, the emulations with the drawn posterior samples agree well with the target observation when considering the try-to-try variations (see figure 31).

For that approximated posterior, we also investigated possible correlations (see figure 33). We found a strong anticorrelation between parameters b and τ_{w}, as well as a notable positive correlation between a and τ_{w}. The anticorrelation between b and τ_{w} can be explained through the AdEx equations 1 and 2. Specifically, τ_{w} governs how quickly the adaptation current w changes. A higher value of the spike-triggered adaptation b, on the other hand, leads to higher values of the adaptation current w when a spike occurs. Thus, if b increases, the adaptation current w must decrease more rapidly, which can be achieved by reducing the value of τ_{w} to produce a similar voltage trace, and vice versa. On the other hand, the correlation between a and τ_{w} is less straightforward due to the differing potential signs in equation 2, since the change of w also depends on the specific value of the membrane voltage V. The increase or decrease of w does not only depend on the subtreshold adaptation a but also on the current memrbane potential V. However, as we observed in the case of simulation, correlations might also arise due to the specific type of embedding technique used. One must assess the stability of the embeddings by running the SNPE algorithm multiple times to see if consistent results are obtained.

We have seen that the inference results depend on several factors. One key factor is the embedding technique, which has a clear impact on the approximated posterior. The encoder without additional retraining performed worse than the encoder that was retrained simultaneously with the NDE, suggesting that while the encoder learned good features for reconstructing the original trace, these features may not be optimal for parameter inference with the SNPE algorithm. Thus, retraining the encoder makes sense. Moreover, the algorithm converges faster when the encoder is retrained rather than kept static (see figures 23 and 24).

Furthermore, we observed that the inference results also depend on the specific type of target observation whose parameters are being inferred. For the faster observation $\hat{x}_{\text{fast}}$, the results were more imprecise (see figure 35) compared to the results of the slower spiking observation $\hat{x}_{\text{slow}}$ (see figure 29).

On top of that, the inference results also heavily depend on the specific hardware parameters being inferred. Some parameters can be found more precisely than others. As seen in figure 37, the marginals of the parameters V_{th}, V_{r}, and V_{T} are more narrow than those of other parameters. This could be because those parameters impact more distinct properties, such as thresholds and boundaries, which likely have little to no correlation with other parameters, resulting in clearer shifts in the voltage trace. In contrast, adaptation variables are more complex, as multiple parameters can influence changes in inter-spike intervals, making them harder to isolate.

In the 8D inference problem, the algorithm had difficulty identifying a well-defined region in the entire parameter space (see figure 37). When we increased the complexity of the NDE, the results deteriorated even further (see figure 44). Since standard sampling performed poorly, we were compelled to use sampling from a truncated posterior. This indicates that a significant portion of the approximate posterior lies outside the support of the prior. Therefore, we may have needed more simulations per round, as the larger network might require additional training information.

This may have been due to the poor performance of standard sampling methods, which prompted us to use truncated sampling. While truncated sampling does speed up the process, it comes with several disadvantages. It can negatively affect results by introducing bias, reducing variance, and potentially missing important global features of the posterior distribution, which can lead to overconfidence and a skewed parameter estimation.

As a note, fewer rounds of SNPE can also be sufficient for many cases. In our work, we performed multiple rounds mainly to study the development of the embeddings, as we did not have much time for more extensive testing.

6. Outlook

The future goal of this work is to accurately emulate recorded voltage traces of a biological neuron on the BrainScaleS-2 hardware. However, achieving this goal requires inferring all 24 variable hardware parameters. Since our model already struggled to accurately infer eight parameters, there is much work ahead of us. However, the inference pipeline can still be optimized in many ways.

First of all, extensive systematic testing needs to be conducted. This includes finding optimal hyperparameters for the neural density estimator (NDE) and exploring other NDEs, such as neural spline flows (Durkan et al., 2019). Additionally, we need to identify better hyperparameters for the autoencoder and test the relationship between the sequential neural posterior estimation (SNPE) algorithm and different embedding techniques. For further potential optimization tests, see chapter 5.

One major step that could optimize and reduce variability in the entire inference pipeline is to investigate the specific reasons behind the trial-to-trial variations. If we can address and diminish these underlying issues, the parameters can likely be determined more accurately.

If one wants to infer even more parameters, it may be necessary to implement even better autoencoders to compress the time series data. A prominent candidate for that are "temporal convolutional autoencoders" (Bai, Kolter, and Koltun, 2018; Lea et al., 2017; Z. Li, Sun, et al., 2022). These models leverage causal convolutions, where only the present and past data points of the time series are convolved, excluding any future data points. Additionally, they utilize layers of dilated convolutions to gain a larger receptive field, thereby allowing for a more extensive contextual understanding of the data. Furthermore, improvements in autoencoder techniques, such as incorporating adaptive loss terms with regularization, can enhance model performance (Tschannen, Bachem, and Lucic, 2018).

Aside from its denoising effect, the autoencoder can also function as an initial check to determine whether a trace is even possible to emulate on the hardware. In this context, the Autoencoder can serve as an anomaly detector. The idea is that the Autoencoder is trained on all possible voltage traces that the BrainScaleS-2 hardware can produce. As a result, the Autoencoder can reconstruct those traces very well. However, if a trace is fed into the encoder that significantly differs from the training data, the reconstruction loss will be much higher. By setting a threshold, this reconstruction loss can be used as

an anomaly metric, thus serving as a first predictive check to assess whether inference can be successful in the first place.

If one finally obtains a good approximation of the high-dimensional posterior, more analyses and diagnostic tests can be performed. First, it is essential to find a suitable metric to quantify the similarity between two time series. For instance, using the mean squared error can be problematic, as even a single time shift or adaptation shift in the traces can accumulate and affect all subsequent time points in the time series. Once an appropriate metric is established, systematic posterior predictive checkss (PPCs) can be conducted to assess the quality of the posterior.

If the posterior is too confident, one can also combine an ensemble of multiple posteriors to get closer to the true posterior. The aim is to approximate the same posterior with different SNPE runs, as the results may vary. By averaging over multiple posteriors, one may come closer to the true posterior. The final sampling is then performed by sampling equally from all posteriors.

Furthermore, one can perform sensitivity analysis and correlation analysis in high-dimensional space to better understand the relationships between parameters. Through posterior analysis, one can learn a great deal about the relationships of hardware parameters, which may even be useful for designing the next generation of hardware (e.g., to mitigate certain unwanted relationships that one might want to avoid). For more analytical methods, see Lueckmann, Boelts, et al., 2021 and Tolley et al., 2024.

References

Aremu, Oluseun Omotola, David Hyland-Wood, and Peter Ross McAree (2020). "A machine learning approach to circumventing the curse of dimensionality in discontinuous time series machine data". In: *Reliability Engineering & System Safety*. DOI: https://doi.org/10.1016/j.ress.2019.106706.

Ashraf, Mohsena et al. (2023). "A Survey on Dimensionality Reduction Techniques for Time-Series Data". In: *IEEE Access*. DOI: 10.1109/ACCESS.2023.3269693.

Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun (2018). "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In: DOI: 10.48550/arXiv.1803.01271.

Benesty, Jacob et al. (2009). *Noise Reduction in Speech Processing - Pearson Correlation Coefficient*. Springer Berlin Heidelberg. ISBN: 978-3-642-00296-0. DOI: 10.1007/978-3-642-00296-0_5.

Berkhof, Johannes, Ive van Mechelen, and Herbert Hoijtink (2000). "Posterior predictive checks: Principles and discussion". In: *Computational Statistics*. DOI: 10.1007/s001800000038.

Bialek, William et al. (1991). "Reading a Neural Code". In: *Science*. DOI: 10.1126/science.2063199.

Billaudelle, Sebastian et al. (2022). "An accurate and flexible analog emulation of AdEx neuron dynamics in silicon". In: *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. DOI: 10.1109/ICECS202256217.2022.9971058.

Brette, Romain and Wulfram Gerstner (2005). "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity". In: *J Neurophysiol*. DOI: 10.1152/jn.00686.2005.

Campo, Adrià Tauste (2020). "Inferring neural information flow from spiking data". In: *Computational and structural biotechnology journal*. DOI: 10.1016/j.csbj.2020.09.007.

Chen, Tingting et al. (2020). "Unsupervised Anomaly Detection of Industrial Robots Using Sliding-Window Convolutional Variational Autoencoder". In: *IEEE Access*. DOI: 10.1109/ACCESS.2020.2977892.

Chiang, Hsin-Tien et al. (2019). "Noise Reduction in ECG Signals Using Fully Convolutional Denoising Autoencoders". In: *IEEE Access*. DOI: 10.1109/ACCESS.2019.2912036.

Chiarot, Giacomo and Claudio Silvestri (2023). "Time Series Compression Survey". In: *Association for Computing Machinery*. DOI: 10.1145/3560814.

Constantine, Paul G., Eric Dow, and Qiqi Wang (2014). "Active Subspace Methods in Theory and Practice: Applications to Kriging Surfaces". In: *SIAM Journal on Scientific Computing*. DOI: 10.1137/130916138.

Cranmer, Kyle, Johann Brehmer, and Filles Louppe (2020). "The frontier of simulation-based inference". In: *Proceedings for the Sackler Colloquia at the US National Academy of Sciences*. DOI: 10.48550/arXiv.1911.01429.

Dasan, Evangelin and Ithayarani Panneerselvam (2021). "A novel dimensionality reduction approach for ECG signal via convolutional denoising autoencoder with LSTM". In: *Biomedical Signal Processing and Control*. DOI: 10.1016/j.bspc.2020.102225.

Davison, Andrew P. et al. (2009). "PyNN: a common interface for neuronal network simulators". In: *Frontiers in Neuroinformatics*. DOI: 10.3389/neuro.11.011.2008.

Deistler, Michael, Pedro J. Goncalves, and Jakob H. Macke (2022). "Truncated proposals for scalable and hassle-free simulation-based inference". In: *Advances in Neural Information Processing Systems*. URL: https://openreview.net/forum?id=QW98XBAqNRa.

Durkan, Conor et al. (2019). "Neural Spline Flows". In: *SIAM Journal on Scientific Computing*. URL: https://arxiv.org/abs/1906.04032.

Fourcaud-Trocmé, Nicolas et al. (2003). "How Spike Generation Mechanisms Determine the Neuronal Response to Fluctuating Inputs". In: *The Journal of Neuroscience*. DOI: 10.1523/JNEUROSCI.23-37-11628.2003.

Gerstner, Wulfram and Werner Kistler (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press. ISBN: 0 521 89079 9. URL: https://neuronaldynamics.epfl.ch/online/index.html.

Goncalves, Pedro J et al. (2020). "Training deep neural density estimators to identify mechanistic models of neural dynamics". In: *eLife*. DOI: 10.7554/eLife.56261.

Greenberg, David S., Marcel Nonnenmacher, and Jakob H. Macke (2019). "Automatic Posterior Transformation for Likelihood-free Inference". In: *Proceedings of the 36th International Conference on Machine Learning*. DOI: 10.48550/arXiv.1905.07488.

Gulati, Anil (2015). "Understanding neurogenesis in the adult human brain". In: *Indian J Pharmacol*. DOI: 10.4103/0253-7613.169598.

Guo, Yifan et al. (2018). "Multidimensional Time Series Anomaly Detection: A GRU-based Gaussian Mixture Variational Autoencoder Approach". In: *PMLR*. URL: https://proceedings.mlr.press/v95/guo18a.html.

Hasler, Jennifer and Harry B. Marr (2013). "Finding a roadmap to achieve large neuromorphic hardware systems". In: *Frontiers in Neuroscience*. DOI: 10.3389/fnins.2013.00118.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation*. DOI: 10.1162/neco.1997.9.8.1735.

Huhle, Jakob (2024). *Exploration of the AdEx parameter space through simulation and simulation-based-inference*. URL: https://www.kip.uni-heidelberg.de/vision/publications/reports/.

Iberri, D. (2007). *File: Action potential.svg*. Accessed: 20024-09-27. URL: https://upload.wikimedia.org/wikipedia/commons/b/bc/Neuron_Hand-tuned.svg.

Indiveri, Giacomo et al. (2011). "Neuromorphic Silicon Neuron Circuits". In: *Frontiers in Neuroscience*. DOI: 10.3389/fnins.2011.00073.

Izhikevich, Eugene M. (2003). "Simple Model of Spiking Neurons". In: *IEEE Transactions on Neuronal Networks*. DOI: 10.1109/TNN.2003.820440.

J. Touboul, R. Brette (2008). "Dynamics and bifurcations of the adaptive exponential integrate-and-fire model". In: *Biol Cybern*. DOI: 10.1007/s00422-008-0267-4.

Jarosz, Q. (2009). *File: Neuron Hand-tuned.svg*. Accessed: 20024-09-27. URL: https://upload.wikimedia.org/wikipedia/commons/4/4a/Action_potential.svg.

Jolivet, Renaud et al. (2008). "A benchmark test for a quantitative assessment of simple neuron models". In: *J Neurosci Methods*. DOI: 10.1016/j.jneumeth.2007.11.006.

Kaiser, Jakob et al. (2023). "Simulation-based inference for model parameterization on analog neuromorphic hardware". In: *Neuromorphic Computing and Engineering*. DOI: 10.1088/2634-4386/ad046d. URL: https://dx.doi.org/10.1088/2634-4386/ad046d.

Kalra, Dayal and Maissam Barkeshli (2024). *Why Warmup the Learning Rate? Underlying Mechanisms and Improvements*. DOI: 10.48550/arXiv.2406.09405.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR*. URL: https://api.semanticscholar.org/CorpusID:6628106.

Lea, Colin et al. (2017). "Temporal Convolutional Networks for Action Segmentation and Detection". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/CVPR.2017.113.

Li, Pengzhi, Yan Pei, and Jianqiang Li (2023). "A comprehensive survey on design and application of autoencoder in deep learning". In: *Applied Soft Computing*. DOI: 10.1016/j.asoc.2023.110176.

Li, Zhiyuan and Sanjeev Arora (2019). *An Exponential Learning Rate Schedule for Deep Learning*. DOI: 10.48550/arXiv.1910.07454.

Li, Zhiyuan, Yu Sun, et al. (2022). "Unsupervised Machine Anomaly Detection Using Autoencoder and Temporal Convolutional Network". In: *IEEE Transactions on Instrumentation and Measurement*. DOI: 10.1109/TIM.2022.3212547.

Lima, Felipe Tomazelli and Vinicius M.A. Souza (2023). "A Large Comparison of Normalization Methods on Time Series". In: *Big Data Research*. DOI: https://doi.org/10.1016/j.bdr.2023.100407.

Lueckmann, Jan-Matthis, Jan Boelts, et al. (2021). *Benchmarking Simulation-Based Inference*. URL: https://arxiv.org/abs/2101.04653.

Lueckmann, Jan-Matthis, Pedro J Goncalves, et al. (2017). "Flexible statistical inference for mechanistic models of neural dynamics". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/addfa9b7e234254d26e9c7f2af1005cb-Paper.pdf.

Marder, Eve and Adam L Taylor (2011). "Multiple models to capture the variability in biological neurons and networks". In: *Nature Neuroscience*. DOI: 10.1038/nn.2735.

Mark W Barnett, Philip M Larkman (2007). *The action potential; Practical Neurology*. 2nd. Birkhauser. ISBN: 978-3-030-59316-2.

Martin, John H. (2021). *Neuroanatomy: Text and Atlas*. McGraw Hill. ISBN: 978-1-259-64248-7.

Martín-Araguz, A et al. (2002). "Neuroscience in ancient Egypt and in the school of Alexandria". In: *Rev Neurol*.

Mueller, Eric C. (2014). "Novel Operation Modes of Accelerated Neuromorphic Hardware". In: *PhD thesis. Ruprecht-Karls-Universität Heidelberg*. URL: http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3112.

Naud, Richard et al. (2008). "Firing patterns in the adaptive exponential integrate-and-fire model". In: *Biological Cybernetics*. DOI: 10.1007/s00422-008-0264-7.

Oliveira, Marcos A. de et al. (2023). "Time Series Compression for IoT: A Systematic Literature Review". In: *Wireless Communications and Mobile Computing*. DOI: 10.1155/2023/5025255.

Papamakarios, George, Eric Nalisnick, et al. (2021). "Normalizing Flows for Probabilistic Modeling and Inference". In: *Journal of Machine Learning Research*. URL: http://jmlr.org/papers/v22/19-1028.html.

Papamakarios, George, Theo Pavlakou, and Iain Murray (2017). "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6c1da886822c67822bcf3679d04369fa-Paper.pdf.

Paszke, Adam et al. (2019). "PyTorch: an imperative style, high-performance deep learning library". In: *Curran Associates Inc.* DOI: 10.5555/3454287.3455008.

Pehle, Christian et al. (2022). "The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity". In: *Frontiers in Neuroscience*. DOI: 10.3389/fnins.2022.795876.

Richter, Mathias (2020). *Inverse Problems; Basics, Theory and Applications in Geophysics*. 2nd. Birkhauser. ISBN: 978-3-030-59316-2.

Sagheer, Alaa and Mostafa Kotb (2019). "Unsupervised Pre-training of a Deep LSTM-based Stacked Autoencoder for Multivariate Time Series Forecasting Problems". In: *Scientific Reports*. DOI: 10.1038/s41598-019-55320-6.

Schmidhuber, Jürgen (2015). "Deep learning in neural networks: An overview". In: *Neural Networks*. DOI: 10.1016/j.neunet.2014.09.003.

Shinomoto, Shigeru et al. (2009). "Relating neuronal firing patterns to functional differentiation of cerebral cortex". In: *PLoS Comput Biol*. DOI: 10.1371/journal.pcbi.1000433.

Stimberg, Marcel, Romain Brette, and Dan FM Goodman (2019). "Brian 2, an intuitive and efficient neural simulator". In: *eLife*. DOI: 10.7554/eLife.47314. URL: https://doi.org/10.7554/eLife.47314.

Tejero-Canteroe, Alvaro et al. (2020). "sbi: A toolkit for simulation-based inference". In: *Journal of Open Source Software*. DOI: 10.21105/joss.02505.

Tolley, Nicholas et al. (2024). "Methods and considerations for estimating parameters in biophysically detailed neural models with simulation based inference". In: *PLOS Computational Biology*. DOI: 10.1371/journal.pcbi.1011108.

Tschannen, Michael, Olivier Bachem, and Mario Lucic (2018). "Recent advances in autoencoder-based representation learning". In: *Third workshop on Bayesian Deep Learning (NeurIPS 2018)*. URL: http://www.nari.ee.ethz.ch/pubs/p/autoenc2018.

Van Geit, W, E. De Schutter, and P. Achard (2008). "Automated neuron model optimization techniques: a review". In: *Biological Cybernetics*. DOI: 10.1007/s00422-008-0257-6.

Wan, Zhiqiang, Yazhou Zhang, and Haibo He (2017). *Variational autoencoder based synthetic data generation for imbalanced learning*. DOI: 10.1109/SSCI.2017.8285168.

Wang, Fei et al. (2019). "A novel ECG signal compression method using spindle convolutional auto-encoder". In: *Computer Methods and Programs in Biomedicine*. DOI: 10.1016/j.cmpb.2019.03.019.

Weeks, Michael and Magdy Bayoumi (2002). "Discrete Wavelet Transform: Architectures, Design and Performance Issues". In: *Journal of VLSI Signal Processing*. DOI: 10.1023/A:1023648531542.

Windhorst, Uwe and Håkan Johansson (1999). *Modern Techniques in Neuroscience Research*. ISBN: 978-3540644606.

Yildirim, Ozal, Ru San Tan, and U. Rajendra Acharya (2018). "An efficient compression of ECG signals using deep convolutional autoencoders". In: *Cognitive Systems Research*. DOI: 10.1016/j.cogsys.2018.07.004.

Zheng, Yi et al. (2016). "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification". In: *Frontiers of Computer Science*. DOI: 10.1007/s11704-015-4478-2.

Acronyms

AdEx adaptive exponential integrate-and-fire.

CONV-AE convolutional autoencoder.

CONV-LSTM-AE convolutional-LSTM autoencoder.

FCNN fully connected neural network.

FPGA field-programmable gate array.

LSTM long short-term memory.

MAF masked autoregressive flow.

NDE neural density estimator.

PPC posterior predictive checks.

SNPE sequential neural posterior estimation.

SNR signal-to-noise ratio.

A. Appendix

A.1. Fixed hardware parameters

Table 7: Fixed hardware parameter settings throughout all experiments.

parameter	setting
reset_enable_multiplication	True
threshold_enable	True
adaptation_enable_pulse	True
exponential_enable	True
adaptation_enable	True
adaptation_invert_a	False
adaptation_invert_b	False
constant_current_type	source
refractory_period (hardware µs)	3

A.2. Model architectures

Table 8: Conv-AE model overview. A batch size of 32 was used during training.

Layer (type)	Input Shape	Output Shape	Kernel Shape	Param #
Conv_AE_b	(1024)	(1024)	–	–
+– Conv_Encoder_b	(1024)	(32)	–	–
+– Sequential	(1, 1024)	(1, 32)	–	–
+– Conv1d	(1, 1024)	(32, 1024)	5	192
+– ReLU	(32, 1024)	(32, 1024)	–	–
+– BatchNorm1d	(32, 1024)	(32, 1024)	–	64
+– MaxPool1d	(32, 1024)	(32, 512)	2	–
+– Conv1d	(32, 512)	(16, 512)	3	1,552
+– ReLU	(16, 512)	(16, 512)	–	–
+– BatchNorm1d	(16, 512)	(16, 512)	–	32
+– MaxPool1d	(16, 512)	(16, 256)	2	–
+– Conv1d	(16, 256)	(64, 256)	11	11,328
+– ReLU	(64, 256)	(64, 256)	–	–
+– MaxPool1d	(64, 256)	(64, 128)	2	–
+– Conv1d	(64, 128)	(128, 128)	13	106,624
+– ReLU	(128, 128)	(128, 128)	–	–
+– MaxPool1d	(128, 128)	(128, 64)	2	–
+– Conv1d	(128, 64)	(1, 64)	3	385
+– ReLU	(1, 64)	(1, 64)	–	–
+– MaxPool1d	(1, 64)	(1, 32)	2	–
+– Conv_Decoder_b	(32)	(1024)	–	–
+– Sequential	(1, 32)	(1, 1024)	–	–
+– Conv1d	(1, 32)	(128, 32)	3	512
+– ReLU	(128, 32)	(128, 32)	–	–
+– Upsample	(128, 32)	(128, 64)	–	–
+– Conv1d	(128, 64)	(64, 64)	13	106,560
+– ReLU	(64, 64)	(64, 64)	–	–
+– Upsample	(64, 64)	(64, 128)	–	–
+– Conv1d	(64, 128)	(16, 128)	11	11,280
+– ReLU	(16, 128)	(16, 128)	–	–
+– Upsample	(16, 128)	(16, 256)	–	–
+– Conv1d	(16, 256)	(32, 256)	3	1,568
+– ReLU	(32, 256)	(32, 256)	–	–
+– Upsample	(32, 256)	(32, 512)	–	–
+– Conv1d	(32, 512)	(1, 512)	5	161
+– Upsample	(1, 512)	(1, 1024)	–	–
Trainable params:		240,258		
Total mult-adds (M):		842.49		
Input size (MB):		0.13		
Forward/backward pass size (MB):		34.23		
Params size (MB):		0.96		
Estimated Total Size (MB):		35.32		

Table 9: Conv-LSTM-AE model overview. A batch size of 32 was used during training.

Layer (type)	Input Shape	Output Shape	Kernel Shape	Param #
Conv_LSTM_AE	(1024)	(1024)	–	–
+– Conv_LSTM_Encoder	(1024)	(60)	–	–
+– Sequential	(1, 1024)	(1, 32)	–	–
+– Conv1d	(1, 1024)	(10, 1024)	7	80
+– BatchNorm1d	(10, 1024)	(10, 1024)	–	20
+– Tanh, Dropout	(10, 1024)	(10, 1024)	–	–
+– MaxPool1d	(10, 1024)	(10, 512)	2	–
+– Conv1d	(10, 512)	(32, 511)	6	1,952
+– BatchNorm1d	(32, 511)	(32, 511)	–	64
+– Tanh, Dropout	(32, 511)	(32, 511)	–	–
+– MaxPool1d	(32, 511)	(32, 255)	2	–
+– Conv1d	(32, 255)	(32, 255)	5	5,152
+– BatchNorm1d	(32, 255)	(32, 255)	–	64
+– Tanh, Dropout	(32, 255)	(32, 255)	–	–
+– MaxPool1d	(32, 255)	(32, 127)	2	–
+– Conv1d	(32, 127)	(64, 127)	7	14,400
+– Tanh, Dropout	(64, 127)	(64, 127)	–	–
+– Conv1d	(64, 127)	(64, 127)	11	45,120
+– BatchNorm1d	(64, 127)	(64, 127)	–	128
+– Tanh, Dropout	(64, 127)	(64, 127)	–	–
+– MaxPool1d	(64, 127)	(64, 63)	2	–
+– Conv1d	(64, 63)	(128, 63)	11	90,240
+– Tanh, Dropout	(128, 63)	(128, 63)	–	–
+– Conv1d	(128, 63)	(64, 63)	11	90,176
+– BatchNorm1d	(64, 63)	(64, 63)	–	128
+– Tanh, Dropout	(64, 63)	(64, 63)	–	–
+– MaxPool1d	(64, 63)	(64, 31)	2	–
+– Conv1d	(64, 31)	(1, 32)	32	2,049
+– Tanh	(1, 32)	(1, 32)	–	–
+– LSTM	(32, 1)	(32, 20)	–	1,840
+– Sequential	(20)	(60)	–	–
+– Linear	(20)	(60)	–	1,260
+– Tanh	(60)	(60)	–	–
+– Conv_LSTM_Decoder	(60)	(1024)	–	–
+– Sequential	(1, 60)	(10, 3840)	–	–
+– Conv1d	(1, 60)	(1, 60)	32	33
+– Tanh	(1, 60)	(1, 60)	–	–
+– Upsample	(1, 60)	(1, 120)	–	–
+– Conv1d	(1, 120)	(64, 120)	11	768
+– Tanh	(64, 120)	(64, 120)	–	–
+– Conv1d	(64, 120)	(128, 120)	11	90,240
+– Tanh	(128, 120)	(128, 120)	–	–
+– Upsample	(128, 120)	(128, 240)	–	–
+– Conv1d	(128, 240)	(64, 240)	11	90,176
+– Tanh	(64, 240)	(64, 240)	–	–
+– Conv1d	(64, 240)	(64, 240)	7	28,736
+– Tanh	(64, 240)	(64, 240)	–	–
+– Upsample	(64, 240)	(64, 480)	–	–
+– Conv1d	(64, 480)	(32, 480)	5	10,272
+– Tanh	(32, 480)	(32, 480)	–	–
+– Upsample	(32, 480)	(32, 960)	–	–
+– Conv1d	(32, 960)	(32, 960)	6	6,176
+– Tanh	(32, 960)	(32, 960)	–	–
+– Upsample	(32, 960)	(32, 1920)	–	–
+– Conv1d	(32, 1920)	(10, 1920)	7	2,250
+– Tanh	(10, 1920)	(10, 1920)	–	–
+– Upsample	(10, 1920)	(10, 3840)	–	–
+– Sequential	(38400)	–	–	–
+– Linear	(38400)	(1024)	–	39,322624
+– Sigmoid	(1024)	(1024)	–	–
Trainable params:		39,803,948		
Input size (MB):		0.13		
Forward/backward pass size (MB):		59.10		
Params size (MB):		159.22		
Estimated Total Size (MB):		218.45		

A.3. Datasets

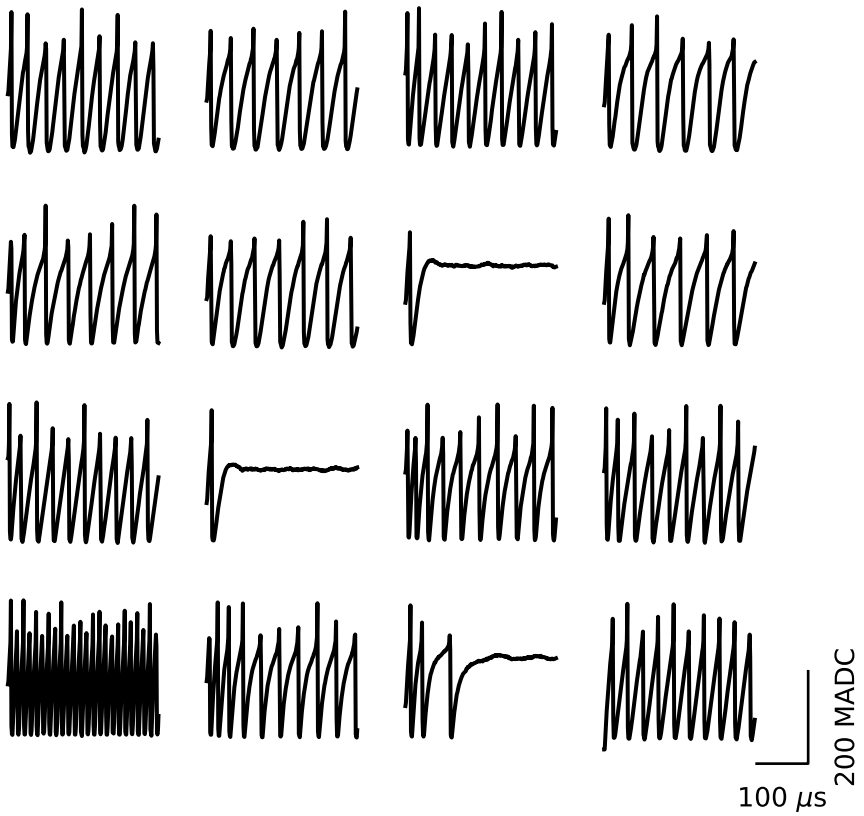


Figure 38: Voltage traces of the 2D dataset with parameter setting from table 1. The traces were interpolated down to 1024 datapoints and only the first 300 datapoints are displayed for visualization purposes.

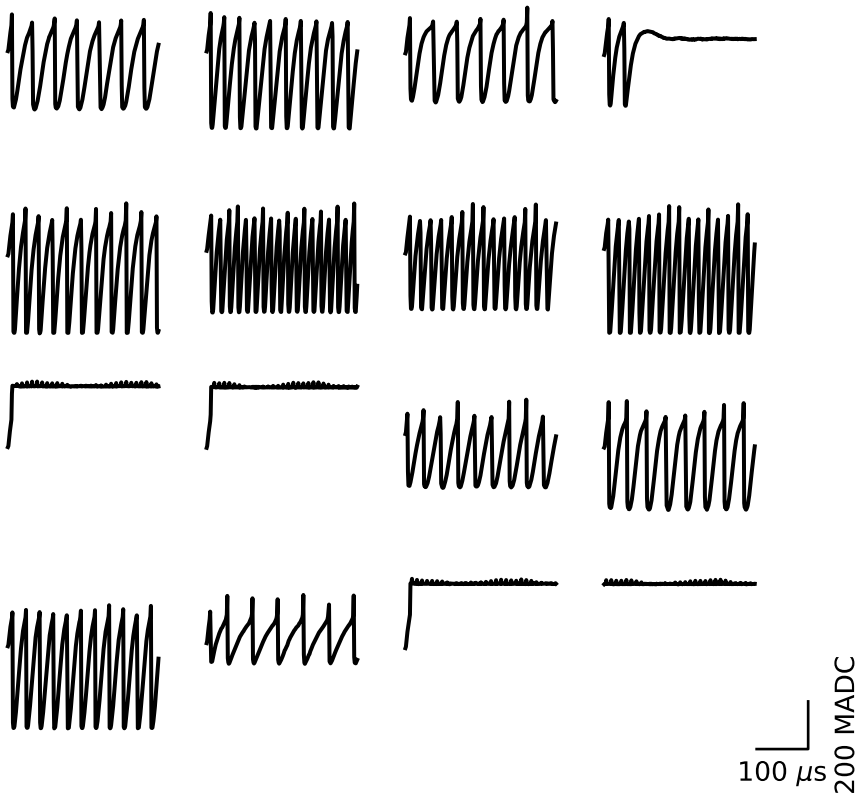


Figure 39: Voltage traces of the 8D dataset with parameter setting from table 1. The traces were interpolated down to 1024 datapoints and only the first 300 datapoints are displayed for visualization purposes.

A.4. Autoencoder training

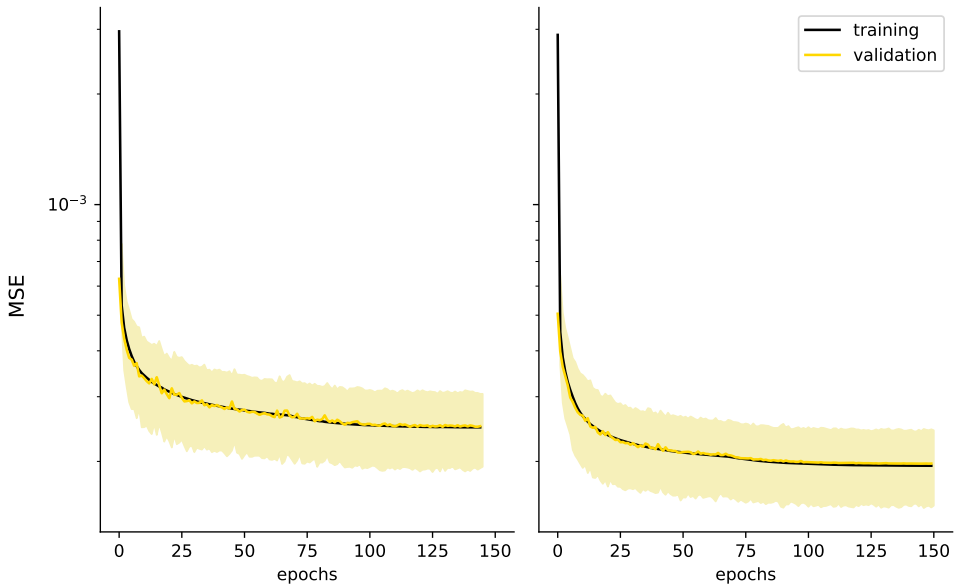


Figure 40: Comparison between the average training and validation losses of two autoencoder models. One standard deviation of the validation loss is shown, too. **Left:** convolutional autoencoder (CONV-AE) where weighted loss was applied during training; **Right:** CONV-AE without a weighted loss during training; the training for both models was conducted on the 4D emulation dataset with parameter settings of table 1.

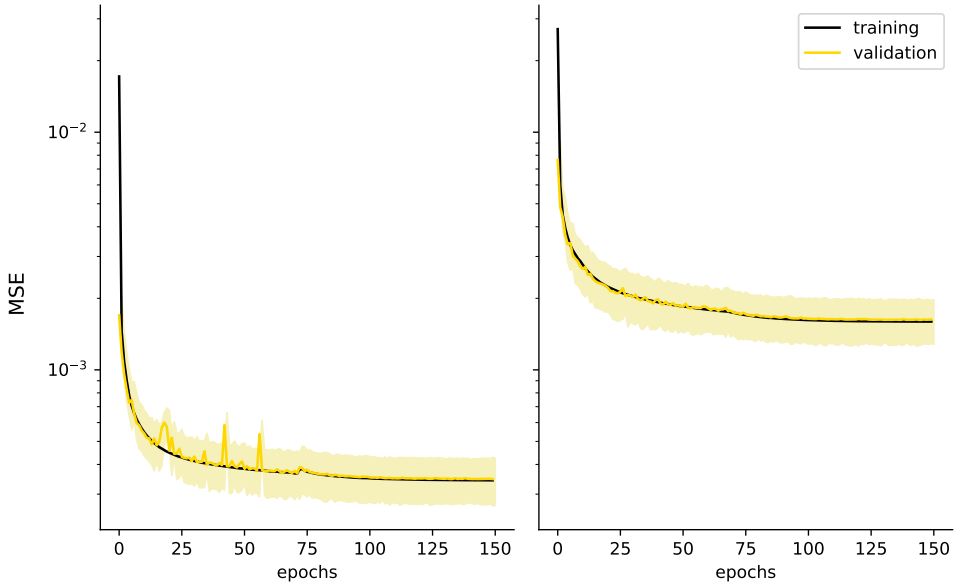


Figure 41: Comparison between the average training and validation losses of two autoencoder models with a latent space dimension of 32. One standard deviation of the validation loss is shown, too. **Left:** CONV-AE which was trained on the 2D dataset; **Right:** CONV-AE which was trained on the 8D dataset; the values of the parameters for those datasets can be found in table 1.

A.5. Simulation-based inference

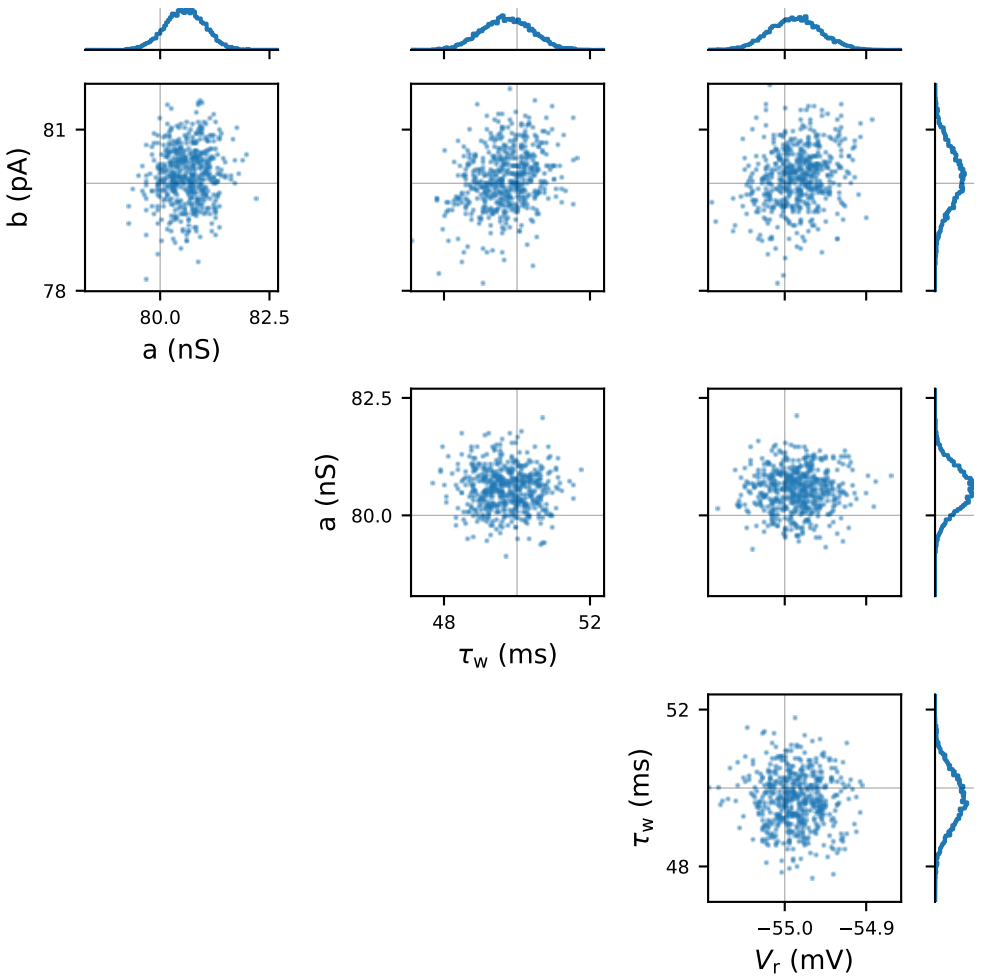


Figure 42: Pair plot of 10,000 drawn samples from the approximated posterior. The true parameters are located at the intersection of the grey lines and are set according to the values in table 4. Wavelet transforms using the "Haar-wavelet" were used as the data embedding technique. Coefficient calculation was performed up to decomposition level 4, and only the approximation coefficients were fed into the SNPE algorithm. For more information, see Huhle, 2024. 20 rounds of inference with 1000 simulations each were performed.

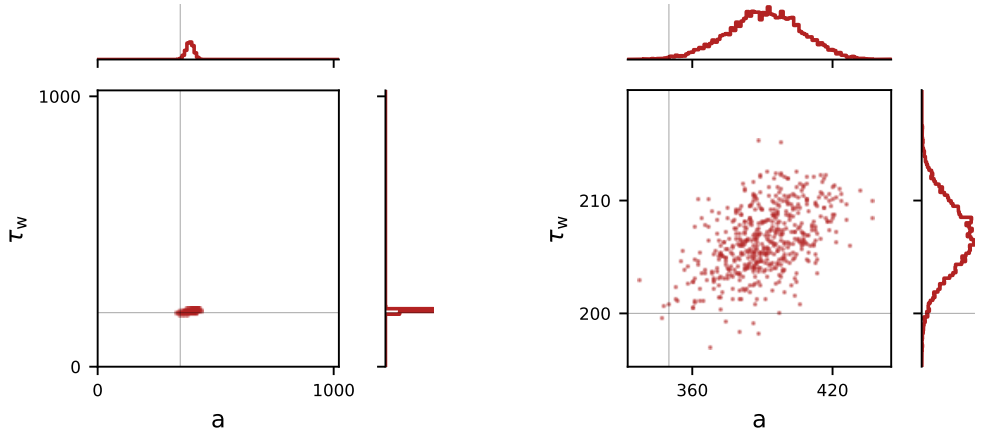


Figure 43: Samples drawn from the 2D posterior for the target observation $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE with additional concurrent retraining with the NDE was used as the embedding technique. 20 inference rounds with 1000 simulations each were performed. **Left:** view of the whole parameter range; **Right:** zoomed-in view of the plotted samples.

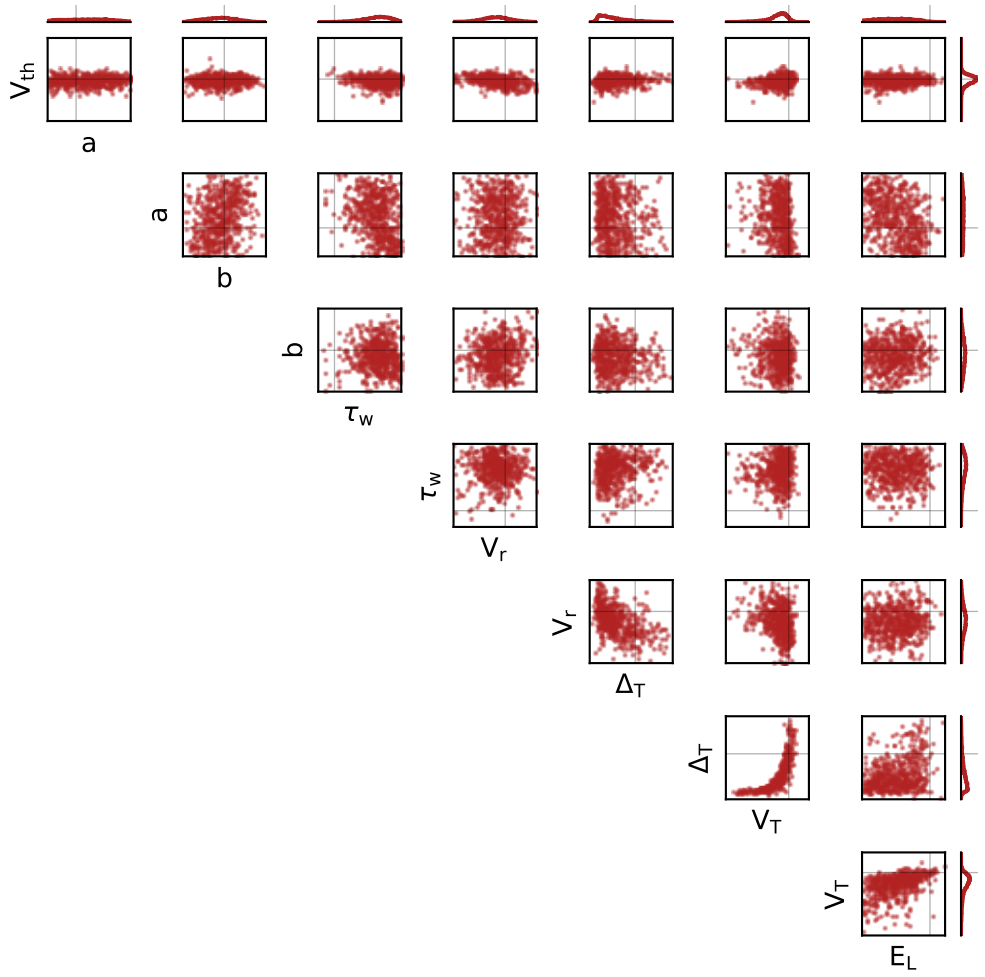


Figure 44: Samples drawn from the 4D posterior for the target observation $\hat{x}_{\text{slow}}$ with parameter settings in tables 1 and 5. The encoder of the CONV-AE with a latent space size of 64 and additional concurrent retraining with the NDE of the SNPE algorithm was used as the embedding technique. 20 inference rounds were performed. 4000 simulations were conducted in the first round while 2000 simulations were carried out in each subsequent round. The plot displays the whole possible parameter range from 0 to 1022 for each parameter.

A.6. Experiment environment and data

The state of the used software is displayed in table 11. Additional changes which are still under review can be found in table 10. The emulation experiments were performed in a software container provided by the research group; the used container is 2024-04-17_1.img. For the simulations, a newer version of the *Brian* simulator was needed. Therefore, a custom conda environment with *Brian* 2.7.0. was used. A file with all the packages in this environment can be found in /ley/users/jhuhle/BA/conda_env.txt. All experiment data can be found in /ley/users/jhuhle/BA.

Table 10: Changesets which are still under review.

Title	Changeset ID
feat: use brian2 as a simulator	22949
dataset	23097
Autoencoder	23216
SBI	23416
Plotting	23517

Table 11: Software state

Repository	git hash
model-hw-adex-sbi	29df7f256b15f892cad4a5346223d1b51883860c
pynn-brainscales	57370c7479f4a4c372ad944c22bec0b5ee30f61b
code-format	09f3a985a6f264359b10a6a129dd6dce7e55c9e8
haldls	237983b173c164d225a2f5398d7e72ef60de7397
grenade	6e2d453aaf305f297027ba0132bc70d13444dda7
calix	a706868c6ba285b1f8fd7cdef1a19d7328e02912
logger	73dadb3ce413c521845ef7d36f818073eee4fefa
halco	a97040a732ab1ba954e077616303a18acf623092
hate	35b3cb211cabbbc5c01036ae7878a73e338166c4
fisch	6120fc0ac0d90b3c66a212b3cc5cc25034bf584e
libnux	66b9c67bc114f82add677c6095f38843c23c4cd7
hxcomm	95abf25670bd8cb7cc5b499cde56f653130cf20c
rant	722edd57c9e42462a660db8a1febb0211ffad07c
ztl	b6745261d8bfdce44516d58d632c3c73834839d2
pywrap	5e2af30e9593882b471d3cd02df00b93f13ff479
lib-boost-patches	136c5b41cb046afe2c726aa4646928bf5190622e
sctrltp	1d854f953f7e8c8ead44406a22bb80421ca3857c
hwdb	f7262189b0e55b686896a3dea952065c2f1a3789
visions-slurm	8f41ea4f5bd1573d8f4623e9ed698a29f30036a3
flange	28e729d59df3b4ff380f84351c40d4da3086bed8
lib-rcf	000185eb11db4d54cb6b12b09af54cf742741036
bss-hw-params	b7be7827b51536804f0bda76f8ba4be693df23a8
paramopt	ddf6c49df5b44d5fd42e314433432015f8738409

Danksagung (Acknowledgement)

Zuallererst möchte ich natürlich Jakob Kaiser danken. Er hat mich Küken unter seine Fittiche genommen und mich mit viel Geduld herangezüchtet. Dabei hat er immer wieder Ordnung in verwirrte Gedanken gebracht und viele Dinge dann eben auch siebenmal erklärt, wenn ich sie schon wieder vergessen hatte. Selbst jetzt ist er mental noch recht stabil, obwohl ich immer noch nicht die drei git commands beherrsche, die ich seit sechs Monaten benutzen sollte. Seine Expertise, große Geduld und Bereitschaft, mir auch an Wochenenden zu Hilfe zu kommen, haben zu dieser Arbeit entscheidend beigetragen. Man hätte sich wohl keinen besseren Cheffe suchen können!

Außerdem möchte ich natürlich auch meinen Bürokameraden Carl, Simon und Arik für die tolle Büroatmosphäre danken. Besonders letztere hat unter meiner Tyrannei wahrscheinlich sehr gelitten, da ich es nie gemisst habe, ihn seiner Stifte und sonstiger Büroutensilien zu berauben.

Auch möchte ich Phillip Spiegler danken, welcher mir immer wieder mal Fragen beantwortete und auch eine vorherige Arbeit Korrektur gelesen hat. Zudem bin ich auch sehr dankbar, dass er mich an meinem letzten Tag hier beim Dart spielen gewinnen lassen hat.

Selbstverständlich gilt mein Dank auch allen anderen Electronics Vision(s) Mitgliedern für die tolle Zeit hier in dieser Arbeitsgruppe.

Zuletzt möchte ich mich noch bei der Putzfrau und dem Hausmeister Michael entschuldigen, die ich wohl sehr verstört haben muss, als ich sie immer wieder schlaftrunken und stöhnend bei ihrem Arbeitsbeginn frühmorgens begrüßte. Auch dem EINC gilt mein Dank, welches in den letzten Wochen wortwörtlich mein Zuhause geworden ist. Es hütete mich stets in tiefer Nacht, als ich mal wieder mit verwirrtem Geist und gebrochenem Willen durch dessen großen Hallen und langen Gängen umherwanderte. Die Akustik des Gebäudes beglückte mich sehr, da sie nachts den Sound des hardstyle Technos klanglich, sowie auch lautstärke-technisch, vorteilhaft unterstützte. Bei WG-gesucht bekommt das EINC dennoch nur vier von fünf Sternen, da ich ab und an durchaus schon einen Herd vermisst habe.

The work carried out in this bachelor Thesis used systems, which received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements Nos. 720270, 785907 and 945539 (Human Brain Project, HBP) and Horizon Europe grant agreement No. 101147319 (EBRAINS 2.0)