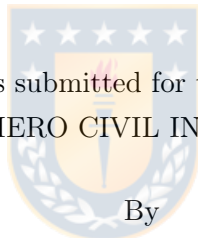




University of Concepción
Faculty of Engineering - Ingeniería Civil Informática

Bayesian parameter estimation using amortized variational inference



Thesis submitted for the degree of
INGENIERO CIVIL INFORMÁTICO

By

ALEXIS ELOY SÁNCHEZ RODRÍGUEZ

Concepción, Chile

January, 2020

Supervisor: Guillermo Cabrera Vives
Committee 1: Cecilia Hernández Rivas
Committee 2: Diego Seco Naveiras
Department of Computer Science
Faculty of Engineering
University of Concepción

©Se autoriza la reproducción total o parcial, con fines académicos, por cualquier medio o procedimiento, incluyendo la cita bibliográfica del documento.



Abstract

Through the use of models, it is possible to express information about a process or data being analyzed. These models seek to explain or predict the process of interest. Models with a known and fixed number of parameters, known as parametric models, let us interpret more easily the phenomena being studied by incorporating assumptions directly in the modeling part through the parameters. The parameters of the model must be estimated to accomplish a specific task related to the data being used. One way of finding these parameters is by using the Bayesian approach, which provides information about the uncertainty of the model being used as well as including prior information into the learning process. Methods commonly used for Bayesian inference, such as Markov Chain Monte Carlo or the faster but approximate Variational Inference methods are not suited when the model is used multiple times on different sets of data. This is because they work on one set of observations at a time, which means that the same procedure must be repeated from scratch to find the parameters of the model that work best for the new data. Instead, we provide experiments on synthetic data that show how Amortized Variational Inference can be used to obtain results comparable to more precise methods while reducing the inference time for new observations by making use of global information learned via an inference network. Our results show that our approximate inference procedure can provide results similar to classic methods even in the presence of noise for simple models.

Contents

1	Introduction	5
2	Bibliography discussion	9
2.1	Bayesian Inference	9
2.2	Markov Chain Monte Carlo	9
2.2.1	Metropolis-Hastings	10
2.2.2	Hamiltonian Monte Carlo	10
2.2.3	Disadvantages	11
2.3	Variational Inference	12
2.4	Amortized Variational Inference	15
3	Developed Model	18
3.1	Data	18
3.2	Inference procedure	20
4	Experiments and results	26
4.1	Experimental setup	26
4.2	Results	29
4.3	Discussion	38
5	Conclusion	40



List of Tables

4.1	-ELBO, negative log likelihood and divergence on the dataset corresponding to the linear parametric model as σ increases.	39
-----	--	----



List of Figures

3.1	Examples from each dataset, on the left, we have a simple linear model where the added noise is $\sigma = 0.1$, on the right side we have our second model that provides shapes similar to supernovae, the noise injected in this case corresponds to $\sigma = 0.01$	20
3.2	Probabilistic graphical model for both the linear and supernova models used in this work. We show the priors for the parameters of each model. Note that we consider the M curves and N data points of each one. The value of σ is assumed known and constant.	23
3.3	Architecture of the CNN used in this work. Residual connections are shown with dashed lines, 1x1 convolutions are used when increasing the number of channels. All convolutions are done with kernel of size 3 and stride 1, except on 1x1 convolutions, where the stride is 2.	24
4.1	Trace plots for three examples of the linear model. The vertical lines represent the true parameters that were used to generate the data. All examples use $\sigma = 0.5$	29
4.2	Mean fit when sampling parameters a and b from the posterior distributions obtained by MCMC. Data corresponds to the one used in figure 4.1.	30
4.3	Trace plots for three examples of the supernova parametric model. The vertical lines represent the true parameters that we used to generate the data. All examples use $\sigma = 0.05$	31
4.4	Mean fit when sampling parameters t_{rise} and t_{fall} from the posterior distributions obtained by MCMC. Data corresponds to the one used in figure 4.3.	32
4.5	Training curves for both models for different levels of noise σ in the dataset, the solid lines represent the training data while the dashed lines corresponds to the validation data.	33
4.6	Results on the test set for the inference network and MCMC. In (a) the dataset has $\sigma = 0.1$, while the supernova model depicted in (b) has $\sigma = 0.01$	34
4.7	Results on the test set for the inference network and MCMC, for a corresponding to the linear model the dataset has $\sigma = 0.9$, while for the supernova model depicted in b the noise corresponds to $\sigma = 0.09$	36

4.8 Noise in the parameters of the models as the noise σ in the dataset increases, in green we see the behaviour of MCMC, while blue corresponds to the inference network trained. 37



1. Introduction

Often it is desirable to explain some process or find patterns in the data that is being observed. It may be interesting doing this to perform tasks such as predicting whether or not a picture is of our favorite animal, which would be a classification task. Or maybe we want to predict the temperature of an object given certain information, which would correspond to a regression task given its continuous nature. Independent of the task we wish to perform, we build models that leverage the available data at our disposal. The models seek to explain the process that we are interested in, for example, by assuming that the color of the pixels in our picture helps to identify which animal is present in it. A model can broadly be classified into two types: parametric and non-parametric. Parametric models have a known and fixed number of parameters and can include assumptions directly in the model, for example, assuming a linear relationship between certain variables. On the other hand, the non-parametric models, although its name may imply the opposite, do have parameters, but the number and form of these depend directly on the data that is analyzed. In this work, we focus on parametric models. One example of a parametric model would be a linear regression model, where the prediction is linear in the parameters of the model.

Once a model for the problem at hand is selected, the parameters needed to perform the desired task must be found. One approach to finding these parameters is, for example, a maximum likelihood approach [1], where the parameters are chosen such that they maximize the likelihood. This quantity corresponds to how probable the data is for a given value of the model parameters [1]. Maximizing the likelihood corresponds to finding the parameters that make the observed data most probable under the proposed model. One disadvantage of this approach is the fact that a single point estimate for the parameters of the model is obtained.

We consider the Bayesian approach to infer or learn the parameters of the model. In this approach, the probabilities represent a degree of belief or uncertainty that a certain event will occur. The central quantity that we are interested in computing in Bayesian Inference is the posterior distribution of the model parameters. The posterior corresponds to the belief about the possible parameter values after we update the prior knowledge with the observed data. This prior knowledge is expressed in the form of a prior distribution, which does not depend on the data. On the other hand, the knowledge about the data and how it relates to the parameters comes in the form of the likelihood which was described

above. These three concepts are related through Bayes Theorem [1], used in order to compute the posterior distribution:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}. \quad (1.1)$$

In the above equation θ corresponds to the model parameters, while D corresponds to the data. In turn, the prior distribution is expressed as $p(\theta)$, while the likelihood is expressed as $p(D|\theta)$, finally, the quantity $p(D)$ serves as a normalization constant that ensures that the posterior $p(\theta|D)$ remains a valid distribution. This quantity $p(D)$ is called the evidence or marginal likelihood and is computed as $p(D) = \int p(D|\theta)p(\theta)d\theta$.

The likelihood $p(D|\theta)$ is the quantity that is maximized when using the maximum likelihood approach to estimate the model parameters. On the other hand the prior distribution $p(\theta)$, expresses the beliefs or assumptions about the parameters before the data is observed. To illustrate this, consider a model where the age of an individual is used as a parameter, we can incorporate information about its age, for example, we can say that on average the age of the individual is between 20 and 30 by using a normal distribution with mean of 25 and an standard deviation of 5.

Note that the posterior obtained from the Bayesian approach differs from a maximum likelihood method in that the result is a distribution for the parameters of the model. This distribution provides more information about the uncertainty of the model which enables further analysis, for example, taking into account extreme parameter values, or making certain decisions based on the uncertainty of the model. The importance of this probabilistic approach and applications are discussed further in [2].

An important problem of the Bayesian methods is precisely the computation of the posterior, this turns out to be a difficult procedure because of the evidence $p(D)$ which can include an intractable integral or an exponential number of terms for more complex models. Recall that $p(D) = \int p(D|\theta)p(\theta)d\theta$, notice that the evidence considers all possible values for the parameters. This integral becomes intractable for many models of interest, for example, hierarchical models with non-linearities such as neural networks are almost always intractable.

One of the main approaches for performing inference corresponds to the Markov Chain Monte Carlo (MCMC) methods[3]. In MCMC approximate inference is performed by constructing a sequence of events known as a Markov Chain in such a way that we can arrive at the true posterior asymptotically. One of the problems that MCMC methods possess, is their running time, where complex models may take a long time, and do not possess a clear or unique stopping criterion for convergence [4]. This means that when using large datasets or complex models, MCMC methods do not scale.

A common alternative to MCMC methods is using Variational Inference[5] to approximate the posterior distribution. This family of methods casts inference as an optimization problem. Here, an approximate distribution q is proposed and an objective function is minimized to find the parameters of q , commonly the objective used is the Evidence Lower

Bound (ELBO). Variational Inference methods result in faster inference procedures when compared with MCMC. However, this results in a worse approximation to the posterior when compared with MCMC, since we are restricted by our choice of q .

Instead of learning the parameters of a model a single time as we have talked until now, consider a situation where we wish to do this procedure multiple times for different sets of observations. One example where one would like to do this is, a service where we would like to determine what each user would buy based on previous purchases. In this case the cost of using MCMC is even higher, given that we run the method in each instance of the problem. This results in spending a lot of time running the same procedure multiple times, only with different observations. On the other hand variational methods while faster, still suffer from needing to perform the inference for each new observation, even when the model remains the same.

This problem can be solved with Amortized Variational Inference [6], where instead of inferring the parameters ϕ of our approximation q_ϕ each time, we learn a function $g_w : x \rightarrow \phi$. Now the parameters ϕ of the approximation are computed with a function g with parameters w , meaning we now learn global parameters w , instead of local ones. This enables fast inference by simply using the learned function to compute the parameters of the approximated distribution in the following way: $q_{g(x)}(\theta)$. We propose to use Amortized variational inference to learn the approximate posterior $q_\phi(\theta)$ for a known and fixed parametric model $p(x|w)$, while using a deep neural network with weights w as $g_w(x)$. We present a comparison between MCMC and amortized variational inference with neural networks to infer the parameters of two parametric models. The models for which we compare both methods consists in a simple linear model of the form $ax + b$ and a simplified version of the model used in [7]. We evaluate both methods on synthetic data that we generate and describe later in section 3.1.

The general objective this work is the Bayesian estimation of the parameters of two models using Amortized Variational Inference with a deep neural network. We compare its performance with an MCMC algorithm on simulated data, from which we can compare the estimated parameters with the ground truth. The specific objectives of this work are the following:

- Define parametric models and the corresponding simulated data.
- Specify the MCMC algorithm to be used as the baseline for Bayesian parameter estimation.
- Description of the Amortized Variational Inference procedure.
- Describe the neural network architecture to be used in the Amortized Variational Inference procedure.
- Training of the neural network achitecture.

- Experiments and comparisons between the proposed Amortized Variational Inference approach and the baseline MCMC algorithm.

The rest of this work is structured the following way: we review the related work of Bayesian inference, including MCMC and variational inference including works which use Deep Learning. We describe both our proposed method of using Amortized Variational Inference and the data we generate to evaluate both the baseline and our proposed method. After that we proceed to the experiments and results of this work. Finally we discuss our obtained results and possible future work.



2. Bibliography discussion

In this section we first discuss briefly the problem of Bayesian inference, then we mention two general and widely used approaches to solve this: Markov Chain Monte Carlo and Variational inference, including their advantages and disadvantages. Finally, we conclude this section with a discussion of amortized inference, which corresponds to the method used in this work.

2.1 Bayesian Inference

We are interested in performing Bayesian inference over the parameters θ of a model. The central piece of this procedure is computing the posterior distribution $p(\theta|x)$ which expresses the distribution of parameters after we have observed the data x . Using the Bayes theorem we can express the posterior distribution as:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}, \text{ Where}$$
$$p(x) = \int p(x|\theta)p(\theta)d\theta.$$

As discussed previously, computing the exact posterior distribution may not be possible. For example, the previous expression for computing the posterior may not have an analytical solution at all. In other situations, the term $p(x)$, also called marginal likelihood or evidence, may not be tractable. This can be due to a variety of reasons, these include: an integral with no closed form solution, or a large number of parameters that make it impossible to perform enumeration.

2.2 Markov Chain Monte Carlo

One of the most popular approaches to performing Bayesian inference corresponds to the family of Markov Chain Monte Carlo methods. Broadly speaking these methods work by constructing a Markov Chain which, given an initial state and transition probabilities T , move from state i to $i + 1$ using T until the desired distribution is reached. For example, the posterior distribution over the model parameters which is what we are after.

More formally, a Markov Chain is a stochastic model defined over a series of random variables X_n , where the n -th random variable depends only on the previous step, that is:

$$p(x_n|x_{n-1}, x_{n-2}, \dots, x_1) = p(x_n|x_{n-1}).$$

This means that given an initial probability for x_0 , $p(x_0)$, we can arrive at the next state x_1 and so on using the transition probabilities denoted as T_i . A Markov Chain is said to be homogeneous if its transition probabilities T_i remain the same at every step. Additionally a distribution is invariant or stationary if, given a Markov Chain and transitions T , the distribution does not change for successive steps in the chain [3, 1].

We set up our Markov Chains to arrive at the distribution of interest, for example the posterior distribution of a model parameters. We also want our target distribution to be invariant given the Markov Chain. This way we can sample from this distribution as we take more steps in the chain, without changing the distribution. Another property desired for Markov Chains, is called ergodicity, this means that as $n \rightarrow \infty$ we approach the target distribution. This property ensures that a properly designed MCMC method will converge to the true distribution with enough steps, enabling exact estimation of the parameters of a model.

We briefly describe two MCMC algorithms: Metropolis-Hastings and Hamiltonian Monte Carlo. Both algorithms are related, and we employ a sampler based on Hamiltonian Monte Carlo as our baseline for comparison with the proposed solution to the problem.

2.2.1 Metropolis-Hastings

The first method which we discuss is the Metropolis-Hastings algorithm [8, 9]. This method works by using a proposal distribution q , from which at each step we draw a sample x^* , this sample is accepted as the next step of the chain with probability:

$$A_i(x^*, x^i) = \min \left(1, \frac{p(x^*)q_i(x^i|x^*)}{p(x^i)q_i(x^*|x^i)} \right). \quad (2.1)$$

We make two observations, first, if we reject the sample from the proposal distribution we consider the next step of the chain the same as the previous one. Second, our proposal distribution can be dependant on the current step, for example, we can choose q to be a normal distribution with mean at the current value x_i and with some predefined variance which will allow for exploration of the posterior at random.

2.2.2 Hamiltonian Monte Carlo

The second MCMC algorithm discussed is Hamiltonian Monte Carlo(HMC), also called Hybrid Monte Carlo [10]. This class of algorithms produce proposals for the Metropolis algorithm described previously. The proposals take into account gradient information, in particular, the gradient of the logarithm of the target probability. For this method,

we introduce additional "momentum" variables and express the "energy" of the system as a couple of differential equations. At every step of the Markov chain we simulate the dynamics of the system and accept the given candidate using the Metropolis Hastings proposal. For an introduction to the method, [11] provides a conceptual overview of HMC, from it's formulation to it's advantages and where it can prove useful.

One downside of HMC algorithms is the need to tune hyper-parameters. Since system being simulated in HMC needs to be integrated there exist parameters associated with the process. Two of the parameters involved are the step size ϵ and the number of steps L of the integrator. The No U-turn sampler(NUTS)[12] has been proposed as an extension of HMC, where the parameter L is adaptatively set. The NUTS sampler exhibits performance comparable with a hand-tuned HMC sampler, meaning that it can be used to perform inference over a large number of models without requiring as much tuning as other methods. This sampler is the default algorithm for inference in some probabilistic programming languages, for example in PyMC [13] and Stan [14].

Some work where HMC has been used successfully include [15], where HMC is used to perform Bayesian inference over neural networks. Meanwhile [16] shows the performance of HMC on a hierarchical model, and compare it with a Metropolis Hastings algorithm, showing that HMC has a better time with the model considering its high dimensionality.

2.2.3 Disadvantages

Some disadvantages of MCMC methods that have not been mentioned:

- While they converge to the desired distribution under appropriate conditions, this is on the limit when the number of steps approaches infinity. This poses a problem since we need to stop the method at some point and check for convergence. A review of convergence diagnostics is provided in [17]. One commonly used method used to assess convergence is to run multiple chains in parallel from different starting points. After the chains are done, we can, for example, observe the trace plot, which shows the sampled values at each step of the chain. With this trace plot of the chains we can verify if all the chains arrive at the same distribution.
- The samples are correlated since each step in the chain is dependent on the previous one. This problem can be solved by techniques such as thinning [18], where we only consider every k th sample in the chain, however, it may not be necessary to use thinning, as the use of the full chain may offer higher accuracy as discussed in [19] when reporting an estimate about the chain, such as the mean of the posterior.
- Related to the first problem mentioned: before the chain converges to the stationary distribution, the samples that are produced by the chain are not of the target distribution. Consider for example, a bad initial state which has low probability of being from the target distribution, as we move further along the chain the initial

samples are not representative of the distribution we desire. These samples are discarded, and called burn in or warm up samples.

Note that all the mentioned MCMC methods only work on a single problem instance. This means that we run MCMC methods on each single problem that we want to perform inference on. Even if they correspond to same model, only with different data.

If we have a large number of instances where we wish to perform inference, in addition with a complex enough model, the time spent on performing MCMC can be significant. This is due to the fact that it is not taking into account global information between the different instances of the problem.

2.3 Variational Inference

One way to perform faster inference when compared with MCMC is to use approximate inference methods, one family of methods corresponds to variational inference[20]. Here, an inference problem is transformed into an optimization one making it more efficient.

To perform variational inference we propose a distribution $q_\phi(\theta)$ to approximate the intractable posterior distribution over parameters $p(\theta|x)$. Note that we assume that this distribution has parameters ϕ . The goal is to find the parameters ϕ that make our approximation q_ϕ close to the intractable posterior. We can measure how close q_ϕ is to p using the Kullback-Leibler divergence:

$$D_{KL}(q_\phi(\theta|x)||p(\theta|x)) = \mathbb{E}_{q_\phi(\theta|x)} \left[\log \frac{q_\phi(\theta|x)}{p(\theta|x)} \right]. \quad (2.2)$$

The Kullback-Leibler divergence is also known as the *relative entropy* between both distributions. From an information theory perspective it represents how much information is lost when encoding a symbol using the distribution p , while the real one is q [1, 21].

Thus in variational inference we are interested in minimizing the Kullback-Leibler divergence between the proposed approximation and the true posterior. This quantity is only zero when $q_\phi(\theta|x) = p(\theta|x)$, that is, the approximation and the true posterior are the same. Our objective is finding the variational parameters ϕ that make our approximation close to the intractable posterior. For example, if our approximation q_ϕ was a normal distribution, we would find the mean and standard deviation such that the divergence is minimized. Note that we have not placed restrictions on q_ϕ , aside from assuming the existence of parameters ϕ , which means we can make q_ϕ as flexible as needed. Often this means trading speed vs quality of approximation. Note that the Kullback-Leibler divergence is asymmetric, therefore:

$$D_{KL}(q_\phi(\theta|x)||p(\theta|x)) \neq D_{KL}(p(\theta|x)||q_\phi(\theta|x)).$$

Consider that $q_\phi(\theta|x)$ is our variational approximation for which we want to find ϕ and $p(\theta|x)$ is the posterior distribution of the model parameters. When p is the first argument

of the divergence we call this the *forward* divergence. The forward divergence has the effect of allowing $q_\phi(\theta|x)$ to be large in sections where $p(\theta|x)$ is small. This happens because in order to minimize this divergence having a lower p is enough, ignoring the probability of q . The case where q is the first argument in the divergence is called the *reverse* divergence. It encourages $q_\phi(\theta|x)$ to have higher probability when $p(\theta|x)$ does and lower probability in cases where $p(\theta|x)$ is low. This means that in the case of p being a multimodal distribution, and q a unimodal approximation, q would match one of the modes in the reverse divergence. However, the forward divergence would spread q across p . The discussion for the difference between forward and backward divergence can be found in [1]. In the rest of this work, we use the reverse Kullback Leibler divergence (Eq. 2.2), as is commonly done in variational inference, where the approximation q_ϕ is the first argument.

Directly optimizing the divergence between q_ϕ and p is not possible, this is due to the fact that $p(\theta|x)$ is the quantity we wish to compute, and it contains the evidence $p(x)$ which as mentioned before, is often intractable. This is clearly seen if we start from Eq. (2.2) and expand the posterior distribution:

$$D_{KL}(q(\theta|\phi)||p(\theta|x)) = \mathbb{E}_{q_\phi} \left[\log \frac{q_\phi(\theta|x)}{p(\theta|x)} \right] \quad (2.3)$$

$$= \mathbb{E}_{q_\phi} [\log q_\phi(\theta|x)] - \mathbb{E}_{q_\phi} \left[\log \frac{p(x, \theta)}{p(x)} \right] \quad (2.4)$$

$$= \mathbb{E}_{q_\phi} [\log q_\phi(\theta|x)] - \mathbb{E}_{q_\phi} [\log p(x, \theta)] + \mathbb{E}_{q_\phi} [\log p(x)]. \quad (2.5)$$

Since we cannot optimize the divergence between the approximation and the posterior directly, variational inference methods instead optimize a lower bound. This is called the Evidence Lower Bound(ELBO), defined as:

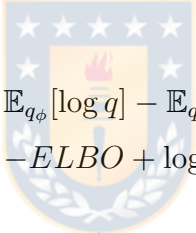
$$ELBO = \mathbb{E}_{q_\phi} [\log p(x|\theta)] - D_{KL}(q_\phi(\theta)||p(\theta)) \quad (2.6)$$

$$= \mathbb{E}_{q_\phi} \left[\log p(x|\theta) - \log \frac{q_\phi(\theta|x)}{p(\theta)} \right]. \quad (2.7)$$

In the ELBO, the first term is the expected value of the data likelihood when sampling θ from the approximate posterior. Maximizing this term encourages learning a model that is more likely to have generated the observed data. The second term is the negative KL divergence between the approximate posterior and the prior over the model parameters. Maximizing this term makes it so that the posterior distribution remains close to the prior and is sometimes referred to as a regularization term. To better understand why the ELBO is used we start from the evidence $p(x)$:

$$\begin{aligned}
p(x) &= \int p(x, \theta) d\theta \\
&= \int p(x, \theta) \frac{q_\phi(\theta)}{q_\phi(\theta)} d\theta \\
&= \mathbb{E}_{q_\phi} \left[\frac{p(x, \theta)}{q_\phi(\theta)} \right] \\
\log p(x) &= \log \mathbb{E}_q \left[\frac{p(x, \theta)}{q(\theta)} \right] \\
&\geq \mathbb{E}_{q_\phi} \left[\log \frac{p(x, \theta)}{q(\theta)} \right], \text{ Using Jensen's inequality} \\
&= \mathbb{E}_{q_\phi} [\log p(x|\theta)] + \mathbb{E}_q [\log p(\theta)] - \mathbb{E}_q [\log q(\theta)] \\
&= \mathbb{E}_{q_\phi} [\log p(x|\theta)] - D_{KL}(q(\theta)||p(\theta)) \\
&= \text{ELBO}.
\end{aligned}$$

Meaning the ELBO, as its name implies, is a lower bound on the evidence. If we look again at our previous formulation of the divergence between the approximation and the true posterior:



$$\begin{aligned}
D_{KL}(q_\phi(\theta)||p(\theta|x)) &= \mathbb{E}_{q_\phi} [\log q] - \mathbb{E}_{q_\phi} [\log p(x, \theta)] + \mathbb{E}_{q_\phi} [\log p(x)] \\
&= -\text{ELBO} + \log p(x).
\end{aligned}$$

We see that the divergence between the variational approximation and the true posterior can be minimized by means of maximizing the ELBO. Also, if we look at the relationship between the ELBO, $p(x)$ and $D_{KL}(q_\phi(\theta|x)||p(\theta|x))$ it becomes clear that the divergence is the exact distance that exists from the ELBO to $p(x)$. Another fact about this classical variational inference approach that we have not mentioned before is that for the optimization procedure of the ELBO, we often try to arrive at an analytical expression that can be optimized directly, so a simple $q_\phi(\theta|x)$ can help the optimization.

For a review of classic variational inference see [5]. Also [1] provides a view of variational inference in the context of Machine Learning and approximate inference. A more recent review of variational inference is done in [22], where methods that enable variational inference to scale to larger datasets, among other advances, are described. These methods include: Stochastic Variational Inference (SVI) [23], where we use noisy estimates of the gradient for the optimization procedure enabling even the use of mini batches of data for large datasets. Other methods described include Black Box Variational Inference [24], where a general approach for optimizing the ELBO is described, making it possible to perform inference even when the analytical solution to the optimization problem is not available, using estimates of the gradient. Finally, amortized variational inference is also explained, which corresponds to the technique we employ in this work and discuss next.

2.4 Amortized Variational Inference

So far we have discussed MCMC and Variational Inference, both of these methods work on an instance by instance basis. This means that when we want to perform inference on a large number of problems, only with different observations, we run the same method multiple times. For MCMC this implies monitoring convergence of all of these procedures and a higher total time, and, while variational inference is faster we note that neither MCMC or variational inference are exploiting global knowledge about the problem. Since we are solving the same problem multiple times we could use additional information to solve it faster.

This approach can be used to describe amortized inference, where we leverage existing information in solving the problem. For example, experiments performed and discussed in [6] show that multiple related inference problems are not, in fact, processed independently by human reasoning, and that the order of these problems can affect the result if we exploit the previous known knowledge. In particular, the experiments in [6] involve asking humans multiple queries based on information provided about a hierarchy of relationships. Their experiments were designed so that it was possible to infer the correct answer based on previous queries, and show that in cases where such previous information is available the performance improves.

In particular, since we are interested in performing fast inference multiple times, we talk about amortized variational inference as a method to accomplish this. Suppose that instead of having just data for a single instance or problem, we have multiple cases. For example, having multiple time series for which we wish to perform inference, instead of just a single one. Let $x_{i,j}$ denote the observations or data points available, with $i \in \{1 \dots M\}$ and $j \in \{1 \dots N\}$, where i represents the i -th process or case for which we wish to infer the parameters of a model, and we have N data points for each case. In the rest of this section we omit the index j , denoting by x_i all the data points of a single instance of the problem. Consider a parametric model f with parameters θ for which we wish to perform Bayesian inference on each x_i to obtain the posterior distribution over the parameters. Notice that now we have a set of parameters θ_i for each x_i that we wish to infer. Next, following variational methods, instead of computing directly the posterior $p(\theta_i|x_i)$, we use an approximate distribution $q_{\phi_i}(\theta_i|x_i)$, governed by parameters ϕ_i for each instance of the problem. In amortized inference we are interested in using a function $g_w : x_i \rightarrow \phi_i$, that is, a function with parameters w that maps the data points to the variational parameters of our approximation q_{ϕ_i} . This means that we can use $q_{g_w(x_i)}(\theta_i|x_i)$ as our approximation. Importantly, using amortized inference allows us to learn global parameters w that define the approximation function g , instead of local parameters ϕ_i for each instance of the problem, thus we only need to find w shared by all data points. The work done in [6] mentions amortized inference in probabilistic reasoning, suggesting that reusing sub queries improves performance, using amortized variational inference is related in learning

this common knowledge in the form of the parameters of g .

One advantage of amortized variational inference is that once we have g performing inference over a new observation is as simple as evaluating g and plugging it into our variational approximation $q_\phi(\theta|x)$. This is useful in situations where the number of times that we wish to perform inference is large and the model is complex enough that an approximation is useful.

An important part of using amortized variational inference is the function g that is used to amortize the variational parameters. This function must be flexible enough to learn the global information across the different instances. It is also useful if we are able to compute g faster than simply using variational inference on a problem.

Currently the preferred technique to learn the global representation used in amortized inference is via the use of Deep Learning, these methods directly learn features from the representation of the data[25]. This is done with multiple layers of abstraction using function composition. Deep Learning techniques have proven to obtain superior performance than traditional methods for tasks such as image classification[26], machine translation [27], generative models[28] and more.

The usage of neural networks for amortized variational inference proves useful both for their capability to learn complex functions, and also due to their ability to be accelerated on dedicated hardware such as GPUs. This means that we can perform fast inference once the model has been trained.

An example of a simple model where amortized variational inference is when learning the latent variables in the Variational Autoencoder(VAE) [29] [30]. The VAE is a generative model that seeks to generate data similar to the one it is being trained on. This model assumes the generative process depends on a set of latent variables. It uses two neural networks: a generative network that learns to generate data from a given prior distribution of latent variables and an inference or recognition network that performs amortized variational inference over the set of latent variables. These two networks are also called the decoder and encoder, respectively, due to similarities with autoencoder models. Once trained, the decoder takes samples from the prior distribution and provides examples similar to the data used to train the neural network, for example, generating images of digits. On the other hand the encoder is used for training the model, acting as the approximation function g of amortized variational inference, and mapping from the input to the set of variational parameters ϕ_i . Importantly, these works also describe an estimator known as the reparameterization trick. This estimator can produce lower variance estimates of the gradient for the optimization procedure [31].

Amortized inference is also mentioned in [32], where its usage is explored for probabilistic programming with experiments for different models such as a Gaussian Mixture Model, showing that amortized inference obtains performance comparable to that of a Mean field approximation. In [33] they amortize the inference procedure by learning an inverse to the probabilistic model, this is done by learning from prior or posterior samples.

However one difference between [33] and our work is the fact that we consider the problem of performing inference from scratch, learning the information accross different cases in the parameters of the approximation function g .

In [34] a recognition network is trained to perform amortized inference for the posterior of spikes in a signal, which is then used then for different generative models. They apply this work to inference the spikes in signals that serve to their generative procedure, and their results show similar results even to MCMC.

These methods also use Stochastic Variational Inference (SVI) [23], briefly mentioned before. SVI provides scalable variational inference by making updates in the variational parameters with subsamples of the data instead of using the whole data set at each step. This results in using noisy estimates of the gradient when optimization takes place. SVI enables fast variational inference with the gradient estimates that are computed in the optimization procedure for training neural networks.

To the best of our knowledge using amortized variational inference where the generative model is not a neural network has not been explored extensively. The authors of [34] explicitly mention the learning of a generative model, however, it is not the main focus of the work and as such, a more in depth comparison with methods such as MCMC is not provided. Details about the training procedure or difficulties of the amortized inference problem are not provided. In the next section, we describe the model implemented for addressing these issues.



3. Developed Model

In this section we describe the main problem that we solve in this work. To start with, we describe the parametric models that will be used in this work. Using this information we describe the data over which we will perform amortized variational inference and how it is generated. Finally, we will describe the inference procedure that we follow to infer the parameters of the parametric models that we use.

3.1 Data

To evaluate the use of amortized variational inference we make use of two parametric models for which we will infer the posterior distribution of the parameters. Since the objective of this work is to determine whether or not amortized inference serves as an alternative to MCMC, we make use of synthetic data that we generate from the parametric model of choice. We will be also adding normally distributed noise with zero mean and a standard deviation of σ .

The first parametric model that we consider is arguably the simplest and corresponds to a two parameters model:

$$f(x) = ax + b. \tag{3.1}$$

This model consists of parameters $\theta = \{a, b\}$, where the parameter a describes the slope and b the intercept. This model serves the purpose of evaluating if the inference procedure works at all. We expect that using amortized inference in this model results in a correct estimation of the parameters. We describe the generative procedure to create datasets that follow this model:

$$\begin{aligned} a &\sim \text{Uniform}(0, 10), \\ b &\sim \text{Uniform}(0, 10), \\ x &\sim \text{Uniform}(-1, 1), \\ f(x) &= ax + b, \\ y &\sim \mathcal{N}(f(x), \sigma). \end{aligned}$$

Note that we will use y to refer to the observed output of our model f when taking into account the noise present in the data, where we have defined distributions over the parameters as well as x . Note that here f both receives and outputs a scalar. The data that we generate consists of several points $\{x_{i,j}, y_{i,j}\}$ where the index i represents the object and j the observation or data point within the object. Additionally, the value of σ is specified when creating the simulated dataset over which we will perform inference. We provide results using different values of σ .

The second model corresponds to a simplified version of the supernova light curve model used in [7] as shown in equation 3.2. The model has parameters that allow it to have a smooth rise until a maximum is reached, with a smooth fall after this. This means the model can generate curves with shapes that mimic those found in supernovae. The model was used in [7] to fit supernova light curves due to its shape. The parameters of the model are a which acts as a scaling parameter, b which shifts the model, t_0 approximately defines where the curve begins rising and t_{rise} and t_{fall} that describe how quickly the curve rises and falls, respectively.

$$f(x) = a \frac{e^{-(x-t_0)/t_{fall}}}{1 + e^{-(x-t_0)/t_{rise}}} + b. \quad (3.2)$$

For this work we only use two of the five parameters of the model. These are t_{rise} and t_{fall} that are responsible for mainly controlling the shape of the resulting function. The rest of the parameters were set to $a = 1$, $b = 0$, $t_0 = 75$ in order to simplify the model to just two parameters, this results in:

$$f(x) = \frac{e^{-(x-75)/t_{rise}}}{1 + e^{-(x-75)/t_{fall}}}. \quad (3.3)$$

The generative procedure that we use to generate data that follows equation 3.3 is the following:

$$\begin{aligned} t_{rise} &\sim \text{Uniform}(0.2, 10), \\ t_{fall} &\sim \text{Uniform}(20, 50), \\ x &\sim \text{Uniform}(0, 150), \\ f(x) &= \frac{e^{-(x-75)/t_{fall}}}{1 + e^{-(x-75)/t_{rise}}}, \\ y &\sim \mathcal{N}(f(x), \sigma). \end{aligned}$$

Once again we assume that y has a mean equal to our parametric model with normally distributed noise σ that is set to a specific value at the moment of generating the dataset.

In the figure 3.1 we present examples from both generative models used to construct the datasets for this work. The examples on the left corresponds to the model of equation 3.1 with $\sigma = 0.1$, while the plot on the right considers examples from the model of equation

Datasets examples

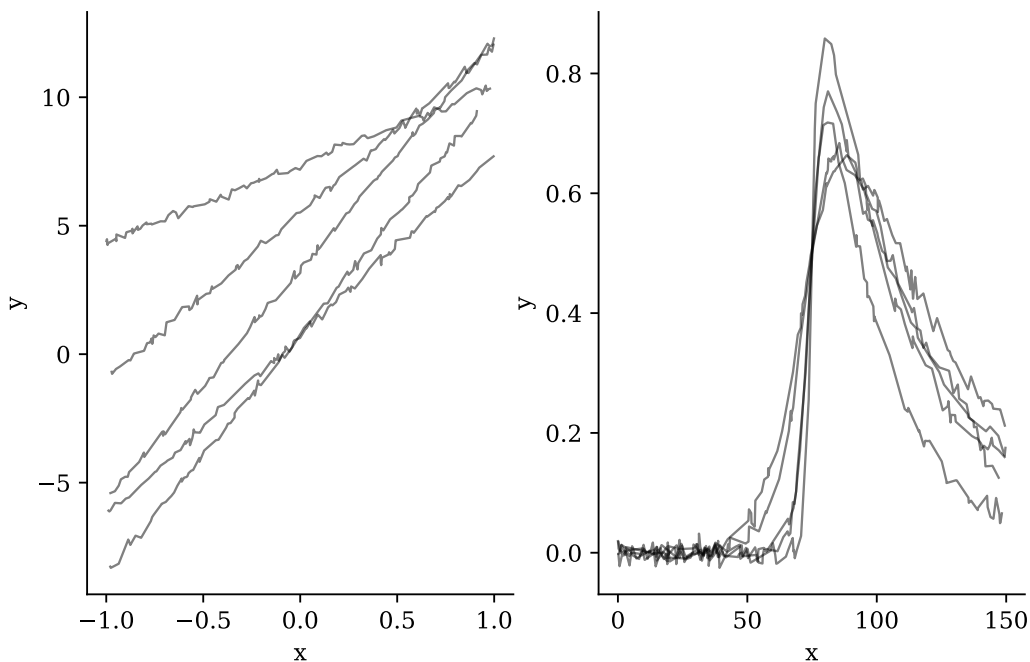


Figure 3.1: Examples from each dataset, on the left, we have a simple linear model where the added noise is $\sigma = 0.1$, on the right side we have our second model that provides shapes similar to supernovae, the noise injected in this case corresponds to $\sigma = 0.01$

3.3 with $\sigma = 0.01$.

3.2 Inference procedure

We are interested in performing Bayesian inference for a parametric model f with parameters θ , specifically for the case where we infer the parameters for multiple sets of observations. We denote as $\{x_{i,j}, y_{i,j}\}$ the set of observations, where the index $i \in \{1 \dots M\}$ denotes a single observation, for example, one time series. On the other hand $j \in \{1 \dots N\}$ corresponds to the points of a specific observation, for example, the individual points of the time series or curve. Here $y_{i,j} = f(x_{i,j})$, that is, the output of the parametric model f with parameters θ . Throughout this work we assume each curve or time series has the same number of points, that is, N remains constant. In the rest of this section x , denoted without indices, corresponds to a specific time series observations. As discussed previously, to do this we compute the posterior over the model parameters:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}.$$

While techniques such as MCMC converge to sampling from the exact posterior, they can be expensive in terms of time for complex models. This is especially true for our problem, where we wish to do inference multiple times. Instead of using MCMC methods,

we use variational inference, where we use $q_\phi(\theta)$ as an approximation to the posterior, and then maximize the ELBO(2.6). Furthermore, we parameterize the parameters ϕ of $q_\phi(\theta|x)$ with a neural network to learn global parameters instead of local ones for each x_i , these global parameters correspond to the weights of the neural network. This means that we amortize the computation, enabling fast inference with just a forward pass through the neural network. Notice, that we assume the parametric model f is known, and do not use a neural network to model $p(x|\theta)$ as it is typically done with deep generative models, such as the VAE. Instead our approach directly uses the parametric model f in the computation of $p(x|\theta)$, this approach would be similar to taking a VAE and replacing the decoder or generator network with f .

The literature around these methods involve the use of latent variables, commonly written as z for this kind of models. These latent variables consider both the model parameters θ , as well as other possible causes. Here we assume that the latent variables z consists only of the parameters θ .

In order to perform inference we maximize the ELBO. To do this we employ an algorithm such as Stochastic Gradient Descent (SGD) or some extension as done when training neural networks, this means we need to differentiate the ELBO. We take a step back and consider differentiating the expectation of a function $h(z)$ under a distribution $q_\phi(z)$ with respect to the ϕ . Additionally consider $z \sim q_\phi$, where z would be for example, the parameters θ . For this we can construct a naive estimator:

$$\begin{aligned}
 \nabla_\phi \mathbb{E}_{q_\phi}[h(z)] &= \nabla_\phi \int q_\phi(z)h(z)dz, \\
 &= \int \nabla_\phi q_\phi(z)h(z)dz, \\
 &= \int h(z)\nabla_\phi q_\phi(z)dz, \\
 &= \int h(z)\nabla_\phi q_\phi(z)\frac{q_\phi(z)}{q_\phi(z)}dz, \\
 &= \int h(z)q_\phi(z)\nabla_\phi \log q_\phi(z)dz, \\
 &= \mathbb{E}_{q_\phi}[h(z)\nabla_\phi \log q_\phi(z)].
 \end{aligned}$$

We can use monte carlo to estimate this quantity, but this estimator exhibits the problem that it can produce high variance estimates of the gradient, requiring many samples in order to bring the variance of the expectation down [35]. However, [29] and [30] propose using the reparameterization trick as a way to produce estimates with lower variance. Instead of sampling directly the latent variable z from q_ϕ , the reparameterization trick uses a function $g_\phi(\epsilon, x)$ as a transformation such that through the use of an auxiliary noise variable ϵ that comes from a distribution p , we can express z deterministically. More precisely instead of $z \sim q_\phi(\theta|x)$ we use the following:

$$z = g_\phi(\epsilon, x), \text{ with } \epsilon \sim p(\epsilon).$$

Where the sampling process is now done in the variable ϵ , and using the transformation g we can instead recover z . As an example consider $z \sim \mathcal{N}(\mu, \sigma)$, then $g_\phi(\epsilon, x) = \mu + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$. With this reparameterization we use the estimate $\mathbb{E}_{p(\epsilon)}[h(g_\phi(\epsilon, x))]$ instead of the original $\mathbb{E}_{q_\phi}[h(z)]$, thus we have:

$$\nabla_\phi \mathbb{E}_{q_\phi}[h(z)] = \nabla_\phi \mathbb{E}_{p(\epsilon)}[h(g_\phi(\epsilon, x))] \quad (3.4)$$

$$= \mathbb{E}_{p(\epsilon)}[\nabla_\phi h(g_\phi(\epsilon, x))]. \quad (3.5)$$

This last expression allows us to work with the expected value of the gradient of the function instead of differentiating its expected value, reducing the variance considerably. The reparameterized expectation can be approximated with a Monte Carlo estimate: $\frac{1}{L} \sum_{l=1}^L h(g_\phi(\epsilon, x))$ where L corresponds to the number of samples used to form the estimator. The use of Monte Carlo can also be applied to the equation 3.5 to obtain an estimate of the derivative.

A more in depth explanation of the requirements to use the reparameterization trick is mentioned in the original work, as well as in [36]. Where the authors propose an alternative way to reparameterize z , enabling the use of more distributions such as the gamma, via implicit differentiation of the transformation g .

As mentioned above, we can express the ELBO as an expectation (Eq. 2.7), this means the previous discussion on estimates of the gradient of expectations of functions can be applied to the ELBO. Furthermore, since we can decompose the ELBO into both a likelihood term and a Kullback-Leibler divergence (Eq. 2.2) term, depending on our choice of prior and posterior distributions we can use analytical expressions for the divergence. This way, a lower variance estimate of the gradient can be obtained, where we only need to use monte carlo to estimate the likelihood. We use this last information to express the ELBO estimate \mathcal{L} that we will employ for the optimization procedure:

$$\begin{aligned} ELBO &= \mathbb{E}_q[\log p(x|\theta)] - D_{KL}(q_\phi(\theta)||p(\theta)) \\ \mathcal{L}(\theta, \phi; x_i) &= \frac{1}{L} \sum_{i=1}^L \log p(x_i|z_{i,l}) - D_{KL}(q_\phi(\theta|x_i)||p(\theta)). \end{aligned}$$

Where the variable z_l has been sampled using the reparameterization trick described before, that is $z_{i,l} = g_\phi(\epsilon_l, x_i)$ with $\epsilon_l \sim p(\epsilon)$. The second term corresponding to the divergence is computed analytically when available, such as when we use both a normal prior and posterior. Additionally, we make use of mini batches of data for training, which results in a new estimate:

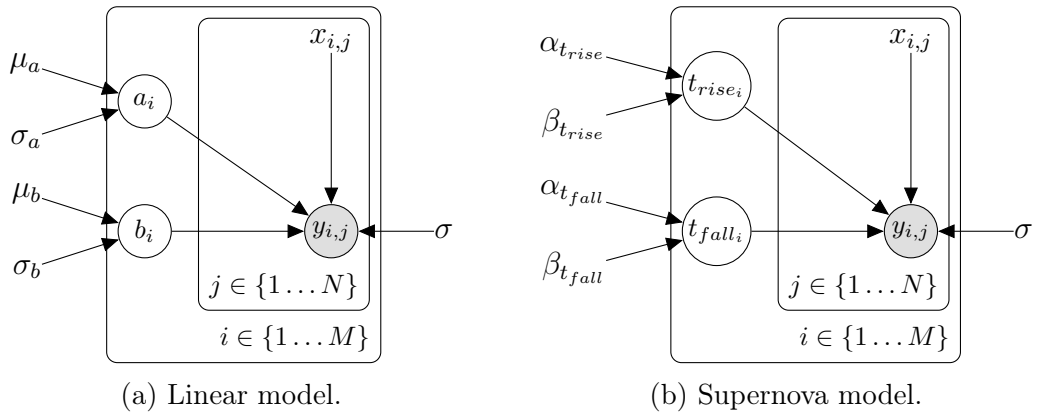


Figure 3.2: Probabilistic graphical model for both the linear and supernova models used in this work. We show the priors for the parameters of each model. Note that we consider the M curves and N data points of each one. The value of σ is assumed known and constant.

$$\mathcal{L}^M(\theta, \phi; \mathcal{X}) = \frac{N}{M} \sum_{i=1}^M \mathcal{L}(\theta, \phi; x_i). \quad (3.6)$$

We use \mathcal{L}^M to train the neural network such that the *ELBO* is optimized with a particular choice of $q_\phi(\theta)$.

So far we have described the general optimization of the ELBO, however we still need to define the components needed to perform inference. These are: the variational approximation $q_\phi(\theta|x)$, the likelihood function $p(x|\theta)$ and the prior $p(\theta)$.

We use a normal likelihood in both the linear parametric model and the supernova parametric model, where the mean is defined by the parametric model f using the parameters θ sampled from the variational approximation and with noise σ which we assume knowledge of.

As for the prior distributions used in this work we choose different priors for each model. For the linear model we place a normal distribution with zero mean and unit standard deviation for the parameters a and b . While in the supernova model we use gamma distributions as the prior to enforce positive parameters, more concretely, we use a $\Gamma(3.25, 0.63)$ for the t_{rise} parameter and $\Gamma(16.3, 0.47)$ for t_{fall} . The distributions are chosen such that they have an approximate mean of 5 and 35 with standard deviations of 5 and 15 for the parameters t_{rise} and t_{fall} respectively. Using gamma distributions provided more stable training than using $\log t_{rise}$ and $\log t_{fall}$ for obtaining positive parameters.

Figures 3.2a and 3.2b show an overview of the prior and likelihood that we use for inference in the models. The rectangles with N and M represent the fact that we have N data points for each curve, with a total of M curves. Each of the parameters has a prior shared by all M models, which correspond to normal priors for the linear model and gamma priors in the case of the supernova model. The likelihood is expressed in the generative process of y , where the parametric model f has parameters $\theta_i = \{a_i, b_i\}$ in the linear model, and $\theta_i = \{t_{rise}, t_{fall}\}$ for the supernova model. In addition, the model f

takes as input $x_{i,j}$ both cases to produce the mean of the normal distribution used in the likelihood. Finally, the standard deviation is set to the known σ of the dataset.

We illustrate the priors used for the parameters of the two models and their relationship with the likelihood through the parametric model f .

At this point knowing both the prior and the likelihood of the models we can employ a MCMC method to compute a posterior distribution over the parameters of the models. In the case of variational inference we need to specify a variational distribution $q_\phi(\theta|x)$ as an approximation to the posterior. We employ a normal distribution for the posterior of the parameters a and b in the linear model (Eq. 3.1) and for the supernova parametric model (Eq. 3.3) we instead use a gamma distribution on both t_{rise} and t_{fall} . The reasoning for using a gamma posterior is once again that we can force positive values of the parameters with the gamma distribution.

Since we use both a normal posterior and prior in the linear model, we can obtain an analytical expression for the divergence term of our objective function as described in Eq. 3.6. This means that we obtain lower variance estimates of the gradient than if we were to use monte carlo for this term, since we would only need to compute an expectation for the likelihood term. For the supernova model, given that we use gamma distributions both for the variational posterior and the prior, we use the approach described in [36]. Using this approach for the gamma distribution it is possible to construct a reparameterization that uses the derivative of the Cumulative Distribution Function. This once again enables lower variance gradients for the gamma distribution.

In order to perform the inference procedure that has been proposed, we must additionally describe the function that maps from observations to the variational parameters. We employ a 1d convolutional neural network with residual connections [37] depicted in figure 3.3. The use of convolutional networks for time series is mentioned in [38, 39], where architectures such as the one used exhibit a good baseline performance.

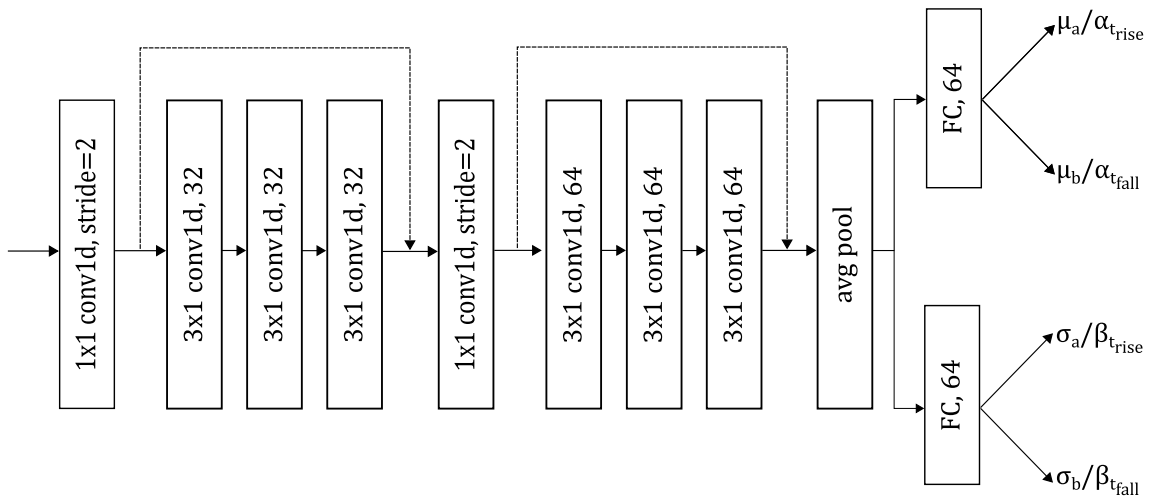


Figure 3.3: Architecture of the CNN used in this work. Residual connections are shown with dashed lines, 1×1 convolutions are used when increasing the number of channels. All convolutions are done with kernel of size 3 and stride 1, except on 1×1 convolutions, where the stride is 2.

This architecture consists of a series of 1d-convolutions followed by Batch Normalization[40] and ReLU activation functions. The 1x1 convolutions with stride 2 are used when we increase the number of channels from the initial 3, to 32 and later to 64, where the first 3 channels correspond to the observations in x , y and σ . After the use of 1x1 convolutions we use residual connections in the blocks that follow, depicted as dashed lines in the figure above. Finally at the end of the network we use global average pooling before the fully connected layers to predict the parameters of the variational approximation. The output of the network corresponds to the means μ and standard deviations σ for the normal posterior used in the first model (Eq. 3.1) and the parameters of rate α and concentration β of the gamma posterior used for the second model (Eq. 3.3).

To summarize, we are interested in performing Bayesian Inference and obtain the posterior distribution of the parameters of a given model f with parameters θ . In particular, we are interested in the case where we have multiple sets of observations, for example, multiple time series. In this case, techniques such as MCMC or Variational Inference would need to be run for each instance of the problem.

We propose using Amortized Variational Inference to obtain an approximation to the true posterior. To evaluate our method, we generate two artificial datasets from two parametric models and added normally distributed noise. We then defined the components needed to perform Bayesian Inference for these models: the likelihood and the prior. Additionally for variational inference we defined the variational distributions $q_\phi(\theta|x)$ used. Finally we described a neural network architecture used to amortize the parameters of the variational approximation, which enables learning a global set of parameters (the weights of the neural network) instead of local variational parameters for each observation.

4. Experiments and results

4.1 Experimental setup

To test our proposed model we generate the datasets described previously for different levels of noise σ . In total, we consider 10 different values for σ , for a total of 10 datasets to be used for each model. The values of σ for the linear parametric model (Eq. 3.1) start from 0.1 and increase by 0.1 until 1.0. For the supernova parametric model (Eq. 3.3) we start from $\sigma = 0.01$ and increase it by 0.01 until we arrive at a total of $\sigma = 0.1$.

For each of the 20 datasets, 10 for each model with different values of σ , we generate a total of $M = 10,000$ simulated time series where each one has a total of $N = 100$ points. From these 10,000 we set aside 200 curves for testing, where the 9800 remaining are split into 9000 for training and 800 for validation of the neural network. The datasets that we generate are similar to the ones shown in figure 3.1, only that we consider multiple values of σ .

We train a neural network for each of the 20 datasets. As input to the neural network we use x_i, y_i, σ_i , where σ_i is the noise that is present on each point, note that the model would be able to work using only x_i, y_i . The input x_i, y_i, σ_i is used such that each one is a channel in the convolutional neural network. For the implementation of the neural network we use the PyTorch [41] library.

We follow the same training procedure for all datasets: we train the neural network with Adam [42] with a weight decay of 10^{-4} as described in [43], using a learning rate of 10^{-4} , and parameters $(\beta_1, \beta_2) = (0.9, 0.999)$ for the optimizer. We train all neural networks for a total of 300 epochs with early stopping, where if the ELBO over the validation data has not improved for 10 epochs we stop the training. We use a batch size of 128 for all of the neural networks trained. Additionally, we consider $L = 1$ for the monte carlo estimate of the likelihood in Eq. 3.6. Finally, before initiating the training we initialize the parameters of the neural network according to [44].

For the supernova parametric model (Eq. 3.3), we improve stability when training by using the natural logarithm of the model and the logarithm of the sum of exponentials, also called the logsumexp trick (Eq. 4.1), to improve numerical stability. With this we can avoid overflow in the computation of sum of the exponentials before taking the logarithm. After that we can use an exponential function to recover the original desired output.

$$\begin{aligned} \text{logsumexp}(x_1, \dots, x_n) &= \log \sum_{i=1}^n \exp(x_i), \\ &= x^* + \log \sum_{i=1}^n \exp(x_i - x^*), \end{aligned} \quad (4.1)$$

Where $x^* = \max_{1 \leq i \leq n} \{x_i\}$.

$$f(x) = \exp(-\text{logsumexp}(\frac{(x-75)}{t_{fall}}, \frac{(x-75)}{t_{fall}} - \frac{(x-75)}{t_{rise}})). \quad (4.2)$$

Additionally we normalize the reconstruction of the neural network, that is, after we sample the parameters and reconstruct y_i using t_{rise} and t_{fall} sampled from the posterior, we perform min-max normalization, where we set the new value of $f(x)$ to:

$$\begin{aligned} h &= f(x) \\ f(\bar{x}) &= \frac{h - \min(h)}{\max(h) - \min(h)} \end{aligned}$$

This later normalization step allow us to train the model which has exponentials more easily, however, note that in modifying this value the noise that is used in the likelihood does change, we do not take this change into account and instead use the original σ value of the dataset. This normalization is used only for the parametric supernova model and not in the case of the first one, corresponding to a simple linear relationship.

After training we test the neural network on the remaining 200 instances of the corresponding dataset. We also evaluate the inference of MCMC to serve as our baseline, in particular, we use the NUTS sampler from Stan [14] to fit the corresponding model. With MCMC we can compare the quality of the variational approximation with respect to a more precise method where the sampler is run on each instance of the model. Note that this sampler also uses gradient information in order to sample from the posterior distribution.

For MCMC we use a total of 4 parallel chains, with a burn-in or warm up of 2000 samples, a total of 4000 samples per chain with thinning of 1 which means we skip 1 sample in the chain. With this choice of parameters for the sampler we end up with a total of 2000 samples per chain or 8000 in total. Importantly note that the use of normalization for the supernova model is exclusive to the neural network procedure used for inference, we do not employ said normalization for the MCMC sampler.

We measure the time taken by Amortized Variational Inference to train the neural network a total of five times. For this purpose, the network is trained for the total of 300 epochs without early stopping to measure a worst case scenario. We use the supernova parametric model with noise added of $\sigma = 0.05$ for measuring time. The supernova model

is used since it has higher computational cost when compared to the linear model. We measure the time that it takes the neural network and MCMC in estimating the model parameters for the 200 curves used for testing. The time taken to complete inference on the testing data is also measured five times.



4.2 Results

The average time it took the neural network to train with 300 epochs was 2187.24 ± 32.54 seconds. Inference on the 200 testing curves took the neural network an average time of 2.27 ± 1.07 seconds. On the other hand MCMC took an average time of 2047.17 ± 140.41 seconds on the same 200 curves used for testing and measuring time.

In order to demonstrate that the MCMC sampler is run until convergence we present trace plots on three examples from each model. We consider $\sigma = 0.5$ for the examples of the linear model and $\sigma = 0.05$ for the supernova model.

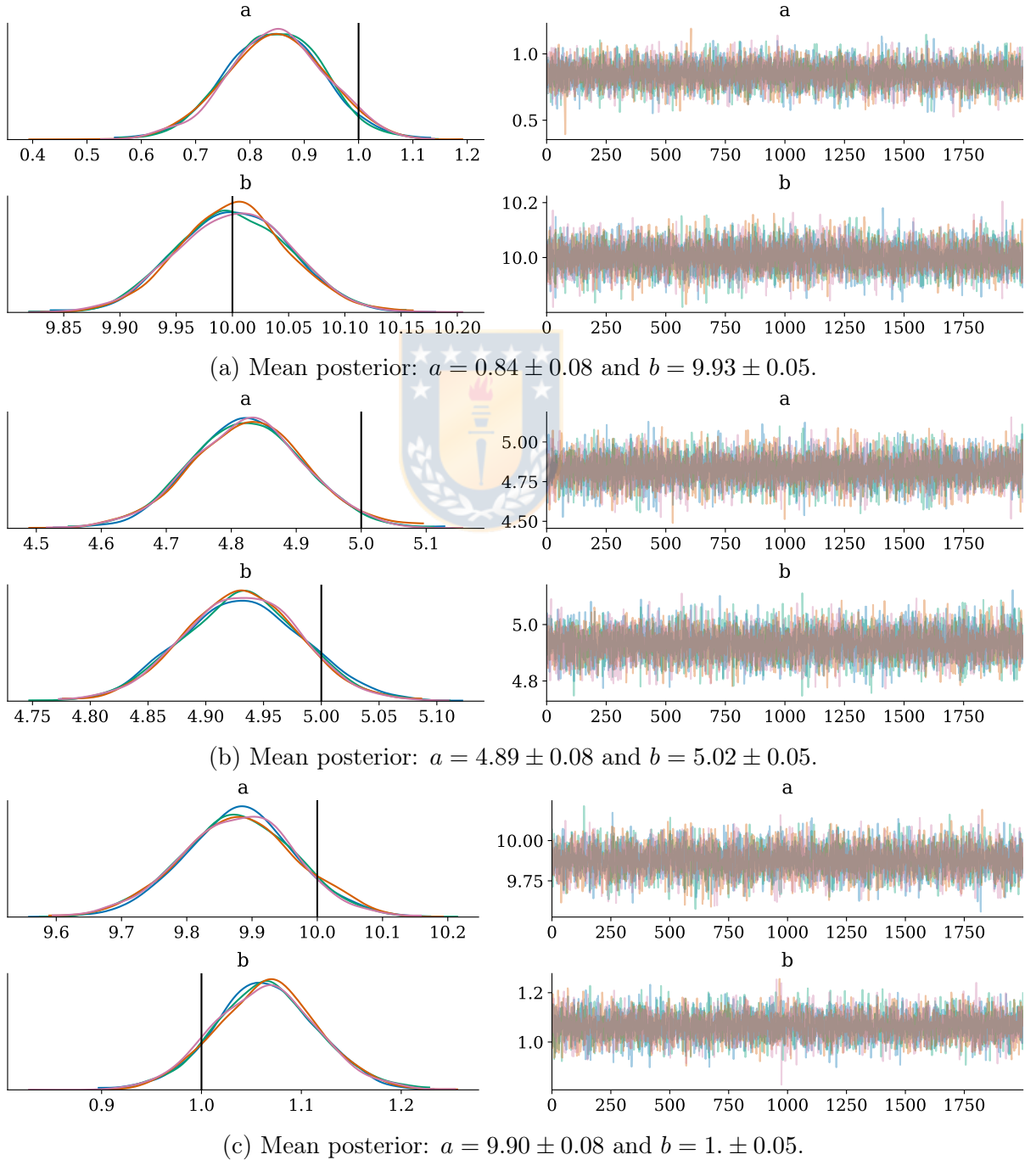


Figure 4.1: Trace plots for three examples of the linear model. The vertical lines represent the true parameters that were used to generate the data. All examples use $\sigma = 0.5$.

In figure 4.1 we show the three trace plots corresponding to the linear parametric model. We see that we do not recover always the original parameters exactly. In the first trace plot of figure 4.1 we get a mean value of a and b of 0.84 and 9.93 respectively, where the true values are 1. and 10., resulting in an error of 0.16 and 0.07 if using the mean of the posterior distribution. For the second trace plot we obtain an error of 0.11 for the parameters a and 0.02 for the parameter b when using the mean of 4.89 and 5.02. Finally, in the last example we get an error between the true value and the mean of the posterior of 0.10 for the parameter a and 0 for the parameter b .

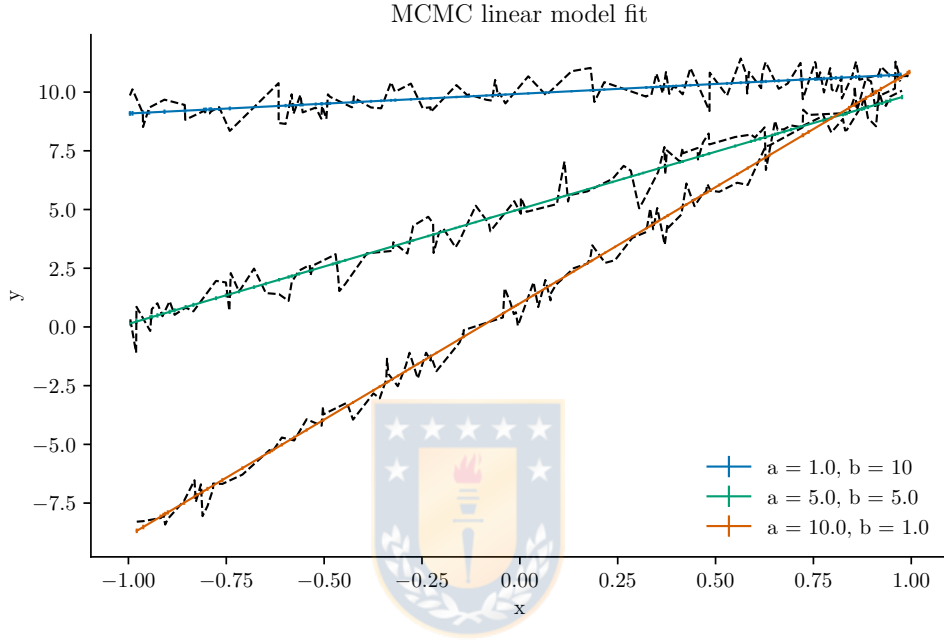


Figure 4.2: Mean fit when sampling parameters a and b from the posterior distributions obtained by MCMC. Data corresponds to the one used in figure 4.1.

In addition to the trace plots provided for the linear model, we also plot the mean value of the model when sampling the parameters a and b from the posterior distribution. This is shown in figure 4.2, where we see that each fit is close to the noisy observations. We measure the error between observations and the mean fit using the Mean Absolute Error (MAE), shown in equation 4.3. The Mean Absolute Error represents the average error between the predictions and the true values. The MAE we obtain for the three examples are 0.41 for the first trace plot, 0.44 for the second and 0.38 for the third. All three examples have errors that are within the added noise that was used in the data generation process, corresponding to $\sigma = 0.5$ in this example.

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (4.3)$$

For the supernova parametric model we show the three example trace plots in figure 4.3. As with the previous linear model, we do not recover exactly the true parameters, depicted with the vertical line. For the first trace plot we have a mean posterior of

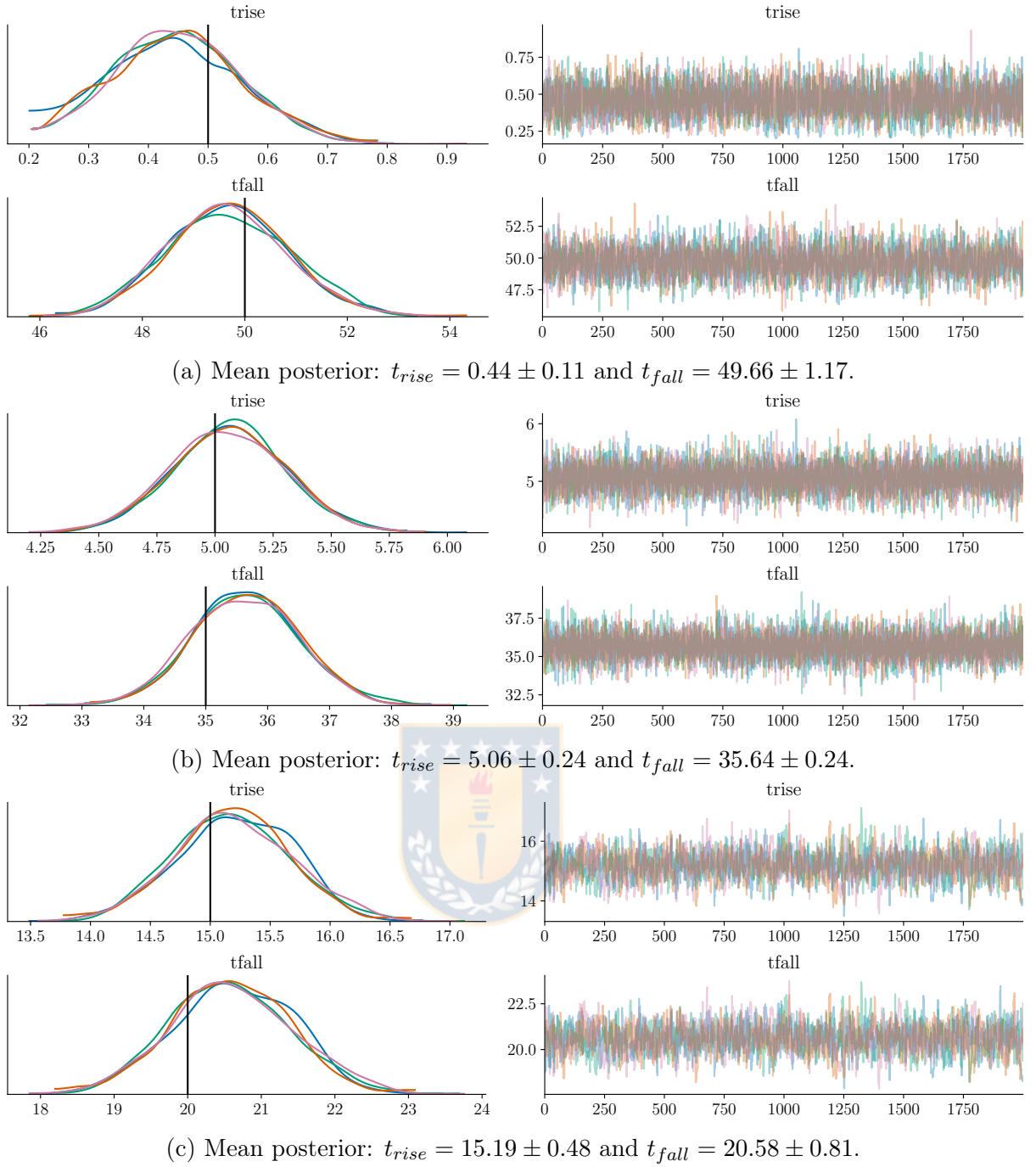


Figure 4.3: Trace plots for three examples of the supernova parametric model. The vertical lines represent the true parameters that we used to generate the data. All examples use $\sigma = 0.05$.

$t_{rise} = 0.44 \pm 0.11$ and $t_{fall} = 49.66 \pm 1.17$ which results in an error of 0.06 and 0.34 respectively when using the mean value. For the second trace plot the mean of the posterior parameters is $t_{rise} = 5.06 \pm 0.24$ and $t_{fall} = 35.64 \pm 0.24$ with an error of 0.06 and 0.64 from the true parameters of 5. for t_{rise} and $t_{fall} = 35.64$. Finally, the last trace plot shows a mean posterior of $t_{rise} = 15.19 \pm 0.48$ and $t_{fall} = 20.58 \pm 0.81$ with an error of 0.19 and 0.81.

In figure 4.4 we show the mean of the parametric supernova model when the parameters t_{rise} and t_{fall} are sampled from the posterior distributions. We obtain a MAE of 0.04

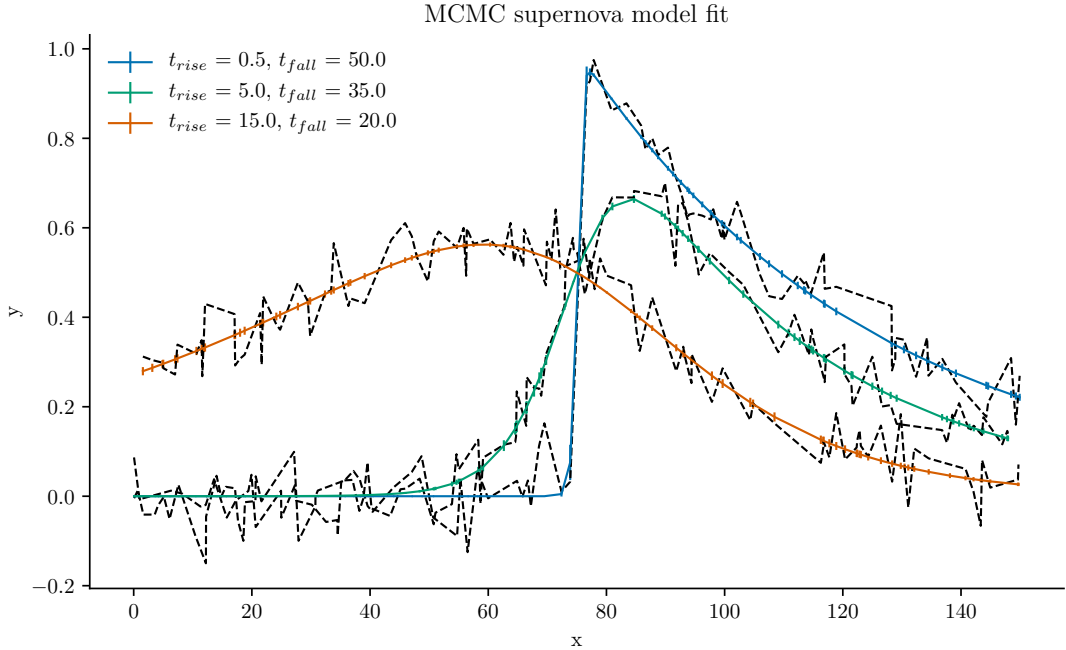


Figure 4.4: Mean fit when sampling parameters t_{rise} and t_{fall} from the posterior distributions obtained by MCMC. Data corresponds to the one used in figure 4.3.

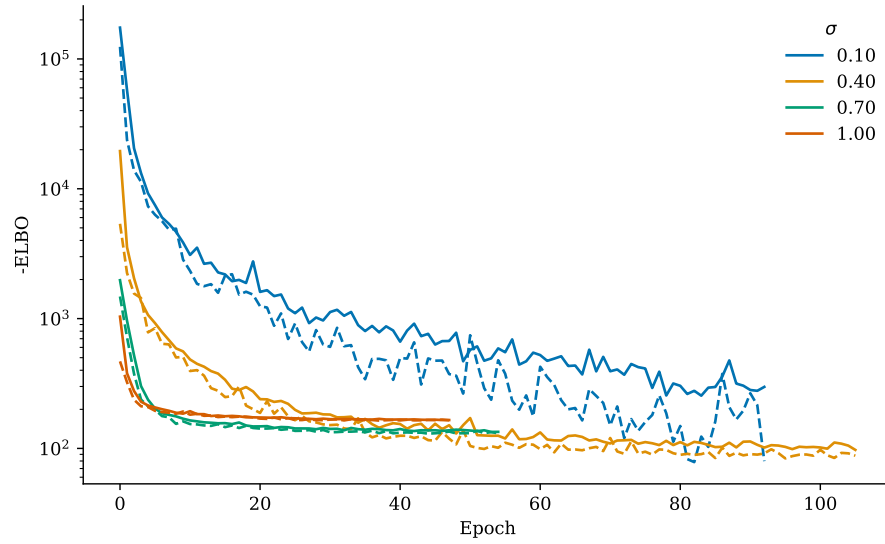
across all fits, this shows that even though we do not recover the exact parameters in the posterior, we obtain a good fit of the model to the data.

For the neural network we present the training curves information for both models in figure 4.5 with some levels of sigma. We show that the neural network has been trained until convergence, even considering the use of early stopping.

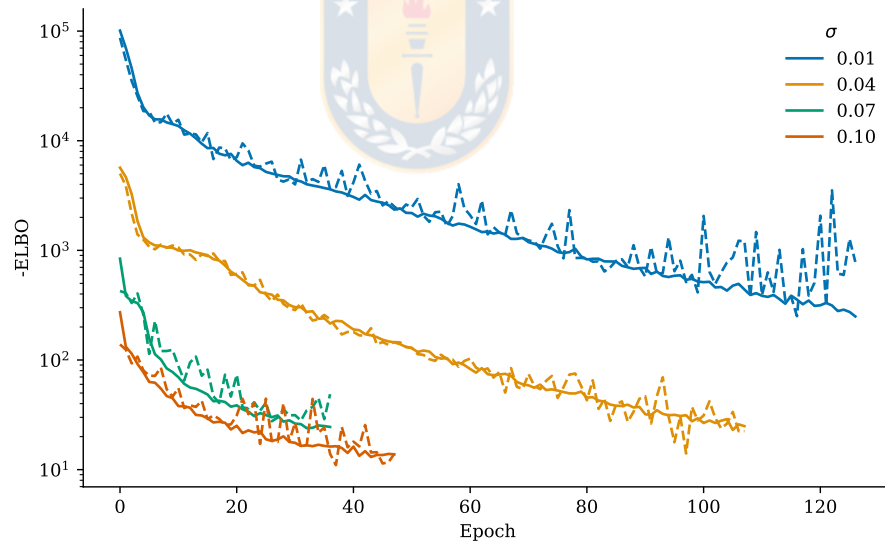
To show the results on the test data we plot in figure 4.6 the true parameter values on the x-axis and the mean and standard deviation from the posterior on the y-axis both by our method and by MCMC. We show that MCMC is more precise in recovering the true parameters that generated the data, however, note that each MCMC prediction is the result of running the MCMC sampler again, while the amortized approximation just performs a single forward pass through the neural network. The results that we present in figure 4.6 correspond to the lowest amount of noise present in the generated datasets, with $\sigma = 0.1$ for the linear model and $\sigma = 0.01$ for the supernova one.

We see that for the parameters of the linear model both the neural network and MCMC are able to recover the original parameters. However, for the supernova parametric model the proposed inference procedure fails to perform as well as MCMC. This is seen more clearly in figure 4.6b for higher values of the parameter t_{fall} , where the inference network produces higher estimates. This could be due to the normalization procedure that we use in this case, since we do not take into account normalization of the noise of the dataset. Finally, note that the proposed model produces estimates with higher error or standard deviation, meaning that we are less certain about the predicted value.

Since the performance of the neural network is expected to diminish as we increase the noise in the dataset we also present results for when $\sigma = 0.9$ and $\sigma = 0.09$ for the



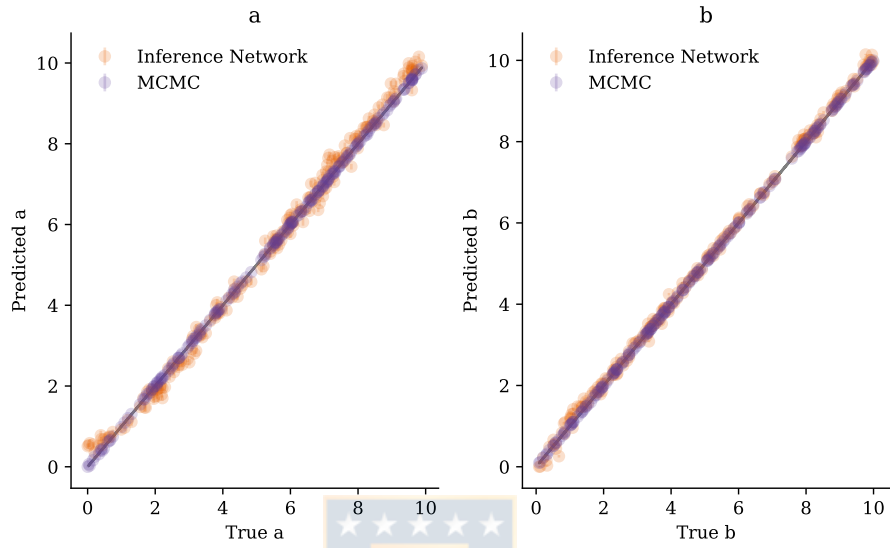
(a) Linear parametric model training curve.



(b) Supernova parametric model training curve.

Figure 4.5: Training curves for both models for different levels of noise σ in the dataset, the solid lines represent the training data while the dashed lines corresponds to the validation data.

(a) Results for the parameters a and b of the linear model



(b) Results for the parameters t_{rise} and t_{fall} of the supernova model

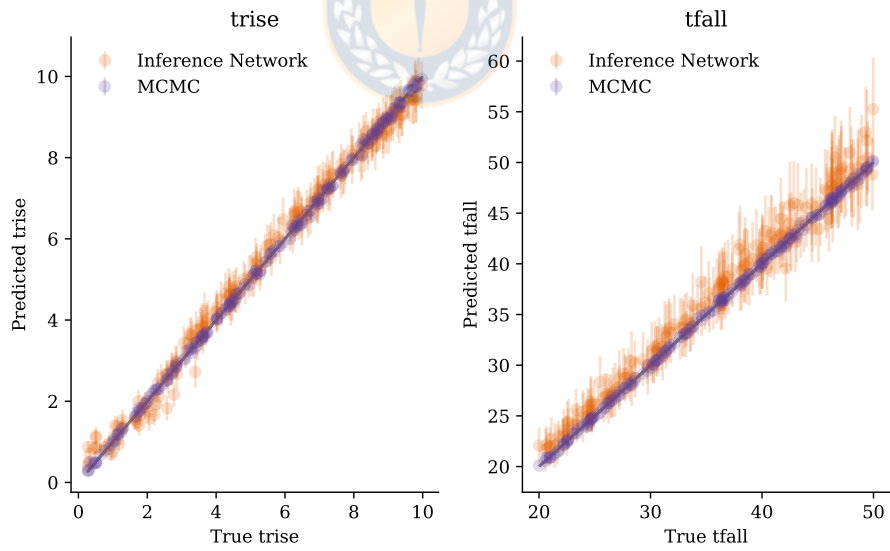


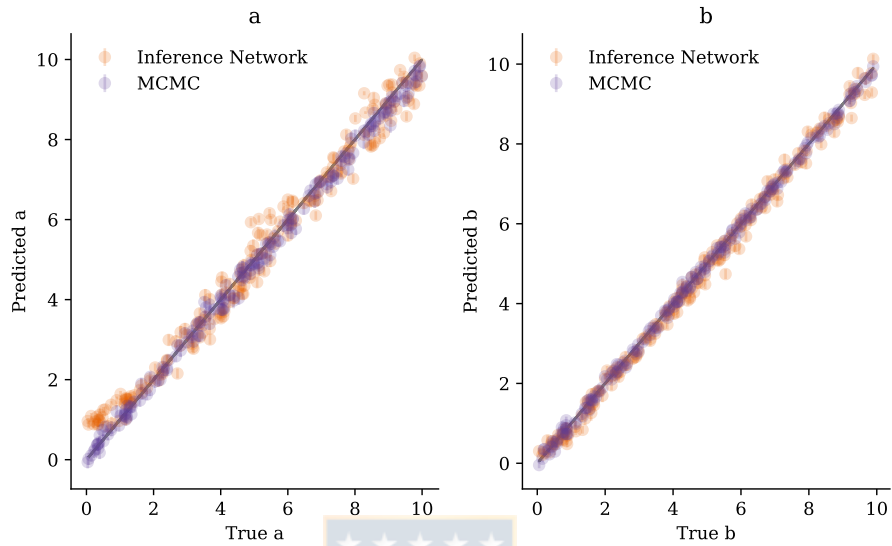
Figure 4.6: Results on the test set for the inference network and MCMC. In (a) the dataset has $\sigma = 0.1$, while the supernova model depicted in (b) has $\sigma = 0.01$.

linear and supernova models respectively. This is shown in figure 4.7 where we see the inference network exhibit notable worst performance in the supernova parametric model with parameters t_{rise} and t_{fall} . It can be seen that the mean prediction from the posterior is higher than the true value consistently, however, the deviation in these predictions is higher as well.

In order to show how the standard deviation of the predicted parameters behaves as we increase the amount of noise in the dataset, we plot the noise σ in the dataset versus the mean and standard deviation of the error in Figure 4.8. We would expect MCMC to scale linearly as we increase σ in the dataset and for the inference network to not scale as good. The results show that MCMC behaves as predicted, with the inference network being erratic on the results from the linear model in Figure 4.8a , with the performance being as expected for the supernova model parameters in Figure 4.8b.



(a) Results for the parameters a and b of the linear model



(b) Results for the parameters t_{rise} and t_{fall} of the supernova model

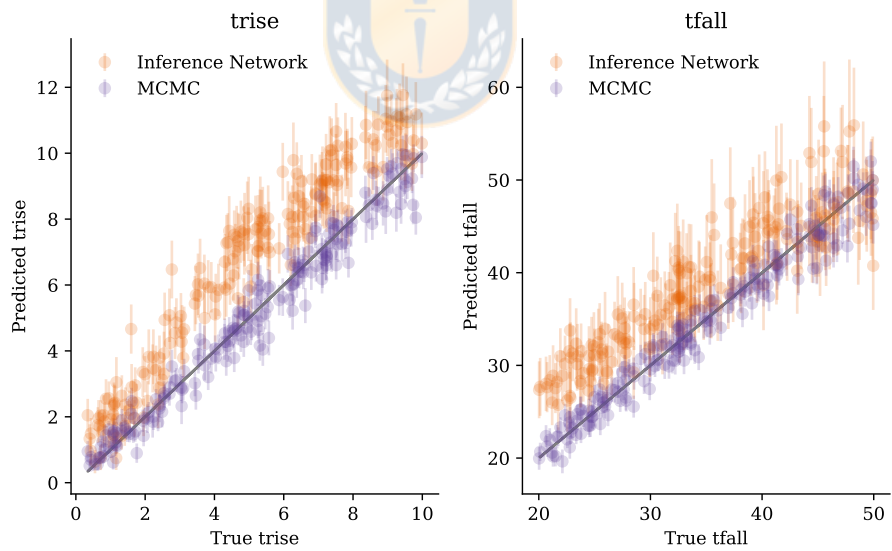


Figure 4.7: Results on the test set for the inference network and MCMC, for a corresponding to the linear model the dataset has $\sigma = 0.9$, while for the supernova model depicted in b the noise corresponds to $\sigma = 0.09$.

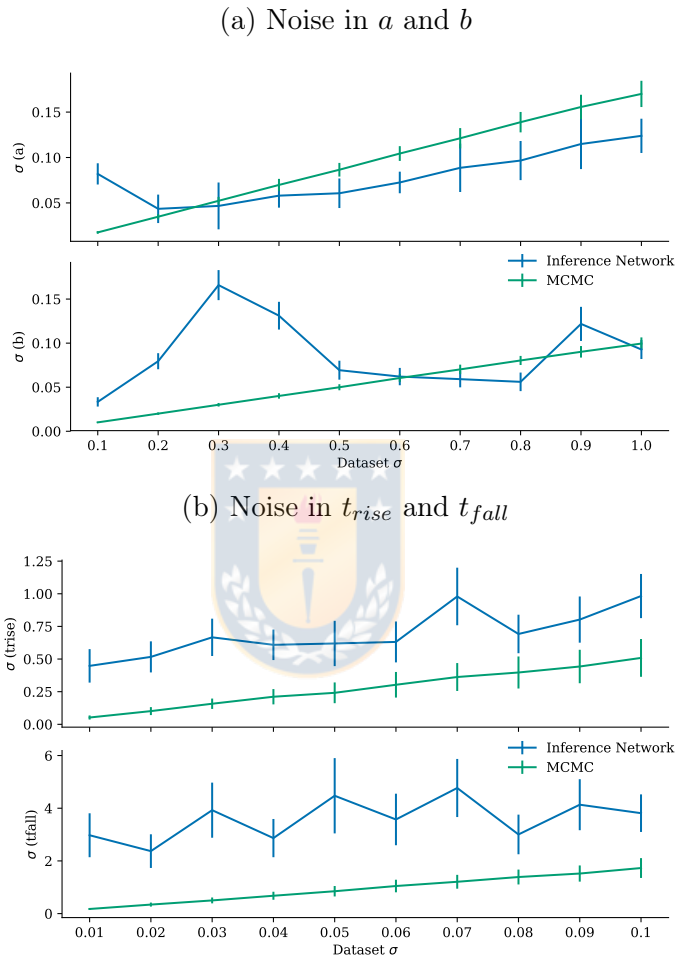


Figure 4.8: Noise in the parameters of the models as the noise σ in the dataset increases, in green we see the behaviour of MCMC, while blue corresponds to the inference network trained.

4.3 Discussion

The results in terms of execution time show that when performing inference the neural network is faster than MCMC. However, there is a need to perform a training step previously. This training step can be as slow as performing MCMC on the test dataset but, the training step has considerably more data: 9000 examples for training and 600 for validation versus 200 for testing. Consider two factors that make MCMC slower for performing inference in this situation. First, we are dealing with 4 parallel chains for each of the 200 curves with a work done on CPU, this means we need a high core count to distribute the work. Second, MCMC is inherently a sequential algorithm, where each step depends on the previous one. On the other hand, the neural network can easily be parallelized on GPUs to achieve lower times when compared with MCMC, as well as avoiding the sequential nature of MCMC. The neural network can be trained in less epochs or with better hardware resulting in less time overall. Additionally, one benefit of the neural network is that once trained the inference time is almost a hundred times faster than MCMC.

Our results show that Amortized Variational Inference is capable of obtaining comparable results to a MCMC sampler when the model is simple like the parametric linear model (Eq. 3.1). The results show this is the case when the noise is as small as $\sigma = 0.1$ or as large as $\sigma = 0.9$ as demonstrated in the figures 4.6 and 4.7.

As for the slightly more complex supernova parametric model (Eq. 3.3) which contains exponential functions, the performance of the amortized variational inference procedure is close to MCMC when the noise is small as seen in Figure 4.6b. However, as the noise in the dataset increases we see the performance degrades (see Figure 4.7b), where we are far from the true parameter value, while MCMC still manages to remain close to the line that represents the true parameters.

The good performance of MCMC is to be expected, since we run an individual algorithm for each instance of the dataset. With MCMC we recover the true parameters from the posterior distribution, even when the noise is relatively high, as shown in figures 4.1 and 4.3, where $\sigma = 0.5$ and $\sigma = 0.05$ for each model respectively. Even though MCMC does not recover exactly the parameters of the supernova model, we see that it may be due simply to the noise, since the mean fit when sampling from the posterior results in an acceptable reconstruction with $\text{MAE} = 0.04$ in the three examples that are provided (see Figure 4.4).

Although our proposed inference procedure does not manage to recover the true parameters in the supernova model, it exhibits high variance in its predicted parameters. This high variance can be interpreted as the model having low confidence in the prediction that is making. We believe that the poor performance may be due to the use of normalization (Eq. 4.1) on this model which may affect the noise used in the computation. However, removing this normalization leads to a less stable training, due to exploding

Table 4.1: -ELBO, negative log likelihood and divergence on the dataset corresponding to the linear parametric model as σ increases.

σ	-ELBO	-LL	D_{KL}
0.1	79.74	60.21	19.54
0.2	58.12	39.79	18.32
0.3	97.69	81.21	16.48
0.4	87.90	69.68	18.22
0.5	101.21	82.32	18.90
0.6	116.88	97.61	19.27
0.7	130.78	110.30	20.48
0.8	150.83	122.05	18.77
0.9	153.08	134.94	18.13
1.0	164.72	145.52	19.20

gradients when we perform backpropagation.

As for the results that show how the algorithms behave when the noise in the respective dataset increases (Figure 4.8). MCMC behaves as we expect, seeing an increase proportional to the increase in σ in every parameter. On the other hand, the inference network presents an erratic behaviour for the parameters of the linear model a and b , where we see an unexpected spike around $\sigma = 0.3$, while for t_{rise} and t_{fall} the rise in noise was expected to be higher, considering that it tends to estimate higher values for the parameters when the noise is near $\sigma = 0.9$.

In table 4.1 we explicitly show the validation -ELBO (Eq. 2.6, including both the expected negative log likelihood term and Kullback-Leibler divergence term as σ increases for the linear model. We see that for $\sigma = 0.3$ the models experience a sudden jump in the ELBO, and that this is caused by a reduction in the likelihood value. This could mean that the neural network fails to learn the mapping from the data to the variational parameters that correctly reconstructs the observed data.

5. Conclusion

We have used amortized variational inference for two parametric models, the first model being a linear one (Eq. 3.1), while the second model corresponds to a supernova parametric model (Eq. 3.3) and compared their performances with an MCMC method. We evaluate both our proposed method and MCMC on synthetic data, generated using the two parametric models. We provide results when different amounts of noise are added to the data, allowing us to compare the performance of both algorithms in presence of different levels of noise. We see how both methods behave in presence of more noise in the data, when the model is simple or slightly more complex.

Our work suggests that for simple models we are able to estimate the posterior of the parameters as well as MCMC, even as we increase the amount of noise present in the data. However, for slightly complex models the gap between the two compared methods increases. This may be due to the use of normalization in the supernova model, where if the noise is high our method fails to recover the original parameters that were used to generate the data. On the other hand, our method exhibits a high uncertainty in the predicted value, indicating that it is not confident in its prediction when the noise is high.

This work shows promising results that more complex models may be approximated correctly with more complex techniques. The implemented method is one of the simplest forms to perform amortized inference and does not differ significantly from the works of [29] and [30].

Future work regarding the use of amortized variational inference include:

- Performing inference for models over real data. One example where performing Bayesian inference multiple times may be desired is in the analysis of multiple time series, where we wish to fit a model for each time series and obtain estimates of the parameters for each one.
- Inference for more complex models, these could be either in the number of parameters or in the computation of the model. The former is useful for more expressive models or models where we have a high number of parameters for which we wish to perform Bayesian inference, while the latter is interesting in that we could have a model which itself has a high computational cost which could be avoided once we train the inference network.
- Different neural network architectures, in this work we have use a 1d convolutional

neural network, however for other data types we may wish to consider another architecture, for example, in time series the use of recurrent neural networks can better model time dependencies.

- More powerful posterior approximations, we have used a simple posterior approximation scheme that is not flexible enough for more complex posterior distributions such as multi modal ones. Using more expressive posterior approximations would improve the performance of the model. These could be in the form of better prior distributions [45] or better posteriors either via a tighter bound on the ELBO[46] or with flexible distributions [47].



Glossary

CNN Convolutional Neural Network.

ELBO Evidence Lower Bound.

HMC Hamiltonian Monte Carlo.

MAE Mean Absolute Error.

MCMC Markov Chain Monte Carlo.

NUTS No U-Turn Sampler.

ReLU Rectified Linear Unit.

SGD Stochastic Gradient Descent.

SVI Stochastic Variational Inference.

VAE Variational Auto Encoder.



Bibliography

- [1] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics). Berlin, Heidelberg: Springer-Verlag, 2006.
- [2] Z. Ghahramani, “Probabilistic machine learning and artificial intelligence,” Nature, vol. 521, no. 7553, p. 452, 2015.
- [3] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, Handbook of markov chain monte carlo. CRC press, 2011.
- [4] S. P. Brooks and G. O. Roberts, “Assessing convergence of markov chain monte carlo algorithms,” Statistics and Computing, vol. 8, no. 4, pp. 319–335, 1998.
- [5] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, “Variational inference: A review for statisticians,” Journal of the American Statistical Association, vol. 112, no. 518, pp. 859–877, 2017.
- [6] S. Gershman and N. Goodman, “Amortized inference in probabilistic reasoning,” in Proceedings of the annual meeting of the cognitive science society, vol. 36, 2014.
- [7] Bazin, G., Palanque-Delabrouille, N., Rich, J., Ruhlmann-Kleider, V., Aubourg, E., Le Guillou, L., Astier, P., Balland, C., Basa, S., Carlberg, R. G., Conley, A., Fouchez, D., Guy, J., Hardin, D., Hook, I. M., Howell, D. A., Pain, R., Perrett, K., Pritchett, C. J., Regnault, N., Sullivan, M., Antilogus, P., Arsenijevic, V., Baumont, S., Fabbro, S., Le Du, J., Lidman, C., Mouchet, M., Mourão, A., and Walker, E. S., “The core-collapse rate from the supernova legacy survey,” A&A, vol. 499, no. 3, pp. 653–660, 2009.
- [8] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” The journal of chemical physics, vol. 21, no. 6, pp. 1087–1092, 1953.
- [9] W. K. Hastings, “Monte carlo sampling methods using markov chains and their applications,” 1970.
- [10] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, “Hybrid monte carlo,” Physics letters B, vol. 195, no. 2, pp. 216–222, 1987.

- [11] M. Betancourt, “A conceptual introduction to hamiltonian monte carlo,” arXiv preprint arXiv:1701.02434, 2017.
- [12] M. D. Hoffman and A. Gelman, “The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo.,” Journal of Machine Learning Research, vol. 15, no. 1, pp. 1593–1623, 2014.
- [13] A. Patil, D. Huard, and C. J. Fonnesbeck, “Pymc: Bayesian stochastic modelling in python,” Journal of statistical software, vol. 35, no. 4, p. 1, 2010.
- [14] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, “Stan: A probabilistic programming language,” Journal of statistical software, vol. 76, no. 1, 2017.
- [15] R. M. Neal, Bayesian learning for neural networks, vol. 118. Springer Science & Business Media, 2012.
- [16] M. Betancourt and M. Girolami, “Hamiltonian monte carlo for hierarchical models,” Current trends in Bayesian methodology with applications, vol. 79, p. 30, 2015.
- [17] M. K. Cowles and B. P. Carlin, “Markov chain monte carlo convergence diagnostics: a comparative review,” Journal of the American Statistical Association, vol. 91, no. 434, pp. 883–904, 1996.
- [18] R. E. Kass, B. P. Carlin, A. Gelman, and R. M. Neal, “Markov chain monte carlo in practice: a roundtable discussion,” The American Statistician, vol. 52, no. 2, pp. 93–100, 1998.
- [19] W. A. Link and M. J. Eaton, “On thinning of chains in mcmc,” Methods in ecology and evolution, vol. 3, no. 1, pp. 112–115, 2012.
- [20] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, “An introduction to variational methods for graphical models,” Machine learning, vol. 37, no. 2, pp. 183–233, 1999.
- [21] D. J. C. MacKay, Information Theory, Inference & Learning Algorithms. New York, NY, USA: Cambridge University Press, 2002.
- [22] C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt, “Advances in variational inference,” IEEE transactions on pattern analysis and machine intelligence, 2018.
- [23] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, “Stochastic variational inference,” The Journal of Machine Learning Research, vol. 14, no. 1, pp. 1303–1347, 2013.
- [24] R. Ranganath, S. Gerrish, and D. Blei, “Black box variational inference,” in Artificial Intelligence and Statistics, pp. 814–822, 2014.

- [25] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” nature, vol. 521, no. 7553, p. 436, 2015.
- [26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, pp. 1097–1105, 2012.
- [27] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in Advances in neural information processing systems, pp. 3104–3112, 2014.
- [28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in Advances in neural information processing systems, pp. 2672–2680, 2014.
- [29] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” arXiv preprint arXiv:1312.6114, 2013.
- [30] D. J. Rezende, S. Mohamed, and D. Wierstra, “Stochastic backpropagation and approximate inference in deep generative models,” arXiv preprint arXiv:1401.4082, 2014.
- [31] Y. Gal, “Uncertainty in deep learning,” 2016.
- [32] D. Ritchie, P. Horsfall, and N. D. Goodman, “Deep amortized inference for probabilistic programs,” arXiv preprint arXiv:1610.05735, 2016.
- [33] A. Stuhlmüller, J. Taylor, and N. Goodman, “Learning stochastic inverses,” in Advances in neural information processing systems, pp. 3048–3056, 2013.
- [34] A. Speiser, J. Yan, E. W. Archer, L. Buesing, S. C. Turaga, and J. H. Macke, “Fast amortized inference of neural activity from calcium imaging data with variational autoencoders,” in Advances in Neural Information Processing Systems 30 (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4024–4034, Curran Associates, Inc., 2017.
- [35] J. Paisley, D. M. Blei, and M. I. Jordan, “Variational bayesian inference with stochastic search,” in Proceedings of the 29th International Conference on International Conference on Machine Learning, ICML’12, (USA), pp. 1363–1370, Omnipress, 2012.
- [36] M. Figurnov, S. Mohamed, and A. Mnih, “Implicit reparameterization gradients,” in Proceedings of the 32Nd International Conference on Neural Information Processing Systems, NIPS’18, (USA), pp. 439–450, Curran Associates Inc., 2018.
- [37] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

- [38] Z. Wang, W. Yan, and T. Oates, “Time series classification from scratch with deep neural networks: A strong baseline,” in 2017 international joint conference on neural networks (IJCNN), pp. 1578–1585, IEEE, 2017.
- [39] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: a review,” Data Mining and Knowledge Discovery, vol. 33, no. 4, pp. 917–963, 2019.
- [40] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” arXiv preprint arXiv:1502.03167, 2015.
- [41] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.
- [42] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.
- [43] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in International Conference on Learning Representations, 2019.
- [44] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256, 2010.
- [45] J. M. Tomczak and M. Welling, “Vae with a vampprior,” arXiv preprint arXiv:1705.07120, 2017.
- [46] Y. Burda, R. Grosse, and R. Salakhutdinov, “Importance weighted autoencoders,” arXiv preprint arXiv:1509.00519, 2015.
- [47] D. J. Rezende and S. Mohamed, “Variational inference with normalizing flows,” arXiv preprint arXiv:1505.05770, 2015.