Master’s Thesis 2023   30 ECTS
Faculty of Science and Technology
Kristin Tøndel

Estimating posterior distributions of synaptic strengths in a neural network using Approximate Bayesian Computation and Sequential Neural Posterior Estimation

Håkon Strand
Data Science

Abstract

In computational neuroscience, it is common to represent natural phenomena through mathematical models in the form of coupled differential equations. These models allow us to study neural networks in the brain by simulating and measuring neural activity, specifically spiking activity. This thesis examines a two-population neural network, consisting of one inhibitory and one excitatory population, to estimate the posterior distributions of four synaptic strength parameters. By varying these parameters in our simulations, we can analyze differences in spiking activity.

From the spiking data, represented by histograms, we computed four summary statistics per population (eight in total): mean firing rate, Fano factor, mean interspike interval and coefficient of variation. We then construct three likelihood-free methods in order to create posterior distributions of the synaptic strength parameters based on either the summary statistics or the raw output data. The posteriors are constructed using Approximate Bayesian Computation (ABC) and Sequential Neural Posterior Estimation (SNPE) methods. One rejection-based ABC method, with included linear regression adjustment, is constructed, as well as two SNPE methods: one using the summary statistics and the other using an embedding network (consisting of a convolutional neural network) to extract it's own metrics based on the raw output data.

Furthermore, we aim to evaluate the findings to best replicate the values of an observation. The observation is randomly selected from the simulated data, and then simulated an additional 100 times with fixed parameter values to account for the stochastic properties of the network model. The mean of these additional simulations served as the observed summary statistics values, and the sample closest to this mean was selected as the raw output observation.

To evaluate the posteriors, we examine the extent to which the methods are capable of restricting the synaptic strength parameters, comparing this with the observation's output in terms of both summary statistics and raw output data. The SNPE method using an embedding network is the least capable of restricting the parameter domain and replicating the observation results. The linear regression adjusted ABC method shows some improvement over this, while the SNPE method without an embedding network appears to be the most successful in both restricting the parameter domain and replicating the observation output.

Acknowledgements

This thesis concludes the final work of my two-year Master's degree in Data Science at the Faculty of Science and Technology, NMBU.

First of all, I would like to thank my supervisor Kristin Tøndel for continuous support through meetings, feedback and discussions throughout the semester. A special thank you to Jan-Eirik Welle Skaar and Nicolai Haug for both technical and theoretical assistance and discussions. A thank you to Alexander Stasik for constructive suggestions and input. To Gaute T. Einevoll for both valuable input and introducing me to the field of neuroscience.

A big thank you to the students at "Studentenes hus" TF6-206. Your technical and theoretical contributions, proofreading, and even your not-so-productive talks have made this experience much more enjoyable.

Lastly, thank you to family members for your support and proofreading efforts.

Abbreviations

ABC Approximate Bayesian Computation

ANN Artificial Neural Network

AP Action Potential

CNN Convolutional Neural Network

CV Coefficient of Variation

ISI InsterSpike Interval

LFI Likelihood-Free Inference

LIF Leaky Integrate-and-Fire (neurons)

MAF Masked Autoregressive Flow

PSD Power Spectral Density

PDF Probability Density Function

SNPE Sequential Neural Posterior Estimation

SR Synchronous Regular

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The human brain is a complex and dynamic system that, despite decades of research, still holds many unanswered questions. It is comprised of numerous specialized regions responsible for controlling various aspects of human behaviour. Central to the brain's function are neurons, which form intricate networks in order to communicate. Studying these neurons, and ultimately the networks they form, is a critical and fascinating area of research.

1.1 Background

The exploration of the brain's complex mechanisms often requires the use of models to represent its biological characteristics. In this context, a model refers to mathematical descriptions of the biophysical properties of, for example, neurons or interconnected networks of neurons. In the field of computational neuroscience, this implies the ability to represent biological processes within computational models, essentially serving as virtual laboratories. These virtual laboratories enable scientists to simulate behaviour under diverse conditions in more controlled environments, thus proving to be an invaluable asset. However, the construction of these models can be difficult as they need to accurately represent the natural behaviour they intend to emulate.

Over the last few decades, a diversity of different models has been developed, ranging from the highly simplified abstractions to the more complex mechanistic models trying to emulate the precise behaviour of neurons. At the lower levels, when examining individual neurons, it is common to use more detailed mathematical representations. This ensures that all relevant information is considered, but also involves less relevant information. While this is feasible for single neurons, it becomes increasingly challenging when combining neurons into larger networks. To manage the escalating complexity of interacting neurons, researchers often simplify individual neuron models in order to focus on their most essential features. This creates a trade-off between simulation power and complexity, as more simulations can be performed on simpler mathematical models, but the output must still retain relevant information.

There are various ways of representing the input parameters and output data for these simulations, whether they represent single neurons or interconnected networks. The input parameters ultimately shape the behaviour and interactions in the system, and are, in the context of this thesis, broadly categorized as either fixed or varying. The fixed parameters are typically related to the core attributes of the neuron model and the network architecture. On the other hand, varying parameters are dynamically changed in order to alter the system's behaviour. The varying parameters considered in this thesis are the synaptic strengths between the neurons, essentially a parameter responsible for how much influence neuron A has on neuron B.

The output is then produced by inserting both the fixed and varying parameters to sets of coupled differential equations. In our case, the output is measured by spiking activity, one of several metrics to quantify the level of activity within a network. In order to use the computational models to understand how the brain works, we have to fit our models to experimental data. This means that we have to find the parameter values that allow our model to produce output data that resembles the experimentally measured data. To achieve this, we need to analyse the relationships between the input parameters and the model output, e.g., which input parameters have the largest effects on which model outputs (i.e., which outputs should we measure in order to find the most suitable values for which parameters).

A powerful approach to understand the relationships between input parameters and output of a neural network is Bayesian inference. This method enables researchers to update their beliefs about a system's parameters by adding more data. Bayesian inference is a framework which combines prior knowledge with simulated output in order to derive posterior distributions of parameters of interest. These posteriors are essentially probability distributions of parameters given a certain model output. For instance, given a certain set of weather data, the probability distribution of temperature would then yield values probable during such conditions. Similarly could output from a neural network, when combined with posterior distributions of input parameters, provide insights into which values are probable given the output.

The trained posterior could subsequently be used to predict new spiking activity. Under normal circumstances, we would compare these predictions with experimental data. If the predicted output closely matches the experimental data, it would serve as a validation of our predictive model. Furthermore, if the posterior distribution accurately captures the experimental data, it can be used to perform inverse modelling. This would reverse the process: instead of predicting the spiking activity based on given parameter values, we would use observed spiking activity to infer the likely parameter values, as shown in Figure 1.1. Such findings could provide valuable insights into the interactions of parameter values responsible for specific patterns of neural activity.

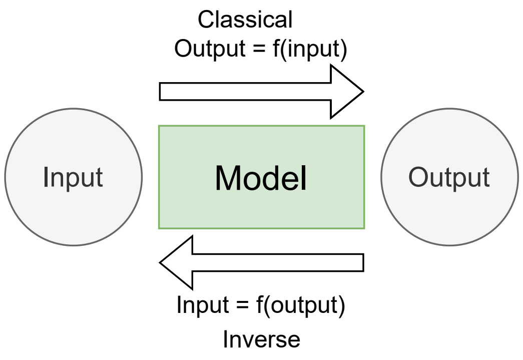

Figure 1.1: Illustration of classical modelling versus inverse modelling, specifically in the context of making predictions using statistical models (e.g., machine learning models). In classical modelling, the output is a function of the input, while in inverse modelling, the input is a function of the output.

A challenge when simulating neural networks is the incorporation of stochastic variables. These representations are commonly referred to as probabilistic, resulting in different outputs for the same parameter inputs. These variations are intended, as probabilistic models by definition are based on probabilities. However, the opposite could also happen, i.e., that different parameter input produces the same output, which could be a manifestation of model sloppiness. Model sloppiness refers to the phenomenon where certain mathematical models are insensitive to variations in parameter inputs, in turn producing the same output despite variations in input parameters. Addressing such inconsistencies adds complexity to the already difficult task of mapping parameter spaces.

In recent years, to address the problem of mapping posteriors, a variety of statistical methods have emerged, ranging from the established Approximate Bayesian Inference (ABC) to more recently introduced machine learning techniques like Sequential Neural Posterior Estimation

(SNPE). These methods are so-called likelihood-free inference (LFI) methods, meaning that the later discussed likelihood function (Chapter 2.4) of a Bayesian inference approach does not need to be present.

1.2 Objective

The primary objective of this thesis is to build posteriors of synaptic strength parameters used in the simulation of a two-population neural network by applying the aforementioned methods. This is done by extracting a single observation, as opposed to the true posterior itself, and then attempting to replicate the results of the parameters from the observation. The implementation is done using the Python programming language, an extensive list of the packages and their purpose can be seen in the appendix Table A.1.

More concretely, the following question is asked: *to what extent can implementations of the likelihood-free methods ABC and SNPE accurately estimate the posterior distribution of synaptic strength parameters of a two-population neural network?* Further, the thesis seeks to evaluate the trade-offs between computational efficiency and accuracy of the resulting posterior parameter estimations, ultimately providing insight into the most efficient strategies.

Chapter 2

Theory

In this chapter, we will delve into the theoretical foundations from which the datasets are created, as well as the methods employed to analyze them. We will begin by examining the biological neurons and their interconnected networks, followed by an examination of the various analytical techniques and approaches used for the analysis. This chapter, together with Chapter 3, aims to give the context needed for the reader to understand the approaches and results later discussed in Chapter 4 and Chapter 5.

2.1 Neurons

There is a large number of neurons in the human brain, roughly estimated to be 86 billion, according to a 2009 study by Azevedo et al. [1]. They consist of mainly three parts, namely the *soma*, *axon* and *dendrites*, as seen in Figure 2.1. The soma, or cell body, houses the nucleus where the neuron's DNA is stored and where proteins related to neurotransmission are made. The axon is responsible for propagating electrical signals in order to communicate with other neurons, which receive these signals through the dendrites. The cell responsible for sending the signal is referred to as the *presynaptic* cell, while the neuron receiving the signal is the *postsynaptic* cell. When a signal is sent through the axon, the presynaptic cell releases neurotransmitters into the gap junctions between itself and the postsynaptic cell. These neurotransmitters bind to the receptors on the postsynaptic cell and determine whether the postsynaptic cell will send signals to other neurons it is connected to or not.

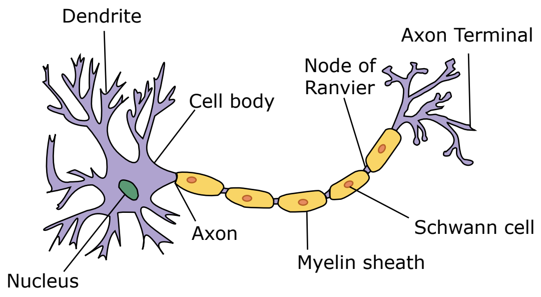

Figure 2.1: Neuron model illustration. Electrical signals are sent through the axon, to the axon terminal. Connected neurons receive the information in the form of released neurotransmitters binding to their dendrites, which in turn is processed throughout the cell body. *The image is used under the Creative Commons license* [2]

2.1.1 Action potential

The electrical signals described above are called *action potentials* (AP) or *spikes*. Spikes happen when the *membrane potential* of a neuron reaches a certain threshold. The membrane potential refers to the difference in electrical charge between the outside and inside of the cell membrane. The outside has a reference value of 0 and the difference between that and the charge inside the neuron determines its potential. When the neuron is inactive it will have a somewhat constant membrane potential called a *resting membrane potential.* As a stimulus occurs, the membrane potential will shift and once the threshold is passed, the neuron spikes. This is a so-called all or nothing phenomenon, meaning that the spike either happens or not. It does not matter whether the threshold is barely passed or surpassed by a large margin, the effect will essentially be the same.

This is because gated ion channels open or close depending on which state the membrane potential is in. Ions are atoms or molecules with either a net positive or negative charge, meaning they either have lost or gained electrons. The most prominent ions responsible for spikes in a neuron are sodium (Na^+), potassium (K^+) and chloride (Cl^-). Once the threshold is passed, voltage-gated ion channels in the membrane activate causing an influx of sodium. This in turn depolarizes the cell (increasing electric potential) further causing the membrane potential to be more positively charged compared with the outside. To repolarize the cell (decrease electric potential), potassium, and sometimes chloride, ion channels open making positively charged potassium ions leave the cell while negatively charged chloride ions enter. The repolarization process hyperpolarizes the neuron making the membrane potential go below the resting potential. The effect of this is a lower chance of spiking multiple times in a row as the membrane potential has a brief period of time below its resting potential [3]. An example curve of how an action potential could look like is shown in Figure 2.2.

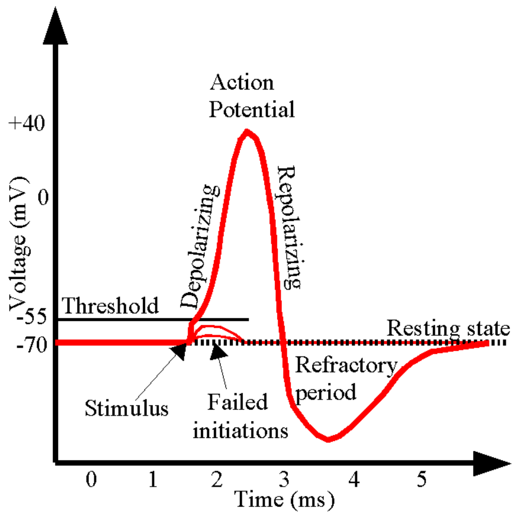

Figure 2.2: Propagation of a neuron action potential. Extracellular stimuli cause depolarization and rapid influx of Na^+ ions into the cell. At its peak, K^+ ions leave the cell while Cl^- ions enter. The hyperpolarization phase, which undershoots the resting potential, creates a refractory period during which the neuron is less likely to spike. *The image is used under the Creative Commons license* [4]

2.2 Cellular models

Scientists have discovered that mathematical models can be used to describe these above described phenomena. Typically, these models are created using coupled differential equations and is commonly known as cellular models. A cellular model describes various characteristics of a cell, enabling computational simulations.

The breakthrough in mathematical modelling of cellular processes was achieved by A. L. Hodgkin and A. F. Huxley in 1952 [5]. By measuring the activity of an axon found in a giant squid, they were able to mathematically model how an action potential propagates within the axon. This was accomplished by translating the axon into an electrical equivalent circuit, focusing primarily on three properties: *capacitance*, *conductance* and *leaky currents*. The capacitance describes the amount of electrical charge stored in the membrane, while the conductance and leaky currents describe ion channel permeability and ion leakage in the membrane, respectively.

Since then, a wide variety of cellular models have emerged, differing in complexity and thereby use case. While more complex cellular models, such as multi-compartmental models, provide more accurate representations of a cell's morphology, they are also more computationally expensive to simulate. Multi-compartmental models attempt to emulate the cell's morphology by dividing the neuron model into compartments. For example, imagine dividing the neuron model in Figure 2.1 into 10 equal compartments and then converting each compartment into an electrical equivalent circuit, as the one illustrated in Figure 2.3. This allows for a more realistic propagation of the electric potential through the cell. However, the individual simulation costs of such models are high, resulting in fewer simulations performed within the same time frame. Alternative approaches to neuron simulations, such as compressing the cell's morphology into a single point (point neurons), could therefore be beneficial.

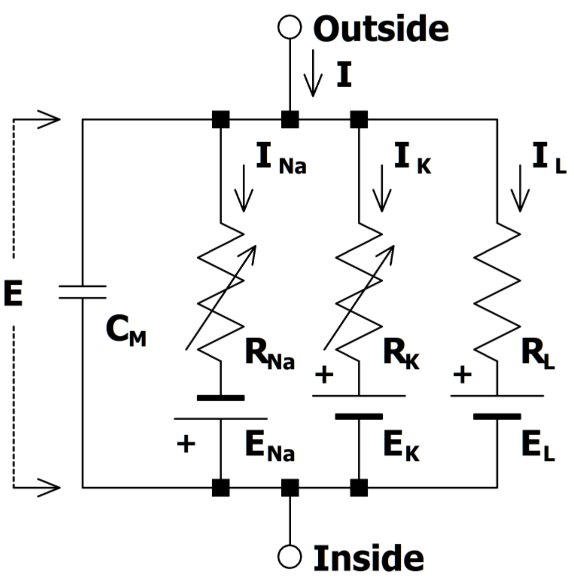

Figure 2.3: Illustration of the Hodgkin-Huxley RC circuit model. Represented by capacitance (C_M), input currents ($I_{Na,\ K}$), resistance ($R_{Na,\ K}$) (reciprocal of conductance, typically denoted g) and reversal potential ($E_{Na,\ K}$) (equilibrium potential of ion). Included is also the leak channels (i.e., I_L, R_L, E_L). *The image is used under the Creative Commons license* [6]

In addition, scientists would need a way to induce spikes in neurons. In single-cell experiments, this could be done by introducing an artificial current into the neuron. When neurons are connected to other neurons, however, the synapse must be mathematically modelled. A straightforward approach is to treat each spike the same way, using a constant value transferred to the postsynaptic neuron given that the presynaptic neuron spikes. More complex methods typically involve stochastic modelling. In biological neurons there are vesicles at the end of the axon containing various neurotransmitters. These vesicles have a chance to open and release its content into the synapse, directly depending on the present concentration of calcium (Ca^{2+}) [7]. The released neurotransmitters that bind to the postsynaptic cell will either cause depolarization or hyperpolarization, determining whether it spikes.

2.2.1 Leaky Integrate-and-Fire neurons

While the the Hodgkin-Huxley model make use of a detailed resistor-capacitor (RC) circuit to simulate the biological neuron, this thesis utilizes a simpler model, namely the *leaky integrate-and-fire* (LIF) point neuron. As previously described, the generation of spikes in neurons involve intricate interactions of ion channels, processes which are not incorporated in the LIF model.

In LIF neuron models the spike generation process is done by integrating the input from connected neurons over time, while simultaneously taking into account a leaky component, hence the term "leaky". When the predefined threshold is surpassed, a spike is said to occur. It is then important to note that the LIF model does not simulate the spike's propagation or the underlying factors leading to its generation, but merely notes the occurrence of the spike, before instantiating a relatively short refractory period (i.e., where the neuron is incapable of spiking).

Mathematically, the subthreshold dynamics (i.e., before the occurrence of a spike) of a LIF model can be expressed by the following differential equation:

$$\tau_m \frac{dV}{dt} = -V(t) + R_m I(t) \tag{2.1}$$

In this equation, τ_m represents the membrane time constant, V the time-dependent membrane potential, R_m the membrane resistance and I the time-dependent input current. This creates a model that balances a passive "leak" of the membrane potential towards its resting sate, and input currents that drive the membrane potential away from said state, towards the threshold.

2.3 Models of biological neural networks

When large numbers of individual neurons form synaptic connections they are referred to as neural networks. As stated in Section 2.2, some neuron models are computationally costly to simulate, and this only increases as more neurons are introduced into a network of neuron models. Typically, simulating these networks often involve numerical integration of thousands of coupled differential equations [8]. Therefore, the number of neurons in a network and the mathematical representation of their characteristics, are of importance for the scientist.

Biological neurons form synaptic connections through a process of axonal and dendritic growth [9]. Both axons and dendrites are extensions of the cell's body and form connections from the axon of one neuron to the dendrite of another. As this process is hard to emulate, the approach of the model used in this thesis is to represent the synaptic connections through probabilities.

When modelling networks of neurons it is common to divide the neurons into groups based on their characteristics, referred to as populations. In the context of this thesis, these characteristics are *excitatory* and *inhibitory*. Excitatory neuron populations are groups of neurons that have a positive impact on neurons they are connected to. In essence, a presynaptic excitatory neuron increases the likelihood of the postsynaptic neuron to generate a spike, while a presynaptic inhibitory neuron decreases the likelihood.

There are several neural network models that is described by using such population dynamics, two of which will be introduced in the following sections.

2.3.1 The Brunel network

The Brunel network is a spiking neural network model proposed by Nicolas Brunel in the year 2000 [10]. It is a well-known network in computational neuroscience due to its simplicity and capability of accurately reproducing neural behaviour. The network consists of N LIF neurons, with a distribution of 80% excitatory and 20% inhibitory neurons.

It is efficient to simulate due to the simplicity of the LIF neuron model, as well as the sparse connectivity of the neurons. The network is set up in such a way that each neuron has C connections, with $C << N$, i.e., the average synaptic connectivity between neurons is relatively low.

Consider an example where neuron A is connected to C other neurons from both excitatory and inhibitory populations. When these connected neurons spike, their respective synaptic output are processed by neuron A. If a connected neuron is excitatory, it causes an increase in membrane potential of neuron A. Conversely, if a connected neuron is inhibitory, the membrane potential of neuron A decreases. In addition to this, all neurons receive inputs from external synapses that are activated independently according to a Poisson process with a rate of v_{ext} [10]. The final output of neuron A is then decided by the total input, from both the local and external sources.

2.3.2 The Potjans-Diesmann network

Introduced in 2014, the Potjans-Diesmann network is another popular network model used as basis for analysis in computational neuroscience [11]. Rather than using two populations of excitatory and inhibitory neurons, like the Brunel network, this network incorporates a total of eight populations distributed across various layers of the neocortex. These layers include a combined L2/3 layer, as well as distinct L4, L5 and L6 layers, each with their own associated population of inhibitory and excitatory neurons.

The proposed network consists of $N = 80\,000$ neurons, interconnected through roughly 0.3 billion synaptic connections, corresponding to 217 million excitatory and 82 million inhibitory [11]. Despite the complexity introduced with additional neuron populations, the Potjans-Diesmann network share similarities with the previously mentioned Brunel network. Notably, it uses the LIF neuron model and connects populations to external synaptic sources, which generate spike trains following the same Poisson process as described earlier. The excitatory and inhibitory synaptic strengths in this network are designed as positive for the excitatory and negative for inhibitory populations. For the inhibitory population, an additional factor, g, is multiplied with the reference value to adjust the synapse strength. The g parameter resembles the one used in this thesis, although our g parameter relates to both populations, and have varying strength metrics based on which population it connects to (in total four since it can connect to the same population).

In the remaining sections of this chapter, we shift our focus to the various methodologies that can be used for estimating the posterior distribution of parameters such as g.

2.4 Bayesian inference

As briefly mentioned in the introduction, Bayesian inference is a framework for updating beliefs about a system by adding additional information. The Bayesian approach to posterior estimation is based on Bayes' theorem, which is shown in Eq. 2.2. The theorem describes the conditional probability of event A given event B ($p(A|B)$) as equal to the conditional probability of event B given event A ($p(B|A)$) times the probability of event A ($p(A)$), divided by the probability of event B ($p(B)$). For example, events A and B could represent specific weather conditions such as rain and cloudiness, respectively. Using Bayes' theorem we can then derive the probability of rain given cloudy weather, assuming the probability functions are known.

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{2.2}$$

However, in the real world, we are often interested in variables that can be described by probability distributions. The Bayesian approach to posterior estimation takes this into account by using probability distributions instead of point probabilities. To make the example more relevant for this thesis, Eq. 2.3 introduces θ, synaptic strength parameters, and y, the data produced by simulating a neural network with these parameter values.

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \tag{2.3}$$

Here, $p(\theta)$ is the prior distribution of θ, $p(y|\theta)$ is the likelihood of observing the data y given the parameters θ, and $p(\theta|y)$ is the posterior distribution of the parameters given the observed data y. The marginal probability $p(y)$ is the integral of the likelihood function $p(y|\theta)$ over all possible values of θ, i.e., $p(y) = \int p(y|\theta)p(\theta)d\theta$ for continuous values of θ or $\sum_\theta p(y|\theta)p(\theta)$ for discrete values of θ [12]. This is used to normalize the posterior, ensuring that it integrates to 1, a requirement for a probability distribution. A commonly used informal representation of this formula, excluding the normalizing parameter, is the following:

$$Posterior \propto Likelihood * Prior \tag{2.4}$$

What is needed to calculate the posterior is then broken down to a likelihood function and a prior distribution. The prior can either be informative, meaning that the prior is based on domain knowledge about the topic at hand and the distribution is biased towards more probable values, or uninformative, meaning that the prior is a uniform distribution over a given parameter space. If we then know the likelihood function, we can calculate the posteriors. This function, however, is often unknown or intractable, such as parameter posteriors for biological neural networks, and one would need to approximate it. In the next sections, we will look into some methods one could use in order to approximate the posterior when the likelihood function is unknown or intractable.

2.4.1 Summary statistics

Before diving deeper into the methods, a crucial part of estimating posteriors in this thesis rely on *summary statistics*. In many instances, the output data can be complex and high-dimensional, making the comparison between observed and simulated data difficult. The primary objective of using summary statistics is then to capture the essential features of the output while simultaneously reducing its dimensionality.

For instance, when simulating behaviour of a biological neural network, the output data could be high-dimensional time series of spike counts. Comparing individual spike counts between samples could be difficult. Instead, using summary statistics that capture essential information, such as *firing rates* and *interspike intervals* (time between spikes), can yield more stable and meaningful comparisons.

There are essentially endless ways of computing these summary statistics, and choosing the correct number, and quality thereof, can be challenging. One of the approaches in this thesis uses the later introduced convolutional neural network (Section 2.4.3) as an embedding network to extract summary statistics, as an attempt to deal with this problem.

2.4.2 Approximate Bayesian Computation

Approximate Bayesian Computation (ABC) is known as a *likelihood-free inference* (LFI) method, meaning that it is used in the aforementioned case of difficult or intractable likelihood functions. Within the framework there are several approaches, one of which uses a rejection based algorithm. By drawing parameter values (θ) from a prior distribution and using a model $\mathcal{M}$, we get our simulated data ($p(y|\theta)$). The summary statistics calculated from these simulations are compared with those of the observed data. Samples, with their corresponding θ values, are subsequently rejected based on $\|y - y_{obs}\| < h$ [13]. Here h is some specified tolerance.

Table 2.1: The rejection algorithm of ABC.

Step	Description
1	Draw θ values from a prior distribution.
2	Generate y based on θ.
3	Calculate the distance between simulated and observed data: $\|y - y_{obs}\|$.
4	Reject θ values given $\|y - y_{obs}\| < h$ for a specified value of h.

Linear regression adjustment

In general, machine learning is the process of complex algorithms learning patterns of a given dataset. There are mainly three subcategories within the field, namely *supervised learning*, *unsupervised learning* and *reinforcement learning* [14]. Within the scope of this thesis, supervised and unsupervised learning are the most relevant. In supervised learning, we provide both a dataset of features and the true outcomes for the individual feature combinations to the machine learning model. The model then attempts to learn a relationship between input and output to predict future outcomes from new input. In unsupervised learning, we do not have a dataset containing the true outcomes. Instead, we rely on the model to extract underlying patterns from the data and group them accordingly.

Linear regression is a statistical method used in machine learning, where independent variables are utilized in order to map a linear relationship with one or more dependent variables. Mathematically, this can be written as $Ax + b = y$ where A represents a matrix of parameter values, x the corresponding coefficients, b the intercept, and y the outcome. The goal is then to estimate the values of the coefficients in order to best fit the observed y given the values of A.

This method can be incorporated as an additional step to the ABC algorithm, occurring after the rejection process. In this case, θ becomes the dependent variable, and y the independent variable trying to predict θ. Later, in Chapter 4, a more detailed explanation of how this further impacts our output will be discussed in relation to the dataset used in the thesis.

2.4.3 Sequential Neural Posterior Estimation

Another LFI method is Sequential Neural Posterior Estimation (SNPE). The method relies on training a deep neural density estimator to estimate the posterior distribution of the varying parameters [15]. This is either done by explicitly passing a simulation model, $\mathcal{M}$, as an argument or appending pre-simulated data, of which the latter is done in this work.

Artificial neural networks

The process of associating simulated data with a posterior parameter distribution in SNPE is done through an artificial neural network (ANN). ANNs are loosely based on the functionality of biological neurons, where input and weights from one layer of artificial neurons determine the output of the next layer of artificial neurons. In Figure 2.4 we can see that input parameters are connected to the individual nodes of the hidden layer. The activation of the node in the hidden layer is dependent of the dot product of the input feature values and their corresponding weights, $z = \sum_j w_j x_j$ (excluding bias, which would add a term b to the equation). This value is then passed to an activation function, with typical choices being Rectified Linear Units (ReLu), Eq. 2.5, or the sigmoid function, Eq. 2.6. The latter is a less modern choice as activation functions in the hidden layers but a popular choice in the output layer of binary problems, as it compresses the output between 0 and 1, mimicking confidence of output value in terms of percentage. ReLu, on the other hand, better addresses the vanishing gradient problem, which is related to larger neural networks where small gradients are constantly multiplied as one moves backward in the network, rendering artificial neurons closer to the input layer incapable of updating weights. Further, the same process is applied between the hidden layer and the output layer, or between hidden layers if there are more.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

This procedure is known as the feedforward process of an ANN, but assuming we are using the popular *backpropagation* algorithm, the learning primarily happens during the backward iteration process. Here the derivative of a user-specified cost function, $C(\hat{y}, y)$, is utilized, where $\hat{y}$ and y is the predicted and true output of a given node in the output layer, respectively. Specifically, the algorithm computes the gradient of the cost function with respect to the weights, applying the chain rule as it propagates through the network. The weights corresponding to connections from nodes between layers are then updated appropriately. Finally, if the specified number of epochs (i.e., training iterations) are completed, the output layer produces the final prediction.

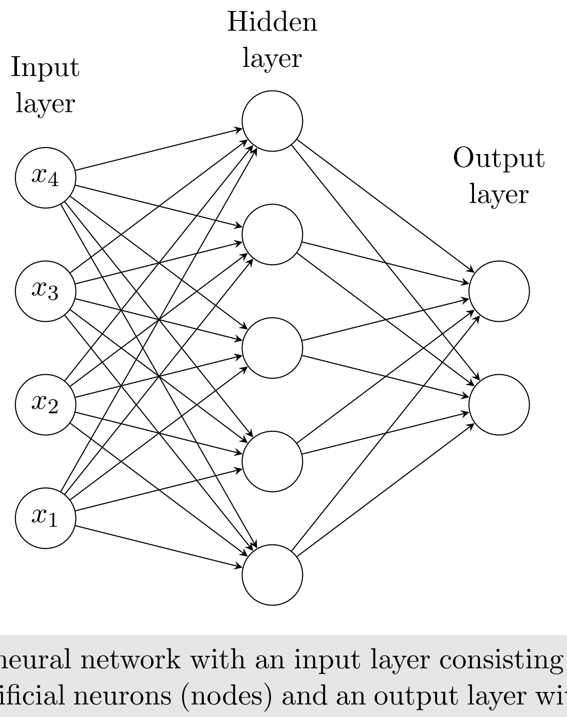

Figure 2.4: An artificial neural network with an input layer consisting of four input features, a hidden layer with five artificial neurons (nodes) and an output layer with two artificial neurons. Input data passes through the layers, where each node employs an activation function to generate an output based on the dot product of the values from the previous nodes and their corresponding weights.

Convolutional neural network

Another method that falls under the ANN umbrella term is the *convolutional neural network* (CNN). This type of network architecture has become particularly popular in dealing with pixel-related processes, such as image recognition [16], due to its ability to intelligently reduce the number of parameters that need to be trained. Traditional ANN models, as the one previously mentioned, use fully connected layers where every neuron between the layers share connections. An increase in the number of neurons, layers and feature parameters leads to a substantial growth in the number of trainable parameters in the network. This is especially true for images, where individual pixels are considered as features. CNNs can also be used as embedding networks, learning relevant information while simultaneously reducing the dimensional space.

There are various ways of building CNNs, but it generally involves the use of convolutional layers, max pooling layers and fully connected layers. Convolutional layers consist of filters, or kernels, that move across all the input data, creating what is called a feature map as output. The number of filters determines how many feature maps to use, which in turn is passed as the channel argument in the subsequent layer.

Max pooling layers also move filters across the input, extracting the maximum value within the filter iteration and mapping it to the output layer. This does not change the channel output in terms of number of feature maps, but is a common technique for reducing the size of each feature map, achieved in conjunction with stride (i.e., the steps taken by the filter between calculations).

Fully connected layers often serve as the endpoint for the network. By converting the channels and feature space into a single vector, the fully connected layers can transform the information into a final output. For instance, when training a model to distinguish images of cats and dogs, the output could be an array of two numbers asserting the model's confidence in each animal respectively. In other words, the spatial component is lost during transformations, and the model is no longer able to determine *where* the animal is on the original image. To address this issue, other CNN techniques, such as the U-Net [17], have been developed, demonstrating that the abovementioned method is just one of many potential implementations.

Masked Autoregressive Flow

In 2018, scientists of the University of Edinburgh published a paper discussing a new method for density estimation, namely the Masked Autoregressive Flow (MAF) [18]. The method is a type of normalizing flow designed to transform simple distributions, e.g., a multivariate Gaussian ($\mathcal{N}$), into more complex ones.

This is achieved by stringing together a sequence of invertible transformation functions, as is the typical approach of normalizing flows, learned by a neural network. Moreover, it incorporates autoregressive properties by only allowing certain dependencies between variables, i.e., $p(x) = \prod_i p(x_i|x_{1:i-1})$.

An important thing to consider when transforming one distribution to another, is the change in density. By calculating the Jacobian determinant and incorporating it in the equation, this change is appropriately accounted for. The masking of the MAF method ensures that the Jacobian matrix of the transformation function is triangular, in turn making the determinant trivial to compute as the product of the diagonal entries. Formally, mapping from the simple distribution $p(u)$ to the learned complex distribution $p(x)$ is written as

$$p(x) = p_u(f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}}{\partial x} \right) \right| \tag{2.7}$$

and the other way around, from $p(x)$ to $p(u)$

$$p(u) = p_x(f(u)) \left| \det \left(\frac{\partial f}{\partial u} \right) \right| \tag{2.8}$$

In the above equations, f represents the learned transformation function parameterized by the neural network, and f^{-1} its invertible counterpart. After the training is complete, which involves maximizing the log probability, we can sample values from the simple Gaussian distribution and transform them into the domain of the complex distribution.

Chapter 3

Data

This chapter aims to provide the reader with an understanding of the dataset used in the thesis, as well as to offer insights into the transformations and modifications done in order to enable the analytical methods proposed in the next chapter.

3.1 Introduction to the dataset

The dataset consists of a two-population biological neural network, featuring one excitatory and one inhibitory population, simulated over 2 500 ms. The simulations are conducted using the NEST simulator version 3.4 [19], with computations performed on supercomputers provided by the Jülich Supercomuting Center (JSC). To account for the large number of spikes occurring at the beginning of the simulation, the relevant time frame is further reduced to start at 500 ms, resulting in a period from 500 ms to 2 500 ms. Further, the simulation output consists of spike counts per 0.1 ms, which are converted to per 1 ms for improved readability and size reduction.

In principle, the network is a reduced and modified version of the Potjans-Diesmann eight population network, extracting the fourth layer of inhibitory and excitatory populations, L4I and L4E. Our network consists of a total of 27 394 leaky integrate-and-fire (LIF) neurons, with approximately 80/20 percentage split between the populations, resulting in 21 915 excitatory and 5 479 inhibitory.

To provide an initial understanding of the data we are working with, Figure 3.1 presents the spiking activity of 100 inhibitory and excitatory neurons from a randomly drawn sample of the dataset. In the uppermost portion of the figure, individual dots represent the spiking activity of a neuron at that given line. The two histograms displayed beneath the spiking activity plot illustrate the density of the neuron spikes over the course of the simulation interval. These histograms are derived from the same set of neurons depicted in the spiking activity plot above.

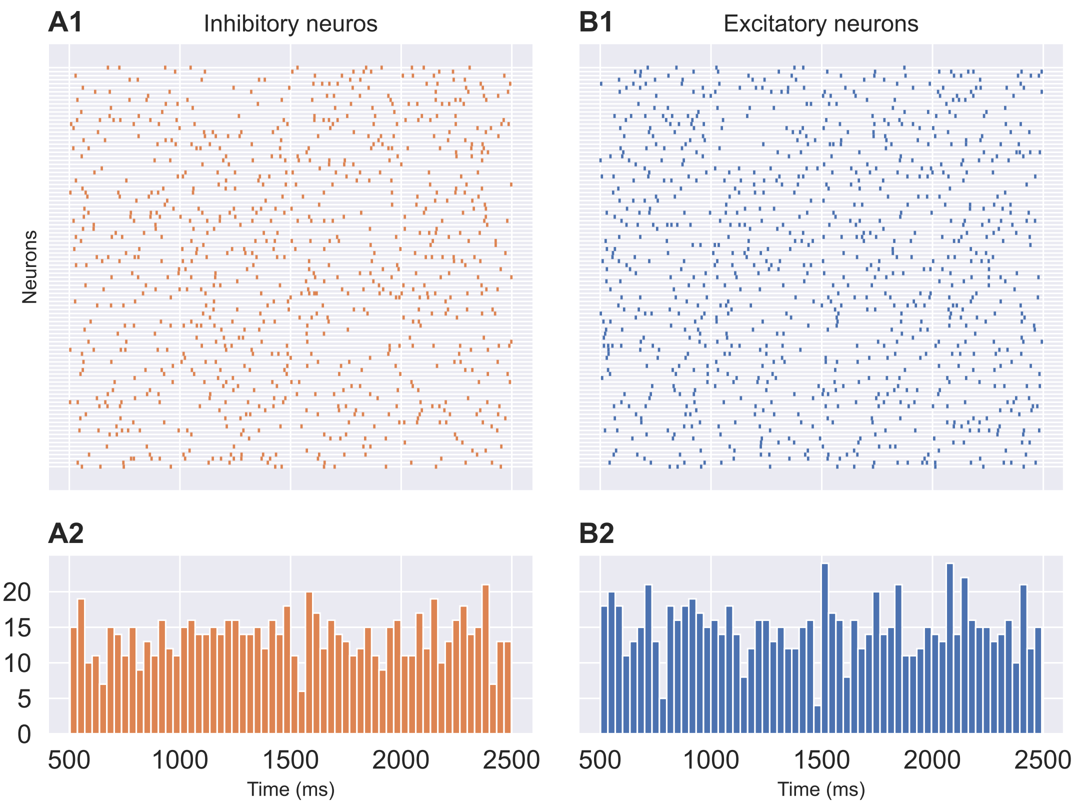

Figure 3.1: **A1-B1**: Raster plot of the spiking activity of 100 inhibitory and excitatory neurons respectively. **A2-B2**: Histogram showing the density of spikes at a given time period. Calculations are based on the same number of neurons illustrated in A1 and B1.

To construct the connections of the network, the predefined number of neurons are assigned to their respective population, be it excitatory or inhibitory. Each neuron within a population has a probability of connecting to others, denoted as, C_{YX} (see Table 3.2). This could either be of the same population (if $Y = X$) or a different population (if $Y \neq X$). Here, Y represents the postsynaptic population, while X represent the presynaptic population, and their connection probability is defined as the probability of forming at least one synaptic connection with a neuron of the postsynaptic population [11].

Table 3.1: Synaptic strength parameters for the two-population model. The format of parameter names is given by $g_{postsynaptic,\ presynaptic}$.

Parameter	**Description**
g_{ee}	`Connection from excitatory to excitatory`
g_{ei}	`Connection from inhibitory to excitatory`
g_{ie}	`Connection from excitatory to inhibitory`
g_{ii}	`Connection from inhibitory to inhibitory`

The varying parameters of the network are the synaptic strengths between the neurons of each population, denoted as g_{ee}, g_{ie}, g_{ei} and g_{ii} (refer to Table 3.1). The notation is such that the postsynaptic neuron is denoted first, followed by the presynaptic neuron. For instance, g_{ie} signifies the relative synaptic strength of an excitatory cell when connecting to a neuron from the inhibitory population. To calculate the synaptic weight, each of the respective synaptic strength parameters are multiplied with J (see Table 3.2), a reference metric of the synaptic strength value. The synaptic strengths (g_{yx}) are drawn from a uniform distribution, as illustrated by Figure 3.2, making it an uninformed prior. The uninformed prior offers a neutral starting point for the analysis and is intended to be updated as the methods discussed in Chapter 4 are applied.

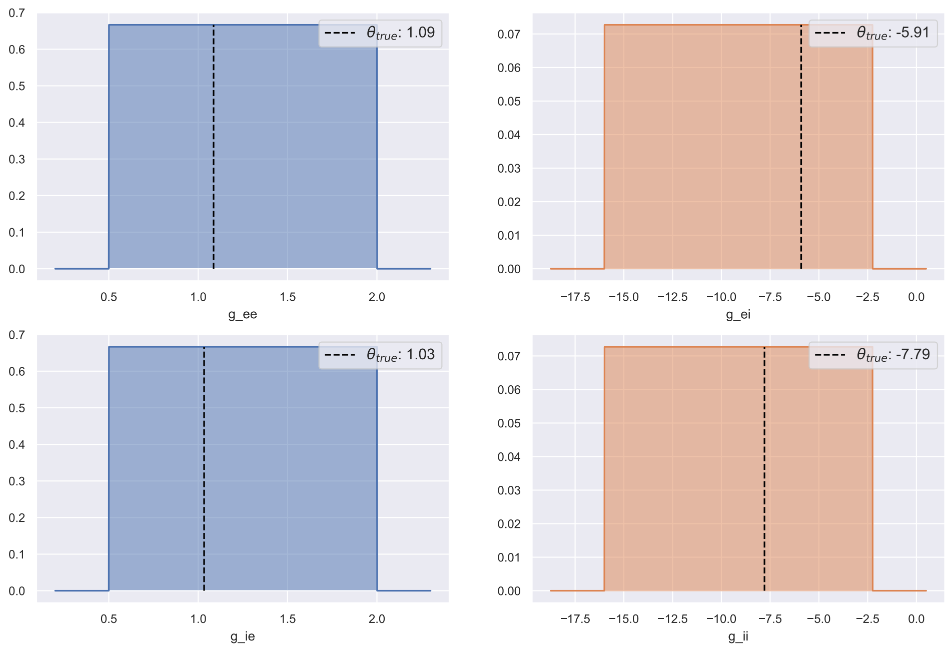

Figure 3.2: The prior distribution of the synaptic strength parameters. The dashed black line in each respective plot represents the observed parameter value. The observation value is extracted as a random sample from the initial set of simulations.

As the network is a reduced and modified version of the original Potjans-Diesmann eight population network, readers interested in further technical details about the network are encouraged to read the previously cited Potjans-Diesmann paper ([11]), as well Romaro et al. [20] and Hagen et al. [21]. For the remaining network and neuron parameters used in this thesis, refer to Table 3.2.

Table 3.2: Network and neuron parameters used for the simulation of the two-population model. Simulations are done with the NEST simulator version 3.4, and computations are performed on supercomputers provided by the Jülich Supercomuting Center (JSC). **C_YX**: The parameter C_YX follows the logic of connecting a neuron from population X onto population Y. C[0, 0] is the connection from excitatory to excitatory, C[0, 1] is from inhibitory to excitatory, C[1, 0] is from excitatory to inhibitory and C[1, 1] is from inhibitory to inhibitory.

Network parameters		
Parameter	**Value**	**Description**
`N_E`	21 915	`Number of excitatory neurons`
`N_I`	5 479	`Number of inhibitory neurons`
`k_ext_E`	2 100	`External input to the excitatory population`
`k_ext_I`	1 900	`Extrnal input to the inhibitory population`
`C_YX`	0.050, 0.135 0.079, 0.160	`Connection probabilities between neurons`
`d_E`	1.5	`Synaptic delay for the excitatory population (ms)`
`d_I`	0.75	`Synaptic delay for the inhibitory poulation (ms)`
`J`	87.81	`Refernce synaptic strength (pA)`
Neuron parameters		
`tau_m`	10.0	`Membrane time constant (ms)`
`t_ref`	2.0	`Refractory period (ms)`
`C_m`	250.0	`Membrane capacitance (pF)`
`E_L`	-65.0	`Resting membrane potential (mV)`
`V_th`	-50.0	`Threshold potential (mV)`
`V_reset`	-65.0	`Reset potential (mV)`
`tau_syn_ex`	0.5	`Excitatory synaptic time constant (ms)`
`tau_syn_in`	0.5	`Inhibitory synaptic time constant (ms)`

3.2 Summary statistics of spike trains

To capture essential information of the network, and reducing the output dimensionality, several summary statistics are generated, specifically *mean firing rate*, *Fano factor*, *mean interspike interval* and *mean coefficient of variation.* These summary statistics are calculated separately for the two populations, resulting in a total of eight statistics. The calculations are based on spike trains of 100 neurons from each population, which helps save disk space and computation time. It is important to note that numerous other summary statistics could have been developed to better capture the essence of the output data. However, given constraints of the python Elephant package [22] (used to generate the statistics) and the use of an embedding net as an alternative approach to summary statistic gathering (covered more extensively in Section 4.1.2), these eight were chosen.

In certain simulations, when the inhibitory activity become too large, the excitatory neurons cease firing. This issue creates problems for certain summary statistic calculations, such as the interspike interval, a measure on the length between spike times. If a neuron fires one time or less during the simulated interval, the interspike interval value is returned as *nan* (not a number). Most statistical models have a difficult time interpreting nan-values, and simulation samples that produce these values are therefore dropped. An alternative approach to address this issue is to merge the two populations and calculate summary statistics based on the combined populations instead. However, this method is not employed, as it is believed that compressing the summary statistics into four dimensions instead of eight would result in a greater loss of information than dropping a limited number of samples containing nan-values. A third, yet unexplored, method could involve using interpolation or clustering techniques to more intelligently compute the missing values.

The summary statistics are calculated as shown by the equations below. To adjust for the summary statistics with mean values, all values are appended to a list and averaged over the total number of neurons belonging to the relevant population. The firing rate (FR), as shown in Eq. 3.1, is the average number of times a neuron spikes per ms. Here, n denotes the total number of spikes from a neuron, and T represents time.

$$\text{FR} = \frac{n}{T} \tag{3.1}$$

The Fano factor (F), Eq. 3.2, quantifies how much a neuron variate around its mean spike ratio [23]. It is calculated by grouping all spike trains (st) and dividing the variance (σ^2) by the mean (μ).

$$\text{F} = \frac{\sigma^2(st)}{\mu(st)} \tag{3.2}$$

The interspike interval (ISI) is the time between occurrences of spikes (Eq. 3.3). Depending on the number of spike occurences, we can end up with a long list of numbers. These are then averaged per neuron, before taking the total average of the population.

$$\text{ISI} = t_{i+1} - ti \tag{3.3}$$

Here, t_{i+1} and t_i denotes the spike time of spike $i+1$ and i, respectively. The coefficient of variation (CV) uses the ISI to calculate the variability of the distribution, by dividing the standard deviation (σ) of the individual ISI by the mean of the ISI, as shown in Eq. 3.4

$$\text{CV} = \frac{\sigma(\text{ISI})}{\mu(\text{ISI})} \tag{3.4}$$

3.3 Observation

In order to capture the target posterior distribution, we need an observation that reflects the output values from a specific data point. The observation is chosen randomly as one of the samples generated from the initial simulation iteration, with the details on the number of simulations per iteration discussed further in Chapter 4. Since this is a probabilistic model, the output generated from the same synaptic parameters may vary. To obtain a more accurate estimate of the output given a set of parameter values, we keep the synaptic strengths fixed and do 100 additional simulations based on these parameter values. The final observation output is then the average of these 100 simulations. The variations of the observed summary statistics are illustrated in Figure 3.3. The dashed orange line represents the base, i.e., the values corresponding to the sample drawn from the initial simulation, while the dashed black line represents the final value as the mean of the 100 simulations.

For reference, the distribution of summary statistics from the first simulation is also displayed (refer to Figure 3.4). The figure provides a basis for comparison between the simulation results and the observation later in Chapter 5. However, certain simulations exhibit *synchronous regular* (SR) activity, meaning that the neurons fire synchronously and with much higher frequency

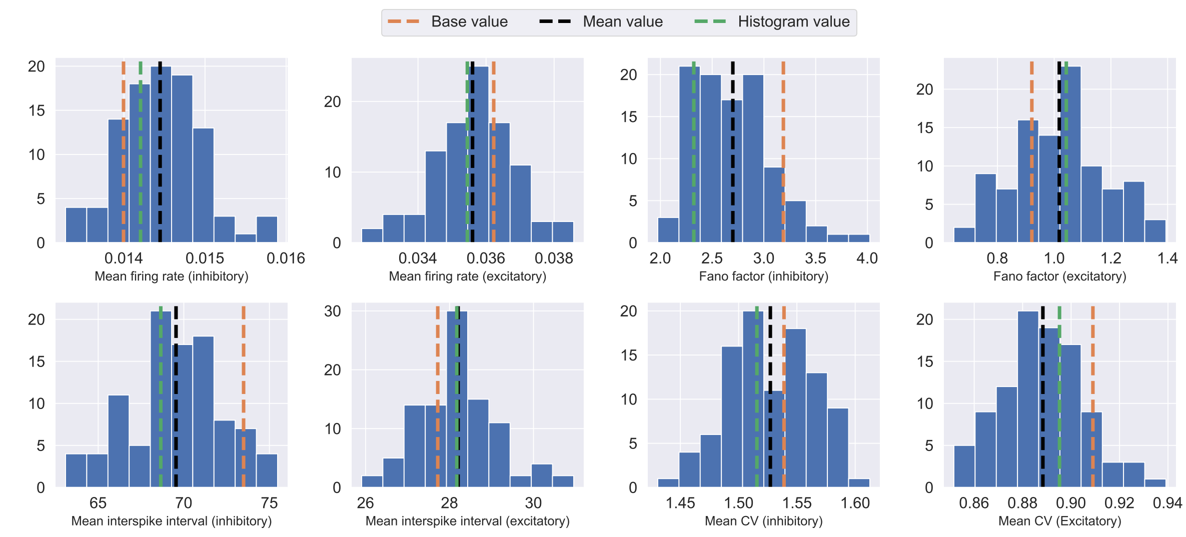

Figure 3.3: Histograms illustrating the distribution of each summary statistic metric. The values corresponding to the original observation from the initial simulation are shown with a dashed orange line. The dashed black line is the mean value, used as the metric for the observation. The green dashed line indicates the individual sample (out of the 100) closest to the mean, and is the sample used for the spike count histogram observation.

compared to the rest. Consequently, this increases the summary statistics domain for a few set of samples, making the visual representation hard to interpret. To make the graph more readable and facilitate comparison with the observation, values with a z-scores above 2 or below -2 (which includes values not in the SR regime) are removed from the plot, accounting for approximately 10% of the dataset.

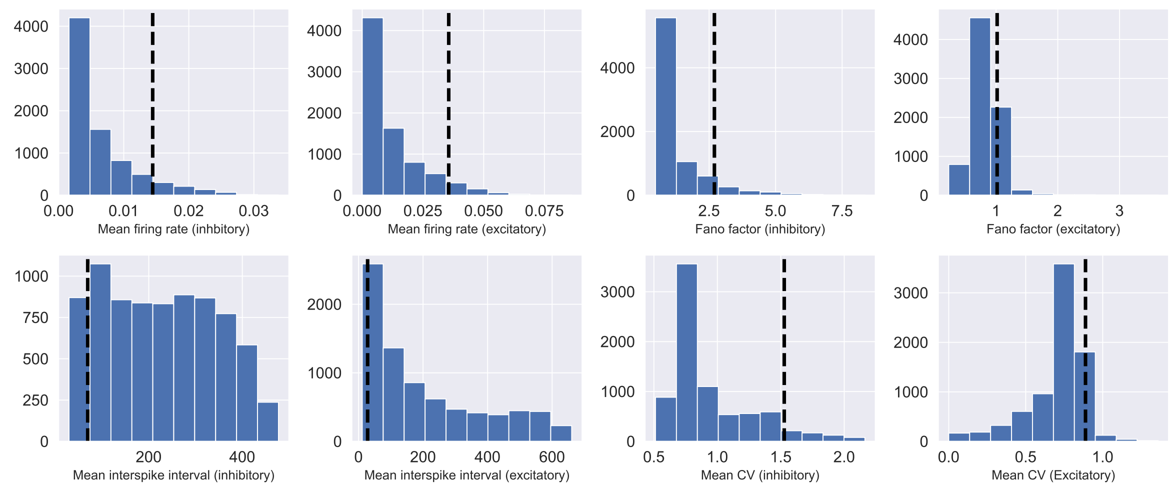

Figure 3.4: Histogram illustrating the distribution of each summary statistic metric from the initial sequence of simulations. Samples with z-scores above 2 and below -2 have been removed for readability (approximately 10% of the dataset). The dashed black line represents the value of the observation.

As previously mentioned, an embedding network is used as an alternative approach to learning summary statistics based on the spike count histogram. However, constructing the final observed histogram by taking the mean spike count from each bin (ms) turned out to be suboptimal.

There are large variations in spike counts in each bin across the 100 simulations, but the average value is approximately the same. This results in bins all containing approximately the same value, even though the individual simulations show significant variation in spike counts between bins. In fact, the intricate correlation between excitatory and inhibitory neurons, as well as their oscillation, is challenging to replicate. To address this issue, the dataset containing the 100 observed simulations was used to extract the sample closest to the mean. The summary statistics were scaled to ease comparison, and the sample with the shortest total Euclidean distance from the mean was chosen, shown in Figure 3.3 as the dotted green line. Figure 3.5 gives a visual representation of the network's oscillation. The opaque blue line at the forefront represents the final histograms used as the observation, contrasted against the other 99 histogram lines depicted in transparent green.

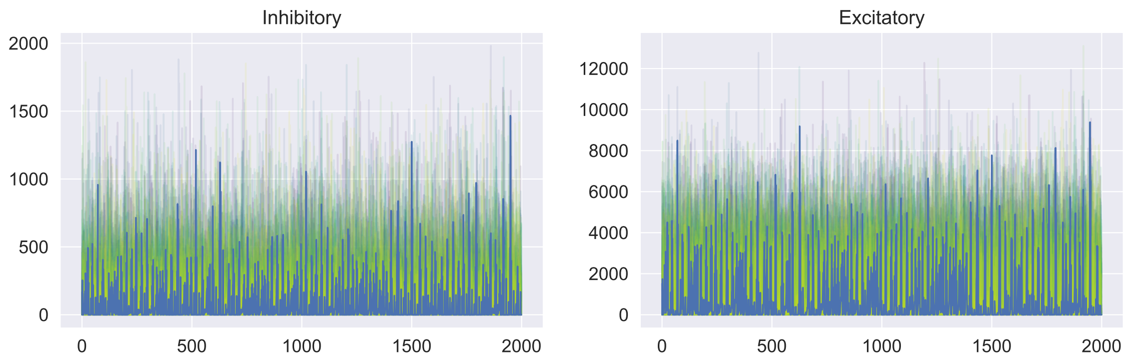

Figure 3.5: Oscillations of the various spike count histograms given by the 100 samples simulated on fixed parameter values from the observed sample. The opaque blue color represents the histogram chosen to represent the observed value. The transparent green color in the background shows the variation in spike counts for the other 99 samples.

Chapter 4

Methods

This chapter aims to provide a comprehensive explanation of the various methods employed to construct parameter posteriors that match the output from the observed data point. Essentially, this involves implementing practical versions of the theoretical Bayesian inference methods introduced in Chapter 2. The procedure consists of the following steps:

1. Simulate $1 \times 10\,000$ samples of a two-population neural network model for 2 500 ms using NEST, drawing synapse parameters (g_{yx}) from a uniform distribution. Reduce the interval to 500ms:2500ms (as discussed in Chapter 3).
2. Drop nan-values to end up with a total of 8 793 samples.
3. Extract a random sample to be used as observed data point, while considering the remaining samples as simulated data.
4. Create a more robust observed sample by simulating 1×100 samples and performing relevant calculations (as discussed in Chapter 3).
5. Construct a posterior distribution based on the following methods:
   - ABC with linear regression adjustment
   - SNPE using MAF and handcrafted summary statistics
   - SNPE using MAF and an embedding network for summary statistic extraction
6. Simulate $3 \times 1\,500$ samples of the same network, drawing parameters from each respective posterior obtained in the previous step.
7. Repeat step 5.
8. Simulate 3×100 samples of the same network, drawing parameters from each respective posterior of the previous step.
9. Evaluate the findings from step 8 against the observed data point.

As briefly mentioned in the introduction, the programming related to this project has been carried out using the Python programming language. The publicly available GitHub repository can be found at github.com/strandxd/masteroppgave-data-2023, while an overview of the most important folders can be found in Table A.2.

4.1 Estimating posteriors

In this thesis, three distinct approaches are constructed: the ABC path with linear regression adjustment, the SNPE path using MAF as a neural density estimator without an embedding net, and the SNPE path using MAF as neural density estimator with an embedding net. These methods all vary in complexity and, consequently, computational cost, with ABC being the least complex and SNPE with an embedding net being the most complex. This contributes to the interest in comparing the methods, as it enables us to ascertain whether augmenting complexity enhances quality, or if such augmentation is wasteful.

4.1.1 ABC with regression adjustment

The initial step involves constructing a general rejection ABC algorithm that compares the summary statistics of the simulated and observed data. Due to differences in scale and the sensitivity of distance metrics to range variations, it is necessary to standardize the data. This is achieved by subtracting the mean and dividing by the standard deviation, like so: $z = \frac{x-\mu}{\sigma}$. The standardized summary statistics can then be compared between variables.

Subsequently, the Euclidean distance between each simulated sample and the observed sample is calculated based on the previously computed standardized output. A quantile of the distances closest to the observed sample is then chosen, which determines the acceptance threshold of the algorithm. In this work, a quantile of 10% is chosen, resulting in a dataset of 880 samples after applying the rejection ABC algorithm to the initial set of 8 793 samples (after droppping nan-values from the set of 10 000).

The quantile-based approach used in this method can make it challenging to obtain the true posterior, as the algorithm may not always select samples that accurately represent the underlying distribution. To account for this, we apply linear regression on the newly constructed dataset. In this case, the simulated summary statistics serve as the regressor (independent variable), and the parameter values act as the response variables (dependent variable). The resulting linear model is then used to adjust the parameters obtained by the previous ABC algorithm based on the difference in predicted output between the observed and simulated summary statistics, informally as follows:

$$\text{Adjusted parameters} = \text{linear\_model.predict}(\text{X}_{\text{obs}}) - \text{linear\_model.predict}(\text{X}) + \text{y} \tag{4.1}$$

where X represents the standardized summary statistics in the simulated dataset, X_{obs} denotes the standardized summary statistics for the observed data, and y is a matrix containing parameter values, with y_i representing each individual parameter value. The *linear_model* refers to the linear regression model that predicts the parameter values. Finally, to calculate the probability density function (PDF) we make use of *gaussian_kde* from the Python SciPy package [24], which also normalizes the output for us. This package is used throughout the project to construct PDFs where needed.

4.1.2 Neural posterior estimation

In this work, two neural posterior estimators are implemented to learn posterior mappings from a prior parameter distribution. Both of these estimators make use of the SBI Python package [15] and, specifically, the SNPE algorithm. However, they differ in their approaches: one employs handcrafted summary statistics, while the other takes the spike count histogram as input and attempts to learn its own summary statistics through an embedding network.

The initial step in the SNPE implementation without an embedding network is to define a neural network architecture. In this case, the MAF is used as a neural density estimator. Additionally, the number of hidden features, number of transformations, and learning rate are specified. Furthermore, the data is standardized independently for each column, which involves calculating a z-score for each value within its respective column. The optimization process is discussed in detail in Section 4.2, but the final hyperparameter values for the different simulation iterations can be found in Table 4.1. The iterations are distributed as follows: 8 793 for the first iteration (after dropping nan-values of the 10 000 samples), 1 500 for the second iteration. The third iteration, containing 100, only uses the raw output and are not considered as part of this.

Table 4.1: The final hyperparameter values of SNPE without an embedding network after optimizing the network.

Name	**Value iteration 1**	**Value iteration 2**
`Hidden features`	`172`	`185`
`Transformations`	`14`	`2`
`Learning rate`	`0.000481`	`0.000439`

In the SNPE implementation, there is an option to include the simulation model in the process (in our case the NEST implementation). However, as this work relies on an external supercomputer to perform the simulations, the simulated dataset is instead appended to the SNPE object along with the relevant parameter values. In the end, the observed summary statistics are passed as arguments to obtain a posterior distribution. As previously mentioned, for the first iteration, the prior is a uniform distribution, but for the subsequent simulation iterations, the prior is updated to be the posterior estimation found from simulation $i - 1$.

The alternative SNPE implementation utilizes a CNN as an embedding network to summarize the raw output data, specifically the spike count histogram. The details of the CNN network architecture can be found in Table 4.3. This approach introduces a new hyperparameter to optimize: the number of summary statistics to use. The final hyperparameter values can be viewed in Table 4.2. As for the standardization of parameters, the varying synapse strengths are standardized independently, like the SNPE method described above. However, the spike count histogram is standardized using a structured approach. This involves calculating z-scores based on the entire histogram from both channels, i.e., both populations, in order to encapsulate the relationships between the populations.

Table 4.2: The final hyperparameter values of SNPE with an embedding network after optimizing the network.

Name	**Value iteration 1**	**Value iteration 2**
`Hidden features`	`152`	`254`
`Transformations`	`6`	`6`
`Learning rate`	`0.000036`	`0.000091`
`Summary statistics`	`52`	`7`

In this approach, we pass the parameter values together with their spike count histogram as training data instead of the summary statistics. The histograms are processed by the embedding network, which in turn outputs summary statistic values. To obtain the posterior, we pass the observed spike count histogram to the model. The subsequent simulations follow the same pattern of prior updates as the methods discussed earlier.

Table 4.3: Summary of the CNN embedding network. Num sumstats (number of summary statistics) is the parameter to optimize.

From layer	To layer	Output Shape
`Input`	`Conv1`	`(256, 4, 2000)`
`Conv1`	`MaxPool1`	`(256, 4, 1000)`
`MaxPool1`	`Conv2`	`(256, 8, 1000)`
`Conv2`	`MaxPool2`	`(256, 8, 500)`
`MaxPool2`	`Flatten`	`(256, 4000)`
`Flatten`	`FC1`	`(256, 350)`
`FC1`	`FC2`	`(256, num sumstats)`

4.2 Optimization

There are various methods of optimizing hyperparameters, ranging from brute force approaches such as grid search to random searches over a given hyperparameter space and combinations of the two. Although these methods are popular and straightforward to implement, they lack the ability to intelligently narrow in on promising areas to limit the search space. Optuna [25], an optimization software introduced in 2019, implements a Bayesian algorithm known as Tree-Structured Parzen Estimator (TPE), which enables dynamic hyperparameter optimization. Since this thesis focuses on Bayesian statistics, the choice of Optuna seems fitting, and its Python package allows for easy compatibility with PyTorch, which is also required for SNPE implementation using SBI.

The Bayesian aspect of the algorithm refers to the fact that we start with an initial belief, our entire hyperparameter search space, and iteratively update this belief as we gather information on how the hyperparameter combinations impact performance [26]. The initial hyperparameter values are chosen randomly, before grouping the results of these combinations into either a *good* or *bad* distribution, $l(x)$ and $g(x)$ respectively. The next iteration is then chosen as the maximum of $\frac{g(x)}{l(x)}$ [27]. This ensures that we draw hyperparameters from a distribution more likely to perform well, thereby increasing the likelihood of faster convergence in the optimization process.

The hyperparameter space used in this thesis can be viewed in Table 4.4. As previously mentioned, summary statistics as a hyperparameter are only applicable to the version implementing an embedding network, but the remaining hyperparameters are used for both variations.

Table 4.4: Values assosiated with hyperparameter optimization for both methods of the implemented SNPE algorithm.
*Summary statistics are only used as a hyperparameter during the SNPE implementation using an embedding network.

Name	Lower bound	Upper bound	Scale
`Hidden features`	`32`	`256`	`Linear`
`Transformations`	`2`	`14`	`Linear`
`Learning rate`	`0.00001`	`0.01`	`Logarithmic`
`Summary statistics*`	`15`	`70`	`Linear`

Chapter 5

Results

Evaluating the estimated posterior is challenging, as we do not have the true posterior for comparison. As a result, the evaluation process relies less on objective metrics and more on analyzing the output. In this chapter, we present the findings from the previously described methods.

5.1 Inspecting the posterior distribution

While not certain, it is reasonable to assume that values closer to the observed parameters have a higher likelihood of reproducing similar output. However, it is essential to consider this as a four-dimensional problem rather than a one-dimensional one, where the joint interaction of all four parameters is what produces the output.

Figure 5.1 shows the density estimations of the various parameter spaces throughout the simulated iterations. The base simulation, further referenced as *simulation 0*, serves as the foundation for further work. The equal probability of drawing parameters within the space results in a significant variation in simulated output. Given our randomly chosen observation, we aim to narrow this output to replicate what the observation yields.

Posterior 1 is the result of applying the Bayesian inference methods on simulation 0, with 8 793 samples (after dropping nan-values from the original 10 000). All applied methods manage to narrow the parameter space down closer to the observed parameter value, though to varying degrees between them and the individual parameters. g_{ee} and g_{ei} appear to be the most constrained by all methods, while the other two parameters display more varying results between the methods and are generally less constrained. SNPE without an embedding network, further referred to as novel SNPE, seems to be the most effective, with high densities either on or close to the observed parameters. SNPE with an embedding network, further referred to as embedding SNPE, has similar maximum densities close to the observed parameters but is generally less restrictive, meaning that the distribution curve is flatter overall. The regression-adjusted ABC method (ABC for short) highly overlaps the others on parameters g_{ee} and g_{ei} but shifts further to the right and left for g_{ie} and g_{ii}, respectively. This is not necessarily an indication of a poor posterior estimation, but given the simpler nature of this method compared to the others and the fact that it is the outlier of the three, it is reasonable to be wary of the results.

Posterior 2 is constructed by drawing parameters from posterior 1, with 1 500 samples. It is expected that the different methods should continue to optimize the posterior distribution by narrowing it down. This is indeed what happens for the ABC and the novel SNPE method. However, the embedding SNPE approach shows less improvement, as its distribution curves

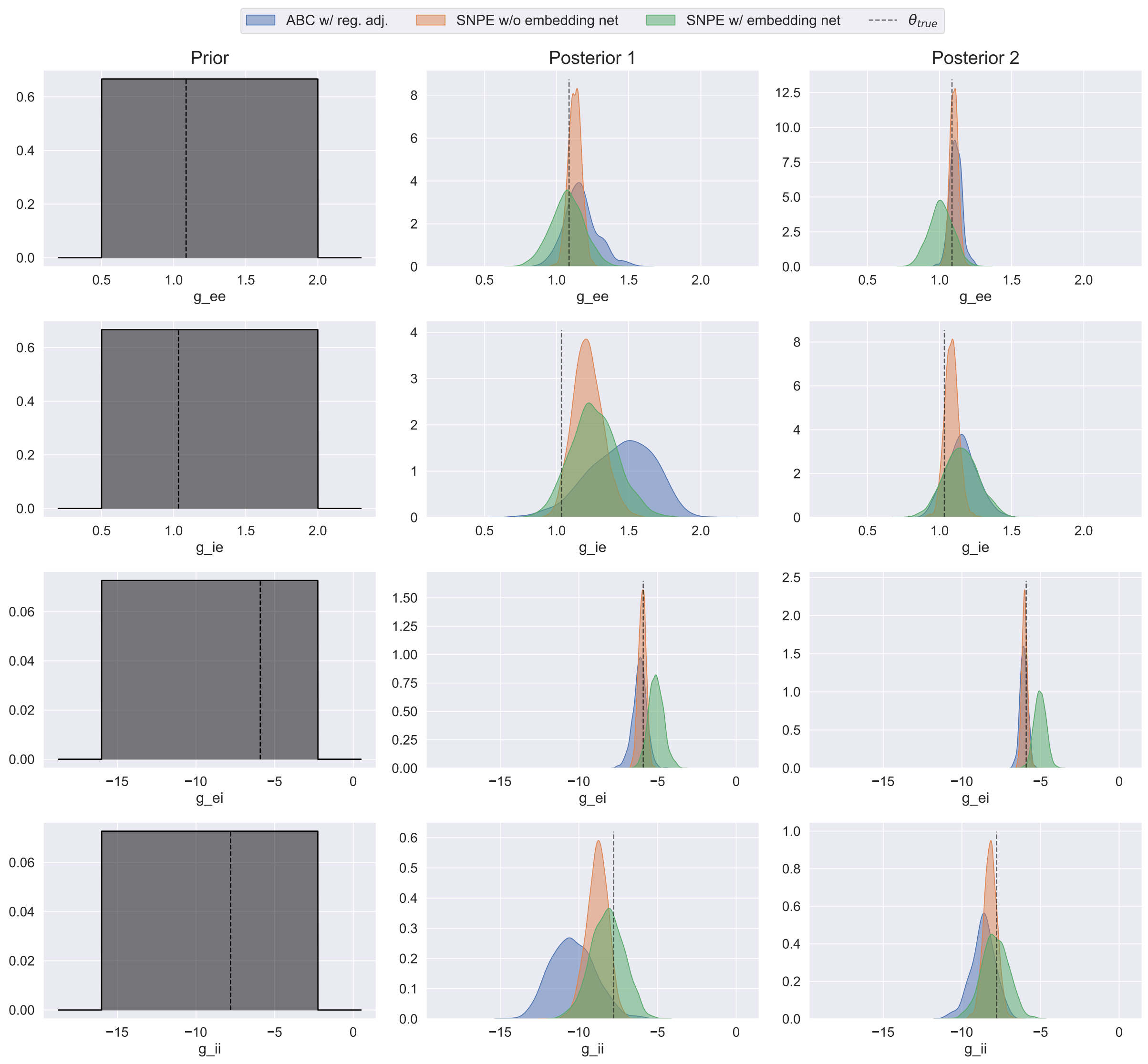

Figure 5.1: Plot showing the posterior distributions of each method, related to the prior (leftmost column), color coded as shown by label. The leftmost column shows the prior, the middle column shows the posterior (posterior 1) generated from the prior (8 793 samples), and the rightmost column shows the posterior (posterior 2) generated from posterior 1 (1 500 samples).

appear very similar to those in posterior 1. This could suggest that the method with an embedding net has already found a good estimation of the posterior, but further examination is needed to draw any conclusions. It is worth noting that since the two remaining methods continue to further constrain the parameter space, it might not be the case that the embedding net method has found the optimal constraint. Instead, it could be failing to extract the relevant information needed for further improvement. To better understand the performance of the embedding SNPE method, additional iterations could be implemented. This possibility, along with other potential improvements, is discussed further in the future work section of Chapter 6.

Figure 5.2 illustrates 2D plots of the parameters plotted against each other as a result of posterior 2, with the black marker representing the observed sample's parameter values. The color scheme is the same as in Figure 5.1: ABC is blue, novel SNPE is orange and embedding

SNPE is green. This color scheme holds true for the remainder of the text. Each of the six plots (e.g., A1) has a correlation matrix displaying how the different parameters correlate (e.g., A2). The most notable difference is that the ABC method shows different correlations between the parameters than the two SNPE methods, which are much more similar. The exception is the correlation between g_{ie} and g_{ii} as well as between g_{ee} and g_{ei}, which is highly similar between all three methods.

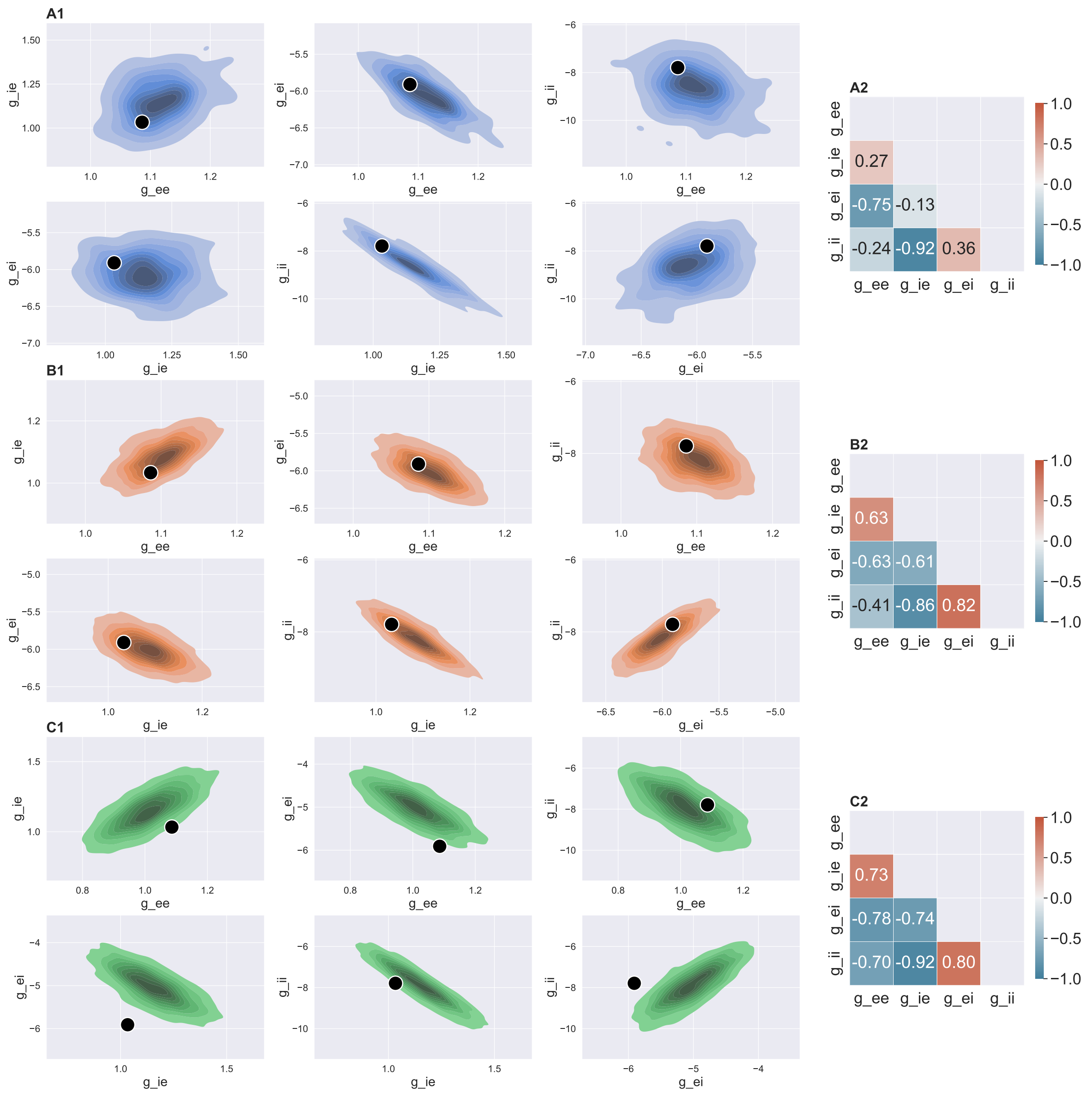

Figure 5.2: Plot illustrating the joint distribution between pairwise parameter combinations, where the densities are from the final posterior (posterior 2). **A1-A2**: The joint distribution of parameters (A1) of the linearly adjusted ABC method and a corresponding correlation matrix between these parameters (A2). **B1-B2**: The joint distribution of parameters (B1) of the SNPE method without an embedding network and a corresponding correlation matrix between these parameters (B2). **C1-C2**: The joint distribution between parameters (C1) of the SNPE method with an embedding network and a corresponding correlation matrix between these parameters (C2)

Additionally, we can see that the darkest area (where the highest densities of values are centered) is closer to the observed data point for the ABC method and the novel SNPE method. The embedding SNPE (green) is less effective in capturing areas around the observed target, which is consistent with what was shown in Figure 5.1. The method with the highest number of samples closest to the observed data point is the novel SNPE method (orange). It also displays high correlations between the variables, very similar to that of the embedding SNPE method, emphasizing the importance of drawing from the joint distribution to account for their dependencies. On the other hand, the ABC method reveals relatively weaker overall correlations between the variables. This might suggest that it is not capturing the linear dependencies between the variables as effectively as the SNPE methods. However, it is essential to note that the parameter correlations of the ABC method are not negligible, but comparatively lower than the other two.

The similarities of the correlation in the novel SNPE and embedding SNPE produces 2D plots that are very similar in shape. Their most significant variability lies between g_{ee} and g_{ii}, where the correlation value of the novel SNPE method indicates much less correlation compared to the embedding SNPE. However, as previously mentioned, the novel SNPE approach focuses more on the areas surrounding the observed data point. With probabilistic models, there is the chance that the output produced by parameters related to the observed data point is unlikely to occur, although this chance is lowered by conducting multiple simulations with fixed parameter values. It is possible that the true posterior is not centered around the observed parameters, but further away in the 4D space. An option then becomes to analyze how well the methods can reproduce the distribution of summary statistics.

5.2 Summary statistic comparison

From the final posterior, *posterior 2*, we can draw parameter values and see how well they resonate with the observed samples' summary statistic distribution. However, one value simulated by drawing parameter values from ABC posterior 2 exhibits SR activity and is therefore dropped, as its values excessively stretch the graphs and hinder interpretability. The red circles from Figure 5.3 show the dropped value and its associated parameter combinations, while the black round circle represents the observation. The novel SNPE method generates no such values, and it would make little sense to drop values from the evenly distributed flat curve produced by the embedding SNPE method, as these values are not definite outliers inconsistent with the remaining data.

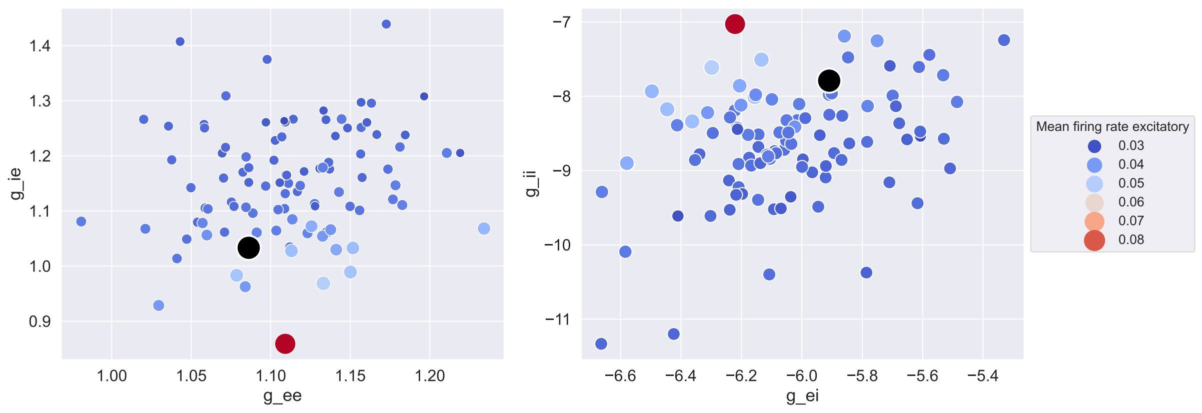

Figure 5.3: Scatter plot showing the distribution of parameters and a corresponding arbitrary summary statistic for the ABC method. The red circle symbolises the sample to be dropped, while the black circle represents the observation for reference.

Figure section **A** of Figure 5.4 presents the ABC method's produced summary statistics (blue) in comparison with the observed summary statistics distribution (purple). While metrics such as the Fano factor for both populations heavily overlap, there is more spread in mean firing rate and interspike interval, two heavily correlated metrics. High firing rates typically involve short interspike intervals and vice versa. Despite these variations, even after removing the outlier, most of the data points seem to revolve around the same ranges as the observed metrics.

As seen in figure section **B** of Figure 5.4, the novel SNPE method seem to overlap the most of the three methods. As this is based on 100 samples, the possibility of it being due to chance has to be considered, so making any definite conclusions is unwise. However, the method presents few values outside the range of the observed summary statistics, and the general distribution curve for the Fano factor is nearly identical, with minimal deviation. The previous section (Section 5.1) revealed that this method is also the most restrictive of the three. These findings further suggest that the posterior distribution is indeed centered around a tight distribution with focus around the observation.

The final figure section, section **C** of Figure 5.4, illustrates the embedding SNPE method. This curve is flatter than the others, producing a broader range of summary statistics. The 2D plot in figure 5.2 illustrated that the joint distribution of parameter combinations resembled the novel SNPE method's shape, although it was less constrained around the observed value. This may indicate that the model successfully captures the underlying interactions between the variables but fails to produce the correct combinations of values. Since this method is not trained on the summary statistics themselves but rather on the raw output data in the form of spike count histograms, the lack of results in this regard may not be all that surprising. It is possible that the summary statistics produced by the embedding net are more capable of tracking the actual oscillations of the raw output data, which is arguably more important than being able to replicate the handcrafted summary statistics. In the next section, we will look further into this possibility.

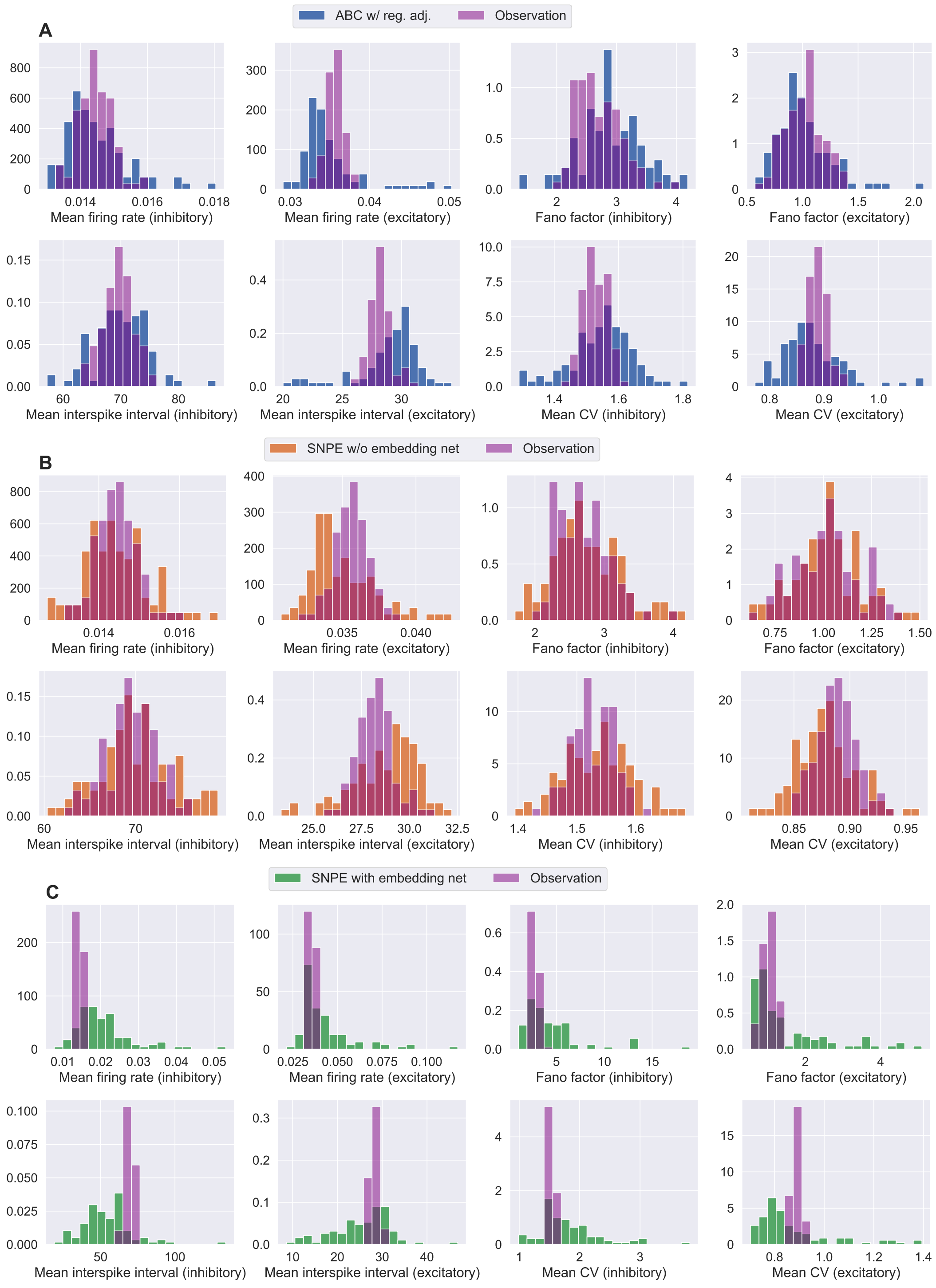

Figure 5.4: Density (y-axis) of summary statistics (x-axis) compared to the observation. Based on 100 simulations from posterior 2 from each respective method. **A**: ABC with regression adjustment (blue) compared to the observation (purple). **B**: SNPE without an embedding net (orange). **C**: SNPE with an embedding net (green).

5.3 Power spectral density

The power spectral density (PSD) is a measure of the distribution of power over various frequency components in a signal. Again, we use the 100 samples created by drawing parameters from posterior 2 of each respective method. We can then transform the time series histograms into a range of frequency values, thereby allowing the identification of the frequency components with the most power (i.e., the most activity). Analyzing the PSD can provide insight into how well our summary statistics are able to summarize the raw output data of the simulations.

Figure 5.5 shows the PSD plots for the three methods: ABC in blue, novel SNPE in orange and embedding SNPE in green. The yellow line represents the observation, and the colored stipples lines indicate values within one (red), two (black) and three (magenta) standard deviations, respectively.

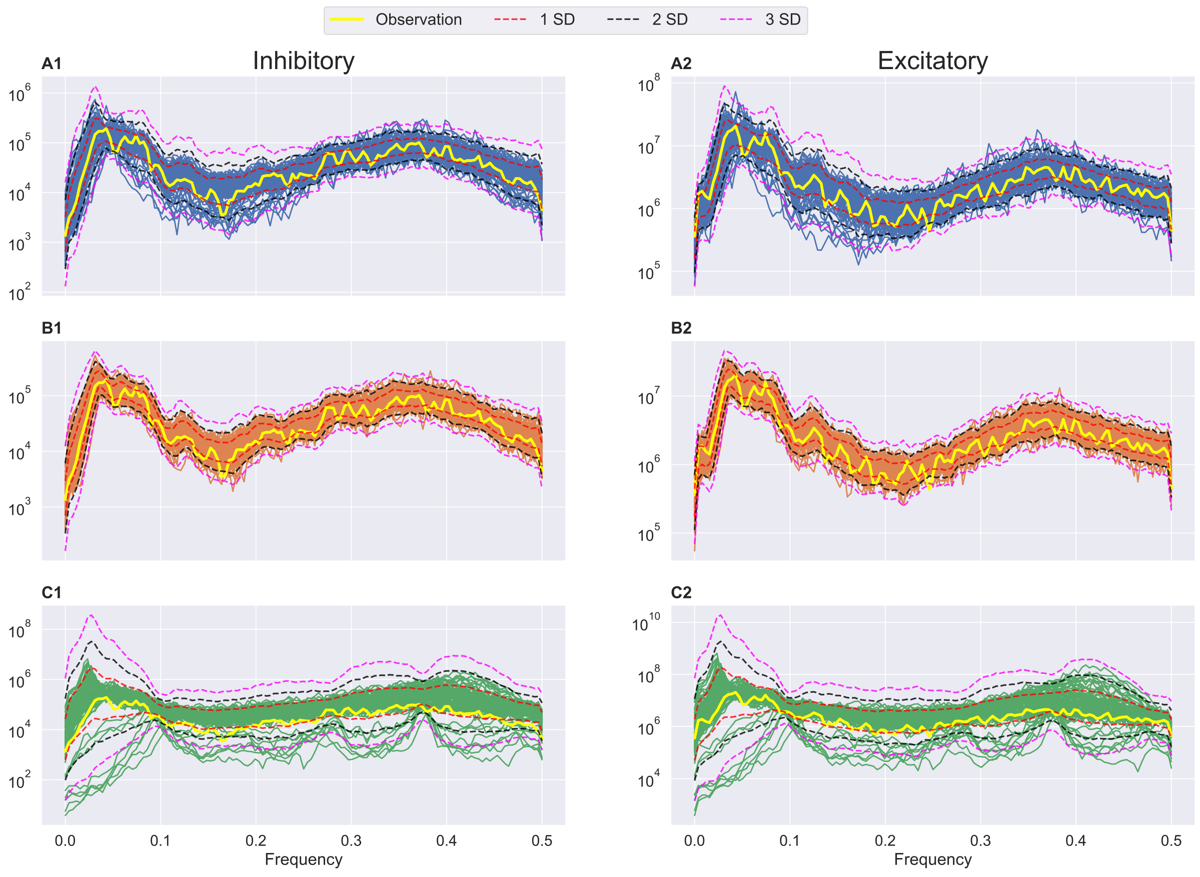

Figure 5.5: **A1-A2**: ABC with regression adjustment (blue). **B1-B2**: SNPE without an embedding net (orange). **C1-C2**: SNPE with an embedding net (green). Power spectral density plot of all three methods, with inhibitory and excitatory population as indicated by the titles, plotting frequency (x-axis) against power (y-axis). The yellow line represents the observation, the red dotted line represents one standard deviation away from the mean of the given method's PSD. The dotted black and magenta lines indicated two and three standard deviations away from the mean, respectively.

The ABC and novel SNPE methods exhibit the most stable oscillations, as evidenced by their narrow standard deviations, few samples outside the third standard deviations, and most samples centered around the observed PSD. Among the two, the novel SNPE method appears to have the least variability and is best able to replicate the observation curve.

In contrast, the embedding SNPE method demonstrates greater variability, which consequently increases the standard deviation. Although most of the values follow the observation curve, they are centered above it, rather than around it, as seen with the other two methods, although it is mostly within the range of one standard deviation. Most of the values that significantly deviate from the observation pattern are found outside the third standard deviation, indicating a few, but large, misses. As these are, by definition, not common occurrences, they pose less of a problem when evaluating the overall performance of the method.

Chapter 6

Discussion

The goal of this thesis has been to explore and compare various LFI methods to estimate a posterior distribution of synaptic parameters in a two-population neural network based on a single set of parameter values, used as observation. The summary statistics of the observed sample were obtained from the mean output of 100 simulations. In total, three methods were constructed: an ABC method with linear regression adjusted parameters, and two SNPE methods using MAF as neural density estimator, one with the use of an embedding net and one without.

As previously mentioned, there is no definite metric to validate such a process. However, the findings in Chapter 5 provide some valuable insights into the performance of these methods. The more restrictive the posterior is around the observed parameter values, the better it seems to reproduce the observed output. This might seem trivial, but given the probabilistic and highly complex nature of the simulation model, one might expect that a wider range of parameter combinations would reproduce the same output. After all, the true posterior might be multimodal and involve parameters outside the investigated parameter space.

Reproducing the summary statistics, as shown in Figure 5.4 of the previous chapter, is also not necessarily trivial to interpret. There is an argument to be made that the true posterior would not reproduce the identical summary statistics distribution, again because of the probabilistic neural model. However, the observed summary statistics seem to have a clear pattern where some values are more likely to be produced. We only compare 100 samples, so expecting an identical match would not be realistic. The argument is still that given simulated samples moving towards infinity of both the observation and posterior, the true posterior would reproduce the same distribution of summary statistics. Given that argument, the more overlapping the 100 samples are, the better. Out of the three methods, the novel SNPE method seems to overlap the most.

One potential limitation of the thesis is choosing the correct summary statistics. It is possible that the chosen metrics may not fully capture the complexity of the model used for the simulations. The compact and highly similar patterns of the PSD plot of the ABC method, and particularly the novel SNPE method, compared to the observation does however indicate that the summary statistics are likely sufficient to compute the underlying structure of the neural model. The embedding SNPE uses a CNN to produce the summary statistics, and these appear to be less informative than the handcrafted ones. This could be attributed to various factors, such as the number of simulated data points, the number of iterations we perform the simulations (i.e., how many sequences of simulations we do), and the quality of the CNN architecture itself. Initially, it was assumed that letting a computer handle the summary statistic gathering would be the better choice, but the results suggest otherwise.

The trade-off in computational efficiency was also an area of interest, not taking into account the simulations, as they were handled by a supercomputer and are considered a separate aspect entirely. The contrast in runtime quickly became clear as the ABC method is computationally inexpensive compared to the two SNPE methods, as it does not require any optimization, but its performance is also less accurate than the novel SNPE method. On the other hand, the embedding SNPE method has a relatively long optimization time, but its performance does not justify the additional computational cost as of now. However, it is important to note that the runtime on a trained model is low, and given more time to optimize the method, it might yield better results. The CNN part of the embedding SNPE method has generally received little optimization focus, so the potential for improvement is especially high for this method.

In conclusion, the novel SNPE method appears to capture the posterior distribution the best, based on its ability to replicate results. The ABC method is also able to replicate at a reasonably satisfactory level, while the embedding SNPE method seems to produce less accurate results. However, there is potential for future optimization improvements of the embedding SNPE method, as it shows similar correlation patterns to that of the novel SNPE, indicating high capability of capturing joint parameter dynamics.

6.1 Future Work

As part of the work for this thesis, some work has also been done on a four-population biological neural network. This network adds two additional layers of the original Potjans-Diesmann network, specifically the combined L2/3 excitatory and L2/3 inhibitory. With an increased network size, there are a number of increased synaptic connection parameters, 16 in total. The realization of the number of simulations needed to obtain reasonable results, combined with limited available CPU hours and the fact that the process would essentially be the same, made the priority shift fully towards the two-population network. It would still be interesting to see whether it would be possible to produce reasonable posteriors of such a network, where the problem is 16-dimensional instead of 4.

Further, it would be interesting to continue improving the work discussed in this thesis. For example, one could look at another parameter space entirely, or expand on the existing one. There is also a chance that the characteristics of our observed sample were particularly easy, or hard, to replicate. Choosing different observations and performing the same process to see how it works would be beneficial in testing the robustness.

Improvements in optimization are also worth exploring, as this is never truly finished. There is also the option of improving the way the final observed spike count histogram is chosen. As of now, this is chosen as the sample whose combined summary statistic distance is closest to the final observation (mean). Training neural networks, either recurrent or convolutional, would most likely require way more than the 100 samples used in this thesis, but could potentially further optimize the path involving the embedding net.

Exploring nonlinear regression adjustment techniques, instead of linear, might also be a valuable improvement to the ABC method. The interactions of the synaptic parameters and the summary statistics are generally nonlinear, so incorporating such techniques could help capture these interactions at a better degree, leading to more accurate and reliable results.

Appendix A

Appendix

A.1 Python packages

Table A.1: Python packages with their respective version and main purpose of use.

Name	Version	Purpose of use
NEST	3.4	Simulating neural networks
Numpy	1.22.1	Calculations and array manipulation
Pandas	1.4.1	Data frame manipulation
Scipy	1.7.3	Calculations
Matplotlib	3.5.1	Plotting
Seaborn	0.11.2	Plotting
SciKit learn	1.0.2	Linear regression and scaling
PyTorch	1.13.1+cpu	CNN network
SBI	0.21.0	SNPE with MAF implementation
Neo	0.11.1	Spike train type conversion
Elephant	0.11.2	Summary statistics of spike trains
Optuna	3.1.1	Optimization

A.2 GitHub files

Table A.2: Overview of the folder structure representing the most important files used in the project. The ”data” folder is primarily used to transform the raw data (from the simulations) into accessible datasets. The ”ABC” and ”SBI” folders focus on implementing ABC and SNPE (both variations) respectively. The ”figures” folder contains files related to plotting, and additional utility functions can be found in the ”helpers” folder. The GitHub page can be found here.

File	Description	Git hash
`2pop/data/*`	`Data extraction from simulations`	`fc3cf8c`
`2pop/abc/*`	`Implementation of ABC methods`	`fc3cf8c`
`2pop/sbi/*`	`Implementation of SBI methods`	`fc3cf8c`
`figures/*`	`Plotting`	`fc3cf8c`
`helpers/*`	`Helper functions`	`fc3cf8c`

A.3 Hardware specifications

Table A.3: Hardware specifications for the desktop computer used to run all files except simulations. Simulations were run on supercomputers provided by Jülich Supercomuting Center.

Name	Version
`Processor`	`Intel(R) Core(TM) i5-8600K CPU @ 3.60GHz`
`RAM`	`8.00 GB`
`OS`	`Windows 10`

Bibliography

[1] F. A. Azevedo et al. “Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain”. In: *The Journal of comparative neurology* 513.5 (2009), pp. 532–541. URL: https://pubmed.ncbi.nlm.nih.gov/19226510/.

[2] Wikimedia Commons. *File:Neuron.svg — Wikimedia Commons, the free media repository.* 2022. URL: https://commons.wikimedia.org/w/index.php?title=File:Neuron.svg&oldid=648452462.

[3] P.J. Duncan and M.J. Shipston. *Chapter Nine - BK Channels and the Control of the Pituitary.* Ed. by Candice Contet. 2016. URL: https://www.sciencedirect.com/science/article/pii/S0074774216300241.

[4] Wikimedia Commons. *File:ActionPotential.png — Wikimedia Commons, the free media repository.* 2022. URL: https://commons.wikimedia.org/w/index.php?title=File:ActionPotential.png&oldid=628255538.

[5] A. L. Hodgkin and A. F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of physiology* 117.4 (1952), pp. 500–44. eprint: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413/pdf/jphysiol01442-0106.pdf. URL: https://physoc.onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004764.

[6] Wikimedia Commons. *File:Hodgkin-Huxley-model.svg — Wikimedia Commons, the free media repository.* 2022. URL: https://commons.wikimedia.org/w/index.php?title=File:Hodgkin-Huxley-model.svg&oldid=665450513.

[7] Südhof T. C. “Calcium control of neurotransmitter release”. In: *Cold Spring Harbor perspectives in biology* 4.1 (2012). DOI: https://doi.org/10.1101/cshperspect.a011353. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3249630/.

[8] T. P. Vogels, K. Rajan, and L. F. Abbott. “Neural network dynamics”. In: *Annual review of neuroscience* 28 (2005), pp. 357–376. DOI: https://doi.org/10.1146/annurev.neuro.28.061604.135637. URL: https://pubmed.ncbi.nlm.nih.gov/16022600/.

[9] T. C. Südhof. “The cell biology of synapse formation”. In: *The Journal of cell biology* 220.7 (2021). DOI: https://doi.org/10.1083/jcb.202103052.

[10] N. Brunel. “Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons”. In: *J Comput Neurosci* 8 (2000), pp. 183–208. DOI: https://doi.org/10.1023/A:1008925309027.

[11] T. C. Potjans and M. Diesmann. “The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model”. In: *Cerebral Cortex* 24.3

(Dec. 2012), pp. 785–806. ISSN: 1047-3211. DOI: https://doi.org/10.1093/cercor/bhs358.

[12] A. Gelman et al. "Bayesian Data Analysis". In: Chapman and Hall/CRC, 2013. Chap. 1: Probability and Inference.

[13] S. A. Sisson, F. Yanan, and M. A. Beaumont. "Handbook of Approximate Bayesian Computation". In: Chapman and Hall/CRC, 2018. Chap. 1: Overview of ABC.

[14] L. Serafeim. *What is Machine Learning: Supervised, Unsupervised, Semi-Supervised and Reinforcement learning methods*. 2020. URL: https://towardsdatascience.com/what-is-machine-learning-a-short-note-on-supervised-unsupervised-semi-supervised-and-aed1573ae9bb.

[15] Alvaro Tejero-Cantero et al. "sbi: A toolkit for simulation-based inference". In: *Journal of Open Source Software* 5.52 (2020), p. 2505. DOI: 10.21105/joss.02505. URL: https://doi.org/10.21105/joss.02505.

[16] Samer L. Hijazi, Rishi Kumar, and Chris Rowen. "Using Convolutional Neural Networks for Image Recognition By". In: (2015). URL: http://www.multimediadocs.com/assets/cadence_emea/documents/using_convolutional_neural_networks_for_image_recognition.pdf.

[17] O. Ronneberger, P Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). URL: http://arxiv.org/abs/1505.04597.

[18] George Papamakarios, Theo Pavlakou, and Iain Murray. "Masked Autoregressive Flow for Density Estimation". In: (2018). URL: https://arxiv.org/abs/1705.07057.

[19] Ankur Sinha et al. *NEST 3.4*. Version 3.4. Feb. 2023. DOI: 10.5281/zenodo.6867800. URL: https://doi.org/10.5281/zenodo.6867800.

[20] C. Romaro et al. "NetPyNE Implementation and Scaling of the Potjans-Diesmann Cortical Microcircuit Model". In: *Neural computation* 33.7 (2021), pp. 1993–2032. DOI: https://doi.org/10.1162/neco_a_01400.

[21] E. Hagen et al. "Hybrid Scheme for Modeling Local Field Potentials from Point-Neuron Networks". In: *Cerebral Cortex* 26.12 (2016), pp. 4461–4496. DOI: https://doi.org/10.1093/cercor/bhw237.

[22] M. Denker et al. *Elephant 0.11.2*. Version v0.11.2. Nov. 2022. DOI: https://doi.org/10.5281/zenodo.7307401.

[23] K. Rajdl, P. Lansky, and L. Kostal. "Fano Factor: A Potentially Useful Information". In: *Frontiers in computational neuroscience* 14 (2020). DOI: https://doi.org/10.3389/fncom.2020.569049.

[24] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[25] Takuya Akiba et al. "Optuna: A Next-Generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery: Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2623–2631. ISBN: 9781450362016. DOI: 10.1145/3292500.3330701. URL: https://doi.org/10.1145/3292500.3330701.

[26] H. Colin. *Building a Tree-Structured Parzen Estimator from Scratch (Kind Of)*. 2023. URL: `https://towardsdatascience.com/building-a-tree-structured-parzen-estimator-from-scratch-kind-of-20ed31770478`.

[27] James Bergstra et al. “Algorithms for Hyper-Parameter Optimization”. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS’11. Granada, Spain: Curran Associates Inc., 2011, pp. 2546–2554. ISBN: 9781618395993.