Master's Thesis : Lightning Gravitational Wave Parameter Inference through Neural Amortization

Auteur : Delaunoy, Arnaud
Promoteur(s) : Louppe, Gilles
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2019-2020
URI/URL : http://hdl.handle.net/2268.2/10554

Lightning Gravitational Wave Parameter Inference through Neural Amortization

Author: Arnaud Delaunoy

Advisor: Gilles Louppe

Thesis submitted in partial fulfilment of the requirements for the Degree of Master of Science (MSc) in Data Science and Engineering

University of Liège - School of Engineering
Academic year 2019-2020

Acknowledgements

I would first like to thank my advisor Gilles Louppe and Antoine Wehenkel who supervised me throughout this thesis. They were always available for a discussion whenever I ran into a trouble spot. I have received plenty of useful advice regarding both technical aspects and the writing of my thesis which made me grow as a researcher.

I'm also thankful to Christoph Weniger, Andrew Williamson, Samaya Nissanke and Tanja Hindered from the GRAPPA Institute of the University of Amsterdam who provided valuable help regarding the physical aspects of the project.

Finally, I would like to thank my family and girlfriend who provided me continuous support throughout my thesis, both during hard and good times. This work would not have been possible without their encouragement and kind and caring listening.

Abstract

Gravitational waves analysis relies on a simulator governed by the nonlinear field equations of general relativity for binary systems. Such analysis is computationally very expensive and necessitates a large-scale exploration of the likelihood surface over the full parameter space. Neural networks have been gaining popularity as tools for gravitational waves analysis for the last few years. They lead to fast gravitational wave detection and parameter inference and hence complement classical slower techniques. This work explores simulation-based inference which relies on likelihood-to-evidence ratio estimation for the parameters of binary black holes mergers. We build a neural network modeling this ratio and use it in place of the simulator allowing to perform parameter inference in a few minutes. The performances are assessed on both gravitational waves generated by the simulator and emitted by real black holes. A scientific paper summarizing the results presented in this thesis is in preparation.

Contents

1 Introduction

Gravitational waves detection has been made possible with the construction of the LIGO detector. The first gravitational wave was detected in 2015. Currently, gravitational wave emission from at least eleven binary black hole mergers and three neutron star binary mergers have been measured. The improvements in sensitivity of the gravitational waves detector network for the next observing run (O4), scheduled to start in late 2021, will lead to unprecedentedly high detection rates of binary merger events of 1 per week to 1 per day [B. P. Abbott et al., 2018]. There is thus an urgent need to develop fast and efficient methods for parameter estimation.

Current analyses are based on sampling methods such as Markov chain Monte Carlo (MCMC) and nested sampling. Those methods have the huge drawback of being slow to run, requiring days to weeks to get satisfactory results. This prevents multi-messenger astronomy which consists in considering multiple messenger signals [Shawhan et al., 2019]. Examples of messenger signals are gravitational waves, gamma-ray bursts and electromagnetic radiations. Information inferred from one messenger can help in the detection of other messenger signals. Fast and accurate inference methods are thus critical.

The **research question** addressed by this thesis is whether deep learning algorithms can be used to speed up the inference of binary black holes mergers' parameters based on a gravitational wave. We investigate simulation-based likelihood-to-evidence ratio estimation methods and explore different neural network architectures to perform this task. Experiments show that this method seems to produce reliable results in a few minutes instead of days. However, results indicate that MCMC better constraint the parameters. Further analyses are needed to understand where those differences come from.

2 Gravitational wave physics

This chapter provides a brief overview of gravitational wave's physics. A model for this phenomenon is then presented.

Gravitational waves modify the gravitational forces applied to the bodies it passes through. Those are represented by a signal indicating the evolution of those modifications over time. Gravitational waves are explained by the theory of general relativity which states that no information can travel faster than the speed of light. This includes information about the positions of masses. When celestial bodies move, the gravitational force they apply on other bodies changes. The modification of their gravitational field travels at the speed of light and is called gravitational wave.

To characterize gravitational waves, a space-time model is used. Space-time is a mathematical model combining the three dimensions of space with the fourth dimension of time. A position in space-time, characterized by 4 coordinates, is called an event. A gravitational wave modifies gravity and hence the forces applied on bodies. The forces modifications lead to the modification of the trajectories of the different bodies and consequently modify space-time. To characterize a gravitational wave, one considers two reference space-time events and analyses the modifications on those events resulting from the gravitational wave. The effect of gravitational waves at a given time is expressed by an adimensional amplitude related to the change of proper distance in the space-time model between the two space-time reference events under the effect of this gravitational wave ["Gravitational waves", 2020]. Formally, let us denote by h the adimensional amplitude that characterizes the effect of a gravitational wave, by l the distance between the two reference space-time events without gravitational wave effect and by δl the distance modification implied by the gravitational wave. The amplitude is expressed:

$$h = \frac{\delta l}{l}. \tag{2.1}$$

Gravitational waves effect evolves over time as bodies keep moving. Consequently, those are represented by a temporal signal indicating the evolution of the amplitude h over time. Gravitational waves are produced by any mass displacement. Therefore, there is a continuous flow of gravitational waves from diverse sources at any time in the space. While most of these waves are unobservable because of their tiny amplitude, gravitational waves produced by pair of massive objects orbiting to each other such as black-holes are strong enough to be detected by dedicated measurement tools ["Introduction to LIGO gravitational waves", 2020].

Gravitational wave's detection systems are based on interferometry. Interferometers are composed of two perpendicular arms through which light travels, those are designed such that no interference appears at the intersection when there are no gravitational waves passing through. Gravitational waves modify the forces applied on those arms and hence the travel time of the light in each arm. A travel time modification makes the light beams to be out of phase at intersection showing interferences. The first detection system to be built is LIGO which is composed of two detectors based at Hanford and Livingston in the USA. Those are usually denoted H1 and L1. Later on, the VIRGO detector based at Santo Stefano a Macerata in Italy and the KAGRA detector based at Gifu Prefecture in Japan became available. Having multiple detectors at disposal offers several advantages. First correlating multiple signals allows mitigating the effect of noise. Second, by nature interferometers cannot be oriented offering no information about the position of the gravitational event. Position can only be inferred by correlating information from multiple detectors.

Gravitational waves are modeled by two components: a waveform model and a noise model independent from each other. The resulting gravitational wave signal is the sum of the output of each model.

We consider a deterministic waveform model. It models the gravitational wave such as it would be perceived without any noise. A waveform model takes as input a set of parameters characterizing the binary black-hole merger. There exist several waveform models, taking different effects into account and some being faster to evaluate than others. The waveform model simulates the gravitational wave emitted by the black holes. The way this gravitational wave is perceived by the detector depends on the detector. This gravitational wave is then projected on the detectors to model the way detectors would perceive it. This projection has two effects. First, the amplitude of the signal is scaled based on the antenna pattern of the detector. Second, the time at which the signal is received is computed based on the position of the detector on Earth and the position of the event. Note that, as Earth rotates, the position of the detectors depends on the time at which the gravitational wave reaches the Earth.

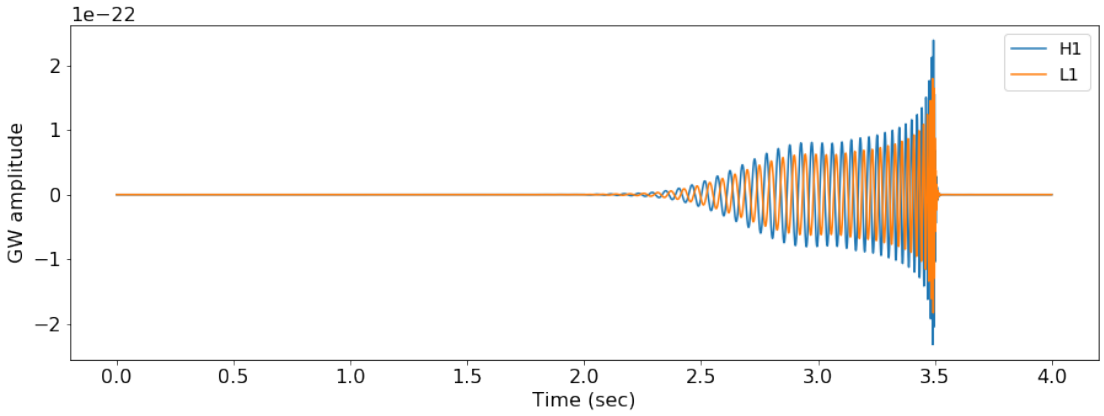


Figure 2.1 – An example of gravitational wave signal projected on the Hanford(H1) and Livingston(L1) detectors.

An example of gravitational wave signal is shown in Figure 2.1. As time passes, the black holes get closer and the signal amplitude increases until the peak. The peak corresponds to the moment at which the black holes merge. After merging, the gravitational amplitude quickly decreases.

The model parameters we consider are divided into 8 intrinsic parameters and 7 extrinsic parameters described in Table 2.1. For a better understanding, a graphical view of those parameters is also provided in Figure 2.2. The sky position, polarization and coalescence time parameters act on the projection of the gravitational wave onto the detectors. The coalescence time determines the position of the detectors on earth. The other parameters determine the gravitational wave emitted. We use the same conventions as the PyCBC Python library. Two of those parameters are the detector frame masses of the black holes. There exists a phenomenon similar to the Doppler effect that affects gravitational waves called redshift. The redshift effect modifies the signal such that a gravitational wave resulting from a binary black-hole merger with masses (m_1, m_2) without redshift effect is indistinguishable from one resulting from a merger with masses $(\frac{m_1}{1+z}, \frac{m_2}{1+z})$ at redshift z [Holz, 2020]. When characterizing a gravitational event, it is common to consider detector frame masses instead of the true masses called source masses. Denoting by m^{det} the detector frame mass and by m^{src} the source mass, under redshift z, those are linked by:

$$m^{src} = \frac{m^{det}}{1+z}. \tag{2.2}$$

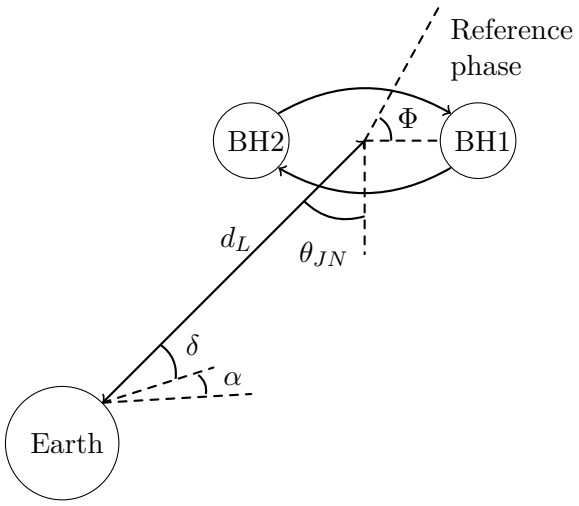


Figure 2.2 – Graphical view of waveform parameters

The noise model characterizes the noise due to the interferometer. Causes of noises are mainly local vibration such as seismic activity or truck driving and related to interferometer's internal components ["LIGO's Dual Detectors", 2020]. Before giving further information about the noise model, let us introduce the notion of power spectral density (PSD). The power spectral density of a random signal describes its power in the frequency domain. Let $\hat{x}$ be a random signal expressed in the frequency domain and denote by S its power spectral density, it can be expressed:

$$S(\omega) = \underset{\hat{x}}{E}\{|\hat{x}(\omega)|^2\}. \tag{2.3}$$

A common noise model is to consider Gaussian noise in the frequency domain. Gaussian noise is a type of noise that admits a normal probability density function. The noise model is conditioned on a given PSD, it has a zero mean and a variance equal to the PSD. The PSD is usually approximated based on real noise events. The probability density function $p_{\hat{x}}$ of such a noise is then expressed:

$$p_{\hat{x}}(\omega) = \frac{1}{\sqrt{2\pi S(\omega)}} \exp\left(-\frac{\omega^2}{2S(\omega)}\right). \tag{2.4}$$

Figure 2.3 shows the same signal as Figure 2.1 with Gaussian noise added. The noise amplitude is higher than the signal amplitude making the signal hard to distinguish.

The analysis of the signal detected by interferometry can be decomposed into two tasks. Those are shown graphically in Figure 2.4. The first task consists in detecting the parts of the signal that contain a gravitational wave generated by the merging of two

Parameter	Symbol	Description
Intrinsic parameters		
Masses	m_1^{det}, m_2^{det}	The detector frame masses of the two black-holes.
Spins	$\boldsymbol{S_1} = (a_1, \theta_1, \phi_1)$ $\boldsymbol{S_2} = (a_2, \theta_2, \phi_2)$	The spins of the two black-holes expressed in spherical coordinates. a_i is the amplitude, θ_i is the polar angle and ϕ_i is the azimuthal angle.
Extrinsic parameters		
Luminosity distance	d_L	The distance between the blacks holes and the Earth.
Inclination angle	θ_{JN}	The inclination angle defined as the angle between the total angular momentum of the binary black-hole merger and the line-of-sight.
Polarization angle	Ψ	The polarization angle relates the frame in which the gravitational wave propagates and the reference frame of the detector.
Coalescence phase	Φ	The orbital phase of the merger at reference frequency.
Sky position	α, δ	Correspond to the direction of the event. Combined the with distance, the sky position fixes the location of the event. The sky position is expressed in terms of right ascension denoted by α and declination δ.
Coalescence time	t_c	Time at which the black holes merge to become one.

Table 2.1 – Parameters of the waveform model

celestial bodies. This work focuses on the second task that is given a detected merger signal, to perform inference on the parameters of the merger that generated the signal.

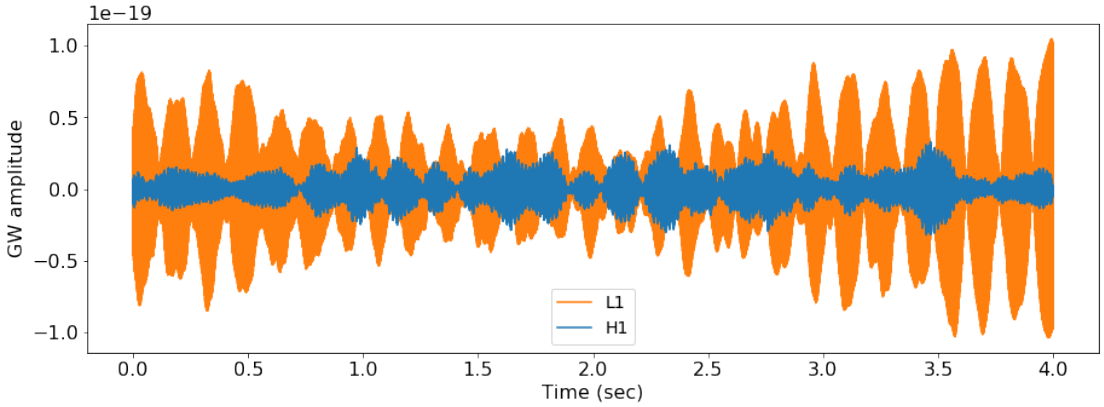


Figure 2.3 – Figure 2.1 gravitational wave signal with Gaussian noise added

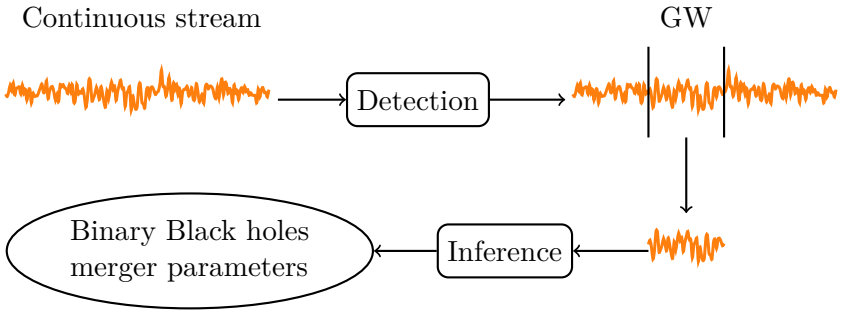


Figure 2.4 – Gravitational wave analysis tasks

3 Deep learning

We introduce in this chapter neural networks in general, as well as some neural network architectures and training techniques that will be used later in this work.

3.1 Neural networks

Neural networks are a type of parametric functions. The parameters of a neural network are to be optimized to match an unknown function implicitly defined by a set of input/output pairs. In this section, the structure of a neural network is presented along with the optimization procedure.

Neurons The basic building block of a neural network is a neuron. A neuron is a parametric function taking as input a set of N values that we denote by $\boldsymbol{h} = (h_0, h_1, .., h_{N-1})$ and outputting a single value that we denote by h'. The parameters of the neuron are N weights that we denote by $\boldsymbol{w} = (w_0, w_1, .., w_{N-1})$ and a bias that we denote by b. The output of the neuron is defined:

$$h' = \sigma(b + \sum_{i=0}^{N-1} h_i w_i), \tag{3.1}$$

where σ is a non-linear function called an activation function. A graphical view of a neuron is provided in Figure 3.1.

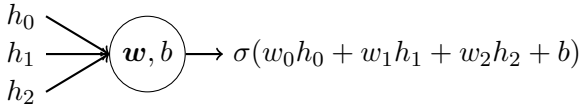


Figure 3.1 – Graphical view of a neuron with 3 inputs.

Neural network A neural network is built by combining neurons. The simpler form of neural networks is the multi-layer perceptron. In a multi-layer perceptron, neurons are arranged in layers, each neuron of a layer takes as input the outputs of the neurons of the previous layer. The first layer is composed of the inputs and the last layer's neurons are the outputs of the multi-layer perceptron. Internal layers are called hidden layers. A graphical view of a multi-layer perceptron is provided in Figure 3.2.

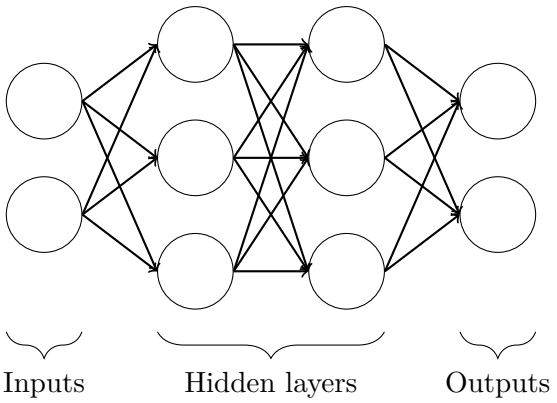


Figure 3.2 – Graphical view of a multi-layer perceptron modeling a function with 2 inputs and 2 outputs and containing 2 hidden layers of 3 neurons each.

Optimization procedure Let us denote by f the neural network and by $\boldsymbol{\phi}$ its parameters (biases and weights). We have at disposal a dataset composed of input/output pairs, we denote by $\boldsymbol{x}$ the inputs and by $\boldsymbol{y}$ the outputs. The goal is to optimize the parameters $\boldsymbol{\phi}$ such that, for new inputs, the neural network outputs the most probable associated outputs. To this end, we define a loss function denoted by L taking as input an output from the dataset and the corresponding prediction of the neural network and outputting a real value. The loss function must be designed such that it is minimized when the two inputs are equal, i.e. the neural network makes the correct prediction. Neural network's parameters are optimized through stochastic gradient descent to minimize the expected value of the loss function. At each step, a batch of B samples is drawn from the dataset, the inputs being denoted by $\boldsymbol{x}_0, \boldsymbol{x}_1, .., \boldsymbol{x}_{B-1}$ and the outputs by $\boldsymbol{y}_0, \boldsymbol{y}_1, .., \boldsymbol{y}_{B-1}$. Each parameter of the neural network denoted by ϕ_i is updated using the following rule:

$$\phi_i \leftarrow \phi_i - \alpha \frac{1}{B} \sum_{j=0}^{B-1} \frac{dL(\boldsymbol{y_j}, f(\boldsymbol{x_j}; \boldsymbol{\phi}))}{d\phi_i}, \tag{3.2}$$

where α is the learning rate.

The partial derivatives are computed using the backpropagation algorithm. This algorithm is based on the chain rule. Denoting by $(o_0, o_1, .., o_{M-1})$ the M inputs of a neuron and

by u its output, the chain rule is expressed as:

$$\frac{du}{dW_i} = \sum_{j=0}^{M-1} \frac{\partial u}{\partial o_j} \frac{do_j}{dW_i}. \tag{3.3}$$

The terms $\frac{\partial u}{\partial o_j}$ depend only on the considered neuron. The structure of a neuron being fixed, its derivative is hard-coded and hence known. The terms $\frac{do_j}{dW_i}$ are computed recursively using the chain rule.

3.2 Convolutional neural network

A convolutional neural network is a neural network composed of convolutional layers. Convolutional layers have been designed to give good performances with inputs that admit a spatial representation. The most common example of such inputs is images. Images are composed of pixels that admit a spatial representation. Each element of the input can itself be composed of several features called channels. As an example, a pixel of a colored image is composed of 3 channels being the values of the red, green and blue components. Convolutions can also be used for temporal signals such as gravitational waves. Convolutional layers may admit an arbitrary number of dimensions. In this work, we will only consider one-dimensional convolutions as temporal signals have time as single dimension.

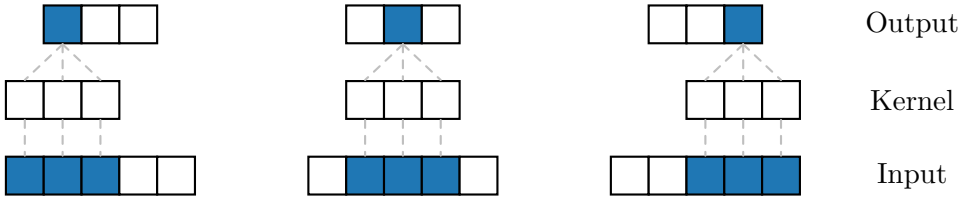


Figure 3.3 – Convolutional layer

Figure 3.3 shows graphically the computation made by a convolutional layer. The input consists of a set of consecutive elements across the considered dimension. We consider an input of 5 elements in the picture. A kernel composed of learnable weights will be applied to input elements to produce the output of the layer. The size of the kernel is arbitrary and fixed when designing the architecture. In the picture, a kernel of size 3 is chosen. An output element is computed by applying the kernel to the corresponding input element channels and the channels of the elements next to it. Each input element is multiplied by its corresponding kernel element, the results are then summed. Let $\boldsymbol{h}$ be the input of the convolution layer, $h_{i,j}$ being the j^{th} channel of its i^{th} element, by $\boldsymbol{k}$ the kernel, $k_{i,j}$ being the j^{th} channel of its i^{th} element, by L the kernel size, by C_{in} the input channel size and by $\boldsymbol{h'}$ the output, h'_i being its i^{th} element. The convolution operator

denoted by $\circledast$ is expressed:

$$\boldsymbol{h'} = \boldsymbol{h} \circledast \boldsymbol{k} \Leftrightarrow h'_i = \sum_{c=0}^{C_{in}-1} \sum_{l=0}^{L-1} h_{i+l,c} k_{l,c}. \tag{3.4}$$

To improve convolutional neural networks' capacity, one may equip hidden layers with several channels. This allows the neural network to model more complex functions as hidden layers can contain more information. We denote by C_{out} the number of output channels. A layer outputting C_{out} channels is equipped with C_{out} kernels. Denoting by $\boldsymbol{h'}_i$ the i^{th} channel of the output and by $\boldsymbol{k}_i$ the i^{th} kernel, $\boldsymbol{h'}_i$ is expressed:

$$\boldsymbol{h'}_i = \boldsymbol{h} \circledast \boldsymbol{k}_i \tag{3.5}$$

Dilated convolutions Dilated convolutional layers are based on dilated convolutions [Yu and Koltun, 2015]. Those are similar to convolutions with the exception that the kernel is not applied on consecutive inputs elements. A dilated convolution of dilation d applies the kernel on input elements equally spaced by a distance d.

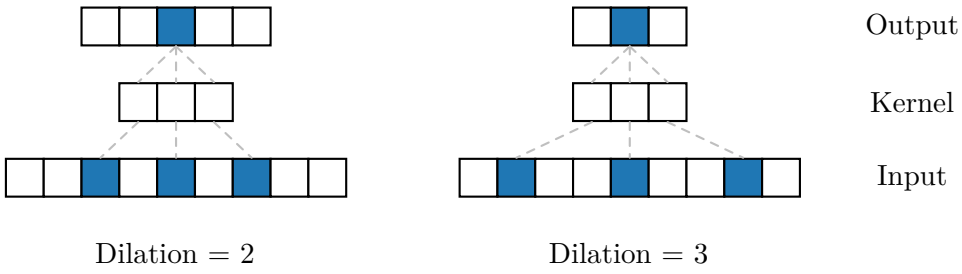


Figure 3.4 – Dilated convolutional layer

Figure 3.4 shows graphically the computation of a dilated convolution for dilations of size 2 and 3. A dilated convolution of dilation 1 is similar to a classic convolution. Using the same notations as for convolutions, a dilated convolution of dilation d is computed the following way:

$$h'_i = \sum_{c=0}^{C_{in}-1} \sum_{l=0}^{L-1} h_{i+dl,c} k_{l,c}. \tag{3.6}$$

Residual neural network A residual network is a type of convolutional neural network based on residual connections [He et al., 2016]. A residual connection is an identity mapping of features at a given layer that are added deeper in the network. It has as objective to allow easier training of the neural network with the assumption that the difference between the input and the output is easier to learn than the output itself.

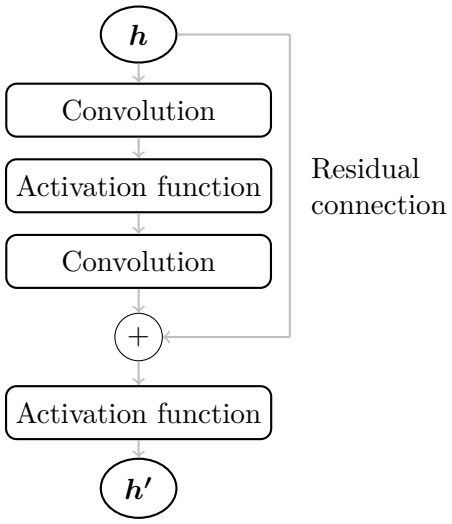


Figure 3.5 – Residual neural network block

Figure 3.5 shows a residual network block. The output is the result of the addition of the original input and the output of two convolutional layers. A residual neural network is composed of a series of such blocks.

3.3 Normalization

It is common in deep learning to normalize the data to improve training. Normalization may take several forms. It may be applied to the training data but also to features inside the neural network through normalization layers.

Data normalization Data normalization consists in shifting and scaling every feature to be of comparable mean and variance. Let us denote by $(\boldsymbol{x}, \boldsymbol{y})$ the dataset, $\boldsymbol{x}$ are the inputs and $\boldsymbol{y}$ the outputs. We consider the inputs $\boldsymbol{x}$ to be composed of N samples of M features and denote by $x_{i,j}$ the j^{th} feature of the i^{th} sample. The first step consists in computing the approximate mean $\hat{\mu}_j$ and variance $\hat{\sigma}_j^2$ of each feature from the dataset:

$$\hat{\mu}_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}, \qquad \hat{\sigma}_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \hat{\mu}_j)^2}. \tag{3.7}$$

The normalized element denoted by $\hat{x}_{i,j}$ is then expressed as:

$$\hat{x}_{i,j} = \frac{x_{i,j} - \hat{\mu}_j}{\hat{\sigma_j}}. \tag{3.8}$$

Batch normalization A batch normalization layer normalizes the features inside the neural network in a similar way to data normalization [Ioffe and Szegedy, 2015]. The difference resides in the fact that since the entire dataset is not available during a training step, mean and variance are then computed at each training step on batch samples. Let us denote by $h_{i,j}$ the j^{th} feature of the i^{th} sample in the batch and by $\tilde{\mu}_j$ and $\hat{\sigma^2}_j$ the approximations of mean and variance of feature j computed on the current batch. Let B be the batch size, the batch mean and variance are computed as:

$$\tilde{\mu}_j = \frac{1}{B}\sum_{i=1}^{B} h_{i,j}, \qquad \tilde{\sigma}_j = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(h_{i,j} - \tilde{\mu}_j)^2}. \tag{3.9}$$

Batch normalization acts differently at training and testing phase. At training phase, the normalized element, $\hat{h}_{i,j}$ is expressed as:

$$\hat{h}_{i,j} = \frac{h_{i,j} - \tilde{\mu}_j}{\tilde{\sigma}_j}. \tag{3.10}$$

At testing phase, running means and variances are used to scale the features. Let μ' and σ'^2 denote running mean and variance. At each training step, the running means and variances are updated:

$$\mu' \leftarrow (\alpha)\mu' + (1-\alpha)\tilde{\mu}, \qquad \sigma'^2 \leftarrow (\alpha)\sigma'^2 + (1-\alpha)\tilde{\sigma}^2, \tag{3.11}$$

α being the momentum. The normalized element is expressed:

$$\hat{h}_{i,j} = \frac{h_{i,j} - \mu'_j}{\sigma'_j}. \tag{3.12}$$

Optionally, in addition to normalization parameters, learnable per-channel scale and bias are applied. Let C be the number of channels of the features to normalize, we denote by $\beta_{1,\dots,C}$ the biases and by $\gamma_{1,\dots,C}$ the scales. Let us denote by $c(j)$ the channel to which feature j belongs. When using those parameters, Equation 3.10 is modified as:

$$\hat{h}_{i,j} = \frac{h_{i,j} - \tilde{\mu}_j}{\tilde{\sigma}_j}\gamma_{c(j)} + \beta_{c(j)}. \tag{3.13}$$

Equation 3.12 is updated similarly.

Layer normalization Layer normalization works similarly to batch normalization with the difference that those compute normalization parameters across features instead of batches. The same normalization parameters are shared for all features in a layer, and different normalization parameters are computed for each sample. The normalization

parameters of a sample i with M features are expressed:

$$\tilde{\mu}_i = \sum_{j=1}^{M} \frac{1}{M} h_{i,j} \qquad \tilde{\sigma}_i = \sqrt{\sum_{j=1}^{M} \frac{1}{M} (h_{i,j} - \tilde{\mu}_i)^2}. \tag{3.14}$$

In opposition to batch normalization, layer normalization acts similarly at training and testing phase, the normalized element $\hat{h}_{i,j}$ is expressed as:

$$\hat{h}_{i,j} = \frac{h_{i,j} - \tilde{\mu}_i}{\tilde{\sigma}_i}. \tag{3.15}$$

Layer normalization uses per feature scale and bias parameters γ and β, those are then of size M. When using those parameters, equation 3.15 is modified:

$$\hat{h}_{i,j} = \frac{h_{i,j} - \tilde{\mu}_i}{\tilde{\sigma}_i} \gamma_j + \beta_j, \tag{3.16}$$

3.4 Hyper networks

Hyper networks were developed by Ha et al., 2016. The idea is to make the weights of the primary network generated by another network called a hyper network, the hyper network being trained by gradient descent, end-to-end with the primary network.

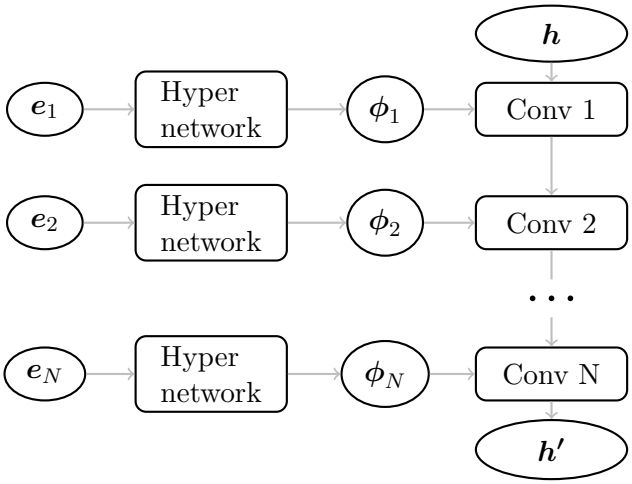


Figure 3.6 – Hyper network

Figure 3.6 shows an architecture based on a hyper network. The convolutions have no learnable internal parameters. Parameters are provided by a hyper network shared between all layers. Each layer has a set of scalar characterizing the layer called embedding. We denote the embedding of the i^{th} layer by $\boldsymbol{e}_i$. This embedding is given as input to

the hyper network to produce parameters for the convolutional layer. We denote the parameters of the i^{th} convolutional layer by $\boldsymbol{\phi}_i$. The learnable parameters of such an architecture are then the embeddings and the parameters of the hyper network. Hyper networks were developed to reduce the number of parameters while keeping the same convolutional structure as an architecture without hyper network. The hyper network is usually small, and the embedding size can be set to an arbitrarily low number, independently from the complexity of the primary network.

3.5 Conditionally parameterized convolutions

Conditionally parameterized convolutions (CondConv) mimic mixtures of experts while being faster to evaluate [Yang et al., 2019]. Let us denote by $\boldsymbol{h}$ the input of the CondConv layer, by $\boldsymbol{h'}$ the output of the layer, by $\boldsymbol{\phi}_i$ the parameters of the i^{th} expert, and by α_i a weight associated to the i^{th} expert. The mixture of experts framework consists in evaluating the convolution for each expert parameter and to combine their results. A mixture of N experts has the following expression:

$$\boldsymbol{h'} = \sum_{i=1}^{N} \alpha_i (\boldsymbol{\phi}_i \circledast \boldsymbol{h}). \tag{3.17}$$

CondConv combines the kernels before evaluating a single convolution. Modifying the computation order this way is equivalent to mixtures of experts while being more computationally efficient since convolutions have to be applied at many different positions in the input while the experts are only combined once per input. A CondConv layer with N experts has the following expression:

$$\boldsymbol{h'} = (\sum_{i=1}^{N} \alpha_i \boldsymbol{\phi}_i) \circledast \boldsymbol{h}. \tag{3.18}$$

The routing weights α_i are computed based on the inputs through a routing function r:

$$\alpha_{1,..,N} = r(\boldsymbol{h}). \tag{3.19}$$

The computational graph of an N experts CondConv is shown graphically in Figure 3.7. It contains N internal learnable kernels that are combined to produce the final kernel of a classical convolution.

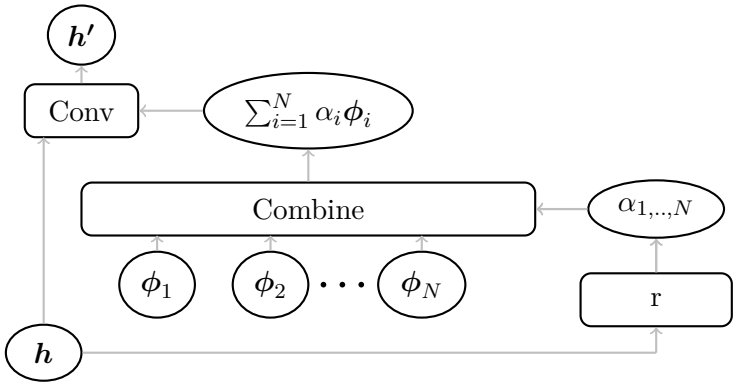


Figure 3.7 – Conditionally parametrized convolution

4 Bayesian inference

In this section, we introduce the task of Bayesian inference with simulators. We review common techniques, with a particular focus on the method used in this work.

4.1 Problem statement

Inference is the task of making a guess on something based on information at disposal. Let $\boldsymbol{x}$ denote the observation and $\boldsymbol{\vartheta}$ the parameters to be inferred. **Bayesian inference** consists in using the Bayes' theorem to perform inference. Let $p(\boldsymbol{\vartheta})$ be a prior distribution over the parameters to be inferred, i.e. a prior knowledge without any information about an observation. $p(\boldsymbol{x}|\boldsymbol{\vartheta})$ is the likelihood, which is the probability density function (pdf) of the observations conditioned on the generating parameters. The evidence $p(\boldsymbol{x})$ is the pdf of the observation. The goal of Bayesian inference is to compute the posterior $p(\boldsymbol{\vartheta}|\boldsymbol{x})$, which is the probability density function over the generating parameters conditioned on an observation. This is done through the Bayes' theorem:

$$p(\boldsymbol{\vartheta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta})}{p(\boldsymbol{x})} == \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta})}{\int_{\boldsymbol{\Theta}} d\boldsymbol{\vartheta}' p(\boldsymbol{x}|\boldsymbol{\vartheta}')p(\boldsymbol{\vartheta}')}, \tag{4.1}$$

where $\boldsymbol{\Theta}$ is the set of possible generating parameters values $\boldsymbol{\vartheta}$.

Simulation based Bayesian inference also called likelihood-free inference (LFI) is a particular instance of Bayesian inference in which the likelihood $p(\boldsymbol{x}|\boldsymbol{\vartheta})$ is intractable. This likelihood is only implicitly defined by a simulator through which samples can be drawn. In addition, the integral at the denominator of Equation 4.1 is also intractable. This all makes Bayesian inference a complex task.

4.2 Bayesian inference methods with tractable likelihood

We address in this section the task of performing Bayesian inference with a tractable likelihood $p(\boldsymbol{x}|\boldsymbol{\vartheta})$ and an intractable evidence $p(\boldsymbol{x})$. To this end Markov chain Monte-Carlo methods are considered.

Markov chain Monte-Carlo (MCMC) methods are used to sample a random variable with a pdf proportional to another known pdf up to a normalizing factor [Metropolis et al., 1953]. For a given observation $\boldsymbol{x}$, the posterior $p(\boldsymbol{\vartheta}|\boldsymbol{x})$ is proportional to $p(\boldsymbol{x}|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta})$. One can then use MCMC to sample from the posterior. A Markov chain is a sequence of events such that probability of each event is only conditioned on the previous event. We denote by $\boldsymbol{\vartheta}_t$ the t^{th} element in the chain. A Markov chain Monte-Carlo algorithm constructs a Markov chain such that the asymptotic distribution of its events follow the target distribution. To this end such algorithms are equipped with a proposal transition probability function $q(\boldsymbol{\vartheta}_{\text{prop}}|\boldsymbol{\vartheta}_t)$ from which it is easy to sample and an acceptance probability $\alpha(\boldsymbol{x}, \boldsymbol{\vartheta}_t, \boldsymbol{\vartheta}_{\text{prop}})$. At each iteration, a proposal sample $\boldsymbol{\vartheta}_{\text{prop}}$ is generated according to the distribution $q(\boldsymbol{\vartheta}_{\text{prop}}|\boldsymbol{\vartheta}_t)$ and accepted with probability $\alpha(\boldsymbol{x}, \boldsymbol{\vartheta}_t, \boldsymbol{\vartheta}_{\text{prop}})$. The pseudo-code of a MCMC algorithm is shown at Algorithm 1.

Algorithm 1 Markov chain Monte-Carlo

$\boldsymbol{x} \leftarrow$ Conditioning observation
$\boldsymbol{\vartheta}_0 \leftarrow \boldsymbol{\vartheta}_0 \sim p(\boldsymbol{\vartheta})$
$t \leftarrow 0$
while !StoppingCriterion **do**
    $\boldsymbol{\vartheta}_{\text{prop}} \leftarrow \boldsymbol{\vartheta}_{\text{prop}} \sim q(\boldsymbol{\vartheta}_{\text{prop}}|\boldsymbol{\vartheta}_t)$
    $u \leftarrow u \sim \text{Uniform}(0, 1)$
    if $u < \alpha(\boldsymbol{x}, \boldsymbol{\vartheta}_t, \boldsymbol{\vartheta}_{\text{prop}})$ **then**
        $\boldsymbol{\vartheta}_{t+1} \leftarrow \boldsymbol{\vartheta}_{\text{prop}}$
    else
        $\boldsymbol{\vartheta}_{t+1} \leftarrow \boldsymbol{\vartheta}_t$
    end if
    $t \leftarrow t + 1$
end while
return $\boldsymbol{\vartheta}_{0:t}$

There exist many Markov chain Monte-Carlo algorithms that mainly differ in the acceptance probability. As an example, we present the Metropolis-Hastings algorithm [Hastings, 1970]. For a given proposal transition probability function $q(\boldsymbol{\vartheta}_{\text{prop}}|\boldsymbol{\vartheta}_t)$, the acceptance probability is expressed:

$$\alpha(\boldsymbol{x}, \boldsymbol{\vartheta}_t, \boldsymbol{\vartheta}_{\text{prop}}) = \min\left(1, \frac{p(\boldsymbol{\vartheta}_{\text{prop}})p(\boldsymbol{x}|\boldsymbol{\vartheta}_{\text{prop}})}{p(\boldsymbol{\vartheta}_t)p(\boldsymbol{x}|\boldsymbol{\vartheta}_t)} \frac{q(\boldsymbol{\vartheta}_{\text{prop}}|\boldsymbol{\vartheta}_t)}{q(\boldsymbol{\vartheta}_t|\boldsymbol{\vartheta}_{\text{prop}})}\right) \tag{4.2}$$

4.3 Review of simulation-based Bayesian inference

In this section, we review methods for performing simulation-based Bayesian inference. Those methods are divided into the approximate Bayesian computation (ABC) method that is not based on surrogate model and methods that construct a surrogate model [Cranmer et al., 2019]. Model-based methods can further be divided into methods that learn a model for the likelihood, the posterior or the likelihood ratio.

Approximate Bayesian computation (ABC) This method inspired by rejection sampling consists in sampling data from the joint probability density function and rejecting values far from the observation [Beaumont et al., 2002]. The procedure is summarized in Algorithm 2. Sampling from the joint density distribution is done by sampling $\boldsymbol{\vartheta}$ from the prior $p(\boldsymbol{\vartheta})$ and then $\boldsymbol{x}'$ from the likelihood $p(\boldsymbol{x}'|\boldsymbol{\vartheta})$ using the simulator. The samples $\boldsymbol{\vartheta}$ that generated observations $\boldsymbol{x}'$ close to the observation $\boldsymbol{x}$ are then retained. To this end, one defines a distance function between two observations, if this distance is lower than a fixed value ϵ, $\boldsymbol{\vartheta}$ is retained. It can be proven that with $\epsilon \to 0$, retained $\boldsymbol{\vartheta}$ follow the posterior distribution $p(\boldsymbol{\vartheta}|\boldsymbol{x})$ conditioned on the observed value of $\boldsymbol{x}$. The posterior distribution can then be approximated through density estimation techniques.

Algorithm 2 Approximate Bayesian computation

```
while !StoppingCriterion do
    ϑ ← ϑ ~ p(ϑ)
    x' ← x' ~ p(x'|ϑ)
    if DISTANCE(x, x') < ε then
        STORE(ϑ)
    end if
end while
```

Modeling the likelihood A second approach consists in building a surrogate of the likelihood $\hat{p}(\boldsymbol{x}|\boldsymbol{\vartheta})$ and using this model to perform inference. Diggle and Gratton, 1984 use kernel density estimation to construct the surrogate. Markov chain Monte Carlo methods can then be used to sample from the posterior conditioned on an observation using the likelihood model [Hastings, 1970; Metropolis et al., 1953].

Modeling the posterior distribution A model $\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})$ of the posterior can be directly learned. A first approach to this problem would be to train a neural network that takes as input the observation $\boldsymbol{x}$ and output parameters of a given probability density function such as a Gaussian mixture. Such models may however be of limited capacity due to the choice of probability density function. A way to increase the model capacity is to use conditional variational auto-encoders [Kingma and Welling, 2013; Rezende et al., 2014;

Sohn et al., 2015]. This framework introduces a set of latent variables $\boldsymbol{z}$. The posterior is then given by:

$$\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x}) = \int r_{\phi_1}(\boldsymbol{z}|\boldsymbol{x}) r_{\phi_2}(\boldsymbol{\vartheta}|\boldsymbol{z}, \boldsymbol{x}) dz, \tag{4.3}$$

where r_{ϕ_1} is the encoder network and r_{ϕ_2} is the decoder network. To train this network, one need to introduce a third network $q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\vartheta})$. Those networks are trained to maximize

$$-KL(q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\vartheta})|r_{\phi_1}(\boldsymbol{z}|\boldsymbol{x})) + E_{q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\vartheta})}\{\log r_{\phi_2}(\boldsymbol{\vartheta}|\boldsymbol{z}, \boldsymbol{x})\}.$$

The KL symbol denotes the Kullback-Leibler divergence and is expressed as:

$$KL(q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\vartheta})|r_{\phi_1}(\boldsymbol{z}|\boldsymbol{x})) = E_{q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\vartheta})}\left\{\log \frac{q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\vartheta})}{r_{\phi_1}(\boldsymbol{z}|\boldsymbol{x})}\right\}. \tag{4.4}$$

The term $-KL(q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\vartheta})|r_{\phi_1}(\boldsymbol{z}|\boldsymbol{x}))$ makes the distributions $q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\vartheta})$ and $r_{\phi_1}(\boldsymbol{z}|\boldsymbol{x})$ close in average. The term $E_{q_{\phi_3}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\vartheta})}\{\log r_{\phi_2}(\boldsymbol{\vartheta}|\boldsymbol{z}, \boldsymbol{x})\}$ encourages the network to predict the correct parameters $\boldsymbol{\vartheta}$.

Normalizing flows can also be used [Kobyzev et al., 2020; Papamakarios et al., 2017; Rezende and Mohamed, 2015]. A normalizing flow is a density estimator that leverages the change of variable theorem by modeling the probability density as being an invertible transformation of another tractable base probability density function. Let $p_{h'}$ be the density to model and p_h the base density, the normalizing flow models an invertible transformation f such that $\boldsymbol{h'} = f(\boldsymbol{h})$, using the change of variable theorem, the density $p_{h'}$ can then be expressed:

$$p_{h'}(\boldsymbol{h'}) = p_h(f^{-1}(\boldsymbol{h'}))\left|\det\left(\frac{\partial f^{-1}}{\partial \boldsymbol{h'}}\right)\right|. \tag{4.5}$$

For Equation 4.5 to be tractable, the transformation f must be designed to have a tractable Jacobian $\left|\det\left(\frac{\partial f^{-1}}{\partial \boldsymbol{h'}}\right)\right|$. To obtain a high capacity model, several transformations are applied end-to-end. A normalizing flow is optimized to maximize the log probability of the training dataset. This framework can be extended to learn probabilities conditioned on an observation by using a conditional base density or by conditioning the transformations.

Modeling the likelihood ratio Another type of method consists in learning a model of the likelihood ratio. Let $\boldsymbol{\vartheta}_0$ be the null hypothesis and $\boldsymbol{\vartheta}_1$ the alternative hypothesis. The likelihood ratio is expressed:

$$\lambda(\boldsymbol{x}; \boldsymbol{\vartheta}_0, \boldsymbol{\vartheta}_1) = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta}_0)}{p(\boldsymbol{x}|\boldsymbol{\vartheta}_1)}. \tag{4.6}$$

Cranmer et al., 2015 show how to learn this likelihood ratio using a classifier trained to discriminate between the two hypotheses. Brehmer et al., 2018a; 2018b; Brehmer et al., 2020; Brehmer et al., 2019; Stoye et al., 2018 extend this method to make use of information about simulator's latent variables, i.e. intermediary states of the simulation process showing improvements in convergence speed and data efficiency.

Active learning To improve the sample efficiency of the methods, active learning can be used. Those methods consist in using the information at disposal to sample efficiently from the simulator reducing the number of samples to draw. Such techniques consider posterior density estimation conditioned on a specific observation that we call $\boldsymbol{x}_0$. The methods aim to build a model that is performing well on this observation, to this end we sample parameters $\boldsymbol{\vartheta}$ from a distribution $\tilde{p}(\boldsymbol{\vartheta})$ different from the prior $p(\boldsymbol{\vartheta})$. The distribution $\tilde{p}(\boldsymbol{\vartheta})$ is chosen based on current knowledge to produce simulated observations close to $\boldsymbol{x}_0$. The training procedure alternates between training a model and drawing new observations from a prior computed based on the current model. The main issue is that using $\tilde{p}(\boldsymbol{\vartheta})$ instead of the true prior biases the posterior probability density. Papamakarios and Murray, 2016 address this problem by multiplying the learned posterior by $p(\boldsymbol{\vartheta})/\tilde{p}(\boldsymbol{\vartheta})$. Lueckmann et al., 2017 use a neural network model and weight the loss assigned to each sample by $p(\boldsymbol{\vartheta})/\tilde{p}(\boldsymbol{\vartheta})$ during training. Papamakarios et al., 2018 build a model for the likelihood instead of the posterior density removing the prior dependency.

Amortization A huge advantage of model-based techniques is the fact that the model needs to be built only once even if we perform inference on several observations. The computational cost of building the model is said to be amortized. In contrast, sampling methods such as approximate Bayesian computation require the whole process to be performed for each observation. This process may require heavy computations when dealing with large scale problems, amortization is therefore of high interest.

4.4 Likelihood-to-evidence ratio based inference

In this section, we focus on the simulation-based likelihood to evidence ratio estimation technique used in this work. This method developed by Hermans et al., 2019 consists in creating a model for the likelihood-to-evidence ratio. Let r denote the likelihood-to-evidence ratio, defined as:

$$r(\boldsymbol{x}, \boldsymbol{\vartheta}) = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})}{p(\boldsymbol{x})}. \tag{4.7}$$

A classifier is trained to discriminate between $(\boldsymbol{x}, \boldsymbol{\vartheta}) \sim p(\boldsymbol{x}, \boldsymbol{\vartheta})$ and $(\boldsymbol{x}, \boldsymbol{\vartheta}) \sim p(\boldsymbol{x})p(\boldsymbol{\vartheta})$.

In practice, first a training set is generated from the distribution $\boldsymbol{\vartheta} \sim p(\boldsymbol{\vartheta})$, $\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{\vartheta})$. At each training step, a batch of $(\boldsymbol{x},\ \boldsymbol{\vartheta})$ pairs is sampled from this dataset and are labeled as 1. The $\boldsymbol{\vartheta}$ samples from those pairs are then permuted such that each $\boldsymbol{\vartheta}$ ends up in a different position than its initial one. Those new pairs are labeled as 0. The classifier is then trained using the binary cross-entropy loss. Let us denote by $s_\theta(\boldsymbol{x}, \boldsymbol{\vartheta})$ the classifier with parameters θ, by L the binary cross-entropy loss, by D a dataset of pairs $(\boldsymbol{x}, \boldsymbol{\vartheta}) \sim p(\boldsymbol{x}, \boldsymbol{\vartheta})$ and by M the batch size. The training algorithm is provided in Algorithm 3.

Algorithm 3 Classifier training algorithm

while not converged do
  $(\boldsymbol{x}, \boldsymbol{\vartheta}) \leftarrow M$ samples $(\boldsymbol{x}, \boldsymbol{\vartheta}) \sim p(\boldsymbol{x}, \boldsymbol{\vartheta})$
  $\boldsymbol{\vartheta}' \leftarrow M$ samples $\boldsymbol{\vartheta}' \sim p(\boldsymbol{\vartheta}')$
  loss $\leftarrow L(s_\theta(\boldsymbol{x}, \boldsymbol{\vartheta}), 1) + L(s_\theta(\boldsymbol{x}, \boldsymbol{\vartheta}'), 0)$
  $\theta \leftarrow \textsc{Optimize}(\theta, \nabla_\theta loss)$
end while
return s_θ

Denoting by $s(\boldsymbol{x}, \boldsymbol{\vartheta})$ the learned classifier, the cross-entropy loss of such a classifier is expressed:

$$L[s] = \int d\boldsymbol{\vartheta} \int d\boldsymbol{x} \underbrace{\frac{1}{2} p(\boldsymbol{x}, \boldsymbol{\vartheta}) \Big[-\log s(\boldsymbol{x}, \boldsymbol{\vartheta}) \Big] + \frac{1}{2} p(\boldsymbol{\vartheta}) p(\boldsymbol{x}) \Big[-\log(1 - s(\boldsymbol{x}, \boldsymbol{\vartheta})) \Big]}_{F(s)}. \tag{4.8}$$

This loss function is minimized for a function $s^*(\boldsymbol{x}, \boldsymbol{\vartheta})$ such that

$$0 = \left. \frac{\delta F}{\delta s} \right|_{s^*} = \frac{1}{2} \left(p(\boldsymbol{x}, \boldsymbol{\vartheta}) \Big[-\frac{1}{s^*(\boldsymbol{x}, \boldsymbol{\vartheta})} \Big] + p(\boldsymbol{x}) p(\boldsymbol{\vartheta}) \Big[\frac{1}{1 - s^*(\boldsymbol{x}, \boldsymbol{\vartheta})} \Big] \right). \tag{4.9}$$

As long as $p(\boldsymbol{\vartheta}) > 0$, this is equivalent to

$$p(\boldsymbol{x}\, \boldsymbol{\vartheta}) \frac{1}{s^*(\boldsymbol{x}, \boldsymbol{\vartheta})} = p(\boldsymbol{x}) p(\boldsymbol{\vartheta}) \frac{1}{1 - s^*(\boldsymbol{x}, \boldsymbol{\vartheta})}. \tag{4.10}$$

Hence, the Bayesian optimal classifier can be expressed as:

$$s^*(\boldsymbol{x}, \boldsymbol{\vartheta}) = \frac{p(\boldsymbol{x}, \boldsymbol{\vartheta})}{p(\boldsymbol{x}, \boldsymbol{\vartheta}) + p(\boldsymbol{x}) p(\boldsymbol{\vartheta})}. \tag{4.11}$$

The likelihood-to-evidence ratio can then be expressed

$$r(\boldsymbol{x}, \boldsymbol{\vartheta}) = \frac{s^*(\boldsymbol{x}, \boldsymbol{\vartheta})}{1 - s^*(\boldsymbol{x}, \boldsymbol{\vartheta})} = \frac{p(\boldsymbol{x}, \boldsymbol{\vartheta})}{p(\boldsymbol{x}) p(\boldsymbol{\vartheta})} = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})}{p(\boldsymbol{x})}. \tag{4.12}$$

Using the learned model s instead of the Bayesian optimal classifier, one obtains an approximation $\hat{r}$ of the likelihood-to-evidence ratio:

$$\hat{r}(\boldsymbol{x}, \boldsymbol{\vartheta}) = \frac{s(\boldsymbol{x}, \boldsymbol{\vartheta})}{1 - s(\boldsymbol{x}, \boldsymbol{\vartheta})}. \tag{4.13}$$

Equation 4.13 is not stable for low values of $s(\boldsymbol{x}, \boldsymbol{\vartheta})$. A workaround is to design the classifier such that it ends with a sigmoid activation. Let us denote by $d(\boldsymbol{x}, \boldsymbol{\vartheta})$ the logit which is the value extracted before the sigmoid activation:

$$s(\boldsymbol{x}, \boldsymbol{\vartheta}) = \frac{1}{1 + \exp(-d(\boldsymbol{x}, \boldsymbol{\vartheta}))}. \tag{4.14}$$

The approximated likelihood-to-evidence ratio can be expressed in the more stable form:

$$\hat{r}(\boldsymbol{x}, \boldsymbol{\vartheta}) = \frac{s(\boldsymbol{x}, \boldsymbol{\vartheta})}{1 - s(\boldsymbol{x}, \boldsymbol{\vartheta})} = \frac{\frac{1}{1+\exp(-d(\boldsymbol{x}, \boldsymbol{\vartheta}))}}{1 - \frac{1}{1+\exp(-d(\boldsymbol{x}, \boldsymbol{\vartheta}))}} = \exp(d(\boldsymbol{x}, \boldsymbol{\vartheta})). \tag{4.15}$$

Finally, an approximation of the posterior density probability function $\hat{p}(\boldsymbol{x}, \boldsymbol{\vartheta})$ can be expressed using the learned likelihood to evidence ratio $\hat{r}(\boldsymbol{x}, \boldsymbol{\vartheta})$ and the tractable prior $p(\boldsymbol{\vartheta})$ as:

$$\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x}) = \hat{r}(\boldsymbol{x}, \boldsymbol{\vartheta}) p(\boldsymbol{\vartheta}). \tag{4.16}$$

5 Parameter estimation from gravitational waves

In this work, we perform Bayesian inference in the context of gravitational waves analysis. The objective is to determine the parameters of a binary black-hole merger given the corresponding observed gravitational wave. We denote by $\boldsymbol{x}$ the gravitational wave and by $\boldsymbol{\vartheta_{all}}$ the parameters of the binary black-hole system.

A particularity of gravitational wave simulation is that the waveform model is deterministic. Using a Gaussian noise model then leads to a tractable likelihood $p(\boldsymbol{x}|\boldsymbol{\vartheta_{all}})$. To evaluate the likelihood, one simulates the waveform with parameters $\boldsymbol{\vartheta_{all}}$. The difference between $\boldsymbol{x}$ and the simulated waveform corresponds to the noise. The probability density of the noise can be computed since the pdf of a Gaussian distribution is tractable. However, in most analyses, one is interested in only a subset of parameters of interest that we denote by $\boldsymbol{\vartheta_i}$, we denote by $\bar{\boldsymbol{\vartheta}}_{\boldsymbol{i}} \in \bar{\boldsymbol{\Theta}}_{\boldsymbol{i}}$ the other parameters. The likelihood considering only the parameters of interest is expressed as:

$$p(\boldsymbol{x}|\boldsymbol{\vartheta_i}) = \int_{\bar{\boldsymbol{\Theta}}_{\boldsymbol{i}}} p(\boldsymbol{x}|\boldsymbol{\vartheta_i}, \bar{\boldsymbol{\vartheta}}_{\boldsymbol{i}}) p(\bar{\boldsymbol{\vartheta}}_{\boldsymbol{i}}) d\bar{\boldsymbol{\vartheta}}_{\boldsymbol{i}}. \tag{5.1}$$

Due to the complexity of the waveform model, the integral cannot be computed analytically making the likelihood **intractable**. For easier notations, we will further denote the parameters of interest $\boldsymbol{\vartheta_i}$ by $\boldsymbol{\vartheta}$.

The intractability of the likelihood makes us rely on simulation-based Bayesian inference techniques. Due to the fast inference goal, amortization is particularly compelling. The model is built at any time **before receiving the gravitational wave**. On gravitational wave detection, the built model is used to perform fast inference. We use the method presented in section 4.4.

In the rest of the section, we explain the different steps of the method. First we define the prior $p(\boldsymbol{\vartheta})$. Then, the training data generation pipeline is presented. Several architectures

of the model are then explored. This section ends with the definition of quantities of interest derived from the posterior model.

5.1 Prior

The first step consists in the design of a prior $p(\boldsymbol{\vartheta})$. For comparison purposes, we use the same priors as B. Abbott et al., 2019. Table 5.1 summarizes the priors used. The used pdfs are defined in Table 5.2. Provided that the model needs to be trained before receiving the signal on which to perform inference, a suitable prior for the coalescence time is impossible to design. As a workaround, we fix this value to a fictive value. Since we use a uniform prior over the sky position, the prior over the relative position of the event to the detectors is uniform no matter the coalescence time prior. Therefore, this does not affect the data generation procedure. At inference time, one has access to the coalescence time. When inferring the sky position, we can, therefore, derive a correction factor to apply to the inferred sky position taking into account the fictive training coalescence time and true coalescence time.

Parameter	Prior	Units
$m_{1,2}$	Uniform(10, 80)	$M_\odot$
d_L	UniformVolume(10, 1000)	Mpc
θ_{JN}	SinAngle(0, π)	rad
Ψ	Uniform(0, 2π)	rad
Φ	Uniform(0, 2π)	rad
$a_{1,2}$	Uniform(0, 0.99)	
$\theta_{1,2}$	UniformSolidAngle	rad
$\phi_{1,2}$		rad
α	UniformSky	rad
δ		rad

Table 5.1 – Prior distribution of gravitational wave simulation parameters

Probability density function	**Description**
Uniform(a, b)	
$f(x) = \begin{cases} \frac{1}{b-a} & x \in [a,b] \\ 0 & \text{otherwise} \end{cases}$	Describes the probability density function of a variable with uniform probability over a given range
SinAngle(a, b)	
$f(x) = \begin{cases} \frac{\sin(x)}{\int_a^b sin(y)\ dy} = \frac{\sin(x)}{\cos(a)-\cos(b)} & x \in [a,b] \\ 0 & \text{otherwise} \end{cases}$	Describes the probability density function of an angle which probability is proportional to its sinus
UniformVolume(a, b)	
$f(x) = \begin{cases} \frac{x^2}{\int_a^b y^2\ dy} = \frac{3x^2}{b^3-a^3} & x \in [a,b] \\ 0 & \text{otherwise} \end{cases}$	Describes the probability density function of the distance from a center O of an object with a position uniformly distributed over a sphere centered in O, the distance being bounded between a and b.
UniformSolidAngle	
$f(\theta,\phi) = \begin{cases} \frac{sin(\theta)}{4\pi} & \theta \in [0,\pi], \phi \in [0,2\pi] \\ 0 & \text{otherwise} \end{cases}$	Describes the probability density function of the spherical coordinates angles of a point uniformly distributed on the surface of a sphere. θ is the polar angle and ϕ the azimuthal angle.
UniformSky	
$f((\alpha,\delta)) = \begin{cases} \frac{cos(\delta)}{4\pi} & \delta \in [-\frac{\pi}{2}, \frac{\pi}{2}], \alpha \in [0,2\pi] \\ 0 & \text{otherwise} \end{cases}$	This distribution is similar to UniformSolidAngle with slight modifications to respect sky location conventions. The azimuthal angle is called right-ascension (α), the polar angle is called declination (δ) and vary in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$

Table 5.2 – Definition of useful probability density functions.

5.2 Data generation process

In this section, we present the procedure to generate the training data. We want to generate samples $(\boldsymbol{x}, \boldsymbol{\vartheta}) \sim p(\boldsymbol{x}|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta})$. This data generation procedure presents several challenges. First, it must be designed to fit as closely as possible to reality. Second, the data must be provided in a form easy to learn by the model, this is done by performing appropriate pre-processing steps. Finally, the data generation procedure must be independent of the gravitational wave on which to perform inference. Indeed, We must train the model before receiving the gravitational wave to amortize the training cost. In a real application, we would, therefore, have no information about the gravitational wave when training the model. Figure 5.1 provides a summary of the data generation procedure. We base our pipeline on the code provided by T. Gebhard and Kilbertus, 2020.

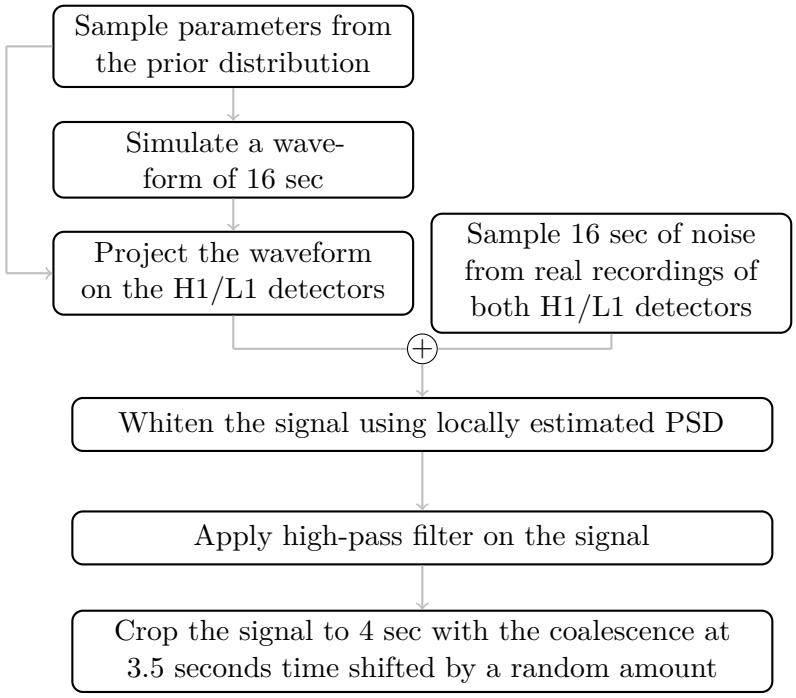


Figure 5.1 – Summary of the synthetic training data generation process

The first step consists in sampling parameters from the prior and then to evaluate the waveform model. For the sake of validation against classical methods, we work with the IMRPhenomPv2 waveform model. Note that the speed of the methods used in this work is independent of the waveform model's complexity. The waveform model could be replaced by any other model. We generate 32 seconds of signal. The generated waveform is then projected on the Hanford and Livingston detectors from LIGO as explained in Section 2.

The next step consists in generating noise that will be added to the projected waveform. As mentioned in Section 2, a common model is to consider Gaussian noise in the frequency domain. This model is conditioned on the noise power spectral density (PSD). This PSD changes over time and is usually approximated on noise recorded around the event. In this application, since we train the neural network before receiving the gravitational wave on which to perform inference, this PSD is not available. The alternative chosen to remove the dependency on the PSD is to make abstraction of a noise model and sample directly from real noise recordings. Using real noise samples also have the advantage to better fit reality. However, doing so is equivalent to marginalizing over the PSDs since we sample noise recordings ranging over a wide period and hence over a wide range of PSDs. This leads to a loss of information compared to methods that do not perform amortization and use the information of the PSD. This analysis focuses on data from the O1 run, i.e. the first running session of the detectors. We sample noise recordings from this run excluding the detected gravitational waves. Consequently, our model is marginalized with respect to those samples and only works on events from this run. However, the method could be extended to other runs by generating synthetic data and training a model for each run.

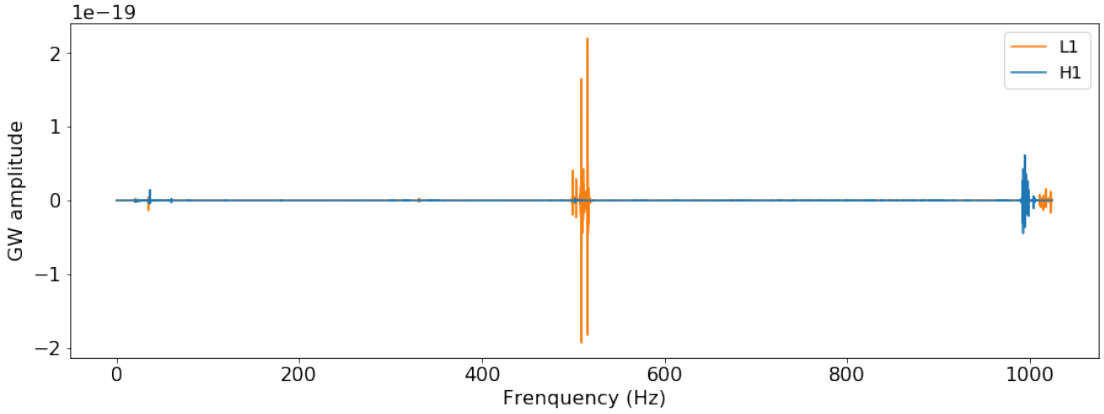
Once the waveform and the noise have been generated and added, we perform pre-processing steps in order to provide easy to learn data. The first pre-processing step consists in whitening the signal. Whitening equalizes the spectrum of the signal. Let us denote by $\hat{X}$ the signal in the frequency domain and by S the PSD estimated on the 32 seconds of the generated signal. The whitened signal $\tilde{X}(\omega)$ at frequency ω is expressed as:

$$\tilde{X}(\omega) = \frac{\hat{X}(\omega)}{\sqrt{S(\omega)}}. \tag{5.2}$$

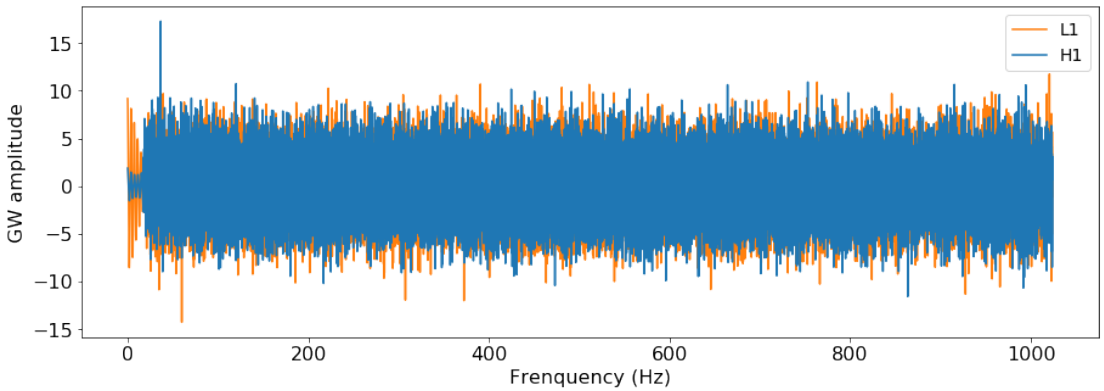
Whitening eases the task of marginalizing over the PSDs since it removes most of the dependence.

Noise is typically high at low frequency. Therefore, a second preprocessing step consist in applying a high-pass filter to the signal. We filter out the frequencies under 20Hz. Figure 5.2 shows the effect of whitening and high-pass filtering on a noisy signal. Amplitude of the different frequencies are more similar and the amplitude of frequencies below 20 Hz is reduced.

Whitening corrupts the edges of the signal making those be of abnormally high amplitude. The signal is therefore cropped to keep 4 seconds of it. This removes the corrupted edges. The 4 seconds window is such that the coalescence time lies at 3.5 seconds to which a small random shift is applied. We sample this random shift uniformly



(a) Before



(b) After

Figure 5.2 – Effect of whitening and highpass filtering

between -0.1 and 0.1 seconds. The objective followed by setting this random shift is to prevent the network from overfitting on the position of the coalescence time in the signal. Indeed, in opposition to simulated data that come with exact knowledge about the coalescence time, real gravitational events' coalescence times are estimated and hence are not known with exact precision.

5.3 Neural networks

As described in Section 4.4, we will build in this section a classifier taking as input a gravitational wave $\boldsymbol{x}$ and some parameters $\boldsymbol{\vartheta}$. This classifier will be trained to discriminate between $(\boldsymbol{x}, \boldsymbol{\vartheta})$ pairs generated following $(\boldsymbol{x}, \boldsymbol{\vartheta}) \sim p(\boldsymbol{x}|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta})$ and $(\boldsymbol{x}, \boldsymbol{\vartheta})$ pairs generated following $(\boldsymbol{x}, \boldsymbol{\vartheta}) \sim p(\boldsymbol{x})p(\boldsymbol{\vartheta})$.

We feed to the network 4 seconds of gravitational wave signal sampled at 2048 Hz. The signal is therefore composed of 8192 samples. Since they provided complementary

information, we use both the Hanford and Livingston detectors. Using both detectors allows mitigating the effect of noise since the noise perceived by those detectors is different. It also allows performing inference on the parameters that act on the projection of the waveform on the detectors. Those parameters are impossible to infer with only one detector. The signals provided by the detectors are provided in two different channels to the model. In the O1 run, only Hanford and Livingston were available. In more recent runs, the Virgo and Kagra detectors are available. The architecture can be adapted by adding more channels to the input. We aim to construct the posterior density of the 2 parameters of interest. The size of the parameters fed to the network is then 2.

5.3.1 Base architectures

We first consider two base architectures: one based on dilated convolutions and the other on a residual neural network. Variants of those architectures are then explored.

Dilated convolutions The first architecture considered is inspired by the one used by T. D. Gebhard et al., 2019 for gravitational wave detection. A sketch of this architecture is shown in Figure 5.3. The network is composed of two parts. The first part uses convolutional layers to compress the gravitational wave into features. The second part is a multi-layer perceptron that takes as input those features concatenated to the parameters and output a real number between 0 and 1. A value close to 1 indicates that the gravitational wave is likely to have been generated by the parameters given as input. In opposition, a value close to 0 indicates that the gravitational wave is likely to have been generated by other parameters. The closer the value to the extremes, the higher the probability of the claim.

Let us consider a M channels convolutional part of the neural network. It is first composed of a convolutional layer with a kernel size of 1 mapping the 2 channels input to a M channels signal. This signal is then fed to 13 dilated convolution layers with M input and output channels each and a kernel size of 2. Starting from 0, the i^{th} convolution layer has a dilation of 2^i. The size of the signal is therefore reduced by 2^i at this layer. We have got

$$\sum_{i=0}^{12} 2^i = 2^{13} - 1 = 8191.$$

At the end of the 13 dilated convolutional layers, the signal, therefore, reduces to a single element with M channels. Note that even if the signal length reduces quickly due to the dilated convolutions, this architecture makes the final element depend on all the input signal points. This is illustrated in Figure 5.3. The black arrows represent the dependencies. The highlighted neuron at the output of the third layer depends on $8 = 2^3$ input values.

The major advantage of this architecture is to be invariant to a shift of the signal neglecting border effects. Convolutional layers apply the same kernel at each place in the signal, those are therefore by nature invariant to shift. The convolutional part reduces the signal to 1 sample, hence the multilayer perceptron has no information about the initial positions in the signal. This property is of high interest in gravitational wave parameter inference since this step follows a gravitational wave detection step which may imperfectly estimate the coalescence time. Having an architecture resistant to shift allows to be less sensitive to such errors. Note that border effects make this shift invariance property not perfectly satisfied. In the rest of this work, we will denote this architecture by *dilated convolutions*.

Residual neural network The second architecture considered is a residual neural network [He et al., 2016]. The skeleton is the same as for dilated convolutions showed in Figure 5.3. The signal is first passed to a convolutional network mapping it to features that are then concatenated to the parameters and fed to a multilayer perceptron. The difference resides in the nature of the convolutional network. We here replace the 13 dilated convolutions by 10 ResNet blocks. We will denote this architecture by *resnet*.

We reduce the dimension of the signal by 2 every 2 ResNet blocks. The final dimension is, therefore, greater than 1 making this network not benefit from the shift invariance property. However, the random shift applied to training data should make the learned network still slightly resistant to shifts.

5.3.2 Conditional parameterization

In this section, we consider improvements of the two architectures described above. The key idea is that it would be valuable to get information about the parameters $\boldsymbol{\vartheta}$ in the convolutional part of the network. Two improvements are considered, one using HyperNetworks and one using conditionally parameterized convolutions. In total, six architectures are then considered, the two base architectures and their two respective variants each.

HyperNetwork The first improvement considered is the use of HyperNetworks [Ha et al., 2016]. We make use of this architecture to integrate information about the parameters $\boldsymbol{\vartheta}$ by feeding $\boldsymbol{\vartheta}$ along with the embeddings to the hyper network. The parameters $\boldsymbol{\vartheta}$, therefore, act on the convolutions, the modified architecture is shown in Figure 5.4.

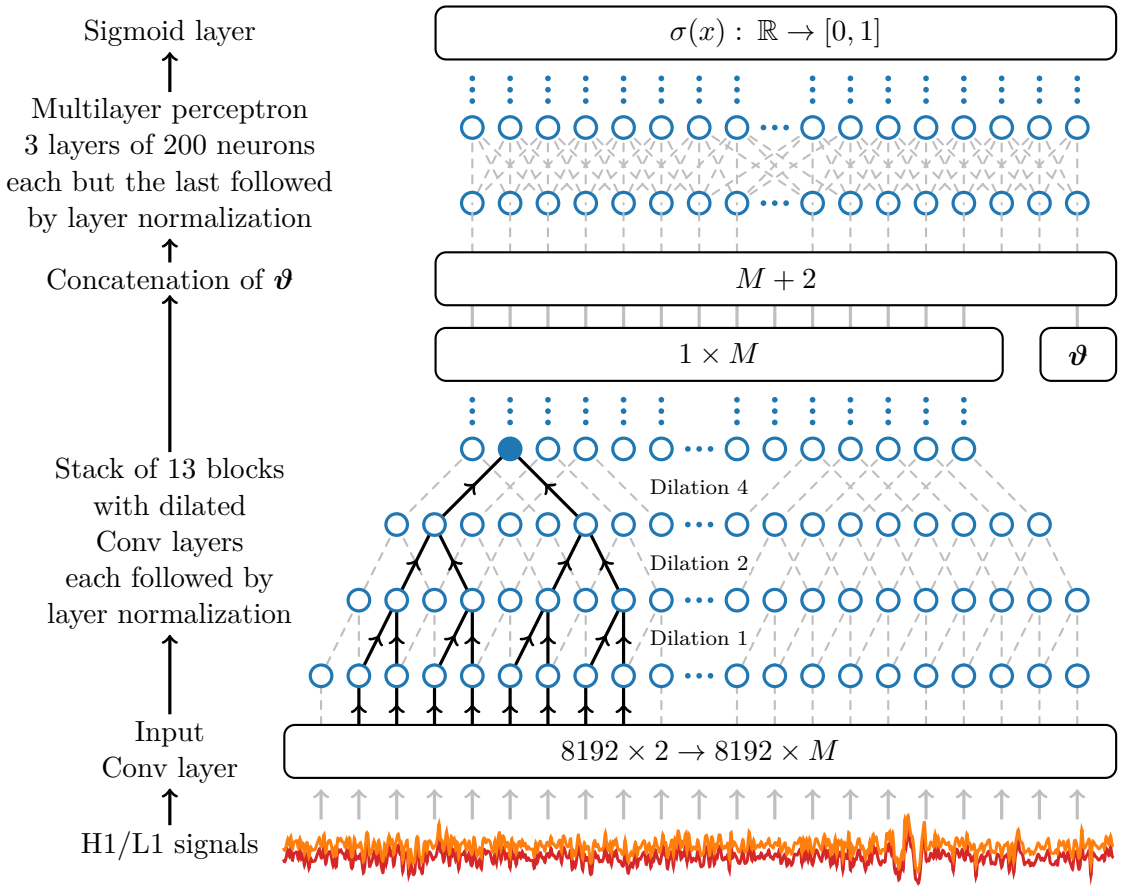


Figure 5.3 – M channels dilated convolution neural network architecture. A first convolutional layer extends the number of channels. Those features are then mapped to M features through a convolutional neural network composed of 13 dilated convolutions. The features are then passed along with parameters $\boldsymbol{\vartheta}$ in a 3 layer multilayer perceptron to produce a value in the range [0, 1] indicating the probability of the signal to have been generated by parameters $\boldsymbol{\vartheta}$.
Figure adapted from T. D. Gebhard et al., 2019.

Conditionally parameterized convolutions Following the same idea of feeding information about the parameters $\boldsymbol{\vartheta}$ to the convolutional network as for HyperNetworks, conditionally parameterized convolutions (CondConv) could be used [Yang et al., 2019].

We take advantage of this framework to feed information about the parameters $\boldsymbol{\vartheta}$ in the convolutional network by making the routing function a function of $\boldsymbol{\vartheta}$. The routing function of an N experts CondConv layer is composed of a small multilayer perceptron taking $\boldsymbol{\vartheta}$ as input and ending by a sigmoid activation which outputs N values between 0 and 1, each layer having its own routing function. The modified CondConv block is showed in Figure 5.5. The final architecture is therefore similar to Figure 5.3 in which the dilated convolutions or ResNet blocks are replaced by CondConvs taking both $\boldsymbol{x}$ and

$\boldsymbol{\vartheta}$ as input.

Feeding information about $\boldsymbol{\vartheta}$ to the convolutional part is not the only benefit of CondConvs. Indeed, the initial objective of CondConvs being to speed up mixtures of experts is highly compelling for our application. It allows reducing the number of channels and to rely on experts to keep a high capacity model and increase inference speed.

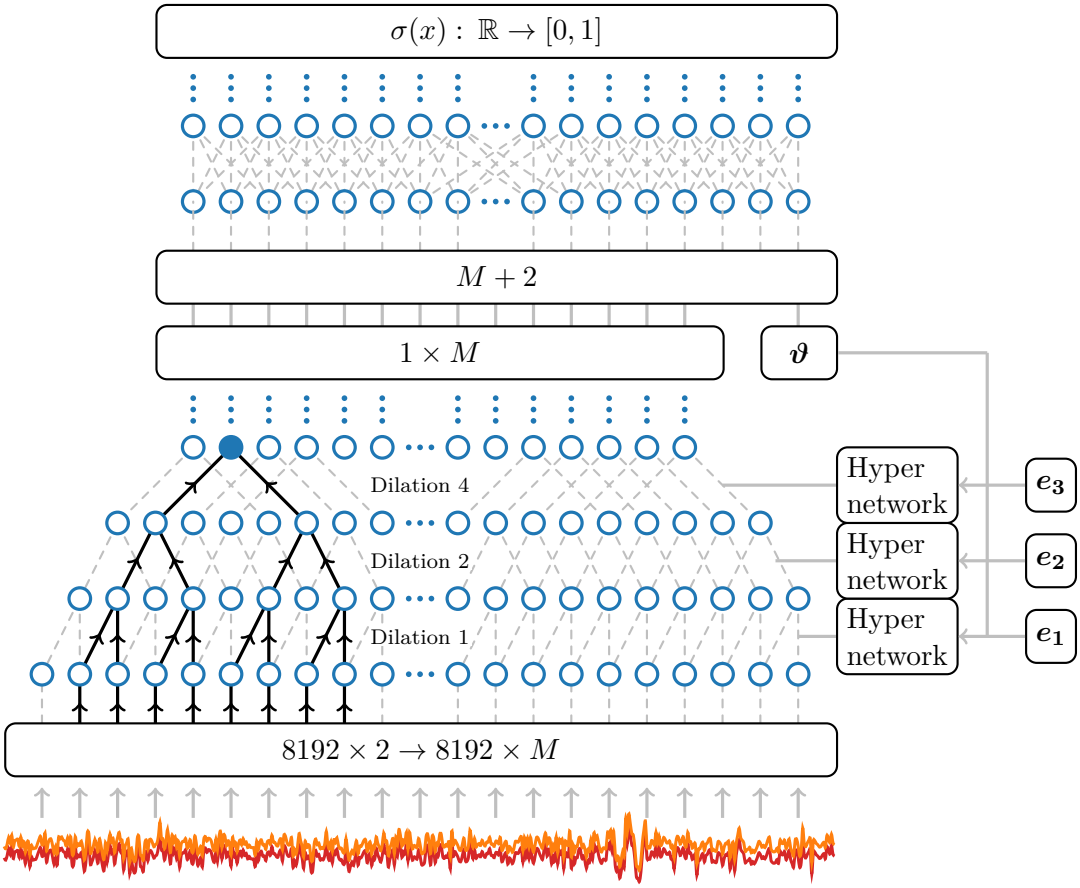


Figure 5.4 – M channels dilated convolutions hyper network architecture. $\boldsymbol{e_i}$ denotes the embeddings.

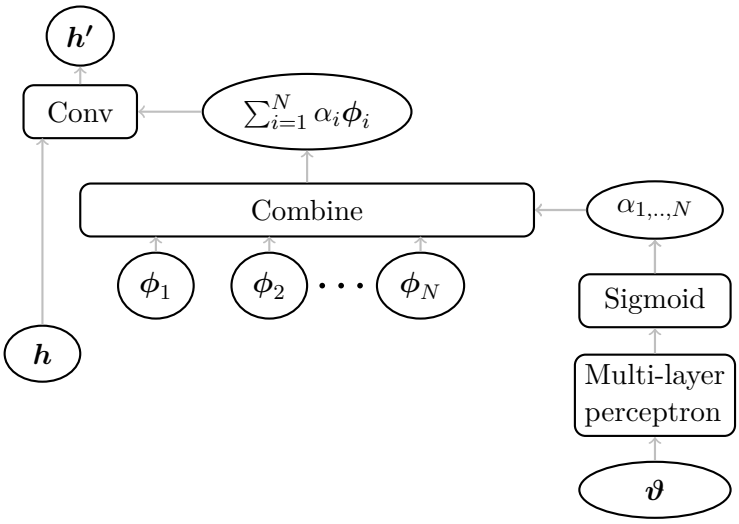


Figure 5.5 – Modified N experts CondConv block to include information about $\boldsymbol{\vartheta}$.

5.3.3 Shift invariance property under normalization

For the shift invariance property of the dilation convolutions based architecture to hold, one needs to be careful with normalization. The normalization steps should also be independent of the position of the points in the signal.

Let us first consider data normalization. To not break the shift invariance property, all signal points should be scaled similarly. To this end, we compute mean and variance considering all signals points as being the same feature. The data mean is shared between all points and is the mean of all points in all samples, the same holds for the variance. They are therefore scaled using the same parameters independently from their position in the signal.

The second normalization step consists in normalizing the features inside the network. Although batch normalization is a classical approach, we will prefer layer normalization that computes shared mean and variance for all features. Classical normalization, however, computes per feature scale (γ) and bias (β). In this work, we use shared scale and bias parameters.

5.4 Inference

Once the classifier is trained, one can evaluate the approximate posterior probability density function $\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})$. However, point estimates of the posterior are not very informative. In this section, we aim to derive informative quantities from the posterior probability

density function. We consider two quantities: the maximum a posteriori estimator (MAP) and credible intervals. All those quantities are defined considering the true posterior $p(\boldsymbol{\vartheta}|\boldsymbol{x})$. We approximate those using the learned posterior $\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})$.

Maximum a posteriori estimator Given an observation $\boldsymbol{x}$, the maximum a posteriori is the combination of parameters that maximize the posterior probability density function. Denoting by $\boldsymbol{\Theta}$ the set of possible parameters, the maximum a posteriori estimator $\hat{\boldsymbol{\vartheta}}$ is expressed:

$$\hat{\boldsymbol{\vartheta}} = \arg\max_{\boldsymbol{\vartheta}\in\boldsymbol{\Theta}} p(\boldsymbol{\vartheta}|\boldsymbol{x}) \simeq \arg\max_{\boldsymbol{\vartheta}\in\boldsymbol{\Theta}} \hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x}) = \arg\max_{\boldsymbol{\vartheta}\in\boldsymbol{\Theta}} \hat{r}(\boldsymbol{\vartheta},\boldsymbol{x})p(\boldsymbol{\vartheta}). \tag{5.3}$$

Credible-intervals A credible-interval of size T over the parameters that generated an observation $\boldsymbol{x}$ is a set of parameters that we will denote by $\boldsymbol{\Theta_T}$ such that the unknown generating parameters fall in this set with a probability T according to the posterior density. Mathematically, $\boldsymbol{\Theta_T}$ is a set of parameters such that:

$$T = \int_{\boldsymbol{\vartheta}\in\boldsymbol{\Theta_T}} p(\boldsymbol{\vartheta}|\boldsymbol{x})d\boldsymbol{\vartheta} \simeq \int_{\boldsymbol{\vartheta}\in\boldsymbol{\Theta_T}} \hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})d\boldsymbol{\vartheta}. \tag{5.4}$$

There exist many intervals that satisfy this criterion. We will consider the smallest one, i.e. the one that contains the parameters of highest probability density.

Quantites approximation Those quantities must be approximated by evaluating the learned posterior $\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})$. To this end, we evaluate it on an equally spaced parameter grid over the prior $p(\boldsymbol{\vartheta})$ support. The maximum a posteriori estimator is then approximated as being the point of highest density. The integral required to compute the credible interval is approximated considering each parameter point has the same density as its closest evaluated point.

Although $\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})$ should integrate to 1 if the model was perfectly modeling reality, it is not the case due to imperfect modeling. To partially correct this imperfect modeling, we normalize the outputs by multiplying the obtained densities by a constant chosen such that $\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})$ integrates to 1.

6 Experiments

This chapter presents the results achieved by the method described in Section 4.4. The first section introduces the experimental setup used to produce those results. The second section evaluates the quality of the different neural network architectures proposed. In the third section, we evaluate the method on data generated using the generation pipeline described in Section 5.2. Finally, we evaluate the method on a real gravitational wave and we compare the credible intervals produced with the one obtained by slower sampling methods.

6.1 Experimental setup

We present here the setup used in the following experiments. We generate 10^6 training samples, 2×10^5 validation samples and 2×10^5 testing samples using the pipeline described in Section 5.2. The network is then trained on 2 GPUs in parallel. To ease training, we perform curriculum learning. We first train the neural network for 5 epochs on 10^5 training samples generated with a UniformVolume(10, 100) luminosity distance prior. This learned model is then used as a starting point for the training on the final dataset. Credible intervals and the maximum a posteriori estimator are then approximated by computing a 200×200 posterior points grid.

During the experiments, we perform inference on transformations of the base parameters defined in Table 2.1. Those transformations are shown in Table 6.1. We work with 4 parameter pairs that are usually considered in standard gravitational wave analyses:

- Maximal and minimal mass $(m^{\text{det}}_{\text{max}}, m^{\text{det}}_{\text{min}})$
- Luminosity distance and inclination angle (d_L, θ_{JN})
- Effective spin and mass ratio (χ_{eff}, q)

- Sky position expressed in right ascension and declination (α, δ)

Parameter	**Symbol**	**Formula**
Maximum mass	$m_{\text{max}}^{\text{det}}$	$m_{\text{max}}^{\text{det}} = \max(m_1^{\text{det}}, m_2^{\text{det}})$
Minimum mass	$m_{\text{min}}^{\text{det}}$	$m_{\text{min}}^{\text{det}} = \min(m_1^{\text{det}}, m_2^{\text{det}})$
Effective spin	χ_{eff}	$\chi_{\text{eff}} = m_1^{\text{det}} a_1 \theta_1 + m_2^{\text{det}} a_2 \theta_2$
mass ratio	q	$q = \frac{\max(m_1^{\text{det}}, m_2^{\text{det}})}{\min(m_1^{\text{det}}, m_2^{\text{det}})}$

Table 6.1 – Parameters tranformations.

6.2 Comparison of neural network architectures

In this section, we compare the performance of the different neural network architectures considered. In total 6 architectures are considered: the two base architectures presented in Section 5.3.1 and their hyper network and conditionally parametrized convolutions (CondConv) variants presented in Section 5.3.2. We use as hyper network a 2-layer neural network such as suggested by Ha et al., 2016. Using an embedding size of 256 gave the best results. When considering the CondConv variant, we also use a 2-layer multi-layer perceptron as routing function. We compare the different architectures on the masses parameter pair and make the assumption that this comparison extends to other parameters. A first comparison is presented in Table 6.2. We evaluate both the binary cross-entropy and the prediction time per sample. The prediction time is measured using an RTX 2080 Ti GPU. The *dilated convolutions* architecture seems to perform better than the *resnet* one. In the remaining, we only consider the *dilated convolutions* architecture.

	Resnet		Dilated Convolutions	
	BCE	Time (ms)	BCE	Time (ms)
Basic 128 channels	0.4733	1.14	0.4216	1.36
HyperNetwork 128 channels	0.4706	0.88	0.4302	1.18
CondConv 4 experts 32 channels	0.4707	0.40	0.4383	0.35

Table 6.2 – Comparison between the resnet and dilated convolutions architectures. The performance is expressed in terms of binary cross-entropy and prediction time per sample and averaged over 2×10^5 testing samples.

	32 channels		128 channels	
	BCE	Time (ms)	BCE	Time (ms)
4 experts	0.4383	0.35	0.4186	1.52
16 experts	0.4309	0.35	0.4193	1.52
32 experts	0.4294	0.35	0.4146	1.52
64 experts	0.4313	0.36	0.4166	1.53

Table 6.3 – Comparison of the CondConv architecture performances with different numbers of experts and channels. The performance is expressed in terms of binary cross-entropy and prediction time per sample and averaged over 2×10^5 testing samples.

We then evaluate how the performance of the CondConv architecture evolves with respect to the number of channels and experts used. Results are shown in Table 6.3. As expected, increasing the number of experts does not increase the prediction time and hence provides an interesting way to increase the model capacity. However, increasing the number of experts does not increase significantly the performance. In our experiment, the 32 experts architecture gave the lowest binary cross-entropy. We will then use 32 experts in the rest of the work.

A final comparison is made in Table 6.4. We first notice that the CondConv architecture seems to perform better than the others while not showing significant improvements. When comparing the performance obtained with different numbers of channels, we notice that increasing the number of channels from 32 to 128 seems to lead to slightly better performance while increasing the prediction time. Increasing the number of channels to 256 does not make the accuracy higher while increasing the prediction time. In conclusion, we would recommend to either use 32 or 128 channels CondConv architectures depending on the time constraints. In the rest of the analysis, we use the 128 channels one that leads to the best accuracy. The prediction time of ± 1.52 ms allows computing a 200x200 posterior points grid in about 1 minute on a single GPU. In addition, it can easily be accelerated using multiple GPU's since this is an easily parallelizable operation. In comparison, MCMC runs we made ran for around 1 week.

Model	BCE	Accuracy	Time (ms)
Dilated convolutions Basic 32 channels	0.4406	78.24%	0.27
Dilated convolutions Basic 128 channels	0.4216	79.44%	1.36
Dilated convolutions HyperNetwork 128 channels	0.4302	79.10%	1.18
Dilated convolutions CondConv 32 experts, 32 channels	0.4294	78.93%	0.35
Dilated convolutions CondConv 32 experts, 128 channels	0.4146	80.01%	1.52
Dilated convolutions CondConv 32 experts, 256 channels	0.4188	79.67%	3.96

Table 6.4 – Comparison of the different neural network architectures. The performance is expressed in terms of binary cross-entropy and prediction time per sample and averaged over 2×10^5 testing samples.

6.3 Evaluation on simulated data

In this section, the method is evaluated on simulated data. The CondConv variant of the dilated convolutions architecture with 32 experts and 128 channels is used. To assess the general performance of the method, we compute several statistics on 1000 simulated samples. Graphical examples of inference made on a part of those simulations are also shown in Figures 6.5, 6.6, 6.7 and 6.8. We compute and evaluate the 50% and 90 % credible intervals as well as the maximum a posteriori estimator.

Credible interval area We compute the area of the contours to evaluate how constrained the parameters are. Figure 6.1 shows the 50% credible interval's areas and Figure 6.2 the 90% credible interval's areas. The credible intervals derived over the masses are of moderated area. However, for some signals, the network has no clue about the parameters. This leads to wide credible intervals. This behavior is well illustrated in Figure 6.5 showing examples of credible intervals over masses on simulated data. Most of those credible intervals are well constrained to the exception of some very wide intervals. Some 50% credible intervals show an area larger than 50% of the prior support, same for the 90% credible intervals. Since we build the credible intervals such that they contain the points of highest density, a $X\%$ credible interval cannot be larger than $X\%$ of the prior support. Such area values are due to an imperfect area estimation procedure. Those credible intervals can however be considered to be of large area. The sky position shows a behavior similar to masses with in average smaller credible intervals, as illustrated in Figure 6.8. The distance and inclination pair and the effective spin and mass ratio pair show few wide credible intervals compared to the masses. This is mainly due to strong priors. High distances are of higher prior than lower ones, the same way low mass ratios are more probable than higher ones. The effective spin and mass ratio pair seems to show two modes, one at low area and one at higher area. Those higher areas are however lower than the ones of other parameter pairs' credible intervals.

Note that high area credible intervals are not necessarily due to a failure of the method. The Bayesian optimal model could also produce high area credible intervals in the presence of a noisy signal. The area of the credible intervals evaluates the ability to perform inference which is different from the quality of the model.

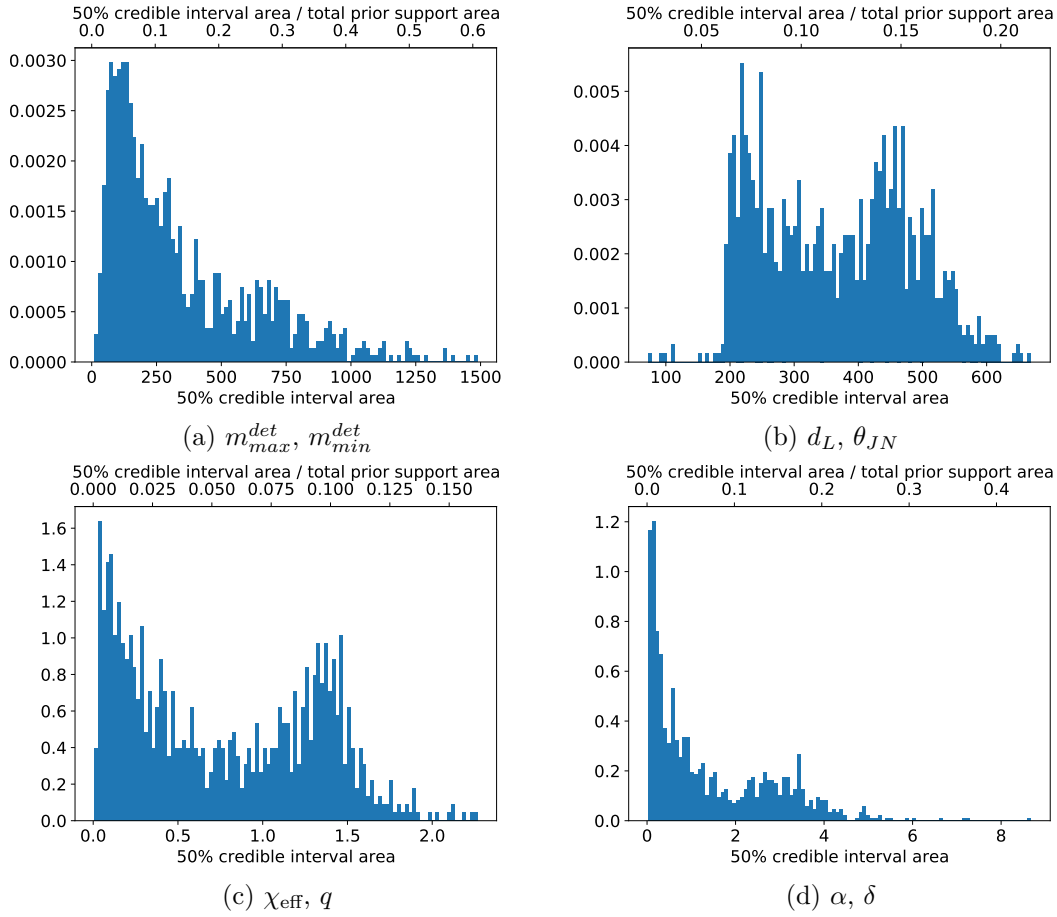
Distance between MAP and exact parameters We then evaluate the quality of the approximated Maximum a posteriori estimator (MAP). We compute its distance to the true parameters. The results are shown in Figure 6.3. In opposition to credible interval areas, the distance between the MAP and exact parameter values does not show a second mode at high distance. This shows that samples that lead to large credible interval areas still get a MAP close to the true parameter values.

	m^{det}_{max}, m^{det}_{min}	d_L,θ_{JN}	χ_{eff}, q	α, δ
50% credible interval hit rate	55.4%	49.4%	55.3%	60.8%
90% credible interval hit rate	90.7%	90.4%	92.2%	93.8%

Table 6.5 – Credible intervals hit rates.

Credible interval hit rate We define the hit rate of a model for a given credible interval by the mean number of times the true parameter values fall inside the derived credible interval. If the credible intervals are well derived, the true value should fall in a X% credible interval X% of the time. The 50% and 90% credible intervals for each considered variable pair hit rates are shown in Table 6.5. We observe that the hit rate is usually slightly too high. This shows that the model is slightly under-confident in its predictions and hence produces large credible intervals than it should. Note that this under-confidence of the model is however moderated and hence shows the ability of the model to produce reliable credible intervals.

Diagnosis of model calibration A calibrated model of the posterior density $\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})$ integrates to 1 over the prior support. We check if the model produced is well calibrated by computing the approximated integral of the posterior conditioned on simulated data. Figure 6.4 shows the distribution of the integral of the inferred posterior over the prior support. This histogram is based on 1000 simulated gravitational waves. The posterior density integrates to a value close to 1 on average, hence no bias is observed. However, it shows a high variance. This illustrates the need to scale the densities to make those integrate to 1 when deriving credible intervals. The variance could potentially be reduced by using ensembles of neural networks, however, this would increase the computational time.



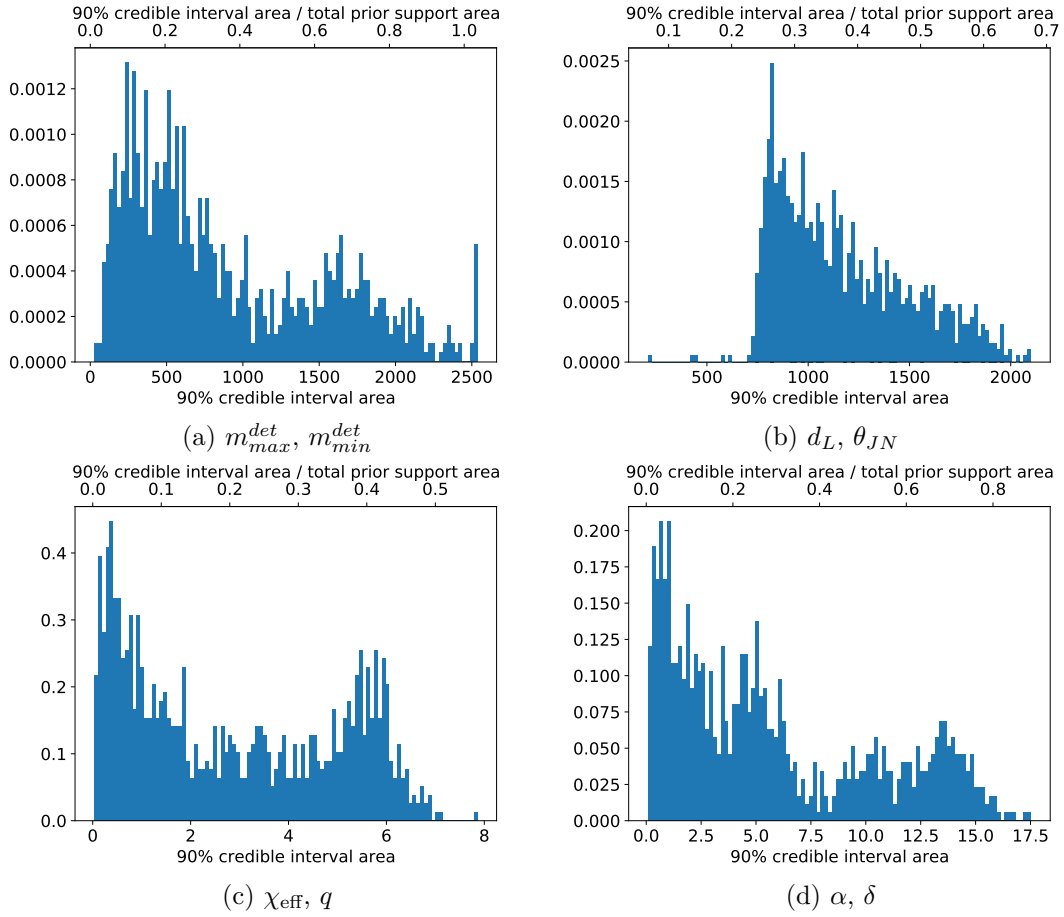
(a) m_{max}^{det}, m_{min}^{det}

(b) d_L, θ_{JN}

(c) χ_{eff}, q

(d) α, δ

Figure 6.1 – Distribution of the inferred 50% credible intervals' area over 1000 samples.



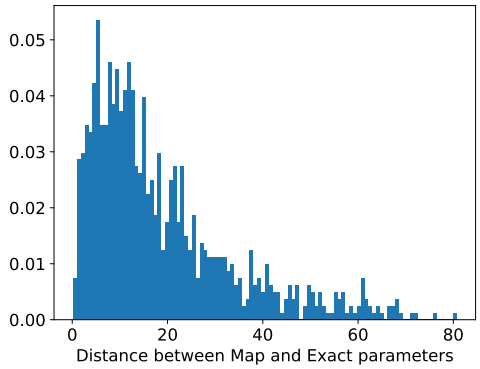
(a) m_{max}^{det}, m_{min}^{det}

(b) d_L, θ_{JN}

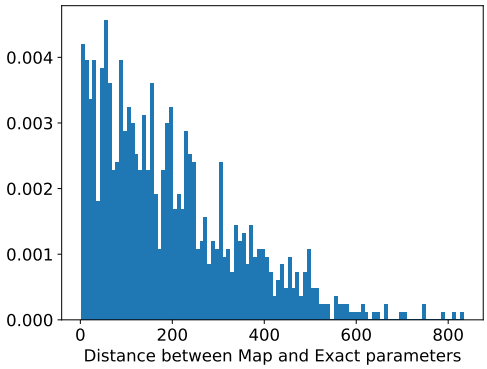
(c) χ_{eff}, q

(d) α, δ

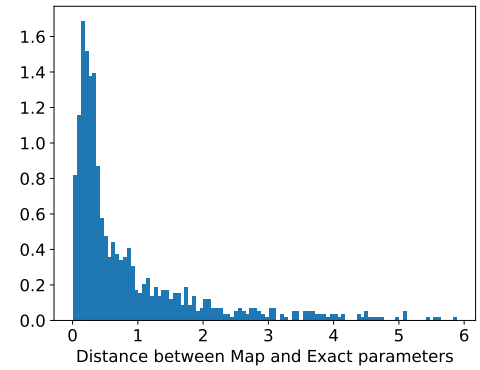
Figure 6.2 – Distribution of the inferred 90% credible intervals' area over 1000 samples.



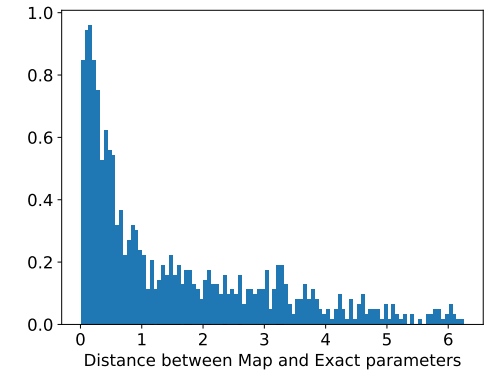
(a) m_{max}^{det}, m_{min}^{det}



(b) d_L, θ_{JN}

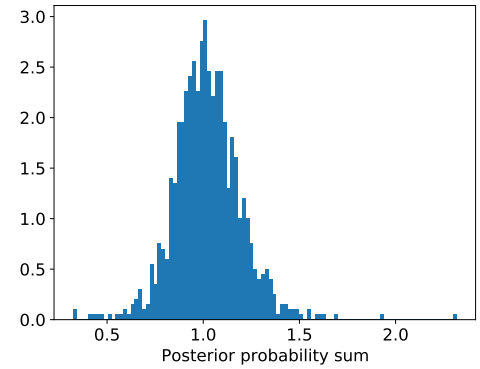


(c) χ_{eff}, q



(d) α, δ

Figure 6.3 – Distribution of distance between the inferred MAP and the exact parameters over 1000 samples.



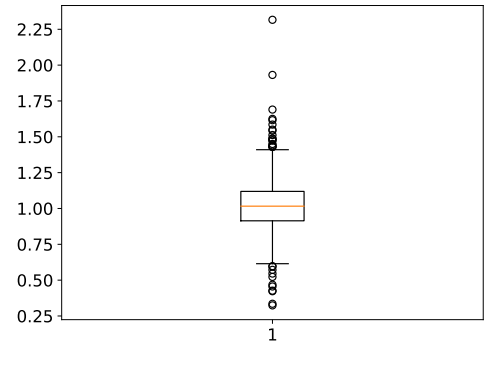


Figure 6.4 – Distribution over 1000 simulated gravitational waves of the integral of the posterior density model $\hat{p}(\boldsymbol{\vartheta}|\boldsymbol{x})$ over the prior support.

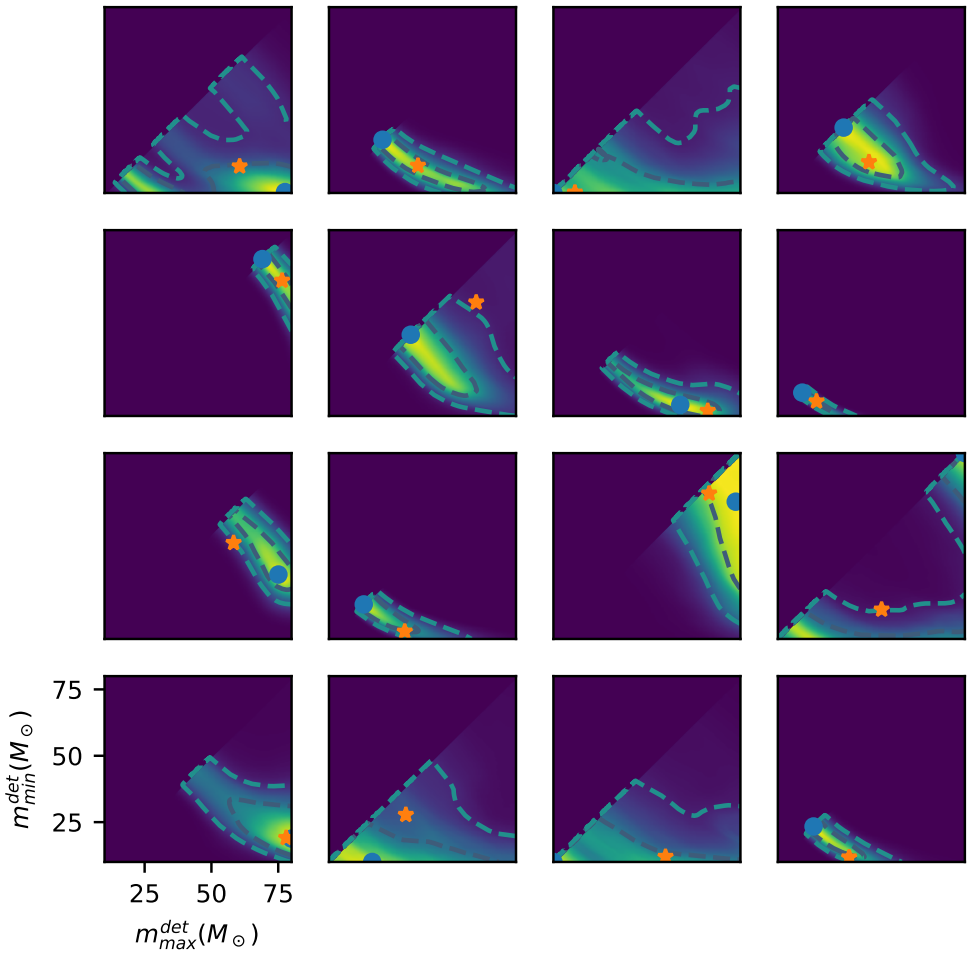


Figure 6.5 – Examples of inference of the masses performed on simulated gravitational waves. The 50% and 90% credible intervals are derived. The blue dot represents the maximum a posteriori estimator and the orange star the true parameters.

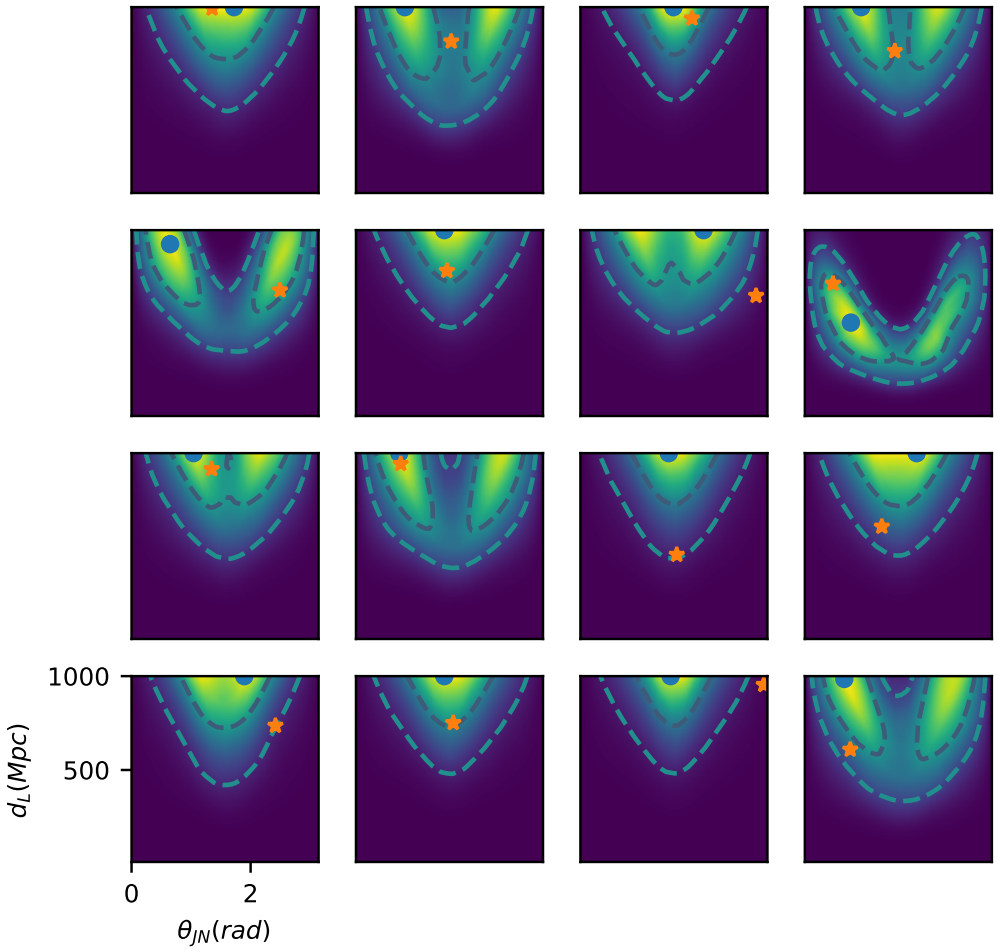


Figure 6.6 – Examples of inference of the distance and inclination performed on simulated gravitational waves. The 50% and 90% credible intervals are derived. The blue dot represents the maximum a posteriori estimator and the orange star the true parameters.

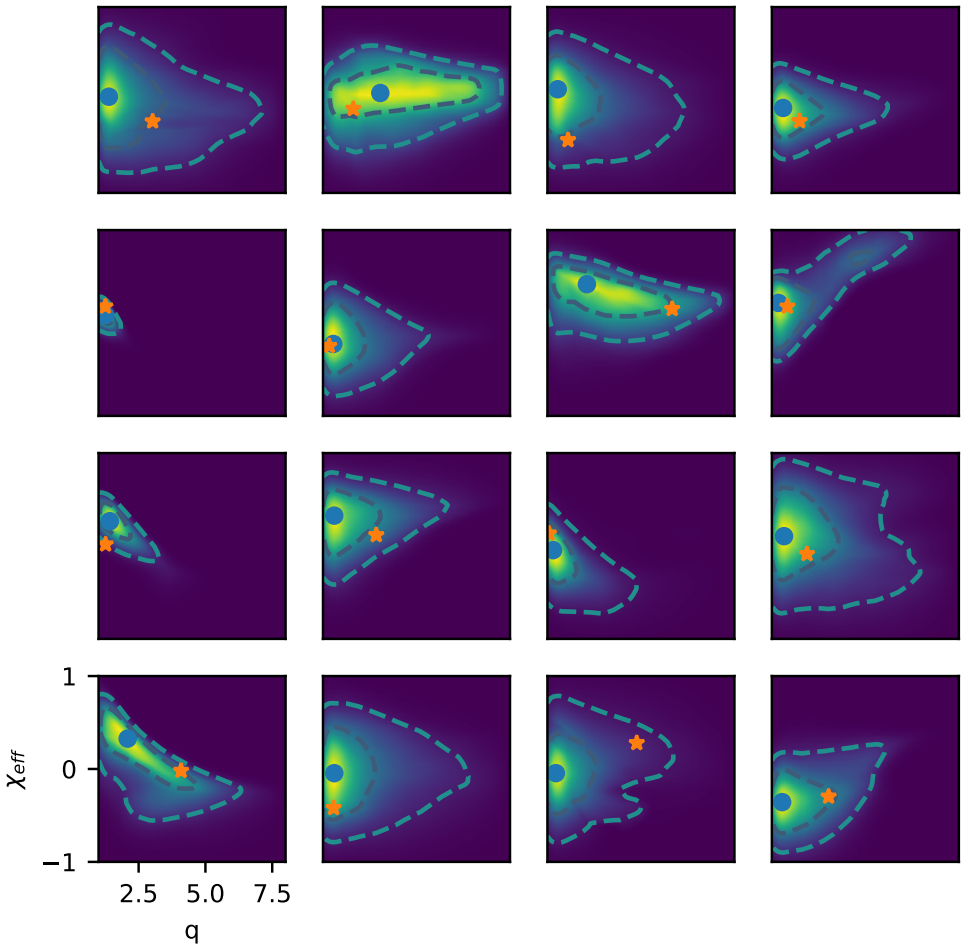


Figure 6.7 – Examples of inference of the effective spin and mass ratio performed on simulated gravitational waves. The 50% and 90% credible intervals are derived. The blue dot represents the maximum a posteriori estimator and the orange star the true parameters.

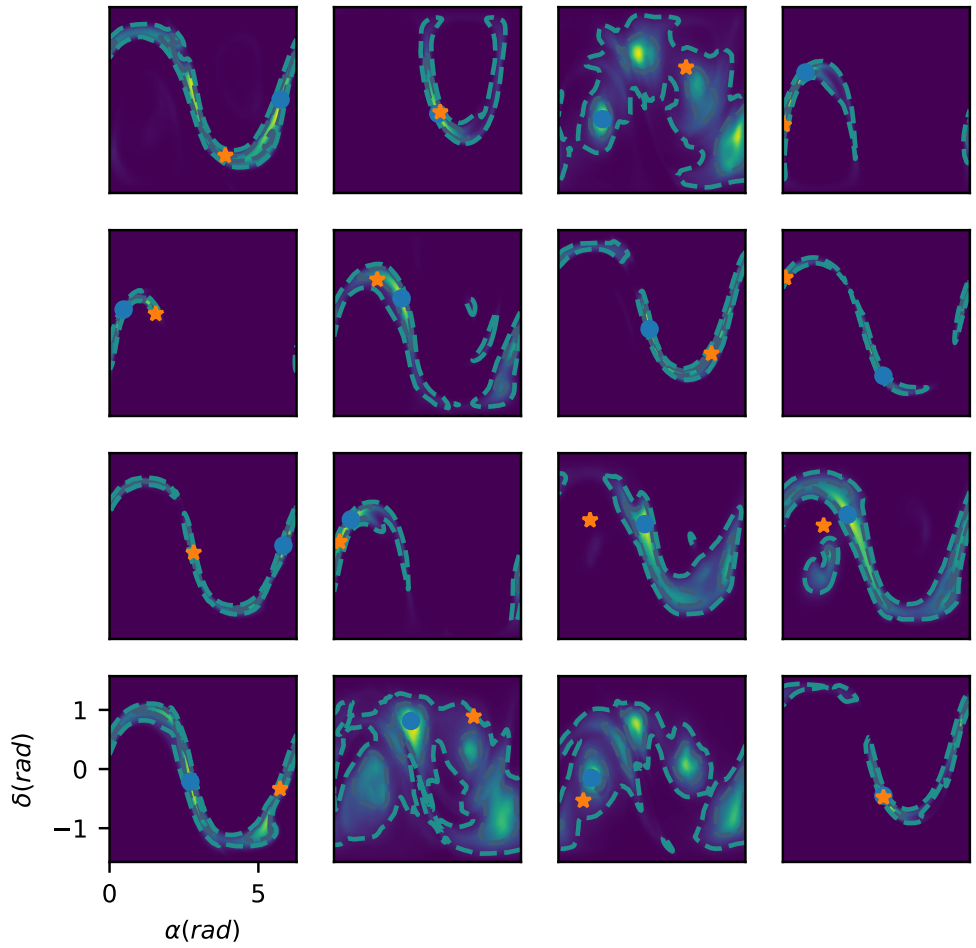


Figure 6.8 – Examples of inference of the sky position performed on simulated gravitational waves. The 50% and 90% credible intervals are derived. The blue dot represents the maximum a posteriori estimator and the orange star the true parameters.

6.4 Evaluation on real data

The method is also evaluated on a real gravitational wave. While testing on simulated data provides a demonstration of the inference capacity. Testing on a real event allows asserting the data generation pipeline fits reality. Since the neural network is trained on noise samples from the O1 run, we evaluate it on the GW150914 gravitational wave which comes from this run.

The true parameters are unknown when dealing with real events. However, since the likelihood-free neural amortization method does not aim to produce better contours than existing ones but to provide a faster way to compute those, comparison can still be made with existing slow methods. We compare our method to results obtained using Monte-Carlo-Markov-Chain (MCMC) sampling. Precisely, we compare to B. Abbott et al., 2019. In their analysis, they model the noise as Gaussian noise in the frequency domain following a given PSD. This model leads to a tractable likelihood when dealing with the full parameters space and hence allows for MCMC methods to draw samples from the posterior distribution. They release samples drawn from this posterior distribution at https://dcc.ligo.org/LIGO-T1800235/public. Credible intervals and the maximum a posteriori estimator are then derived in a similar way to the method we use. The neural network is replaced by a Gaussian kernel density estimator trained on the parameters of interest of the posterior samples produced by MCMC.

Masses Figure 6.9 shows a comparison of the results obtained when performing inference on masses for the GW150914 signal using the MCMC method and the method developed in this work. Both credible intervals are roughly centered at the same place and both maximum a posteriori estimators are close to each other. Our credible intervals are however wider.

Distance and inclination Figure 6.10 shows a comparison of the results obtained when performing inference on the inclination angle and the distance between the merger and earth. Distance credible intervals are very similar using both methods. However, inclination credible intervals show two modes when using the likelihood-free inference technique while only one mode is inferred with the MCMC method. Distinguishing between θ_{JN} and $\pi - \theta_{JN}$ is usually hard since some analyses performed with MCMC on other gravitational waves also show 2 modes [2019].

Effective spin and mass ratio Figure 6.11 shows a comparison of the results obtained when performing inference on the effective spin and the mass ratio. Such as for masses, Maximum a posteriori estimators are close but our method produces wider contour.

Sky position Figure 6.11 shows a comparison of the results obtained when performing inference on the sky position. Sky position is the most constrained parameter pair. The 90% credible intervals produced by our method and MCMC are very close but the 50% credible intervals and MAP lie at different positions. Compared to the other parameters, the size of the 50% credible intervals are however similar.

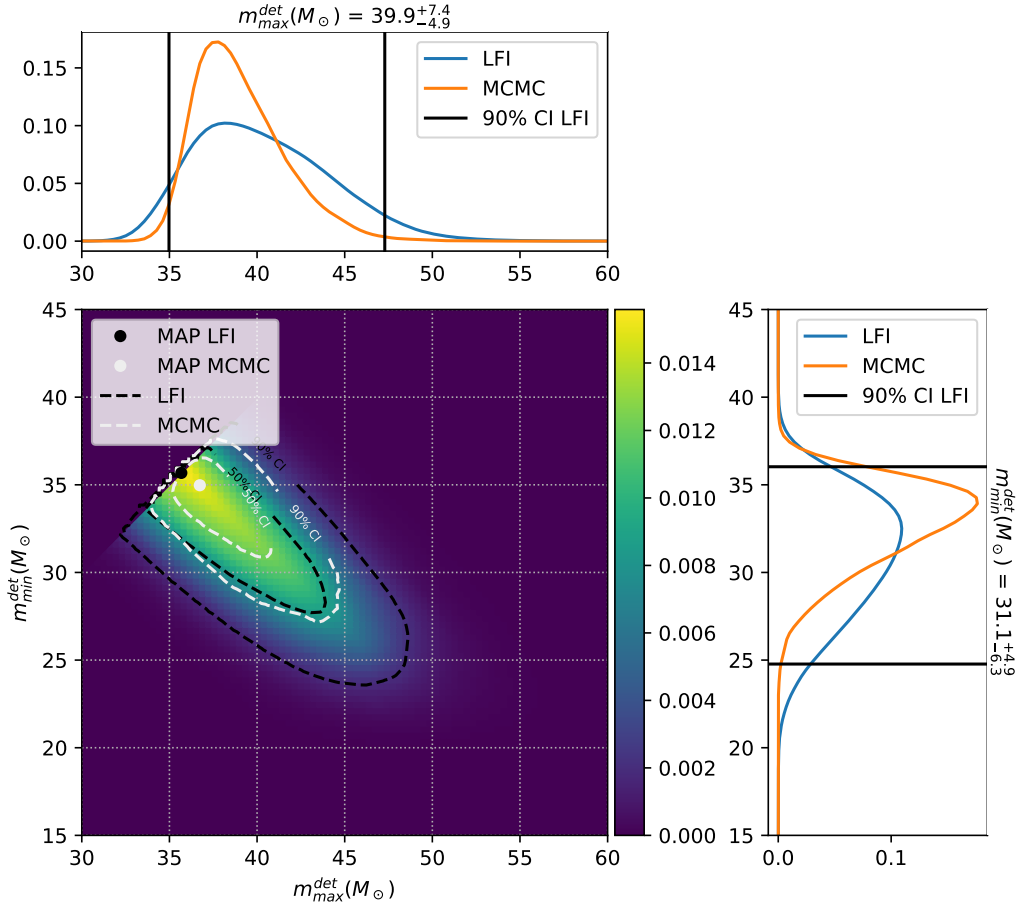


Figure 6.9 – Inference on m^{det}_{max} and m^{det}_{min} for the GW150914 signal. MCMC stands for Markov Chain Monte-Carlo and LFI for Likelihood-Free Inference.

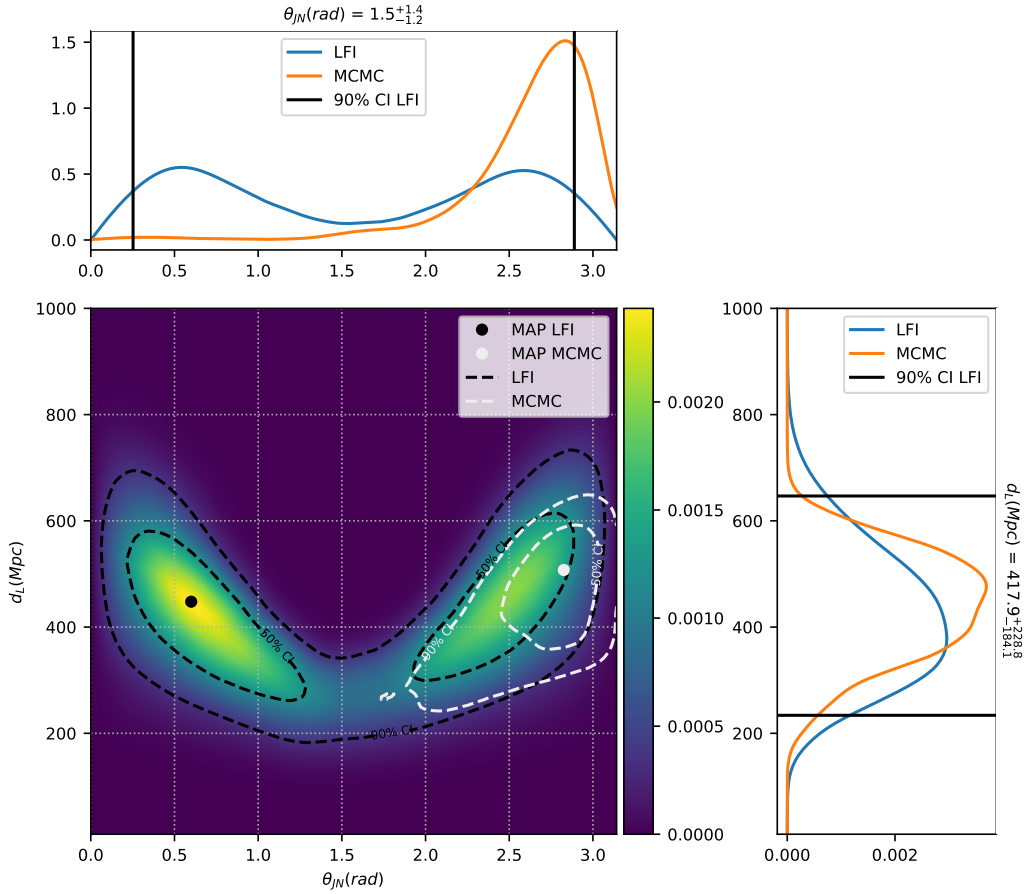


Figure 6.10 – Inference on d_L and θ_{JN} for the GW150914 signal. MCMC stands for Markov Chain Monte-Carlo and LFI for Likelihood-Free Inference.

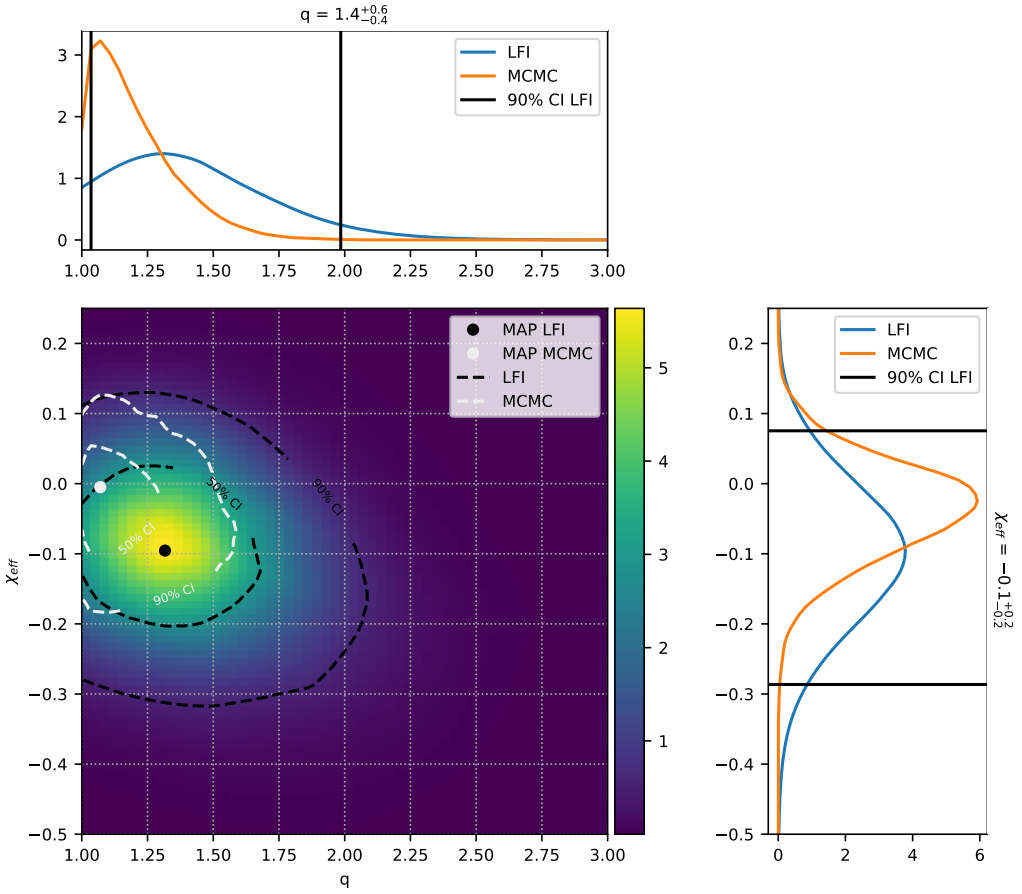


Figure 6.11 – Inference on χ_{eff} and q for the GW150914 signal. MCMC stands for Markov Chain Monte-Carlo and LFI for Likelihood-Free Inference.

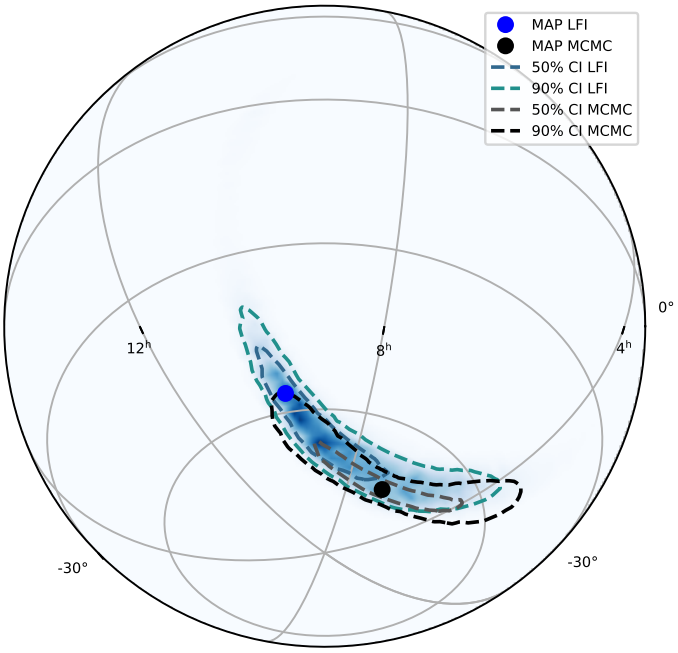


Figure 6.12 – Inference on α and δ for the GW150914 signal. MCMC stands for Markov Chain Monte-Carlo and LFI for Likelihood-Free Inference.

The differences between MCMC and our method's results have multiple causes:

- We train on noise data sampled from the whole O1 run while MCMC is based on a PSD estimated on noise close to the event. Training on the whole O1 run results in a loss of information since we do not include information about the noise around the event. Note that since we want to amortize the training cost and hence train the network before the event happens, information about noise close to the event is not available at training time.
- We use real noise while MCMC uses analytical noise. Therefore, we don't assume a noise model which makes our method able to produce more realistic results.
- The whitening procedure is different. MCMC whiten the gravitational wave using a fixed PSD approximated around the event on which to perform inference. In contrast, we whiten each training gravitational wave with a locally estimated PSD. This could in particular explain the differences in the results obtained for the inclination and the sky position. Information about those parameters lies in the difference in amplitude and time shift between the Hanford and Livingston signals. Whitening with a locally estimated PSD may attenuate the difference in amplitude.
- The methods are different and hence do not have the same accuracy.

7 Related work

Recently, a lot of effort has been made to accelerate the analysis of gravitational waves using machine learning. As shown in this work parameter inference can be highly accelerated with machine learning compared to traditional methods. In parallel to research on parameter inference, using machine learning to accelerate gravitational wave detection which is the task of retrieving parts of the detector signal containing a gravitational wave has also received much attention. We review in this section machine learning methods for both fast gravitational wave detection and parameter inference.

7.1 Gravitational wave detection

Gravitational wave detection is traditionally performed using matched-filtering. It consists in comparing the signal to a set of simulated waveforms. For each signal timestamp and simulated waveform, it outputs the cross-correlation that is as high as the signal is similar to the considered simulated waveform having this timestamp as coalescence time. For this method to be accurate, a large number of simulated waveforms needs to be considered leading to slow computations.

The computational cost can be amortized by using machine learning. George and Huerta, 2018 build a convolutional neural network classifier that is trained to differentiate between signals containing a simulated gravitational wave from a binary black hole merger with noise from signals composed of only noise. This classifier takes fixed length inputs and is applied to the detector signal at several timestamps using a sliding window.

T. D. Gebhard et al., 2019 introduce a fully convolutional architecture allowing to be applied to input data of any dimension. The network outputs a time series where each value indicates if there is a gravitational event at this position. This architecture then allows processing the whole signal once instead of evaluating the network at several timestamps with a sliding window. They also point out the fact that the classifier output should be interpreted carefully since the proportion of the signal containing a

gravitational wave is not the same in training data and in a real detector signal.

Sadeh, 2020 use anomaly detection techniques. They train a recurrent neural network auto-encoder. The encoder takes as input the background before a potential gravitational event, the decoder then outputs the predicted noise following this background noise. This prediction is then compared to the true signal, the more different those are, the likelier there is a gravitational wave. They also use this architecture to directly perform classification based on the decoder output.

Wang et al., 2020 introduce a new neural network architecture inspired by matched filtering. The neural network is first composed of a layer performing match filtering using a limited amount of simulated waveforms. For each timestamp, the maximal resulting value over the considered simulated waveforms is kept. This layer outputs a vector whose size is equal to the initial signal, this vector is then fed to a convolutional neural network.

Krastev, 2020 build a model that is trained to discriminate detectors signals between 3 classes: gravitational wave generated by a binary black hole merger, those generated by a binary neutron star merger and signals without gravitational waves. Since binary neutron stars are of lower mass than black holes, the resulting gravitational waves are weaker and last longer, they then use a time window of 10 seconds. Lin et al., 2020 use a wavelet packet transform which transforms a gravitational wave into a time/frequency matrix to help dealing with binary neutron star mergers. This matrix is then fed to a convolution neural network to perform detection.

Dreissigacker et al., 2019 use deep learning to detect continuous gravitational waves from spinning neutron stars. Those gravitational waves are typically long in the time domain (hours to days) and narrow in the frequency domain, they then take as input the gravitational wave in the frequency domain. Dreissigacker and Prix, 2020 extend this work using multiple detectors and a noise model that takes into account gaps in real data due to detectors not being measuring at all time.

7.2 Parameter inference

Similar to our work, people have also developed machine learning models to accelerate parameter inference. Chua et al., 2019 build a model for the likelihood. They make use of reduced-order modeling which consists in representing a gravitational wave by a small set of features. Under a Gaussian noise model, the likelihood can be expressed in terms of this reduced-order representation. The problem, therefore, reduces to building a model for the reduced representation given the generation parameters. To this end, they train a neural network taking the generation parameters as input and outputting the reduced-order representation. This new model offers several advantages. First, it is faster to evaluate than traditional waveform models. Second, the neural network nature

of the model allows the computation of the derivative of the likelihood allowing the use of derivative-based MCMC methods.

Chatterjee et al., 2019 use deep learning techniques to localize in space a gravitational event. They divide the sky into sectors and train a classifier taking as input a gravitational wave and outputting the sky localization considering each sector as a class. George and Huerta, 2018 perform inference on the parameters by training a model taking as input a gravitational wave and outputting 2 values for the two masses.

A model that only outputs the parameter values minimizing the loss does not give the full posterior distribution. Chua and Vallisneri, 2020 perform inference on the parameters by training a model to output the parameters of a posterior distribution such as a Gaussian mixture. The model takes a reduced-order representation of the gravitational wave as input. They make use of their previous work on building a surrogate for the likelihood [Chua et al., 2019] to efficiently generate training data. Gabbard et al., 2019 use a conditional variational auto-encoder (CVAE) taking as input the gravitational wave to estimate the posterior probability distribution. Those benefit from a higher capacity but are slower to evaluate. Green et al., 2020 make use of autoregressive normalizing flows as density estimator. They compare three types of neural networks. A CVAE, a normalizing flow and the combination of the two. To combine the two, they model all the components of the CVAE with normalizing flows. They show that this last architecture leads to the best results. However, normalizing flows are faster to evaluate than the combined model since they do not require to marginalize over a latent space.

8 Conclusion

In this thesis, a deep learning approach for binary black holes merger parameter inference from gravitational waves is explored. We show how such methods allow a fast analysis of gravitational waves. We also explore several neural network architectures considering both the accuracy and speed of the predictions.

Fast parameter inference We amortize the computational cost by building a surrogate of the posterior using likelihood-to-evidence ratio estimation techniques. This surrogate takes as input a gravitational wave and binary black holes merger parameters and it outputs the posterior probability of the gravitational wave to have been generated by the given parameters. It is then used to perform inference in the order of minutes while classical MCMC techniques take from days to weeks to complete.

A challenge was the design of an appropriate data generation procedure. This procedure had to be designed to provide the detector signals in a way such that valuable information is easy to extract by the neural network. This is done through whitening and filtering. The generation procedure must also be as close as possible to reality. We have chosen to work with real noise samples instead of samples from a noise model. Another challenge inherent to amortization is that the training procedure must be independent of the gravitational wave on which inference will be made. Information about the noise around the event and coalescence time is hence not available at training time. We address this by training the model using noise from the whole O1 run and by whitening with a locally estimated PSD. We also use a fixed fictive coalescence time to train the data and post-process the results accordingly.

We evaluate the performance of our method on simulated gravitational waves and on the GW150914 gravitational wave. Experiments on simulated data assess the ability to perform reliable inference. The experiments performed on GW150914 show that the

method works with real gravitational events. The credible intervals produced are however wider than the one produced using MCMC algorithms. Our method is then a promising complement to classical MCMC techniques. However, further investigation is needed to understand the origins of the differences with MCMC before applying our method to real and routine analysis.

Neural network architectures Another challenge was the design of an appropriate neural network architecture. This architecture must be designed to introduce an inductive bias suited for gravitational wave analysis. We compare two neural network architectures and their HyperNetwork and conditionally parametrized convolutions variants. One architecture is based on ResNet blocks and the other on dilated convolutions. The one based on dilated convolutions shows better results and is designed to be more robust to shifts of the gravitational wave signal. HyperNetwork and conditionally parametrized convolutions provide a way to include information about the input black holes parameters in the convolutional part of the network. For HyperNetworks, we feed the parameters along with the embeddings to the HyperNetwork. When using conditionally parametrized convolutions, we use the parameters as input of the routing function. Conditionally parametrized convolutions show the best accuracy with only a negligible increase in prediction time compared to the classical architecture.

Future work Our method could be improved on several aspects. First, we train the neural network on noise sampled from the whole O1 run and hence do not include information about the noise around the event. We cannot train the neural network on analytical noise generated using a PSD estimated around the event since we want to amortize the training cost and hence train the network before the event happens. A workaround would be to include information about the PSD as an input of the neural network. Introducing a suitable inductive bias for this PSD is, however, not a trivial task.

A second improvement would be to explore better pre-processing steps and in particular different whitening procedures. We whiten the gravitational waves using a locally estimated PSD. This has the advantage to make the noise generated with different PSD to be more similar. It could however also affect the gravitational wave and hence lead to a loss of information about its amplitude. A workaround is to whiten with a fixed PSD. This PSD can either be an analytical PSD or a PSD averaged over several noise samples.

Finally, we highlighted several causes of difference between MCMC and our method. Those are the use of real noise sampled from the whole O1 run instead of Gaussian noise issued from a fixed PSD, the different whitening procedure and the difference of accuracy of the methods. Quantifying the effect of those differences would provide valuable information for the design of further improvements and to assess the quality of

the different methods.

Bibliography

Abbott, B. P., Abbott, R., Abbott, T., Abernathy, M., Acernese, F., Ackley, K., Adams, C., Adams, T., Addesso, P., Adhikari, R., Et al. (2018). Prospects for observing and localizing gravitational-wave transients with advanced ligo, advanced virgo and kagra. *Living Reviews in Relativity*, *21*(1), 3.

Abbott, B., Abbott, R., Abbott, T., Abraham, S., Acernese, F., Ackley, K., Adams, C., Adhikari, R., Adya, V., Affeldt, C., Et al. (2019). Gwtc-1: a gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs. *Physical Review X*, *9*(3), 031040.

Beaumont, M. A., Zhang, W., & Balding, D. J. (2002). Approximate bayesian computation in population genetics. *Genetics*, *162*(4), 2025–2035.

Brehmer, J., Cranmer, K., Louppe, G., & Pavez, J. (2018a). Constraining effective field theories with machine learning. *Physical Review Letters*, *121*(11), 111801.

Brehmer, J., Cranmer, K., Louppe, G., & Pavez, J. (2018b). A guide to constraining effective field theories with machine learning. *Physical Review D*, *98*(5), 052004.

Brehmer, J., Louppe, G., Pavez, J., & Cranmer, K. (2020). Mining gold from implicit models to improve likelihood-free inference. *Proceedings of the National Academy of Sciences*, *117*(10), 5242–5249.

Brehmer, J., Mishra-Sharma, S., Hermans, J., Louppe, G., & Cranmer, K. (2019). Mining for dark matter substructure: inferring subhalo population properties from strong lenses with machine learning. *The Astrophysical Journal*, *886*(1), 49.

Chatterjee, C., Wen, L., Vinsen, K., Kovalam, M., & Datta, A. (2019). Using deep learning to localize gravitational wave sources. *Physical Review D*, *100*(10), 103025.

Chua, A. J., Galley, C. R., & Vallisneri, M. (2019). Reduced-order modeling with artificial neurons for gravitational-wave inference. *Physical review letters*, *122*(21), 211101.

Chua, A. J., & Vallisneri, M. (2020). Learning bayesian posteriors with neural networks for gravitational-wave inference. *Physical Review Letters*, *124*(4), 041102.

Cranmer, K., Brehmer, J., & Louppe, G. (2019). The frontier of simulation-based inference. *arXiv preprint arXiv:1911.01429*, arXiv 1911.01429.

Cranmer, K., Pavez, J., & Louppe, G. (2015). Approximating likelihood ratios with calibrated discriminative classifiers. *arXiv preprint arXiv:1506.02169*.

Diggle, P. J., & Gratton, R. J. (1984). Monte carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society: Series B (Methodological)*, *46*(2), 193–212.

Dreissigacker, C., & Prix, R. (2020). Deep-learning continuous gravitational waves: multiple detectors and realistic noise. *arXiv preprint arXiv:2005.04140*.

Dreissigacker, C., Sharma, R., Messenger, C., Zhao, R., & Prix, R. (2019). Deep-learning continuous gravitational waves. *Physical Review D*, *100*(4), 044009.

Gabbard, H., Messenger, C., Heng, I. S., Tonolini, F., & Murray-Smith, R. (2019). Bayesian parameter estimation using conditional variational autoencoders for gravitational-wave astronomy. *arXiv preprint arXiv:1909.06296*.

Gebhard, T. D., Kilbertus, N., Harry, I., & Schölkopf, B. (2019). Convolutional neural networks: a magic bullet for gravitational-wave detection? *Physical Review D*, *100*(6), arXiv 1904.08693, 063015.

Gebhard, T., & Kilbertus, N. (2020). *Ggwd: generate gravitational-wave data*. https://github.com/timothygebhard/ggwd

George, D., & Huerta, E. (2018). Deep neural networks to enable real-time multimessenger astrophysics. *Physical Review D*, *97*(4), 044039.

Gravitational waves. (2020). Retrieved May 20, 2020, from https://www.cosmos.esa.int/web/lisa-pathfinder/gravitational-waves

Green, S. R., Simpson, C., & Gair, J. (2020). Gravitational-wave parameter estimation with autoregressive neural network flows. *arXiv preprint arXiv:2002.07656*.

Ha, D., Dai, A., & Le, Q. V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*, arXiv 1609.09106.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition, In *Proceedings of the ieee conference on computer vision and pattern recognition*.

Hermans, J., Begy, V., & Louppe, G. (2019). Likelihood-free mcmc with approximate likelihood ratios. *arXiv preprint arXiv:1903.04057*, arXiv 1903.04057.

Holz, D. (2020). *Gravitational wave cosmology lecture 3*. Retrieved May 20, 2020, from http://bccp.berkeley.edu/beach_program/COTB14Holz3.pdf

Introduction to ligo gravitational waves. (2020). Retrieved May 20, 2020, from https://www.ligo.org/science/overview.php

Ioffe, S., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, arXiv 1502.03167.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kobyzev, I., Prince, S., & Brubaker, M. (2020). Normalizing flows: an introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Krastev, P. G. (2020). Real-time detection of gravitational waves from binary neutron stars using artificial neural networks. *Physics Letters B*, 135330.

Ligo's dual detectors. (2020). Retrieved May 20, 2020, from https://www.ligo.caltech.edu/page/ligo-detectors

Lin, B.-J., Li, X.-R., & Yu, W.-L. (2020). Binary neutron stars gravitational wave detection based on wavelet packet analysis and convolutional neural networks. *Frontiers of Physics*, *15*(2), 24602.

Lueckmann, J.-M., Goncalves, P. J., Bassetto, G., Öcal, K., Nonnenmacher, M., & Macke, J. H. (2017). Flexible statistical inference for mechanistic models of neural dynamics, In *Advances in neural information processing systems*.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, *21*(6), 1087–1092.

Papamakarios, G., & Murray, I. (2016). Fast ε-free inference of simulation models with bayesian conditional density estimation, In *Advances in neural information processing systems*.

Papamakarios, G., Pavlakou, T., & Murray, I. (2017). Masked autoregressive flow for density estimation, In *Advances in neural information processing systems*.

Papamakarios, G., Sterratt, D. C., & Murray, I. (2018). Sequential neural likelihood: fast likelihood-free inference with autoregressive flows. *arXiv preprint arXiv:1805.07226*.

Rezende, D. J., & Mohamed, S. (2015). Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*.

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Sadeh, I. (2020). Data-driven detection of multimessenger transients. *The Astrophysical Journal Letters*, *894*(2), L25.

Shawhan, P. S., Brady, P. R., Brazier, A., Cenko, S. B., Juric, M., & Katsavounidis, E. (2019). Data analysis challenges for multi-messenger astrophysics, In *Astronomical data analysis software and systems xxvii*.

Sohn, K., Lee, H., & Yan, X. (2015). Learning structured output representation using deep conditional generative models, In *Advances in neural information processing systems*.

Stoye, M., Brehmer, J., Louppe, G., Pavez, J., & Cranmer, K. (2018). Likelihood-free inference with an improved cross-entropy estimator. *arXiv preprint arXiv:1808.00973*.

Wang, H., Wu, S., Cao, Z., Liu, X., & Zhu, J.-Y. (2020). Gravitational-wave signal recognition of ligo data by deep learning. *Physical Review D*, *101*(10), 104003.

Yang, B., Bender, G., Le, Q. V., & Ngiam, J. (2019). Condconv: conditionally parameterized convolutions for efficient inference, In *Advances in neural information processing systems*.

Yu, F., & Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, arXiv 1511.07122.