Advancing the search for Gravitational Waves using Machine Learning

Hunter Gabbard

Submitted in fulfilment of the requirements for the

Degree of Doctor of Philosophy

School of Physics and Astronomy

College of Science and Engineering

University of Glasgow

October 2021

Abstract

Over 100 years ago Einstein formulated his now famous theory of General Relativity. In his theory he discovered a wave-like solution to the field equations which lead to the beginning of a brand-new astronomical field, Gravitational wave (GW) astronomy.

The Laser Interferometer Gravitational Wave Observatory (LIGO), Virgo, KAGRA (LVK) Collaboration's main focus is the detection of gravitational wave events from some of the most violent and cataclysmic events in the known universe. The LVK detectors are composed of large-scale Michelson Morley interferometers able to detect strain amplitudes on the order of $h \sim 10^{-21}$ from a range of sources including binary black holes, binary neutron stars, neutron star black holes, supernovae and stochastic gravitational waves. Although these events release an incredible amount of energy, the amplitudes of the GWs from such events is also incredibly small.

The LVK uses sophisticated techniques such as matched template filtering and Bayesian inference in order to both detect and infer source parameters from GW events. Although optimal under most cirucumstances, these standard methods are computationally expensive to use. A solution to reducing the computational expense of such techniques is to use machine learning.

In this thesis, I will show how we have employed various machine learning and statistical techniques in order to both optimize the search and show that such machine learning techniques are indeed just as efficient as the gold standard approaches currently used in the collaboration. The first two chapters will give a brief introduction to general relativity as it relates to gravitational wave astronomy, GWs, standard GW data analysis methods and machine learning. Chapters

Contents

List of Tables

List of Figures

Acknowledgements

I would first off like to thank my parents (Lisa and Kurt) for all that they've done for me. Without their constant encouragement growing up and to the present day I would not be where I am now. Additionally, I would like to thank the rest of my family including my siblings (Mallory and Schuyler), my grandfather (Evan Lewis).

To the many advisors that I've had over my short research career including: Prof. Marco Cavaglia, Dr. Florent Robinet, Dr. Soma Mukherjee, Dr. Andrew Lundgren, Prof. Ik Siong Heng and Dr. Chris Messenger.

Declaration

With the exception of chapters 1, 2 and 3, which contain introductory material, all work in this thesis was carried out by the author unless otherwise explicitly stated.

Chapter 1

An Introduction to Gravitational waves, Search Methods and Parameter Estimation Techniques

1.1 Gravitational Waves

Gravitational waves were predicted by Einstein in his theory of general relativity (GR) well over 100 years ago [40]. In his theory, Einstein shows that the more massive an object is, the greater a curvature in spacetime that object creates. This curvature can then have an effect on the motion of objects which encounter it. This is summarized succinctly by John Archibald Wheeler where he states that "Spacetime tells matter how to move; matter tells spacetime how to curve". Einstein formalises the relationship between matter/energy and the curvature of space-time in his field equations as

$$G_{\mu\nu} + \Lambda g_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}, \tag{1.1}$$

where Λ is the cosmological constant (scalar measurement describing the energy density of space), $g_{\mu\nu}$ is the metric which describes the geometric structure of space-time, G is Newton's gravitational constant, c is the speed of light, $T_{\mu\nu}$ is the stress energy tensor which describes the

density, direction, flow of energy in space-time and $G_{\mu\nu}$ is the Einstein tensor defined as

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu} \tag{1.2}$$

where $R_{\mu\nu}$ is the Rici curvature tensor which determines the degree to which matter will tend to change as a function of time (trace of the Riemann tensor) and R is a real valued scalar number describing the degree of curvature.

The Einstein field equations described by e.q. 1.1 unfortunately can only be solved exactly in a limited number of situations. These situations include the Schwartzchild solution for a non-spinning singularity and the Kerr solution for a spinning singularity. If we would like to explore the behavior of more non-linear, time-dependent systems in terms of Einstein's field equations we need to put his equations into a more linear form. This can be done by describing the spacetime metric $g_{\mu\nu}$ in terms of an easy computable known solution in flat spacetime with the addition of some small perturbation.

It turns out that gravitational waves may be defined as small perturbations over the curved background spacetime metric $g_{\mu\nu}$. In Euclidean space, the metric (which is what allows us to compute distances or dot products between two point mass objects), is generally described by the identity matrix in Cartessian coordinates. However, in GR we have to add the time dimension and in flat spacetime this can be described by the Minkowski metric $\bar{g}_{\mu\nu}$ (sometimes also denoted $\eta_{\mu\nu}$) as

$$\bar{g}_{\mu\nu} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.3}$$

Since a gravitational wave is a perturbation (or an additional curvature to the flat spacetime Minkoski metric), we can write the metric with a GW present as

$$g_{\mu\nu} = \bar{g}_{\mu\nu}(x) + h_{\mu\nu}(x), \tag{1.4}$$

where $h_{\mu\nu}(x)$ is the gravitational wave perturbation. After taking the derivative of $g_{\mu\nu}$, we then arrive at the plane wave solution for GWs written as

$$h_{\mu\nu} = \mathrm{Re}[A_{\mu\nu} e^{(ik_\alpha x^\alpha)}], \tag{1.5}$$

where $A_{\mu\nu}$ is the amplitude, k_α is the wavevector and x^α is position in spacetime. Equation 1.5 can be seen to have properties similar to that of an electromagnetic (EM) wave which is typically generated by accelerating charges. However, some major differences are that a GW is generated by accelerating masses which produce higher order quadropole (or multipole) moments and that they are naturally signficantly weaker than EM waves (by 36 orders of magnitude). If we shift equation 1.5 to the Transverse Traceless gauge we arrive at the GW perturbation metric

$$h_{\mu\nu} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & h_+ e^{ic(z-t)} & h_\times e^{ic(z-t)} & 0 \\ 0 & h_\times e^{ic(z-t)} & -h_+ e^{ic(z-t)} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{1.6}$$

where we can clearly see that a GW signal is composed of both "plus" and "cross" polarizations orthogonal to one-another. The effect that a passing GW has on a set of freely floating test particles is illustrated in Figure 1.4. This effect on freely floating test masses is what we try to measure with the LIGO GW detectors. For a full derivation on how GW signals are generated, I refer the reader to [42].

In the following subsection, I will talk about how the detectors function and detect such signals.

1.2 Weber Bar Detector

By the early 1950s technology had progressed enough such that serious attempts at experimentally verifying the existence of GWs by Einstein were possible. One of the earliest and most

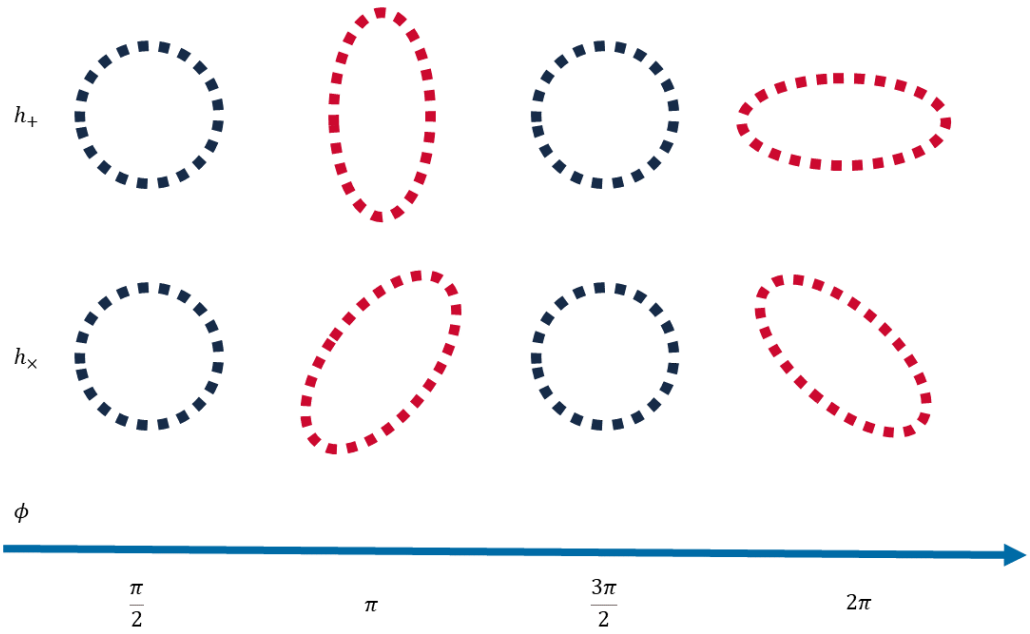


Figure 1.1: An illustration of the h_+ and $h_\times$ polarizations of a GW signal impinging on a set of freely floating test masses as a function of phase ϕ from 0 to 2π.

well-known attempts at doing so was by Joseph Weber at the University of Maryland where he used an instrument known as a resonant-mass detector (Weber Bar Detector) [131]. The Weber Bar operates on the principle that when a GW impinges on the bar (usually made of some type of metal alloy, in this case high Q aluminium) it will cause the bar to stretch and squeeze. If the frequency of the GW is equivalent to the resonant frequency of the bar, a GW would theoretically be detectable up to a strain sensitivity of $h \sim 10^{-16}$. Weber made the claim in 1968 that there was "good evidence" for several detections made by his experiment, but unfortunately no others were able to reproduce his results. Although follow-up results from other independent studies were disappointing Weber's work encouraged many others to build their own improved experiments with breakthrough technological developments at the time and kick-started the subsequent field of GW detection [17].

1.3 The Hulse-Taylor Pulsar

In 1975, Russell Hulse and Joseph Taylor made the first direct observation of a binary pulsar, which subsequently won them the 1993 Nobel Prize in Physics [63]. Importantly, this was also

one of the first pieces of indirect observational evidence for the existence of GWs. This indirect evidence can be attributed to the observation made by both Hulse and Taylor that the period of the binary pulsar appeared to experience orbital decay as a function of time. The orbital decay was thought to likely be attributed to a loss of energy in the system due to GW radiation predicted by GR. The observed period decay as a function of time (in years) is represented in Fig. 1.2. As can be seen in Fig. 1.2, there is a striking level of agreement between the theoretical decay curve predicted by Einstein's GR and the observations made by Hulse and Taylor.

1.4 LIGO Detectors

The LIGO gravitational wave detectors are composed of two detectors, one in Hanford, Washington State and the other in Livingston, Louisiana. There are also other ground-based detectors in Hannover, Germany (GEO), Pisa, Italy (Virgo) and Kamioka, Japan (KAGRA). In addition to ground-based detectors there are eventual plans to build a space-based observatory called the Laser Interferometer Space Antenna (LISA) which will search for super massive binary black holes (among other sources). Each detector can be thought of as a large-scale Michelson-Morley Interferometer composed of two arms orthogonal to each other. Each arm of the detectors is 4km in length. The strain a GW induces on two free point masses can be expressed as

$$h(t) = \frac{2\Delta L}{L}, \tag{1.7}$$

where $h(t)$ is the strain amplitude of the GW as a function of time, ΔL is the relative change in length induced by the GW on the two point masses and L is the baseline length between the point masses without a GW present.

We record the change in length on the two point masses through the use of a ND:Yag 20W laser and large mirrors. Photons emited from the laser pass through a beam splitter and down the two arms of the detectors. The photons then hit end test mass mirrors at both ends and are then caught in a Fabry-Perot signal recylcing cavity in order to effectively increase the baseline length of the arms. Eventually, some photons are released from the Fabry-Perot cavities and return to the beam splitter and are recorded on a set of photodiodes which measure the phase

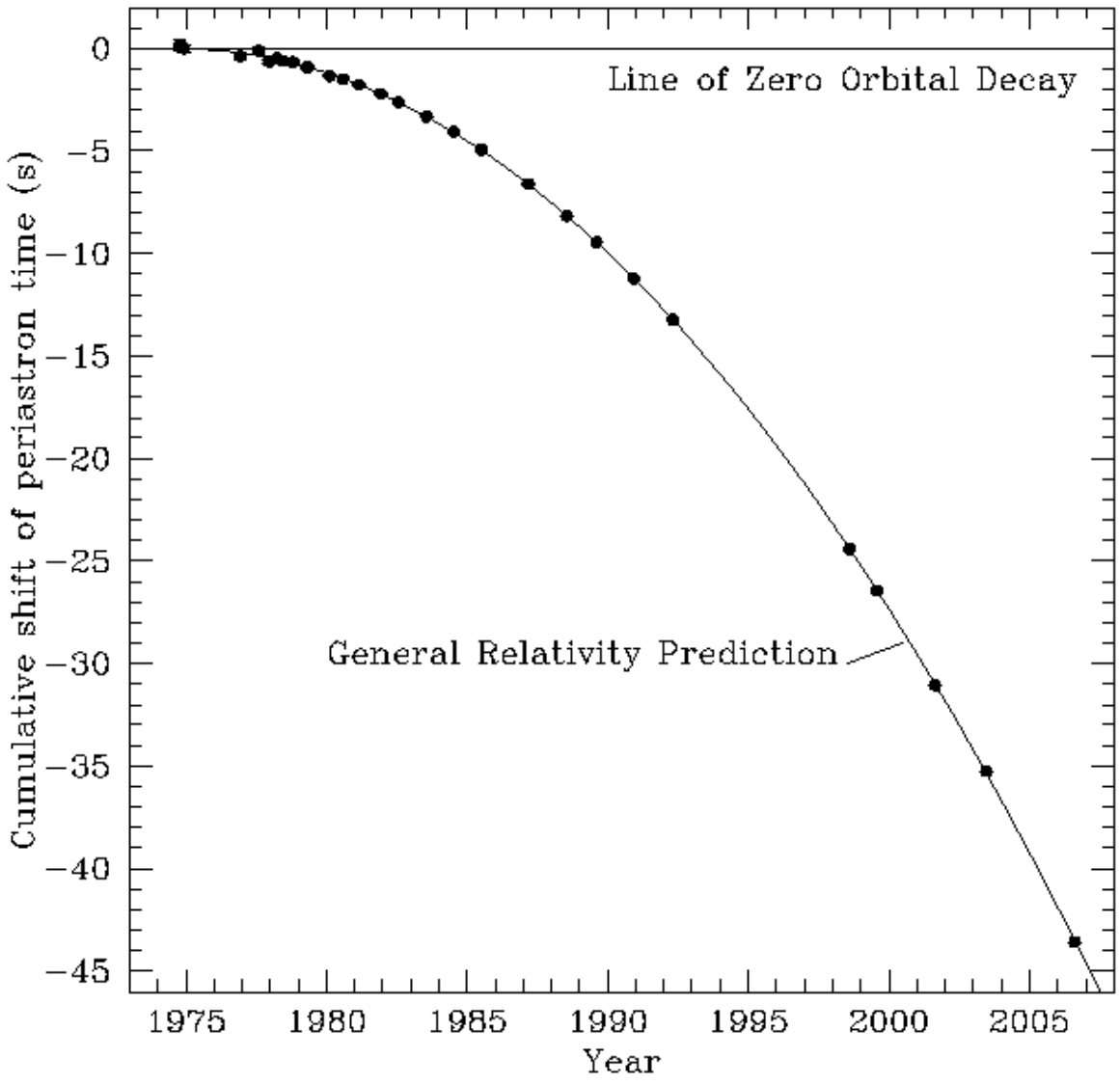


Figure 1.2: This figure shows Hulse and Taylor's observations of binary pulsar `PSR B1913+16` orbital decay as a function of time in years. The orbital decay is quantified by the total cumulative amount the binary system has been offset from it's first observation in 1975 with respect to the binary's perihelion point (black dots). The solid black curve is representative of the theoretical decay curve predicted by GR. `This figure was produced by the authors of` [63].

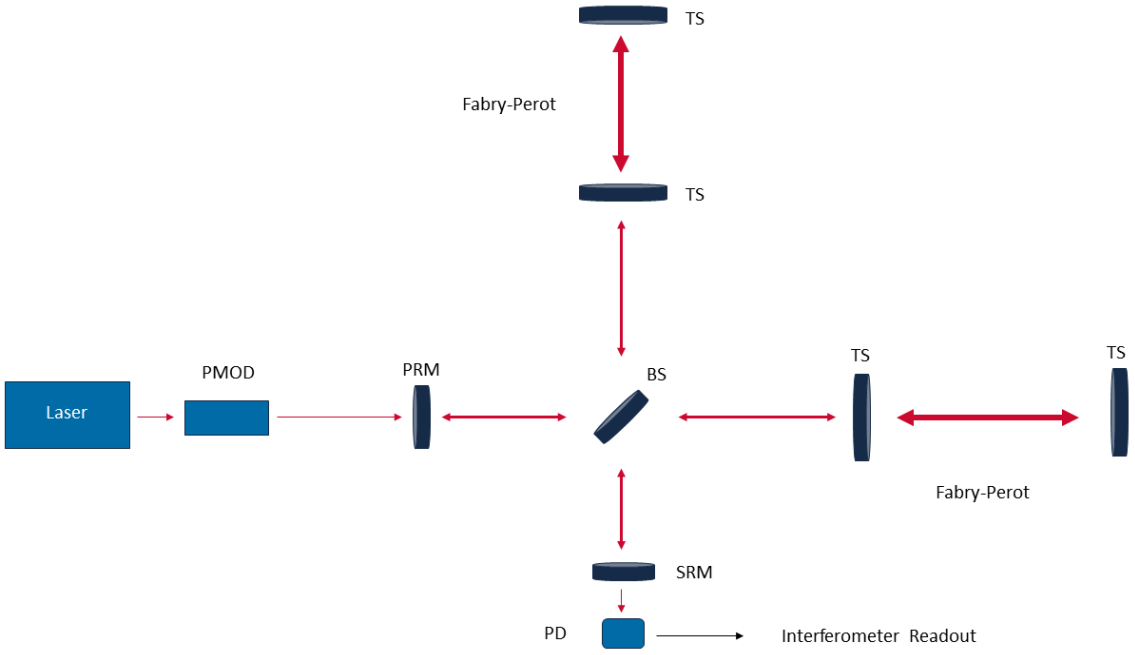


Figure 1.3: An illustration of the LIGO detectors. A 1064nm laser beam is emited from a laser on the left-hand side, it then passes through a phase modulator (PMOD) and enters the power recycling cavity (PRM), this effectively boosts the power of the signal. The laser then passes through a beam splitter (BS), which splits the laser beam path into two seperate parts. Each part travels through an input test mass and hits end test masses at the end of both interferometer arms. The beams are then caught in a Fabry-Perot cavity which acts to extend the distance traveled of the beams, as well as the power. In other words, the cavity "stores" the photons for a long period ($\sim$ 1 ms) which allows a potential GW signal more time to interact with the photons, thus increasing the sensitivity of the interferometer at low frequencies. Some laser light escapes back down both arms and recombines at the BS where the recombined beam passes through a signal recycling mirror (SRM). Finally, the beam hits a set of photodiodes (PD) which produce the interferometer readout which we use to determine whether or not a GW signal is present.

difference between photons from both arms.

Through some algebraic manipulation we can get an estimate on the light travel time difference between the two interferometer arms as

$$\Delta\tau(t) = h(t)\frac{2L}{c} = h(t)\tau_{rt0}. \tag{1.8}$$

where τ_{rt0} is the return trip time down one arm and the phase difference being

$$\Delta\phi(t) = h(t)\tau_{rt0}\frac{2\pi c}{\lambda}. \tag{1.9}$$

Here we can clearly see that the phase difference between the two light signals is scaled by

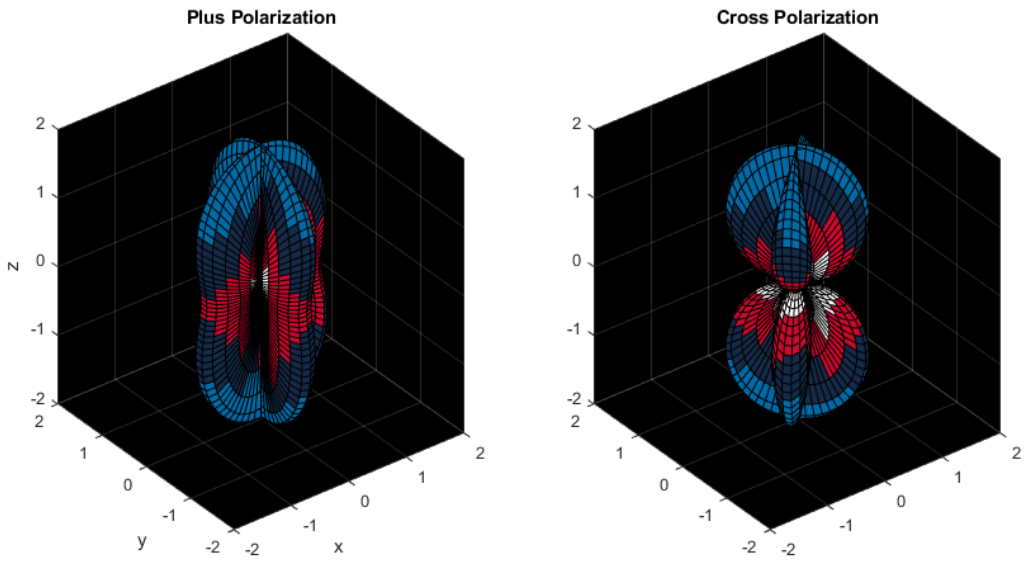


Figure 1.4: An illustration of the LIGO antenna patterns for both the $h_\times$ and h_+ GW polarizations. The detector itself would lie in the x-y plane with one arm along the x-axis and the other along the y-axis.

the length of the interferometer arms L. Because the detectors are tuned such that the photons arriving back at the beam splitter act to destructively interfere with each other (i.e. a dark fringe), there should theoretically be no signal hitting the photodiodes in nominal operating times. When a GW impinges on the detector, it will compress one arm while stretching the other arm. There will then be a detectable difference in phase between the light traveling down both arms. Because of this difference in phase, the light recombining at the beam splitter will no longer destructively interfere and a detectable signal will appear on the photodetectors.

1.4.1 Detector Response

The LIGO detectors are not equally sensitive to all parts of the sky. This is in large part due to their antenna patterns. The antenna pattern of a detector is dependent upon the orientation of the GW source with respect to the detector (with the assumption that it is located at the center of a celestial sphere) given by the expression

$$\mathrm{F}_+ = \frac{(1+\cos^2\theta)\cos(2\phi)\cos(2\psi) - \cos(\theta)\sin(2\phi)\sin(\psi)}{2} \tag{1.10}$$

$$\mathrm{F}_{\times} = \frac{(1+\cos^2\theta)\cos(2\phi)\sin(2\psi) + \cos(\theta)\sin(2\phi)\cos(\psi)}{2} \tag{1.11}$$

where θ is the azimuthal angle, ϕ is the polar angle and ψ is the polarization angle. This is also illustrated pictorially in Fig. 1.4.

Measured GW strain can be expressed as a summation of $h_{\times}$ and h_{+} attenuated by antenna patterns $\mathrm{F}_{\times}$ and F_{+} given by

$$h = \mathrm{F}_{\times}(\theta,\psi,\phi)h_{\times} + \mathrm{F}_{+}(\theta,\psi,\phi)h_{+} \tag{1.12}$$

where h_{+} and $h_{\times}$ are the plus and cross polarizations of the GW signal respectively. As can be seen in e.q. 1.12 , changes in the antenna pattern have a direct impact on the amount of strain measured by the LIGO detectors.

1.4.2 Detector noise

Although a change in phase between the two detector arms can be measured through photon counting, the detectors are also sensitive to non-astrophysical noise sources. These noise sources can drastically affect the sensitivity of the detectors and may also mimic gravitational wave events. Some common noise sources include anthropgenic noise, quantum shot noise, seismic noise and thermal noise.

Seismic noise

Seismic noise largely affects the sensitivity of the detectors in the low frequency regime where an earthquake produces sets of waves which travel both through the earths core/mantel and also along the surface of the earth. When one of these seismic waves hits the detectors it can knock the interferometer arms out of alignment. Closely related to seismic noise (though at different frequencies) anthropogenic noise can result from individuals walking around in the LIGO control rooms or large trucks passing on a nearby highway.

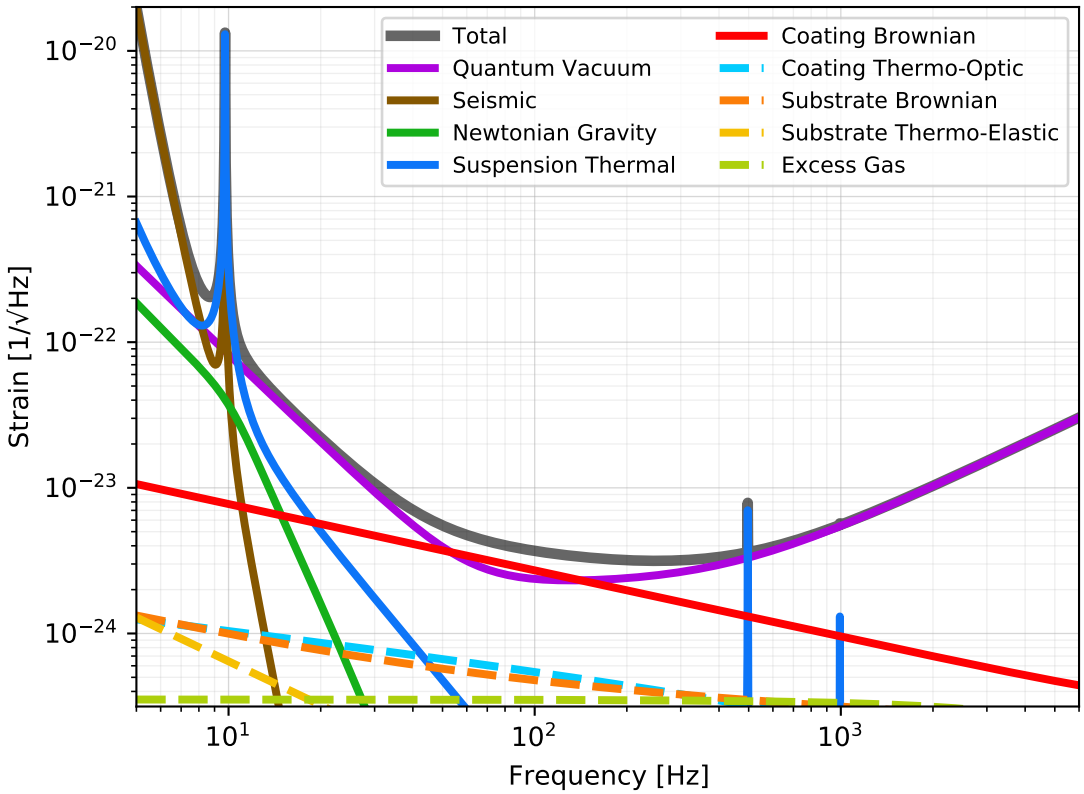


Figure 1.5: The theoretical design sensitivity noise budget curves for Advanced LIGO. As can be seen in the illustration, lower frequencies are largely dominated by seismic motion, mid-range frequencies thermal Brownian motion on the mirrors and higher frequencies quantum vacuum shot noise. Plot was generated using the `pygwinc` computing package [65].

Thermal noise

Thermal noise results from the heat produced by the lasers passing through large coated mirrors in the interferometer. When laser photons pass through these mirrors, the photons subsequently heat up the coating/mirror material. This heat causes the molecules which make up the coating/lens material to behave in random Brownian motion, thus causing the photons to deviate from their intended path.

Quantum noise sources

There are basically two sources of quantum noise in the detectors. The first being quantum shot noise which results from the wave packet-like behavior of light as it travels through a medium. Because of Poisson statistics, we know that the uncertainty on the number of photons that we count arriving at the LIGO photodiodes after recombination at the beam splitter is proportional to the expected number of photons arriving each second. The greater the power of the laser, the greater the quantum shot noise at higher photon frequencies. Shot noise increases by the square root of the optical power of the detector. Shot noise can be expressed mathematically as a limit on the strain sensitivity of the detectors $h(f)_{\mathrm{S}}$ given by

$$h(f)_{\mathrm{S}} = \frac{1}{L}\sqrt{\frac{\hbar c\lambda}{2\pi P}}, \tag{1.13}$$

where L is the arm length of the interferometer, $\hbar$ is Planck's constant, λ is the laser light wavelength, P is the power of the laser and c is the speed of light.

The second type of noise source is quantum radiation pressure noise. Radiation pressure noise arises from the effect of photon momentum transfer onto the test mass mirrors of the detector. When a photon from the detector laser hits a mirror, it transfers some momentum to that mirror. Because all photons do not hit the mirror at the exact same time, there is some variability of pressure exherted on the mirror as a function of time. This can again be mathematically expressed as a limit on the detector strain sensitivity $h(f)_{\mathrm{R}}$ by

$$h(f)_{\mathrm{R}} = \frac{1}{Lmf^2}\sqrt{\frac{\hbar P}{2\pi^3 c\lambda}}, \tag{1.14}$$

where m is the mass of the mirror and f is the frequency of the laser. For a more detailed description of shot noise, see [61].

1.5 LIGO sources

There are a variety of events which can cause significant enough distortions in spacetime to produce GW events. Such events include CBC signals, burst events, continuous GWs from rotating neutron stars and stochastic GWs (leftover echoes from the Big Bang). In this section I will describe these events and how GWs are produced from them.

1.5.1 Compact Binary Coalescencs

CBC signals arise from the collision of massive compact objects moving at high relativistic speeds. Such systems can include binary black holes, neutron star - black hole pairs, binary neutron stars and super massive black hole encounter events.

As the two objects rotate about each other, energy is radiated away in the form of GWs due to the asymmetric motion of the heavy objects. Over the course of millions of years, the two objects will inspiral in towards each other. In so doing the objects will orbit faster, peturbing spacetime to a greater degree and thus releasing more energy in the form GWs. In the final few seconds prior to merger the objects will release a large amount of GW energy collide. If the objects are binary black holes, they will coalesce into a single black hole which will "ring" for a short amount of time. If the objects are binary neutron stars they will typically collide and then produce a large supernovoe event, emiting a large amount of EM light in the process (which importantly can be measured by other telescopes across the spectrum on Earth).

1.5.2 Continuous Waves

CW signals are canonically associated with spinning non-axisymmetric neutron stars. Other more exotic sources can result from clouds of bosons annihilating each around fast-spinning black holes. Neutron stars are the leftover cores of dead stars which have exploded in a super-novae and then collapsed down into an object roughly the mass of our sun and with a radius of a

few km. GW signals are produced from mass quadrupole radiation in the gravitational field and spinning neutron stars can cause such disturbances through cracking and cooling of the crust, internal non-axisymmetric magnetic field flows, and mountains of mass accreted on the surface from a larger companion star [102].

1.5.3 Stochastic Gravitational Waves

Although the more loud signals like GW150914 get the majority of attention these days, there are many more much quieter signals that are present in the detector data streams than are detected with usual CBC detection techniques. To quantify this statement, we can get an exact number for the rate of CBC binary black hole signals occurring at any given time by multiplying the local rate of CBC binary black hole events by a comoving volume (up to a predetermined redshift). If we use a redshift of $z \sim 6$ for this calculation, we wind up with a CBC binary black hole rate of ~ 1 per minute. The vast majority of these events will not lie within the detectibility band of the LIGO detectors and will make up random (or "stochastic") background of events in the noise. The stochastic GW background is not only made up of CBC signals, but any GW event which is low enough in SNR to lie within the general noise background of the detectors.

This stochastic background of GW events could provide valuable insights into fundamental physics if we were able to resolve them from the instrumental/environmental noise background of the detectors. One common example used to explain the potential significance of stochastic GW detection is the Cosmic Microwave Background (CMB). The CMB is the leftover remnants from the Big Bang ($\sim 400,000$ years after) and is the imprint left over by photons at the moment the universe had cooled enough to allow them to travel through space. Unfortunately, we are not able to see any further back since the universe prior to this period was to hot to allow photons to travel. If we were able to resolve the stochastic GW background we would then be able to see even further back in time, right up until the very end of inflation, which is $\sim 10^{-32}$s after the Big Bang. Such accuracy could provide new insights into the early universe, as well as key information about early astrophysical source population properties and formation mechanisms. [103]

1.5.4 Burst signals

When a white dwarf accretes enough mass from a binary companion star such that its total mass exceeds that of Chandrasekhar limit (1.44 solar masses) it may explode in the form of a type 1a supernova. Other supernova events can occur from a variety of other sources such as a supermassive red giant star when it runs out of fuel in its core at the end of its life and implodes on itself. When a supernova occurs, it is possible that GWs may be released in some form if the event happens in a non-symmetric manner, such as if the star is rotating at great speed with minor surface mountain distortions. Because of the complex dynamics of supernova events, it is difficult to model these events and as such commonly use what is known as an unmoddeled search to look for these events. Additionally, there is also the possibility that we may find as yet unknown types of GW sources due to the model independent agnostic nature of this type of search [106].

1.6 Search methods for gravitational wave signals

In this section, I will describe several methods used by the LIGO-Virgo Collaboration (LVC) to search for signals from CBC, GW, burst and stochastic GWs.

1.6.1 CW search method

There are estimated to be roughly $\sim 10^8 - 10^9$ neutrons stars in our own Milky Way Galaxy, of which only ~ 2500 have already been observed by the wider scientific community. One method for detecting CW signals is to go after these already known 2500 neutron stars and perform a targeted search.

Targeted search

The GW strain given by a non-axisymmetric spinning neutron star is typically defined by

$$h_0 = \frac{4\pi^2 G}{c^4}\frac{I_{zz} f^2}{d}\varepsilon, \tag{1.15}$$

where I_{zz} is the moment of intertia around the z-axis of the neutron star, f is the GW frequency (roughly proportional to the star spin frequency) and ε is the elipticity. In a targetted search, it is generally assumed that the sky position, frequency, spin and other parameters are well known. There is also a middle-ground option of performing a directed search where we only assume to know the sky location very well. Nominally, a fast Fourier transform (FFT) is first performed on the data afterwhich a transformation of some sort is then applied (frequency-Hough, "stack-slide", PowerFlux) in order to map the observed data to source properties. After this mapping performed, periods of excesses power above a pre-defined threshold are identified over a large number of frequency bins. There are of course many different variations on the methods listed above cited here [23].

All-sky search

In contrast to both a targeted and a directed search, all-sky searches impose the least amount of constraints on the observable parameter space. Generally speaking, an all-sky search is performed by first breaking up an observed piece of time series data to be analysed into many smaller time segments. These time segments are then analysed coherently, after which the results for each time segment may be recombined in an incoherent manner. This is otherwise known as a semi-coherent search.

In order to combine results from coherent segments incoherently, there are many methods which have been developed over the past several years. Such methods include: stack-slide, time-domain F-statistic, frequency-Hough, Viterbi, and Powerflux. For a full description of these and other methods for combining coherent segments, I refer the reader to the following manuscript [129].

1.6.2 Burst search method

As mentioned previously, burst-like signals are typically not modeled due to the complicated nature of the event and the possibility of detecting as yet unknown signals. Because of the uncertainty in trying to search for this type of signal, we use an unmodelled search which essentially trying to distinguish between unknown signal and noise. Since the search is unmodelled

we don't necessarily have a set of template waveforms which are exactly described by a deep knowledge of numerical relativity, post-Newtonian dynamics or GR. Rather, members of the LIGO burst search team use template waveforms which roughly imitate events of varying statistical properties who have bursts of excess power for a short duration of time. Some of the signal template types include but are not limited to:

- White Nosie burst: A signal which is of zero value and then jumps up to a predefined value in a step-like fashion before sharply falling back down to zero in a step-like fashion.
- String cusp: A delta-like function which is meant to mimic GWs from cosmic string cusps.
- Sine-Gaussian: A sinosoidal waveform which increases exponentially in amplitude, peaking, and then subsequently decreasing exponentially in amplitude. This is meant to mimic the GWs from a ringdown-like event.

Once we have our template bank, the first step of the burst search pipeline is to identify candidate transient events to set aside for further analysis. This is done using the WaveBurst algorithm which tries to identify clusters of times in the data stream which are coincident with each other. The coincidence is determined by times of excess power in the detector, identified through a linear wavelet packet decomposition. This process is done at several different frequency resolutions in order to not miss any features at specific frequency bands. If there is a cluster of times which are identified as having a significant amount of excess power, we then check for whether or not a similar cluster is also present across other detectors which were operational at the time. The significance of the burst-like candidate event is determined by essentially computing the geometric average of the excess signal power across all detectors in multiple frequency bins (with respect to the background noise). There are further complications which may be added to the method described above, but the reader may find more information on more robust signal consistency/coincidence methods in the following manuscript [6]. The Burst search is currently verified for accuracy by injecting signals using the template bank described above.

1.6.3 Stochastic search method

The stochastic search method largely involves attempting to piece out GW stochastic noise from the detector environmental/non-astrophysical noise. We do this by using a method known as cross-correlation analysis. Cross correlation is essentially a (sliding) dot product between two vectors of numbers, where we observe peaks in the dot product where the series are most similar. For LIGO we look for cross correlations across multiple detectors. In LISA we have to use special null combinations of the data where the GW signal is suppressed enough to where there is nearly only detector noise present in the data stream. The suppression is possible to do through the symmetrized Signac method [62]. If this can be done, one can then do an excess power measurement in order to see if there is a signal present in the noise. Alternately, it may also be useful to have a solid understanding of how to model the noise in the LISA detectors in order to distinguish between GW signal and noise.

Normally, it is nearly impossible to distinguish between detector noise and the stochastic GW background in a single detector, which is why we look for correlations between data across multiple detectors. This is made under the assumption that instrumental noise is not correlated between multiple detectors. It turns out that if we take the expected value of the cross correlation of the data between two detectors, we end up with the the variance of the stochastic GW signal (since it is by definition supposed to be present in the data of both detectors).

In order to get a more rigorous statistical estimate of the cross correlation value, one can use a number of frequentist and Bayesian approaches. The frequentist approach involves using a maximum likelihood ratio to compare two models. The two models being a noise+signal model and a noise-alone model. The frequentist approach then computes the maximum likelihood value for both models and estimates a ratio between the two values (divide the maximum likelihood estimate of the signal+noise model by the likelihood estimate of the noise model). The likelihood function which is normally used is a Gaussian multivariate function.

The Bayesian approach again use the likelihood function for each model described above, but additionally need the joint prior probability distributions for both the signal and noise parameters. A flat prior is usually used for the signal variance and Jeffrey's prior is used for the noise variance. One can then construct a join posterior probability distribution for each model in order

to figure out the likelihood of signal being present in the noise. We can quantify this exactly by computing the Bayes factor between the signal+noise model and the noise alone model.

1.6.4 CBC search method

The output of the LIGO gravitational wave detector is a time series produced by the resulting phase shift of two lasers after recombination on a photodiode. Because our detector is not perfectly isolated from all non-astrophysical sources, the output of the detector will be a function of both a gravitational wave impinging on it, as well some noise

$$s(t) = h(t) + n(t), \tag{1.16}$$

where $h(t)$ is the combined strain from a gravitational wave induced on the interferometer and $n(t)$ is the noise contribution. We generally assume that the noise is both stationary and Gaussian, although in reality the detector noise can be non-Gaussian. For those cases where we have non-Gaussianity we run various glitch identification tools and techniques to identify problematic areas of the detector data output.

Assuming Gaussian noise, our problem then becomes, how does one distinguish noise from actual signal? Fortunately, the problem of extracting low SNR signals from the background is not uncommon in physics and the field of statistics and may be accomplished through a technique known as matched template filtering.

Matched template filtering

Given the current level of sensitivity of the detectors, we will be dealing within a regime where the GW signal we are trying to detect will often be burried far below the overall background of the detector noise. Within the context of CBC signals, it is known that as two compact objects inspiral in towards each other that both the frequency and strength of the resultant GW signal increases with time. We can take advantage of the fact that we generally have a good understanding of the form of $h(t)$. If we know exactly the form of $h(t)$, we may construct

a simple filtering technique whereby we multiply the output of the detector $s(t)$ by $h(t)$ and integrate over some observation time. We can describe the noise contribution to the noise-free signal $h(t)$ as

$$\frac{1}{T}\int_0^T dtn(t)h(t) \sim (\frac{\tau_0}{T})^{(1/2)} n_0 h_0. \tag{1.17}$$

In E.q. 1.17 we see that as we increase the observation time T, we see that the overall value of the function tends to zero. So, given enough observation time, we can effectively filter out the contribution from the noise. The keen reader will spot one issue. Given that we may not have an infinite amount of observation time due to the duty cycle of the detectors and the fact that amplitude of the gravitational wave signal is not constant as a function of time, how can we make this filtering technique more optimal given finite observation time T. We define an optimal signal-to-noise ratio

$$SNR_{opt} = 4\int_0^\infty df \frac{|\bar{h}(f)|^2}{S_n(f)}, \tag{1.18}$$

where $|\bar{h}(f)|^2$ is is the inner product of the ideal signal template with itself and $S_n(f)$ is the single-sided noise power spectral density (PSD). In matched template filtering, we generate many thousands of templates and compute the optimal SNR of each. The best matching will contain the highest optimal SNR for that event. If the SNR is above the detection threshold (usually defined as 8), we say that there is a candidate gravitational wave signal present in the data.

Because the LIGO detectors can sometimes have non-Gaussian noise artefacts, these "glitches" can mimic high SNR events. The high SNR glitch events typically contain lots of power within a very small frequency band, distinguishing themselves from GW events. To quantify the likelihood of a candidate high SNR event being a real GW signal, we use a statistic known as the chi-squared test. Specifically, the chi-squared test checks the statistical differences between the distribution of power across a set number of frequency bins in the observed data with frequency

bins from the template which has been determined to match best with the data according to the matched template filtering algorithm.

Frequency bins are chosen such that each individual bin contributes an equal amount to the total matched filter SNR. The Pearson chi-squared statistic is defined as

$$\chi^2 = \sum_{i=1}^{n} \frac{(O(i) - E_i)^2}{E_i} = N \sum_{i=1}^{n} \frac{(O(i)/N - p_i)^2}{p_i}, \tag{1.19}$$

where χ^2 is a finite value which determines how well your observations O match your expected values E, N are all your frequency bins, and p_i is the level of matching in the ith frequency bin. To put this in terms of GW matched template filtering we can plug in our variables describing our observed data and expected template values into the expression to get

$$\chi^2 = p \sum_{i=1}^{p} \left[\left(\frac{\rho^2_{\cos}}{p} - \rho^2_{\cos,i} \right) + \left(\frac{\rho^2_{\sin}}{p} - \rho^2_{\sin,i} \right) \right], \tag{1.20}$$

where p are our frequency bins, $\rho_{\cos}$ and $\rho_{\sin}$ are the matched filter SNR values of the best matching template in both polarisations. The higher the value of E.q. 1.20, the less likely that the observed data matches the best matching template. The SNR previously determined by the matched filtering algorithm is then weighted by the chi-squared value. If the weighted match filter SNR lies below a user predefined value, the candidate event is discarded and not considered for further analyses [124]. The final detection statistic is the quadrature sum of the chi-squared weighted matched filter SNR across all detectors where the event was seen.

Furthermore, we need to ensure that events which we observe in one detector are generally consistent with what we would expect to find in other active GW detectors around the globe. If an event has been identified in one detector, it must also appear in other active detectors within the expected window of time it would require for the GW to travel from one detector to another (roughly equivalent to the speed of light). The amount of time varies depending on sky location of the event, but generally it is expected to arrive at another detector within 15 milliseconds (with added uncertainty due to measurement uncertainty). The same event in multiple detectors are also expected to have the same matching template from the matched filtering algorithm.

Now that we have a detection statistic in the form of the weighted quadrature summed SNR

across multiple detectors described above, we would like a method for determining the statistical significance of this statistic. This can be done by determining the false alarm rate of the detectors as a function of the weighted detection statistic. Because we don't necessarily know how the noise of the detector is going to be distributed due to the varied non-Gaussian nature of the detectors, we have to experimentally measure the false alarm rate. We can measure this through the use of time slides. Time slides are applied by computing the number of detected triggers outside of the light travel time coincidence window (15 milliseconds) for a pre determined number of observation windows. The triggers are calculated using the best matching template for the original candidate event. The time slide windows used with the best matching template, the more sensitive the estimate on the false alarm rate. The false alarm rate is then used in order to assign a p-value to the candidate GW event.

1.7 Bayesian Inference

It is not only important that we detect a GW event, but also that we infer the underlying properties of that event in the form of its source parameters (i.e. component mass, distance, sky location, etc.). In LIGO, the tried and true method for inferring source parameters is done through Bayes inference. Bayesian inference is itself derived from Bayes theorem. Bayes theorem was first proposed by Reverend Thomas Bayes in the 18th century and in it he formulated a new paradigm for thinking about the laws of conditional probability.

Bayesian probability, is a fundamentally different way of interpreting statistics from the more traditional frequentist approach. For a frequentist, an unknown parameter of interest θ is often considered to be a fixed quantity. A frequentist would determine the value of θ through sampling of observational data until enough observational data had been accumulated to form a distribution. From this distribution, a frequentist would then be able to determine confidence intervals on their estimate of θ. On the other hand, a Bayesian does not consider the unknown parameter to be a fixed value, but rather a random variable which is described by a probability distribution. θ may then be inferred through direct application of Bayes theorem. I will describe Bayes theorem in detail and refer the reader to [88] for a more detailed description on frequentist

inference.

To describe succinctly, Bayes theorem states that one can infer the distribution of an unknown parameter by computing the likelihood of a given observation, scaled by our prior belief on the distribution of the unknown parameter. To put it in the context of GW astronomy, given an observed gravitational wave event and some prior assumptions, the posterior can be describes the source parameter values of that signal while also taking into account the uncertainty added by the signal being burried in noise. The posterior can be expressed as

$$p(\theta|d), \tag{1.21}$$

where $p(\theta|d)$ is the probability of the source parameters of the signal (θ being a continuous variable), given some observed data (in the form of a time/frequency series). We assume that the integral over the total posterior is normalised such that

$$\int d\theta p(\theta|d) = 1. \tag{1.22}$$

According to Bayes theorem, we can write the posterior as

$$p(\theta|d) = \frac{p(d|\theta)p(\theta)}{p(d)}. \tag{1.23}$$

Where each term in Eq. 1.7 can be described as

- $p(d|\theta)$: The likelihood of our observable d given source parameters θ.
- $p(\theta)$: Our prior belief on the distribution of source parameters θ.
- $p(d)$: A normalisation factor called the evidence which is integrated over all possible parameters θ.

The prior $p(\theta)$ is largely informed by our understanding on the formation channels of GW sources and our level of understanding on the general physics which govern events. For example, we would intuitively think that the mass of an object should always be positive, so will set

the priors such that the component masses of a GW source must always lie between two positive values. However, if we aren't as knowledgeable about a particular parameter θ, we might try choosing a relatively uninformative prior by employing something like a broad uniform distribution. Choice of prior can also be incredibly influential on the conclusions drawn on some GW parameters such as spin. For further interesting discussions on prior choices see Salvatore et al. [127].

The way in which we define the likelihood $p(d|\theta)$ (our certainty on the observed data) is essentially up to the practitioner. For GW astronomy, we typically define a likelihood which assumes that the detectors operate under Gaussian noise-like conditions. The Gaussian-noise likelihood function can written as

$$p(d|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(d-\mu)^2}{\sigma^2}\right), \tag{1.24}$$

where σ is the detector noise, d is the observed data and μ is a template GW waveform parameterized by source parameters θ.

The evidence $p(d)$ can be defined as

$$p(d) = \int p(d|\theta)p(\theta)d\theta. \tag{1.25}$$

The evidence is usually referred to as the marginal likelihood. Because we are marginalising over all parameters θ, we can think of the evidence as essentially a normalising factor. Since the evidence is just simply a normalising factor and it is prohibitively expensive to compute this factor (because we are integrating over the whole parameter space of θ), most Bayesian practitioners will ignore Eq. 1.28 and rewrite Bayes theorem in the simpler form

$$p(\theta|d) \propto p(d|\theta)p(\theta). \tag{1.26}$$

Exceptions to to this concern model selection where the evidence is required in order to calculate a Bayes factor which computes the liklihood of one model vs. another. In the following subsections I will describe methods for sampling from the posterior $p(\theta|d)$, such as Markov Chain Monte Carlo (MCMC) and Nested Sampling.

Markov Chain Monte Carlo

MCMC is usually used when we would like to try and sample from some distribution p, or approximate the expectation value of some function $f(x)$ which is of a high dimension/complexity. Where p is so complicated that trying to sample from p analytically would be prohibitively expensive. The Monte Carlo portion of MCMC refers to a technique known as Monte Carlo sampling. Monte Carlo sampling means to randomly sample from some distribution. For example, I could choose to randomly sample from a normal distribution $N(0,1)$ or from uniform distribution between -1 and 1. We would define the normal or the uniform distribution that I'm sampling from the proposal distribution. If we randomly sample from the proposal distribution enough times a histogram of the resulting samples should resemble that of the original proposal distribution.

Markov Chain refers to how we are going to go about sampling from that space. Specifically, a Markov Chain is a sequence of numbers where each number in the sequence is dependent on the previous number in the sequence. For example, if we again decided to randomly sample from proposal distribution $N(0,1)$, but instead after each sample is drawn we change the mean of the proposal distribution to be equal to that of the previous sample $N(x_{n-1},1)$, we would end up with something known as a random walk. I will note hear that this approach would not necessarily reproduce the original proposal distribution.

We can use a variety of algorithms to determine how to accept or reject new samples drawn from the proposal distribution. One common algorithm for doing so is the Metropolis-Hastings algorithm. This algorithms works by first calculating the posterior probability value $p(\theta_i|d)$, in other words the height of the posterior, for the new proposed sample, as well as the posterior probability value of the previous sample $p(\theta_{i-1}|d)$ in the Markov Chain. We then compute the ratio of these two values $\frac{p(\theta_i|d)}{p(\theta_{i-1}|d)}$. If this ratio is greater than 1, then we always accept the new proposed sample. If however, the ratio less than 1, then we will not necessarily discard the old posterior probability value. Instead, we determine an acceptance probability based on the ratio. In order to determine whether or not to accept, we draw a uniform random number between 0 and 1 and keep the new proposed sample if the previously calculated ratio value is greater than the randomly drawn number. Mathematically, the probability of acceptance (α) of a new sample

in the Markov Chain can be expressed as

$$\alpha = \begin{cases} \frac{p(\theta_i|d)}{p(\theta_{i-1}|d)} & \text{if } p(\theta_i|d) < p(\theta_{i-1}|d) \\ 1 & \text{if } p(\theta_i|d) \geq p(\theta_{i-1}|d). \end{cases} \tag{1.27}$$

There are a few downsides to using the Metropolis Hastings algorithm. One of those downsides being that we have to choose a starting point for the random walk, which is initially liable to be far from the true posterior. Thus it may take some iterations for the algorithm to walk its way towards areas of high likelihood. We can avoid this issue by discarding an arbitrary number of samples at the beginning of the walk such that the remaining represent a point after which the algorithm has reached a stable equilibrium. We call the discarded samples the burn-in period. Another issue relates to something known as autocorrelation. Predictions on parameters θ are known to be somewhat correlated with each other due to the fact that they are all generated from the same Markov process. The fact that this correlation exists is fine, but excessive correlation can mean that there are issues with the model being used. A helpful method for tempering such correlations can be through the process of thinning. Thinning involves generating a large amount of samples from the proposal distribution, but only keeping every N^{th} sample from that large sample set.

Nested Sampling

Nested sampling is another method which can be used in order to sample from the posterior. However, that wasn't the primary goal in mind when the method was first developed and the posterior is only really obtained as a byproduct (unlike MCMC which *directly* samples from the posterior). An additional motivation for nested sampling relates to the fact that MCMC can have issues when trying to deal with widely spaced multi-modal and degenerate distributions, so an improved method that handles such issues was needed at the time. Nested sampling was first introduced by Skilling in 2004 (later expanded upon in 2006) because he wanted a way to go about computing the evidence (sometimes known as the marginal likelihood) in a feasible way in order to compare different models effectively. The evidence can be described as

$$Z = \int p(\theta)L(\theta)d\theta \tag{1.28}$$

where $p(\theta)$ is the prior distribution and $L(\theta)$ is the likelihood function. Samples from the posterior can then be obtained as a byproduct following the evaluation of the evidence Z. One might naively assume that it would be easy to evaluate this integral by directly computing the expression for each parameter θ. Unfortunately, this procedure can get computationally expensive quickly as the number of inferred parameters θ increases. Rather than integrating over the whole space of parameters θ explicitly, it would be advantageous to redefine Eq. 1.28 so that it was only dependent on a single parameter and an approximate method for computing the integral could be found. The fundamental idea that a complex high-dimensional problem with many inferred parameters may be represented in a simplified 1-dimensional form is one of the key ideas of nested sampling. This then begs the question, how do we go about computing the evidence in a simpler 1-dimensional way? I will describe the details of such an algorithm in the subsequent text.

The nested sampling algorithm can be breifly described as taking the following small steps: dividing the posterior up into more manageable distributions, sampling from each of those distributions, and then subsequently combining the results from each sampling step into a final product. We can start by generating a finite lattice of points sampled stochastically from the prior distribution. These random points are colloquially known as "live" points. Next, we would like to design an algorithm which evolves the "live" points to explore the likelihood space to find the global (or at least close to global) point of maximum likelihood. Ideally we would like to draw from a prior which also decreases in size as we move into regions of the likelihood space which dominates the posterior. Theoretically, the point with the highest likelihood would have a very small region of the prior space to sample from and the point of lowest likelihood would have nearly the whole prior space to sample from.

Because prior volume is by definition normalised to be between 0 and 1, if we framed the problem such that it was likelihood as a function of prior volume being sampled from, we would have a nicely behaved, smooth, monotonically decreasing function which could be integrated

easily. We can start by approximating the evidence integral as a series of N steps given as

$$Z = \sum_{i=1}^{N} L_i w_i, \tag{1.29}$$

where L_i is the lowest likelihood value out of all current "live points" and w_i is a weighting term of that lowest likelihood point defined as

$$w_i = p(\theta_i) d\theta. \tag{1.30}$$

This weight can be thought of as the subset of the prior distribution which is represented by the θ_ith sample. The total prior volume above the current lowest likelihood point can be defined as

$$X(\lambda) = \int_{L(\theta)>\lambda} p(\theta) d\theta, \tag{1.31}$$

where $X(\lambda)$ is the prior volume (a summation over the prior $p(\theta)$) and λ is the likelihood cutoff value defined by the lowest likelihood point. We see that as λ increases, the enclosed prior volume $X(\lambda)$ will then decrease, thus making the integral more tractable as i increases. Because $X(\lambda)$ must lie between 0 and 1 and is a smooth continuous function, we can rewrite the evidence integral in terms of a single variable expression which is far easier to compute than Eq. 1.28

$$Z = \int_0^1 L dX. \tag{1.32}$$

The likelihood L is fairly easy to evaluate for any one sample within the enclosed likelihood space L

We can explicitly evaluate the simplified evidence in Eq. 1.32 by first drawing M random points from the prior distribution $p(\theta)$. We then iterate over a pre-determined number of iterations N. For each iteration i we compute the likelihood of all sampled points and determine the point with the lowest likelihood value. We then compute an estimate on the difference between the amount of prior mass covered by the the current samples above and up to the minimum like-

lihood value sample. We call this estimate the weight w_i and we increment the evidence Z by $L_i w_i$. Finally, we replace the lowest likelihood point L_i with a new point which is sampled randomly through a process like MCMC (subject to the constraint that the new point has a liklihood which is greater than the lowest liklihood point among all current live points). We only need to generate one new point because we can simply reuse the remaining $M-1$ surviving points since they are all better than the lowest likelihood point. Alternatively, one could also generate a new point by genetic mixing of survivor point coordinates [112].

Now that we have a computationally tractable method for computing the evidence integral (e.q. 1.32), we can use this to compute posterior samples. Posterior samples are computed by taking the saved discarded minimum likelihood parameter sample weights w_i from the N steps of the nested sampling algorithm and performing a summation over weights w_i expressed as

$$p(\theta) \approx \frac{\sum_{i=0}^{N} w_i \delta(\theta_i)}{\sum_{i=0}^{N} w_i} \tag{1.33}$$

where $\delta(\theta_i)$ is the Dirac delta function of parameter θ_i at the ith parameter sample. Since both summations in the numerator and denominator of e.q. 1.33 cancel out, we are then left with a set of Dirac delta functions which can be histogrammed in order to compute the probability density function of the posterior.

1.8 GW detections

Since it's inception, LIGO has had several observing runs. Initial operations ran from 2002 to 2010, but no GWs were detected during this time period. During the first observing (September 2015 - January 2016), LIGO detected a total of 3 binary black hole mergers including: GW150914, GW151226 and GW151012. GW150914 was the very first detected binary black hole which had an unusually high SNR. GW151012 was originally labeled as a potential GW event due to its high false alarm rate, but was subsequently upgraded to a confirmed GW event in the GWTC-1 catalogue [119] because its false alarm rate was less than 1 per 30 days (a threshold determined by the LIGO collaboration).

During the second observing run (November 2016 - August 2017) LIGO detected an addi-

tional 7 binary black holes with total masses between $\sim 18.6M_{\odot}$ and $\sim 85.1M_{\odot}$. The second observation run excitingly also saw the very first detection of a BNS event. The BNS event had the highest network SNR of any event over all of O1 and O2. Interestingly, there was also a large non-astrophysical noise transient which overlapped with a portion of the BNS event in the LIGO Livingston detector. This noise transient was successfully mitigated through an excising technique known as time-domain gating [14].

Most recently, during the first half of the third observing run (April 2019 - March 2020) the collaboration made an additional 39 confirmed GW event detections. Of those 39, 3 could be either from a BBH or BNS [119, 120], which may contain the first neutron star black hole (NSBH) signals detected. The increase in number of detections can largely be attributed to higher sensitivities of the detectors during this observation run over previous runs, with a BNS range of 108Mpc, 135Mpc and 45Mpc for Hanford, Livingston and Virgo respectively [120]. The GWTC-2 catalogue contains signals with component masses lower and higher than the lowest and highest component masses contained in all of GWTC-1. The most up-to-date merger rate constraints according to GWTC-2 were also updated to be $\sim 23.9\text{Gpc}^{-3}\text{yr}^{-1}$ for BBHs and $\sim 320\text{Gpc}^{-3}\text{yr}^{-1}$ for BNSs.

1.9 Multi-Messenger Astronomy

After a GW signal has been identified and posterior samples have been obtained, an alert is sent out to other EM partners around the globe in order to perform follow-up analysis. Astronomical partners include instruments which look across the whole range of the EM spectrum including: Radio, Microwave, infrared, visible light, ultra-violet, X-ray and gamma ray. A full Bayesian inference analysis is typically done on all viable GW candidates. In addition to the full Bayesian analysis, the collaboration is also able to produce low-latency parameter estimation products using tools such as `Bayestar` [111]. `Bayestar` operates under the assumption that a large degree of the information contained in a GW signal is encapsulated within a small number of products produced by the search matched template filtering process, namely: the time, amplitude and phase of the signal. Using a simplified likelihood function and the Fisher information matrix,

`Bayestar` is able to produce posterior samples on a limited number of source parameters (sky location, distance and orientation) in under a few minutes which provides a good approximate of the full Bayesian posterior.

Prompt follow-up analysis using tools like `Bayestar` and `Bilby` along with observations from EM partners can provide new insights into fundamental astrophysical processes. For example, as BNS signals reach the final stages of the inspiral phase of the merger, the internal structure of the sources has more of a pronounced effect on the resulting GW signal. Information on the tidal deformability and equation of state of the neutron stars may be gleamed from this part of the GW signal. Tidal deformability is also highly correlated with the predicted masses and spins of the signal [14]. EM follow-up analysis can also be used in tandem with Bayesian analysis in order to produce inferences on the Hubble constant. This can be done by getting accurate estimates on the luminsity distance directly from the GW signal through Bayesian inference and using EM partners to identify a likely host galaxy. For GW170817, LIGO and EM partners were able to infer a Huble constant value of $\sim 70\text{kms}^{-1}\text{Mpc}^{-1}$ [14].

The arrival time (along with an accurate estimation of the luminosity distance) of a GW event can be compared to the observation time of the gamma ray burst from a BNS merger counterpart in order to test the effect gravitational potentials have on BNS EM radiation and BNS GWs (equivalence principle), as well as the speed of gravity [80]. In addition, using best-matching waveform templates for detected GW signals, we can perform tests on the accuracy of general relativity itself by doing residual noise waveform subtraction tests and insprial-merger-ringdown consistency tests. Depending on the length and SNR of the signal, accurate constraints may be placed on the graviton Compton wavelength and non-GR polarization states. Other tests of GR are listed in great detail by [11].

1.10 Summary

In this chapter we provided a brief introduction to Einstein's Field Equations and how those field equations lead to the prediction that GWs exist. We described various sources which are able to produce GWs and how the LIGO detectors physically operate to detect such signals.

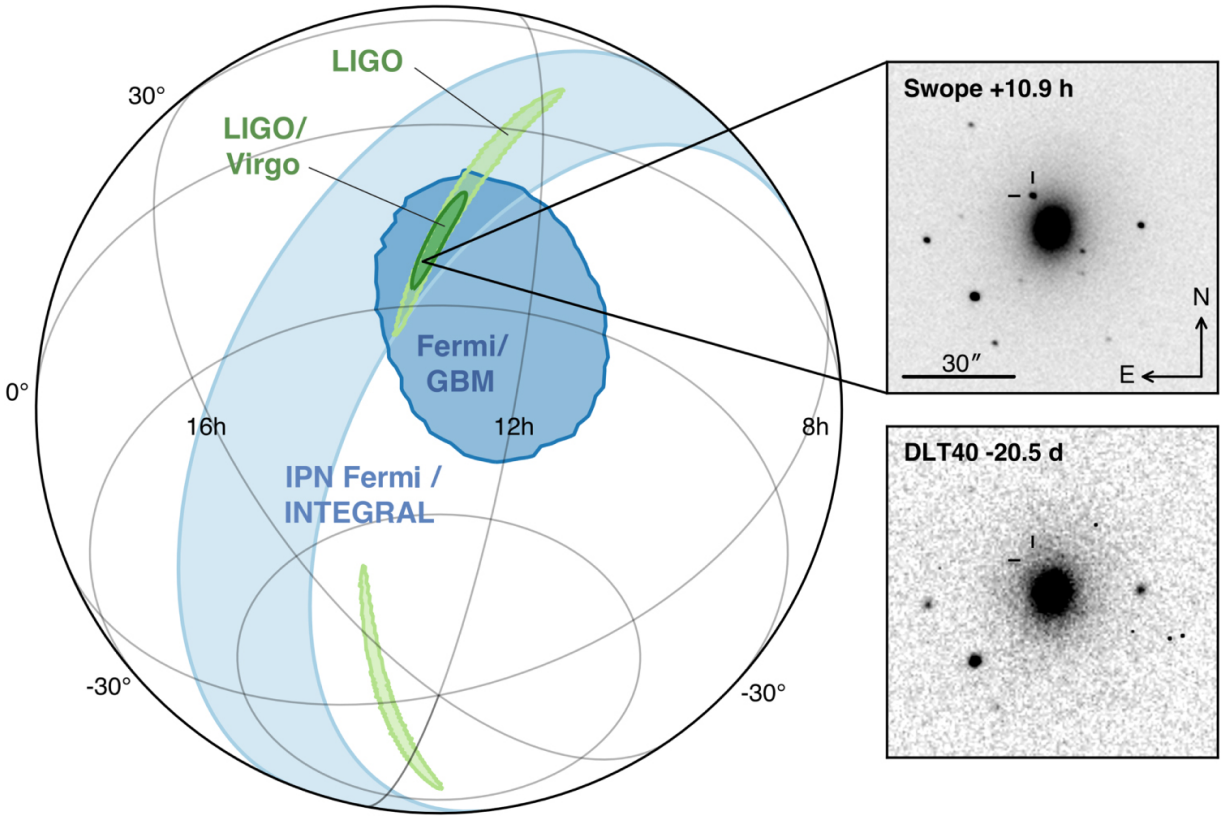


Figure 1.6: Sky localization for the first confirmed detection of a BNS merger by LIGO. Areas shaded in green are the data products from `Bayestar` using both LIGO alone and LIGO/Virgo combined, dark blue are predictions from Fermi/GBM and light blue are predictions from IPN Fermi/INTEGRAL. Black and white images on right-hand side are visible light measurements of a bright event around the galaxy NGC 4993 thought to contain the afterglow of the BNS event. `This figure was produced by the authors of` [81].

We also described the search techniques for all source signals and how we generate predictions on the underlying source parameters of GW signals. We finally end the chapter by discussing multi-messenger astronomy.

Chapter 2

An introduction to machine learning

In this chapter I will explain in detail the machine learning techniques used in this thesis including: fully-connected neural networks [54], convolutional neural network (CNN)s [76] and CVAEs [95]. I will also describe some of the basic principals of how neural networks learn (e.g. backpropogation [**?**]), how they are initialized, best training practices, methods for evaluating performance, and methods for data augmentation/processing.

First off, to dispel any notion of machine learning being thought of as uninterpretable, machine learning is nothing more than statistics and function approximation at the end of the day. The overarching goal being to approximate a function which is trained to find a global minimum (though this rarely is the case in reality and most often a local minimum will suffice) according to a cost function. How one defines this minimised function is the name of the game and there are many methods for doing so.

A machine learning algorithm can perform a variety of objectives including: classification [60], regression [70], anomaly detection [123], denoising [121] and density estimation [97]. Of course, there are many other tasks a machine learning algorithm can tackle, but the three which are primarily used in this thesis are classification, regression and density estimation. In classification, a machine learning algorithm attempts to learn the optimal function to classify a given set of inputs (e.g. a time series or image) and returns as output the likelihood that the given input came from a particular class or set of classes. A machine learning algorithm can also alternatively perform non-linear regression by switching from outputting numeric values

describing class probability to predicting a continuous variable. Finally, we can even have a machine learning algorithm produce probability distributions (e.g. gravitational wave source parameter posteriors) for a given input by enforcing that the algorithm predict moments which describe such a distribution.

2.1 Fully-connected neural networks

So, how do we build a machine learning algorithm? Before diving too deep, lets first define a simple neural network architecture, the perceptron [83]. A perceptron is made up of one neuron which takes a set of inputs, multiplies each input by a learned scalar value called a weight, sums all the multiplied values up and passes them through a re-scaling function to produce a prediction (illustrated in Fig. 2.1). For the sake of simplicity lets presume we want to perform a binary classification task. The perceptron can be defined as a learned function

$$z = \sum_{i=1}^{N} w_i x_i + b \tag{2.1}$$

$$y_p = \sigma(z), \tag{2.2}$$

where y_p is our predicted output (e.g. class label), x is our given input, w is a learned scalar weighting term on x, b is a learned scalar bias term on x and σ is a nonlinear activation function which rescales the output. Because we usually frame the classification problem such that a prediction of 0 represents class 1 and a prediction of 1 represents class 2, we typically apply a sigmoid activation function (Fig. 2.2) since it limits the output to be between 0 and 1.

We also apply a nonlinear activation function in order to allow our perceptron to learn non-linear functions/features [93]. All weights and biases are typically initialized randomly prior to training and there are many schemes for choosing the optimal initialization [77]. In order to approximate more complicated functions, we can string together multiple perceptrons (neurons)

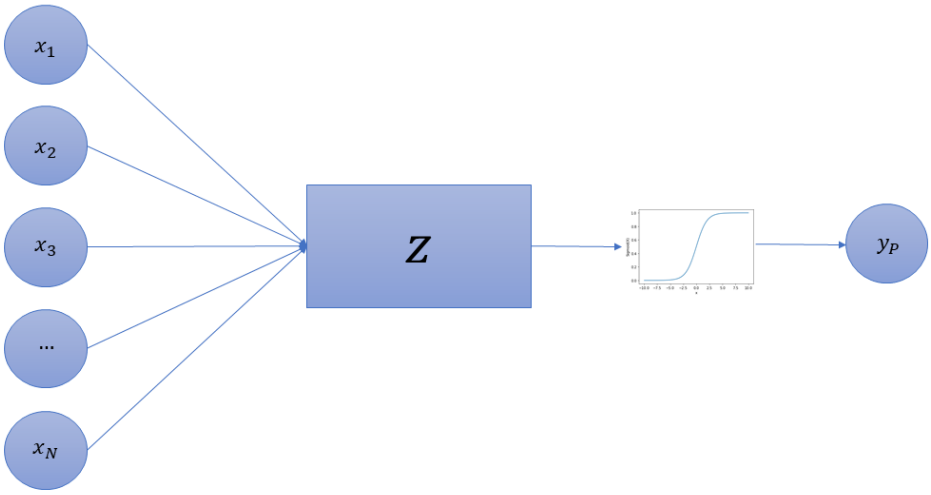


Figure 2.1: A perceptron network whose inputs $x_1 ... x_N$ are each multiplied by a learned set of corresponding weights $w_1 ... w_N$ and summed together in z. The summed value is then passed through an activation function to get the final output of the network.

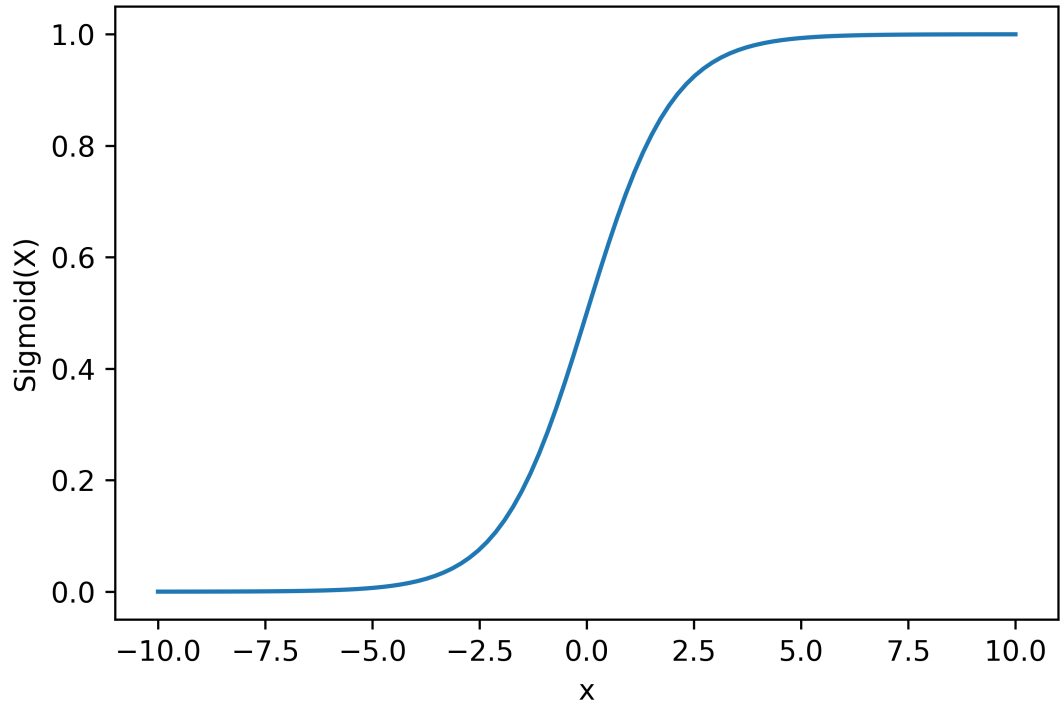


Figure 2.2: A sigmoid activation function plotted over the input range -10 to 10. Given an input over a pre-defined range, the activation function will rescale the input to be between the range of 0 to 1.

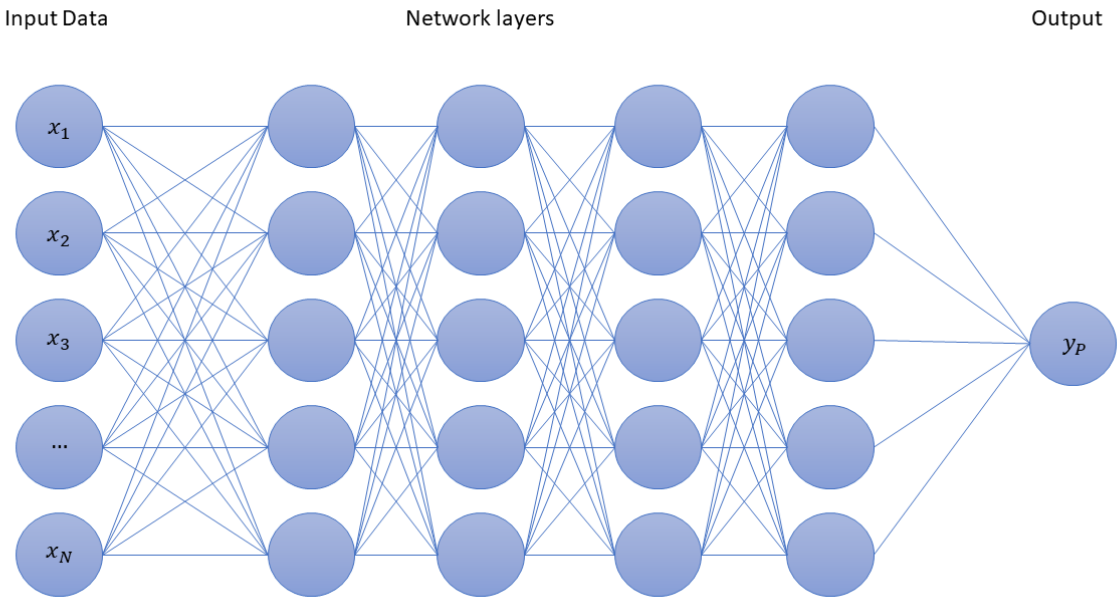


Figure 2.3: A deep fully-connected neural network. Inputs $x_1 ... x_N$ are given as input to the first layer of the neural network and are fully-connected to every node in that layer. The outputs of each node are then passed on to the next set of nodes in the next layer and are also fully-connected. The final layer passes through a final single node which determines the class/value of the given input.

in a layer where each neuron is fully-connected to the given input x (i.e. every element in our input is multiplied by each weight of each neuron in our first layer) [54]. To make a deep network, we can randomly initialize another layer of neurons (of arbitrary size) which takes as input the output from the previous first layer of neurons (illustrated in Fig.2.3). Additional layers can be added in perpetuity, although there are issues that can arise due to a vanishing gradient or hardware memory limitations [98]. During training, weights and biases are updated according to the performance of the algorithm with respect to a loss function which describes how well the algorithm is performing with respect to the true value of each training sample [66]. This loss function can take many forms, but most often a mean squared error is used

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_n^l - y_n^t)^2. \tag{2.3}$$

Where y_n^l is the neural network prediction and y_n^t is the true value. When the predicted label

of nth training sample in the final layer l y_n^l is similar to the nth training sample true label y_n^t, the loss term approaches zero and is at a minimum. The reader's next question may rightly be, how does one go about choosing the right weights w and biases b for each neuron in the network? In other words, how much of an effect will changing $[w,b]$ have on the resulting loss function L. We can quantify the amount to change $[w,b]$ such that the loss is optimised through gradient decent [104]. Specifically, we will be discussing a variant of gradient decent, stochastic gradient decent.

Stochastic gradient decent is an iterative algorithm used to find the global minimum (usually a local minimum is sufficient) of a function. Gradient decent essentially has two decisions it needs to make when choosing new weights and biases: what direction to go and by how much. The most straightforward way of getting this information is through the use of a derivative, specifically the gradient of the loss function L defined as

$$g = -\gamma \nabla L(y^l), \tag{2.4}$$

where γ is a tunable step size scale factor (typically called the learning rate), $\nabla L(y^l)$ is the gradient of the loss with respect to the weights and biases of the network and g is a list of gradients corresponding to each weight and bias between the final output layer and the previous layer. The sign of the gradient for each weight and bias corresponds to the direction of in which to change the weights while the magnitude tells you how sensitive the loss function is to each weight and bias. If there are multiple neurons in the final layer of the network, the contributing gradients for each neuron in the final layer are summed in order to return a representative gradient with respect to all output neurons.

However, this only returns the gradients for the final output layer. In order to appropriately update weights and biases in the previous layers we must propagate the gradient backwards. This process is better known as back propagation.

In back propagation we use the computed gradient from the final layer and compute the gradient between that final layer gradient and the output from the second to last layer. The

individual contributions from each neuron in the second to last layer are summed (same as was done in the final layer computation) and the gradients in the second to last layer are propagated back again to the third to last layer. Mathematically, we can define this process as the derivative of the loss with respect to the outputs of each layer l

$$\nabla L(y^l) = \frac{\delta L}{\delta y^{(l)}}. \tag{2.5}$$

Applying the chain rule and expanding this out for multiple layers we arrive at

$$\nabla L(y^l) = \frac{\delta z^{(l-m)}}{\delta y^{(l-m-1)}} \frac{\delta y^{(l-m)}}{\delta z^{(l-m)}} \cdots \frac{\delta z^{(l)}}{\delta y^{(l-1)}} \frac{\delta y^{(l)}}{\delta z^{(l)}} \frac{\delta L}{\delta y^{(l)}} \tag{2.6}$$

where the total derivative over all neurons in each layer may be quantified and applied to determine how much to shift each weight and bias in order to improve the overall loss of the neural network. This process is repeated until all weights and biases in the network have had their gradients computed. Ideally, we would compute the gradient over more than just one training sample, but rather the whole training set during each training iteration. However, computing the gradient over the whole network for all training samples is abhorrently expensive, so the training set is typically split up into mini-batches of samples in order to reduce the computational cost.

2.2 Training best practices (practical advice for the reader)

I was commonly told when first wadding into the pool of deep learning that the practical implementation/training of deep learning models is largely a dark art. In this section, I would like to dispel that myth by offering some best practices when training any machine learning model.

Training data size

First off, when in doubt the more training data you make available to yourself, the better. Time and again, during a large part of my thesis work, I would spend countless hours tuning and

tweaking my neural network architecture only to run up against some insurmountable wall of impeded progress. Only then to increase the number of training samples by several factors and see a significant increase in performance.

2.2.1 Data pre-processing and augmentation

Data pre-processing

Commonly, our input datasets are in a messy form. In order to help the network learn more efficiently, it is encouraged to perform some pre-procesing steps. It's important to do this because if our input data has large values over a wide range, and our network is trying to learn a mapping from input to prediction, the learned weights may also end up being very large. Large learned variable weights can lead to massive gradients which cause the network to update weights in large step sizes, making the whole network more unstable in the process. We can overcome this by normalising our input dataset to be between the range of zero to one

$$y = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}, \tag{2.7}$$

where y is our new normalised data, x is our original unnormalised data, x_{min} is our unnormalised data minimum value and x_{max} is our unnormalised data maximum value. It can also be advantageous to rescale our input data to a standard normal distribution if our data has widely varying scales by applying

$$y = \frac{x - x_{\text{mean}}}{x_{\text{std}}}, \tag{2.8}$$

where x_{mean} is the mean of our data and x_{std} is the standard devitation of our data. In addition to rescaling our input data, we ocassionally need to also rescale our training labels. This is especially relevant if the activation function in our final neural network layer is fixed to be between some values. For example, if we were using a sigmoid activation function where the values of the output of the sigmoid are only allowed to be between zero and one, then we would also want our training label values to lie between zero and one since the network would not be

able to produce values outside of that range.

Data augmentation

Neural networks can sometimes be composed of hundreds, thousands, even million of parameters to be tuned during training. Given the massive number of parameters, many complex networks require an equally massive number of training samples. But, what do you do if you are only given a limited number of training samples? This is where data augmentation comes to the rescue. In order to generate more training data, we can apply simple shifts to the existing limited dataset and greatly expand the number of training samples. This can be accomplished through many of the following methods:

- Translation: Shift the input data sample in the x/y/z direction.
- Rotation: Rotate data about a fixed point.
- Cropping: Choose a random subset of the input sample, cut out all data outside of subsection, rescale cropped data to original input size.
- Gaussian noise: Add varying amounts of Gaussian noise in order to simulate poor quality signals.

2.2.2 Validation

In order to understand how well your model is performing with respect to the training data, one may plot the resultant average output of the loss function as a function of the number of training iterations. Ideally, we would like the loss to rapidly decrease and to then flatten out. Although this will give us an indication of how well the network is performing on the training set, it does not inform us as to how well the model will generalize to a novel testing set. In order to quantify the generalization ability of the network, we may set aside a small portion of the training set prior to training as a validation set. During training, we can temporarily freeze the weights and run the validation set through the model in order to compute a validation loss.

If we plot both the validation loss and the training loss as a function of time, we again ideally would like to see both curves decrease as a function of training iteration and then level out. However, if we see that the training curve initially decreases and then continues to decrease while the validation curve initially decreases and then subsequently increases we may say that the model has overfit the training data. Overfitting the training set generally indicates that either our model is too complicated and has essentially memorized the training set or that we do not have enough training data to sufficiently cover the entire parameter space. Overfitting can be mitigated by increasing the training set size, adding dropout connections (see Regularization sub section), early stopping of the training run or reducing the complexity of the neural network.

If on the other hand we see that the validation loss decreases far quicker than the training loss then we may say that the model has underfit the data, which is generally an indication that the model needs to have an increased capacity.

2.2.3 Regularization

Regularization is used to help prevent a machine learning model from overfitting the the training data. One of the easiest regularization techniques to implement is that of dropout. Dropout may be implemented across most types of neural network layers (i.e. fully-connected, convolutional filters, etc.) and rarely have any adverse effect on training. During training, if dropout is implemented a subset of the neurons in a layer will be switched off and not used when the gradient is computed and backpropogated through the network for a given training iteration. The percentage of neurons in a layer to switch off is a tunable parameter, though typically it is best to set a value of no more than 50%. Dropout effectively forces the neurons in a dropout layer to be able to identify multiple features, rather than to focus a small subset.

Batch normalization is motivated by the same principle that standard normalization of input to a neural network is motivated. We want to reduce the space over which the neural network has to search. As such, batch normalization is essentially a normalization of the output of individual layers within a neural network to be between zero and one. This normalization prevents weights or biases from becoming too large. It is computed by normalization of the current layer according to the mean and standard deviation of the output of the previous layer and the subsequent

rescaling through two learned variables (α and β) and is formalised as

$$z_n = \alpha \hat{z}_n + \beta, \tag{2.9}$$

where α is a learnable parameter, β is a learnable parameter and $\hat{z}_n$ is calculated by

$$\hat{z}_n = \frac{1}{n} \sum_{i=1}^{n} \frac{z_n - \mu}{\sigma^2 + \varepsilon}, \tag{2.10}$$

where z_n is the output of the neural network layer summed over all outputs in the batch of that layer, $[\mu, \sigma]$ are the mean and standard deviation respectively of the batch of previous layer outputs introduced to the network and ε is a small constant. If it turns out that the application of batch normalization is not optimal, then the network may undo the above normalization in e.q. 2.10 through the optimization of $[\alpha, \beta]$ in each layer.

2.2.4 Hyperparameter optimization

In many instances, after we have spent all our time cleaning the data, choosing a model to solve our problem and then finally training our model, we find that the most tedious and time consuming part of the whole endeavor is all those pesky model hyperparameters. In this sub section, I will describe three approaches used in this thesis to optimize hyperparameter choices.

Random Search

Probably the simplest of the three approaches random search seeks to choose model hyperparameters based off of a predefined prior distribution for each hyperparameter. The user may define whatever distribution they prefer whether that be a multivariate Gaussian distribution or a uniform distribution so long as the bounds of the distribution chosen are within reason.

Grid Search

Grid search is marginally more complicated. In a grid search approach we define both an upper and a lower limit for each hyperparameter. Additionally, we define a step size by which we will increase the the hyperparameter value after each model evaluation (much like steps on a ladder). Each of these hyperparameter ladders are iterated over in an N dimensional grid, where N is representative of N hyperparameters describing the model.

Gaussian Process Bayesian hyperparameter optimization

Bayesian hyperparamter optimization aims to minimize (or maximize) some objective function. In this case, we aim to learn an approximate surrogate model for an objective function which describes the loss function space and to minimize the value of the loss function output. The dimensionality of the search space is parameterized by the number of neural network hyperparameters being optimized and does not usually perform well past 20 dimensions. Gaussian process regression is used in order to minimize the uncertainty on the surrogate model estimate . We then use an acquisition function to sample from the approximate surrogate model to determine the next set of hyperparameters to use during the next round of neural network training [44].

In practice, we first choose an initial set of hyperparameters to use in order to train the network. This can either be randomly chosen from a reasonable prior, or a best guess from the practitioner. Next, the neural network is trained a pre-defined number of training iterations and the network loss function output value is given as input to the Bayesian optimization algorithm. The Bayesian optimization algorithm then updates its surrogate model based off of this new loss value from the neural network given the hyperparameters used and a new set of hyperparameters is chosen by an aquisition function based informed by the Gaussian process regression surrogate model. The above process is repeated for N user pre-defined iterations.

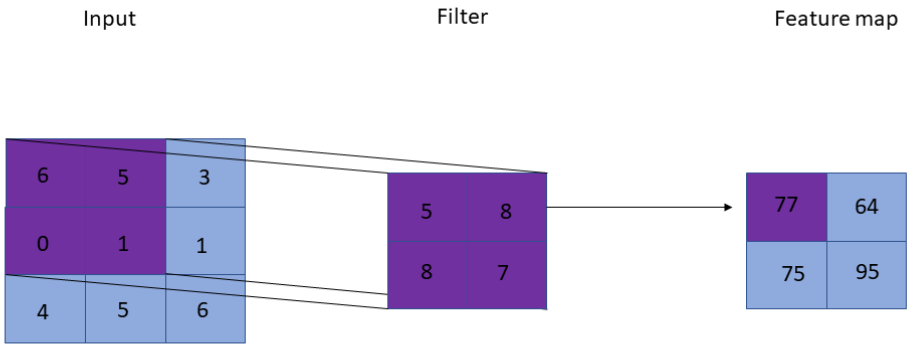
2.3 Convolutional Neural Networks

CNNs were first made popular in the late 1980s to early 1990s and one of the most famous examples is that of the LeNet archetecture made by Yann LeCunn in 1998 [78]. In his paper, LeCunn illustrated that CNNs outperformed other simpler techniques like K Nearest Neighbor and fully-connected neural networks on a variety of image recognition tasks. In recent years CNNs have been applied to many more domains including image object detection, fraud identification, healthcare analysis and many others. In this section I will explain how CNNs work, mechanisms for improving CNN performance, and why they are so powerful for image based tasks.

2.3.1 The convolutional filter

The basic building block of a CNN, convolutional filters are analogous to neurons in a standard fully-connected neural network. A convolutional filter is typically made up of an $N \times N$ matrix of any size that the user wants(for illustraed exmample, see Fig.2.4). Normally, the dimensions are of odd size because all pixels from the input to the filter are enforced to be centered around the output of the filter. This effectively acts to anchor the output of the filter, in a sense acting to interpolate all the anchor pixels neighboring pixels. Without using odd size filters, distortions across multiple layers can cause issues, though are not impossible to overcome through added network complexity.

Values in the $N \times N$ matrix of the filter are initialised randomly with an added bias term. The initialised values in the filter are the weights of the filter. During training, the filter is convolved with its given input where each element in the filter is multiplied by the corresponding overlapping element in the input. All the multiplied values are then summed together to produce the output of the filter. A bias term is then added to the output. We the slide the filter over by 1 column and repeat the convolution process until the edge of the filter reaches the last column of the input. We then slide the filter back to the 1st column of the input and down one row and repeat the convolution process on all columns again. This is done until we have convolved the filter over the whole input.



$$(6 * 5) + (8 * 5) + (0 * 8) + (7 * 1) = 77$$

Figure 2.4: An illustration of a simple CNN filter. The filter is a 2×2 filter randomly initalised with a set of weights for each dimension. Each filter weight is multiplied by the corresponding input dimension over which it is currently covering. All multiplied values are then summed together to produce the output of the filter. The filter slides over one element (if stride is set to 1) and the computation is done again until the whole input space has been covered.

2.3.2 Pooling layers

Pooling is a form of regularization and acts to choose the most important features in each convolution of a filter. We do this because many times CNN filters will focus on minute detailed information, but loose sight of the bigger picture. This means that small changes in the input, will have large effects on the filter. In order to combat this, pooling acts to downsample the input, making it more "fuzzy" in the process, forcing the model to learn broader features. This is done by taking the maximum value over all elements in a convolution and discarding all other values.

2.3.3 Striding

In addition to pooling, another form of regularization is striding. Striding determines the number of pixels a convolutional filter moves after each convolution is performed. Standard techniques nominally employ a stride of 2. Additionally, because we are skipping over some parts of the input when sliding the filter, striding can act to reduce the total memory usage of a CNN.

2.3.4 The fully-connected layers

Following the convolutional filter layers, we now have to convert the output of the CNN to predict a discrete set of classes or regress on some parameter. If performing binary classification, one could simply make the last layer of the CNN two 1D convolutional filters, however this is usually not what is normally done. Typically, a flattening operation is applied to the last convolutional layer where the concatenated feature maps from the previous layer's filters are flattened into a 1D string of elements. This 1D string of elements are then given as input to a fully-connected layer of neurons. One can then add additional fully-connected layers and a final layer equivalent to either the number of classes to predict, or the number of parameters to regress on. A logical question may be why add this extra fully-connected complication? First and least informative answer is that it just what has always been done since time immortal. Second slightly more intuitive answer is that we want to unify the learned features across all CNN filters and use those features to learn how to either classify or regress.

2.4 Conditional Variational Autoencoders

In this section I will describe the network architecture of a CVAE. This will be done by first describing an autoencoder (AE) network, then a variational autoencoder (VAE) network and finally a CVAE network.

Autoencoders

An AE is comprised of two neural networks called an encoder and decoder. The encoder is given an input from a pre-generated training set (in this case, lets assume we are dealing with the MNIST dataset). The output of the encoder network is typically smaller than the dimension of the input, essentially forming a bottleneck. We call the output of the encoder the latent space representation. The predicted numbers from the encoder representing the latent space are then given as input to the decoder network. The decoder then tries to reconstruct the given input to the encoder network (Fig.2.5). We measure how well the network is performing through a loss function given as

$$l(f(x)) = \sum_k (f(x)_k - x_k)^2, \tag{2.11}$$

where $f(x)_k$ is the output from the decoder network on the kth training/testing sample of the AE and x_k is the kth training/testing sample. During training, $f(x)_k$ is trained to be as similar to x_k as possible in order to minimise $l(f(x))$. The lower the value $l(f(x))$ is, the better the network is performing. The loss $l(f(x))$ is then backpropagated through the network adjusting the weights and biases in order to further minimise the loss.

AEs can be used across a variety of applications including: dimensionality reduction [130], image denoising [90], feature extraction [82] and many others. Although powerful, one of the limitations of an AE is the way it represents the latent space. Within the context of AEs as content generators the reader would be forgiven for thinking that, if properly trained, one could just simply sample from the latent space uniformly in order to generate unique content. However, due to the fact that the network is not required to distribute learned latent space representations during training (features are allowed to be encoded anywhere in the latent space), it is not

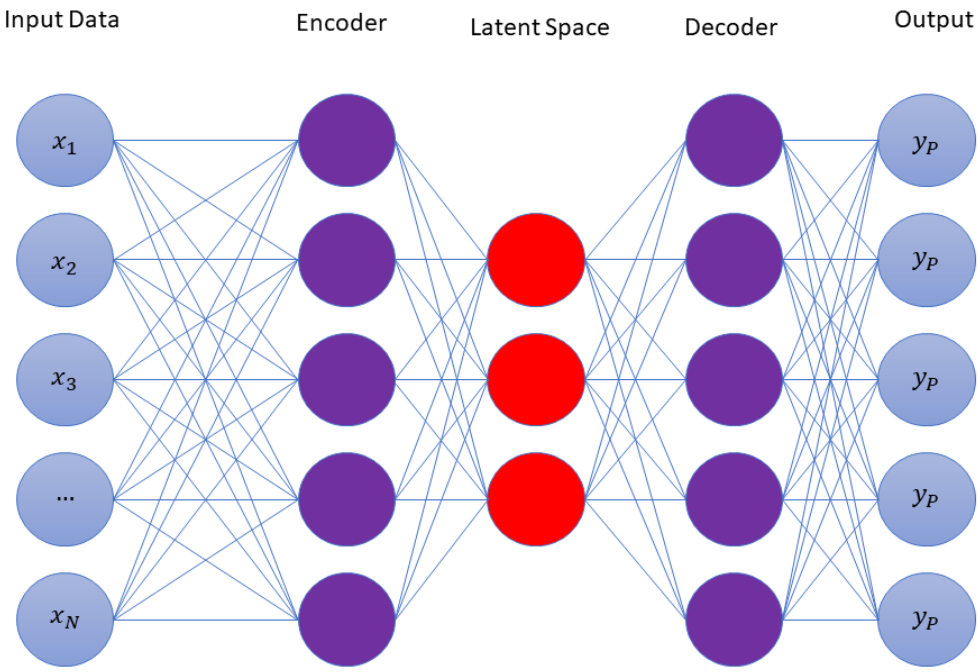


Figure 2.5: An autoencoder network composed of two neural networks defined as the encoder and the decoder networks. Here, both networks are represented as fully-connected networks, but could also be any number of other network architectures such CNNs and LSTM networks. The encoder network takes as input a set of data and compresses the data through a bottleneck called the latent space. The latent space is then given as input to the decoder network which tries to reconstruct the given input $x_1 ... x_N$.

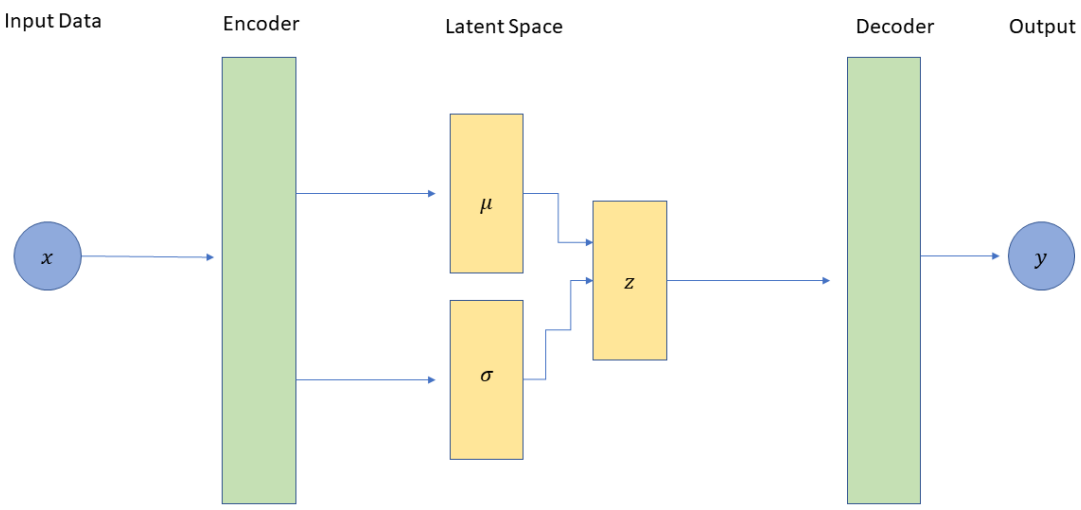


Figure 2.6: A simplified diragram of a variational autoencoder network. The input data x is passed to the encoder neural network which produces predictions on the means μ and standard deviations σ of multi-variate Gaussian distributions describing the latent space. Samples are drawn from the predicted distributions z in order to get samples from the latent space. These latent space samples are then passed to the decoder network which attempts to reconstruct the given input x.

guaranteed that latent space samples will be from a learned part of the latent space.

Variational Autoencoders

We can overcome this, through the use of a VAE (see illustration in Fig.2.6. A VAE is nearly identical to an AE, except for the main differences arising from how the latent space is represented and how the loss function is constructed. If we go back to our original problem, where the encoder is given a sample MNIST digit image; instead of having the encoder network output a single predicted value for each dimension in the latent space, we now have it output both a predicted mean and standard deviation value describing a Gaussian for each dimension. Samples are then drawn from the the predicted distributions and given as input the decoder network. The decoder network produces estimates trying to reconstruct the given input to the encoder network. The loss for a VAE is also slightly different and is represented as

$$l(f(x)) = \sum_k (f(x)_k - x_k)^2 + D_{\mathrm{KL}}[N(\mu_k, \sigma_k), N(0,1)], \tag{2.12}$$

where $f(x_k)$ is the predicted output from the decoder, x_k is the given input to the encoder, D_{KL} is the KL divergence between latent space samples from predicted Gaussians $N(\mu_k, \sigma_k)$ and samples from a mean zero unit variant Gaussian distribution $N(0,1)$.

We can derive this loss function by first assuming that we want to approximate the optimal latent space posterior representation using a neural network. I will be directly following a similar derivation done in [94]. We can define the difference between the approximate and the truth as

$$D_{\mathrm{KL}}(q_\theta(z|x_k)||p(z|x_k)) = -\int q_\theta(z|x_k) \log(\frac{p(z|x_k)}{q_\theta(z|x_k)})dz, \tag{2.13}$$

where $q_\theta(z|x_k)$ is the approximate posterior of the latent space and $p(z|x_k)$ is the optimal representation. We can then substitute Bayes theorem in for the truth $p(z|x_k)$ as

$$D_{\mathrm{KL}}(q_\theta(z|x_k)||p(z|x_k)) = -\int q_\theta(z|x_k) \log(\frac{p(x_k|z)p(z)}{q_\theta(z|x_k)p(x_k)})dz. \tag{2.14}$$

We now separate out the division using the laws of logarithms and distribute the integrand

$$D_{\mathrm{KL}}(q_\theta(z|x_k)||p(z|x_k)) = -\int q_\theta(z|x_k) \log(\frac{p(x_k|z)p(z)}{q_\theta(z|x_k)}) - \log(p(x_k))dz. \tag{2.15}$$

$$-\int q_\theta(z|x_k) \log(\frac{p(x_k|z)p(z)}{q_\theta(z|x_k)}) + \int q_\theta(z|x_k) \log(p(x_k))dz. \tag{2.16}$$

Due to the fact that $q_\theta(z|x_k)$ is a probability distribution, we can assume that integral of $q_\theta(z|x_k)$ is equal to 1, further simplifying the equation. We also know that by definition the KL divergence must be greater than zero, so we can set the whole equation to be greater than or equal to zero

$$-\int q_\theta(z|x_k)\log(\frac{p(x_k|z)p(z)}{q_\theta(z|x_k)})dz+\int \log(p(x_k))\geq 0. \tag{2.17}$$

Since $\log(p(x_k))$ is a constant, we can remove the integrand on the right-hand side

$$-\int q_\theta(z|x_k)\log(\frac{p(x_k|z)p(z)}{q_\theta(z|x_k)})dz+\log(p(x_k))\geq 0. \tag{2.18}$$

Moving the integral over to the right-hand side we get

$$\log(p(x_k))\geq \int q_\theta(z|x_k)\log(\frac{p(x_k|z)p(z)}{q_\theta(z|x_k)})dz. \tag{2.19}$$

Applying rules of logarithms yet again, we can now expand out the above equation into

$$\log(p(x_k))\geq \int q_\theta(z|x_k)[\log(p(x_k|z))+\log(p(z))-\log(q_\theta(z|x_k))]dz. \tag{2.20}$$

We can now recognize that the expression on the right is equivalent to the expectation value over the log terms given as

$$\log(p(x_k))\geq E_{\sim q_\theta(z|x_k)}[\log(p(x_k|z))+\log(p(z))-\log(q_\theta(z|x_k))] \tag{2.21}$$

$$\log(p(x_k))\geq E_{\sim q_\theta(z|x_k)}[\log(p(x_k,z))-\log(q_\theta(z|x_k))]. \tag{2.22}$$

Taking a small look back to equation (2.19), and using the rules of logarithms we can rewrite it as

$$\log(p(x_k))\geq \int q_\theta(z|x_k)\log(\frac{p(z)}{q_\theta(z|x_k)})dz+q_\theta(z|x_k)\log(p(x_k|z))dz \tag{2.23}$$

We can now see clearly that the first expression on the right-hand side of the above inequality is equivalent to the negative KL divergence between our approximate latent space representation $q_\theta(z|x_k)$ and our prior on the latent space $p(z)$

$$\log(p(x_k)) \geq -D_{\mathrm{KL}}(q_\theta(z|x_k)||p(z)) + q_\theta(z|x_k)\log(p(x_k|z))dz. \tag{2.24}$$

Conveniently, we also see that the right most term on the right-hand side of the inequality can be approximated as the expectation value of the log probability of the data x_k given z

$$\log(p(x_k)) \geq -D_{\mathrm{KL}}(q_\theta(z|x_k)||p(z)) + E_{\sim q_\theta(z|x_k)}[\log(p(x_k|z))]. \tag{2.25}$$

And there we have it, the loss function for the VAE! The whole expression to the right of the inequality is typically denoted as the evidence lower bound (ELBO) because it puts a lower bound on the log likelihood of the data $log(p(x))$. For practical purposes, expectation value on the right can be computed analytically by performing the mean squared error between predictions on x_k given latent space samples from $q_\theta(z|x_k)$ and the true x_k values give as input to the encoder. The KL term acts to constrain the form of the approximate posterior $\log(p(x_k))$ according to a chosen prior on the latent space representation $p(z)$ across the whole input parameter space. The KL is essentially a regularization factor which helps the model to learn a well-formed latent space and reduce the likelihood of overfitting to the training data. $p(z)$ is nominally chosen to be represented as a mean zero, unit variant Gaussian distribution. Samples are drawn from predicted means and standard deviations from the encoder $q_\theta(z|x_k)$ and samples are also drawn from a unit variant Gaussian representative of the prior $p(z)$. We can then analytically compute the KL divergence value. The results from the KL and the ELBO calculations are then summed together and the whole term is minimized through backpropogation.

The reparameterization trick

First proposed by Kingma et al. in their seminal VAE paper [73], the reparameterization trick acts to solve a problem that we now have with the above derived loss function for the VAE. That problem being that we cannot backpropagate the gradient computed with respect to the loss through a random node. The random node referred to is the latent space node given as input to the decoder which we define as

$$z = \mu + \sigma, \tag{2.26}$$

where z are samples from our latent space, μ are predicted means describing multivariate Gaussians and σ are predicted standard deviations. We can prove that the above equation is nondifferentiable by first rewriting the expression as an expectation value

$$E_{p(z)}[f(z)], \tag{2.27}$$

If we then go to compute the gradient of the expectation value (which we would normally do when updating the weights in the VAE), then we get the following expression

$$\nabla_\theta E_{p(z)}[f_\theta(z)] = \nabla_\theta \int p(z) f_\theta(z) dz \tag{2.28}$$

$$= \int p(z) \nabla_\theta f_\theta(z) dz \tag{2.29}$$

$$= E_{p(z)}[\nabla_\theta f_\theta(z)] \tag{2.30}$$

where it is shown that the gradient of the expectation value of $f(z)$ is equivalent to the expectation value of the gradient of $f(z)$, which is easily computed since the probability distribution is not a function of weights and biases θ. However, if the probability distribution were a function of θ, which it is since the latent space is produced by the encoder network which is itself a function of weights θ, then we see the following happen when taking the gradient of the expectation value

$$\nabla_\theta E_{p_\theta(z)}[f(z)] = \nabla_\theta \int p_\theta(z) f_\theta(z) dz \tag{2.31}$$

$$= \int \nabla_\theta p_\theta(z) f_\theta(z) dz \tag{2.32}$$

$$= \int p_\theta(z) \nabla_\theta f_\theta(z) dz + \int f_\theta(z) dz \nabla_\theta p_\theta(z) dz \tag{2.33}$$

$$= E_{p_\theta(z)}[\nabla_\theta f_\theta(z)] + \int f_\theta(z) dz \nabla_\theta p_\theta(z) dz \tag{2.34}$$

$$\tag{2.35}$$

Although the first summation term is again easily computed, we now are left with a particularly intractable integral on the right-hand side. Monte Carlo methods do allow us to randomly sample from $p_\theta(z)$, but they do not guarantee that we may take its gradient. We are now left with a bit of problem in how we are expected to compute the integral. However, if we make a small change to equation (2.26) by scaling the standard deviation by a deterministic value, we will see how this then solves our intractable integral problem.

If we parameterize this deterministic value as samples drawn from a mean zero, unit variant Gaussian distribution $\sim N(0,1)$ which we will denote as

$$\varepsilon = N(0,1). \tag{2.36}$$

Then we can rewrite equation (2.26) as

$$z = \mu + \sigma\varepsilon. \tag{2.37}$$

we'll now simplify the expression to be a function g which is parameterized by ε and x, where x is representative of μ and σ

$$z = g_\theta(\varepsilon, x). \tag{2.38}$$

rewriting as an expectation value

$$E_{p_\theta(z)}[f(z^{(i)})] = E_{p(\varepsilon)}[f(g_\theta(\varepsilon, x^{(i)}))]. \tag{2.39}$$

Taking the gradient with respect to the weights and biases of the network θ we get

$$\begin{aligned}\nabla_\theta E_{p_\theta(z)}[f(z^{(i)})] &= \nabla_\theta E_{p(\varepsilon)}[f(g_\theta(\varepsilon, x^{(i)}))] \\ &= E_{p(\varepsilon)}[\nabla_\theta f(g_\theta(\varepsilon, x^{(i)}))].\end{aligned}$$

where we see that indeed we can take the gradient of $f(g_\theta(\varepsilon, x^{(i)}))$ since $g_\theta(\varepsilon, x^{(i)})$ is differentiable with respect to θ. We can now just simply randomly Monte Carlo sample in order to analytically calculate $\nabla_\theta E_{p_\theta(z)}[f(z^{(i)})]$.

Conditional Variational Autoencoders

Through our rigorous derivations in the previous sections we have now formulated a loss function which can be trained to maximize the expected log probability of our decoder's prediction, while also regularizing its model for the latent space to a smooth representation. However, there are some minor drawbacks to the VAE which make it unsuitable for some generative tasks.

For example, if we wanted to train our VAE on the fashion MNIST dataset [133], a database of digit images representative of various classes of fashion items (shoes, shirts, pants), in order to generate new images of fashion items we would run into a problem. This problem is most clearly illustrated when taking a closer look at equation (2.25). In equation (2.25), it can seen that the encoder is solely conditioned on inputs from the training space x_k, but not on what class/type of image x_k is. Similarly, the decoder is solely conditioned on the latent space z. In order to produce a particular class of input from the decoder, we would have to know which part of the latent space each class lives and to then draw from the location of the latent space that our class of interest resides. Knowledge of class location is infeasible to obtain after training because the KL divergence term in equation (2.25) enforces that on average across the whole training set that the predicted latent space locations are representative of a mean zero Gaussian distribution, but does not constrain where individual classes of input may reside.

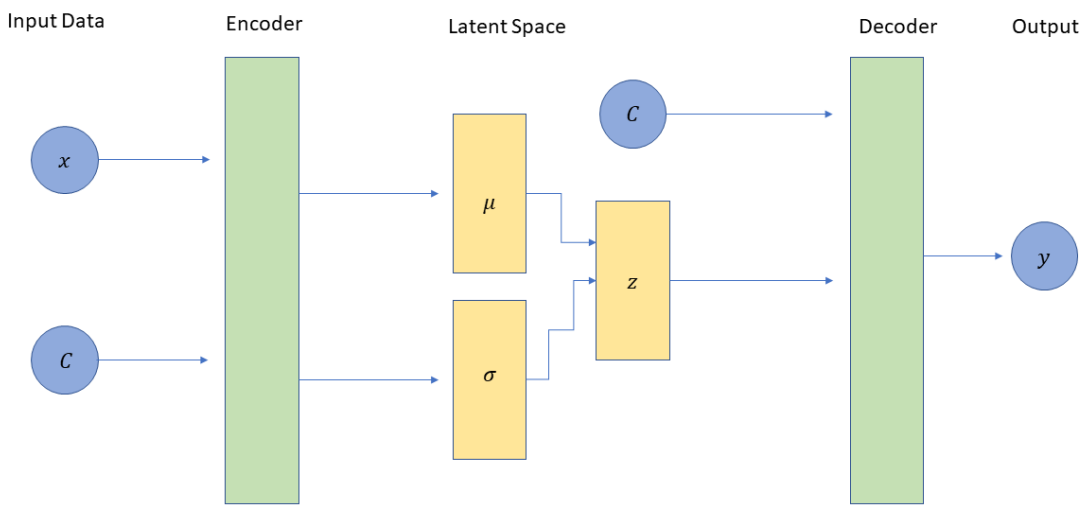


Figure 2.7: A simple CVAE diagram consisting of an encoder and decoder network. x data is given as input to the encoder and latent space samples are given as input to the decoder as was the case with the VAE diagram in Fig.2.6. However, now we additionally condition the encoder on the class of x by appending the class label in numerical form to x. We also condition the decoder on class c by appending it to the latent space samples z.

The natural question then arises, is it possible to condition the network on classes of interest? It turns out that this is possible to do by making a minor update to the VAE loss in equation (2.25)

$$\log(p(x_k)) \geq -D_{\mathrm{KL}}(q_\theta(z|x_k,c)||p(z,c)) + E_{\sim q_\theta(z|x_k,c)}[\log(p(x_k|z,c))]. \tag{2.40}$$

Here, we now condition the encoder $q_\theta(z|x_k,c)$ on both the training images x_k and the class of said images c (see Fig.2.7). In the decoder $p(x_k|z,c)$ we condition on samples from the latent space z and the class c. For classification purposes, the class is usually parameterized as a single scalar number which is appended to the input to the encoder and decoder networks. Now, if we wanted to use the decoder as a fashion image generator, we would only simply need to draw a random sample from the latent space, choose a class number, and then feed both as inputs to the decoder network.

2.5 Summary

In this chapter we introduced the concept of machine learning starting from one of the simplest neural networks, a perceptron. We then built on perceptrons discussed how perceptrons can be strung together to make a neural network layer, and from there layers can be stacked to form a multi-layer deep fully-connected neural network. I also provided some best practices on hyperparameter optimisation and training procedures. We introduce the concept of both a CNN and a CVAE, two of the networks primarily used in the author's thesis work.

Chapter 3

Machine Learning in Gravitational wave Astronomy

It is expected that in the coming years the LIGO detectors will see in increase in sensitivity. With this increase in sensitivity also comes an increase in the number signals which will be detectable by the collaboration. The number of signals per year is expected to be anywhere on the order of 100s to 1000s (depending on the source type). As such, it is imperative that we develop new techniques to deal with this massive influx of detectable sources. In addition, with the increase in signals there is also added pressure to develop more accurate and precise methods for waveform modeling, as well as enhanced techniques for quickly identifying non-astrophysical/environmental noise transients.

machine learning (ML) has been proven to be an excellent resource for tackling the above mentioned problems. Over the past several years, the gravitational wave community has seen a boon in the use of machine learning methods across a variety of application in the detection, parameter estimation and detector characterization algorithms (among many other domains). This surge in applications has seen marked success not only in proof of principles studies, but even studies involving the use of realistic LIGO data. Going forward, it is the hope of myself and other ML GW practitioners that these methods are eventually implemented in realtime during real observing runs in order to further enhance the capabilities of future gravitational astronomy research. In this chapter I will provide an overview of recent advances in ML approaches in GW

astronomy in signal detection, parameter estimation and detector characterization.

3.1 ML for GW detection

Algorithms which search for GWs are typically concerned with sources from three types of sources: CBC, Burst and stochastic GWs. CBC sources being composed of BBH, BNS and NSBH signals, Burst are unmoddeled/poorly modeled signals like supernova or even as yet unknown sources and stochastic GWs result from left over remnants of the Big Bang along with poorly resolved CBC signals from extreme distances. In this section I will outline several recent and seminal studies which use ML to directly detect GWs from a variety of the aforementioned sources listed above.

3.1.1 CBC detection studies

ML was first shown to perform to the same degree of accuracy as that of the standard approach used for signal detection (matched template filtering) by Gabbard et al. [46] and George&Heurta [51]. In Gabbard's work, we used simulated BBH waveforms in Gaussian noise and Gaussian noise alone to perform a binary classification task. Gaussian noise was used because it is known that matched template filtering should be ideal under Gaussian noise conditions, so we should not ideally expect the ML to do any better, but at a maximum match the accuracy of matched template filtering. Using a CNN, Gabbard et al. were able to show for the first time that deep learning could match the efficiency of matched template filtering. In George&Heurta, they also show that a similar CNN approach ("deep filtering") could reproduce predictions from matched template filtering. In addition to signal detection, they use a CNN for the first time to do point-estimate parameter estimation on both BBH and eccentric BBH waveforms which closely match the best-fitting templates predicted by matched template filtering. They additionally compare their machine learning approach to other common ML approaches including: K-nearest neighbor, support vector machines, logistic regression, etc. showing their CNN to have superior results.

There have been numerous follow-up studies which have explored the ML for CBC detection

landscape. Gebhardt et al. took a more critical approach and looked into the feasability of using CNNs under more realistic conditions. They claim that CNNs alone should not be used to claim a detection, but rather they are best suited as a realtime trigger generator which can be used to identify events of interest for later study. They support their claim by showing that a standard binary classification approach does not necessarily tell you where exactly to look in a time series window for an event and does not distinguish well between very loud and very quiet signals (high vs. low SNR). In other words, the "false alarm rate" is directly tied to the training set. They propose a solution to this by utilizing a fully-convolutional approach. The fully-convolutional approach is ideal because it can take an input of any arbitrary size and outputs a value between 0 and 1 for each element in the time series. If a value in the output fully-convolutional time series is 1 that means that there is a GW signal present at that location. They are successfully able to recover both real and simulated GW events in real detector noise.

Gebhardt et al. also perform tests of their approach where they try to fool the network into predicting the presence of a GW signal when none is there. This is done by learning "hypothetical signals" by exploiting the differentialbility of their model to produce what the model "believes" is a GW-like signal. They find that although many of these hypothetical signals do have chirp-like properties, there are a few concerning examples where the signals do not have any GW-like properties and even incredibly unphysical properties such as having only negative strain values. As is mentioned in the manuscript, there is still much study left to be done as to whether or not the examples used are too contrived and not something we would realistically expect to see in the detectors [49].

It was first shown by Krastev et al. that using artificial neural network (ANN)s one could accurately recover BNS signals in Gaussian noise. The problem was posed as a trinary classification task where the ANN was tasekd with distinguishing between BBH, BNS and Gaussian noise alone cases. The neural network was limited to a sliding window of $\sim$ 10s due to computational expense and was trained on more than 100,000 simulated training signals. Their results using ROC curves show that the ANN performs more poorly on BNS signals than on BBH signals. This could be due to a number of reasons including: sliding window size, network architecture choice, signal complexity, etc. The authors do not compare their approach to any

other existing currently used signal detection method such as matched template filtering [75].

In Marlin et al., the authors perform a much more in depth study and use an improved neural network architecture. In order to reduce the overall complexity of the input space they deal with 32s long BNS signals and vary the sample rate as a function of the different segments in time of the signal. For example, the first 16s are sampled at 128Hz, whereas the last second is sampled at 4096Hz. This is feasible because most of the SNR of the signal is contained in the final few seconds. Using this varied sample rate reduces the size of input signal space by a factor of ~ 9 and allows their study to be more realistic and has the ability to reach false alarm rates of about 0.5 per month. In addition to this clever sampling rate scheme, the authors improve on previous architecture approaches by employing an inception modules, temporal convolutions for signal amplification and tailoring inception modules for each different sampled section of the input time series. Their results show a 400% improvement over previous machine learning approaches, with a latency of ~ 10.2s. The authors compare their results to both traditional techniques like matched template filtering and the PyCBC live event trigger generator and find that the machine learning approach is not quite as sensitive as traditional techniques (by ~ 6 times lower). The authors mention that some general improvements could be made to the latency of the ML approach by reducing the complexity of the network, which may also improve the sensitivity [108].

3.1.2 Burst detection studies

The first implementation of a machine learning algorithm for the purpose of identifying Burst-like signals was by Astone et al. in 2018. In their paper they build upon the already existing continuous wave Burst pipeline used by the LIGO collaboration. Specifically, they are interested in searching for core-collapse supernovae events. Their method can be summarized in two steps. One, they prepare time-frequency images of detector data using the cwB event trigger generator which looks for times of excess power in the detectors. Once times of excess power have been identified, time-frequency spectrogram image is constructed for each active detector. In their case, they use three detectors and combine them in such a way that each detector essentially acts as a color channel in an red-green-blue (RBG) color scheme. The authors then use a standard

CNN to classify the identified time-frequency image into either noise or signal+noise. They test their network on a range of signals spanning SNR values from 8 up to ~ 40. They show that their method is overall more efficient than the standard Burst detection pipeline (cwB) at a False Alarm Rate of around 7×10^{-7}.

3.1.3 CW detection studies

For the CW search, the standard amount of time it takes to perform an all-sky coherent search can take well over a month to complete. Dreissigacker et al. try to decrease this computational time with respect to standard CW detection approaches using ResNet neural networks in the first application of deep learning for the CW search. They frame the problem such that their training set consists of two types of input. One, being long 10^6s signals and shorter 10^5s signals. Rather than doing a search over the entire frequency range, they also limit their test cases to a discrete set of frequency bands at (20, 100, 200, 500 and 1000 Hz) due to computational feasibility. They compare their machine learning results to a "gold standard" CW search method pipeline `WEAVE`. `WEAVE` sums coherent F-statistics over semicoherent segments on lattice-based GW template banks, although their study employs the fully coherent search version of the code.

For the training set, they generate over 10,000 training waveforms in the frequency domain with half containing Gaussian noise and the other half containing CW+noise. They found that using a set greater than 10,000 provided little gain in efficiency. After training the authors found that at a fixed false alarm rate, the deep learning approach appears to be competitive ($88-73\%$ detection rate) with the matched filtering search ($\sim 90\%$) on short time scale inputs (10^5s), whereas when tasked with longer time scale inputs (10^6s) it performs a fair bit worse ($69-13\%$). The network did outperform the matched template search by an immense amount in terms of computational speed taking only on the order of a few milliseconds ($3-10$ms) as opposed to the 10^6-10^9s of the matched filter method. As the authors state, there is much more work to be done in order to improve the efficiency of deep neural networks within this subject domain, but there is also much promise given the quality of results achieved thus far.

3.2 ML for GW Bayesian parameter estimation

For many, GW parameter estimation was the "holy grail" of machine learning in the field of gravitational wave astronomy. Although prior to 2019 there were some attempts to compute point estimates for GW source parameters directly, as well as to incorporate ANNs directly into subsections of existing Bayesian inference algorithms [56], there had yet up to that point been an end-to-end algorithm which went from strain to directly computing samples from the Bayesian posterior. In this subsection, I will outline some of the initial attempts to incorporate AI into existing pipelines, ML for point estimate parameter estimation and finally ML to directly compute samples from the Bayesian posterior.

ML was first used in the context of GW parameter estimation by Graff et al. in their modified nested sampling algorithm `BAMBI`. In `BAMBI` the authors exploit the tenants of the universal approximation theorem, which implies that a properly optimized ANN should be able to approximate any sufficiently complicated likelihood function. This is done by essentially replacing the likelihood calculation in the `MultiNest` nested sampling algorithm. After the nested sampling algorithm has produced a sufficient number of samples, their ANN is trained on those samples. The input to the ANN are the sample parameter values and the output is a single value which represents the likelihood values for each of those samples. The ANN is then trained such that it's predictions for the likelihood values matches that produced by the nested sampling algorithm. In order to quantify when the ANN has been trained sufficiently to fully replace the standard likelihood function, a tolerance level is calculated. The tolerance is defined as the standard deviation of the difference between the true log-likelihood values from nested sampling and the predicted values from the ANN. The actual value of this tolerance level may be specified by the user. Once the network has reached an acceptable tolerance level of performance it then replaces the original likelihood function calculation. Periodic tolerance checks are made throughout the rest of the nested sampling iterations to ensure that the ANN is still providing solid predictions.

Generally speaking, the authors find that the algorithm preforms well across a variety of toy model cases (Gaussian shells, Rosenbrock functions, eggbox). They also compare their approach (`BAMBI`) to `MultiNest` on a few real physics examples including trying to predict the

posteriors of 8 cosmological parameters. The gain in speed goes from taking on the order of seconds per likelihood calculation, down to milliseconds, a three order of magnitude increase in computational performance with accurate evidence predictions and posterior probability distributions [56].

Released at the same time as Gabbard et al., Chua et al. also aimed to show for the first time that machine learning methods could accurately approximate Bayesian GW posteriors. Their method used fully-connected neural networks. Specifically, they represent their training waveforms in quite an interesting manner by training a neural network to predict coefficients which describe a reduced order framework in order to quickly produce training GW waveforms. Technically, it's actually two networks which produce the real and imaginary components of the waveform separately. To produce posteriors, Chua et al. again use a fully-connected neural network which takes as input the low-latency training waveforms (in Gaussian noise) and outputs a histogram where each value of the histogram bins is predicted by the network. The loss function is the cross entropy between the natural log of the predicted network histogram bins and the training waveform source parameter values (all summed and minimised during training).

Chua test and train their model on binary black hole waveforms in Gaussian noise with masses ranging from $1.5 \times 10^5 - 10 \times 10^5$ with aligned spins between -1 and 1. They then train their model to predict both the chirp mass and the symmetric mass ratio η. Their results show a good agreement between the histograms produced by their machine learning approach and traditional sampling approaches [33, 34].

Following on from Chua and Gabbard, Green wanted to tackle two outstanding problems which both Chua and Gabbard had yet to solve at the time. With Chua, the main drawback from their approach was the limited number of dimensions that their algorithm could produce as output predictions (no more than 2). For Gabbard, it was the difficulty of dealing with multi-modal posteriors (which was later rectified in subsequent updates to their paper). In Green's first paper, they implemented a new approach using a combination of masked autoregressive flow (MAF)s and CVAEs.

A MAF is a mechanism for mapping simple distributions to more complex distributions, while also retaining the property of being invertible. The complex distribution is dependent

on an inverse Jacobian determinant and the simple distribution (usually represented as some form of a Gaussian distribution). A MAF is normally represented as a neural network where the weights and biases are used to learn this invertible mapping. The simple distributions are typically represented as multivariate Gaussian conditional distributions. In their paper, Green et al. utilize three distinctly different models to model Bayesian posteriors: CVAE alone, MAF alone and a combination of a CVAE and a MAF. The combined CVAE+MAF network, is made by appending flows to the end of both encoders and the decoder network of the CVAE in order to better model the sample space. The authors test their methods on 1s long BBH signals in Gaussian noise sampled at 1024Hz and try to predict a 5 parameter case (m1, m2, phase, time of coalescence, distance) and an eight dimensional case (m1, m2, phase time of coalescence, distance, chi1, chi2, inclination angle). Prior to training, they perform a unique preprocessing step whereby they transform their input via principal component analysis into 100 principal components, thereby reducing the dimensionality of the input and vastly improving the overall computational cost of training/testing their network. They find that all three methods perform reasonably well, although the CVAE alone method is not able to resolve the multi-modal source parameters such as phase [58].

In a subsequent paper, Green et al. did away entirely with their three model approach and went with a pure MAF flow network with spline couplings. Training of their model is done in the standard way and interestingly, they test their model for the first time on a known GW signal in Gaussian noise (GW150914). After training, they find that there approach is able to to closely match a standard Bayesian sampler (`Dynesty`) well with minor inaccuracies on the inclination angle where the ML approach puts more likelihood on one of the two modes in the posterior space [57].

Also using normalizing flows, Williams et al. where able to replace the likelihood calculation stage in an existing nested sampling algorithm with flows in an algorithm dubbed `NESSAI`. Specifically, the authors use coupling flows because of their computational cheapness, although with the added disadvantage of tending to be slightly less flexible than autoregressive flows. The flow is trained during the nested sampling process. Once properly trained, each time a new set of live points is required to be evolved, `NESSAI` can produce on-command a new set of live

points almost instantaneously.

3.3 ML for population inference

Given the recent accumulation of GW detections over the past several LIGO observation runs, we are now able to perform more detailed studies across a population of GW events. It is also expected that as detector upgrades are made and new detectors come online, that we will start to see hundreds to thousands of GW events per year. Such abundunce of signals will only enhance population studies which aid us in understanding GW signal formation channels, the general properties of GW source parameters (masses, redshifts, spins etc.), progenitor merger rates, as well as modifications of GR [16]. In order to constrain certain phenomenological models, it is common to employ the use of Bayes factors between models with different parameters under the same parameterization. Because the Bayes factor involves the computationally costly calculation of marginal model likelihood ratios, it can be intractable to perform some population studies depending on the complexity of the population dataset used.

Wong et al. showed for the first time that a hybrid normalizing flow and hierarchical Bayesian model framework were able to accurately constrain GW population models [132]. Although other attempts at using forms of ML to perform simulation-based population inference have been employed [117, 132], those approaches only worked well up to a limited dataset dimensionality (~ 2 event parameters). Wong et al. show that their technique performs well over what were once considered highly intractable models. They compare their results to simulations using an analytic phenomenological model. Although the authors neglect to take into account event uncertainties and selection bias, their ML technique is shown to scale well up to 3000 GW events with 100 samples per posterior and 6 parameters in ~ 0.1s. Given that their model performs just as efficient as other analytic approaches in a fraction of the time, it is likely that normalizing flows will play a crucial role in testing different state of the art models and even families of models in future GW event catalogue releases. As is the case with most ML approaches, more data and greater ML model complexity may also improve results.

3.4 ML for detector characterization

Nonastrophysical and environmental noise sources can affect the data quality of the detectors adversely. Poor data quality can not only be a source of confusion with doing data analysis, but can even entirely corrupt segments of data. It is the job of the detector characterization group to identify and mitigate such noise disturbances. If we are able to identify and prioritize classes of glitches, this is an important first step in mitigating such events. In this subsection I will describe some recent efforts to use ML to identify and mitigate noise in the LIGO detectors.

In order to better classify known and unknown classes of "glitches", Zevin et al. [136] use a unique combination of human learning and machine learning in a pipeline called `Gravity Spy`. In `Gravity Spy` a set of glitch triggers are chosen for training which occur in "lock" (meaning the detector was in a proper state to search for gravitational waves) and not during times of poor data quality. The glitches themselves are represented as Q-scans, which are essentially time-frequency tilings where Q represents a quality factor of a specific sine-Gaussian waveform.

The initial training set was generated from a set of 10^5 Omicron glitch triggers, from which about 100 glitches per class were identified by eye from trained detector characterization experts. These 100 "gold-standard" glitches were then used to train a preliminary neural network (a basic deep CNN model) to identify the remaining glitches in the rest of the 10^5 Omicron triggers. These triggers are then uploaded to the Zooniverse website where volunteer citizen scientists are able to try classifying the trigger spectrograms by eye. Volunteers are initially trained on "gold standard" glitch images and other well known glitches which have a high probability of belonging to a specific class. After successive successful classifications by volunteers, the difficulty of glitches presented to the volunteers is ramped up quickly. If a glitch is identified enough times by both the machine learning algorithm and volunteers to a high degree of confidence, it is then "retired" whereby it is taken out of the volunteer glitch lookup pool and then added to the machine learning labeled training set to improve the accuracy of the machine learning model.

Overall, `Gravity Spy` has been an incredibly successful use of ML and a unique example of utilizing citizen scientists for great gain. Citizen scientists were able to identify more than

45,000 glitches and several new glitch classes such as "Paired Doves" and "Helix" glitches. The "Paired Doves" class was a particularly useful identification as it closely resembled that of a company binary inspiral signals.

One of the downsides of many ML algorithms (and a topic I'm personally very interested in) is the idea of interpretability. Many ML models such as CNNs, generative adversarial network (GAN)s, CVAEs, etc. while powerful are composed of millions of hyperparameters which interact in complicated non-linear ways. Trying to interpret why a particular network configuration works over another is an active area of ongoing research in the ML community. Genetic algorithms attempt to partially solve this issue. A genetic algorithm is similar to a decision tree, which is made up of mathematical operators and variable place holders. The algorithm can be trained to do both regression and classification tasks. It is trained by first creating an initial population of operands and operators which will be evolved during training. From the initial population some variables are chosen to remain unchanged, while others are chosen to be mutated (changed) based off of a user defined fitness score. There are a variety of methods by which variables may be mutated. Through training and evolution, the optimal set of variables of the tree is chosen and may be accessed post-training in order to see what parts of the input ended up in the tree.

Genetic algorithms were first used in LIGO by Cavaglia et al. in [3]. This was done by training a genetic algorithm over many auxiliary channel degrees of freedom in the detector during periods of observing data where there were known glitch classes (for comparison, a random forest algorithm was also run over the same dataset). They tested their algorithm on two glitch classes from both the first observing run and the second LIGO observing run. The first set are EX magnetometer glitches from O2 and the second set or those from an air compressor seen in O1. Their training set consists of 2000 Omicron triggers (greater than SNR 5.5) and 749 auxiliary channels for the magnetometer glitches and 16 trigger s and 429 auxiliary channels for the compressor glitches both with a training/testing split of 2/3 and 1/3 respectively. When testing, the authors find that the genetic algorithm determines that 7/10 (9/10 for the random forest algorithm) of the most likelely auxiliary channels for the magnetometer gltiches result from the correct magnitometer LIGO detector subsystem (ETX). Although a bit noiser than the

magnitometer set (likely due to the lower number of training samples) both the GP and the RF algorithms are able to correctly identify the source of the most likely auxiliary channels for the air compressor glitches.

What the work of Cavaglia et al. show is that both random forest (RF) and genetic programming (GP) have the ability to work well with complex gravitational wave channel data to identify complicated relationships between gravitaitonal wave subsystems in a human readable fashion. The clear advantage of using such algorithms is in their intpretibility and non-black box-like behavior. The black box nature of many deep ML algorithms is an outstanding problem not only in ML, but also in terms of the acceptability of ML in GW astronomy. Going forward, I believe that more work will need to be done in order to understand what features ML algorithms determine to be most important and in what instances an ML algorithm may be fooled into returning false positives, which I believe algorithms like GP will have an important role to play.

3.5 Summary

Chapter 4

Matching matched template filtering using deep networks for gravitational wave astronomy

4.1 Introduction

The field of gravitational-wave astronomy has seen an explosion of compact binary coalescence detections over the past several years. The first of these were binary black hole detections [9, 10, 12] and more recently the advanced detector network made the first detection of a binary neutron star system [14]. This latter event was seen in conjunction with a gamma-ray burst [53, 80, 107] and multiple post-merger electromagnetic signatures [81]. These detections were made possible by the Advanced Laser Interferometer Gravitational wave Observatory (aLIGO) detectors, as well as the recent joint detections of GW170814 and GW170817 with Advanced Virgo [13, 14]. Over the coming years many more such observations, including BBH, binary neutron stars, as well as other more exotic sources are likely to be observed on a more frequent basis. As such, the need for more efficient search methods will be more pertinent as the detectors increase in sensitivity.

The algorithms used by the search pipelines to make detections [30, 38, 124] are, in general, computationally expensive. The methods used are complex, sophisticated processes computed

over a large parameter space using advanced signal processing techniques. The computational cost to run the search analysis is due to the large parameter space, as well as analysis of the high frequency components of the waveform where high data sample rates are required. Distinguishing noise from signal in these search pipelines is acheived, in part, using a technique known as template based matched-filtering.

Matched-filtering uses a bank [18, 27, 31, 38, 59] of template waveforms [26, 101, 105, 116] each with different component mass and/or spin values. A template bank will span a large astrophysical parameter space since we do not know a priori the true gravitational-waves parameter values. Waveform models that cover the inspiral, merger, and ringdown phases of a compact binary coalescence are based on combining post-Newtonian theory [21, 26, 29, 84], the effective-one-body formalism [28], and numerical relativity simulations [100].

Deep learning is a subset of machine learning which has gained in popularity in recent years [32, 55, 76, 110, 115, 135] with the rapid development of graphics-processing-unit technology. Some successful implementations of deep learning include image processing [69, 76, 137], medical diagnosis [74], and microarray gene expression classification [99]. There has also been some recent success in the field of gravitational-wave astronomy in the form of glitch classification [52, 85, 136] and notably for signal identification [48, 50] where it was first shown that deep learning could be a detection tool [50]. Deep learning is able to perform analyses rapidly since the method's computationally intensive stage is pre-computed during the training prior to the analysis of actual data [54]. This could result in low-latency searches that have the potential to be orders of magnitude faster than other comparable classification methods.

A deep learning algorithm is composed of stacked arrays of processing units, called neurons, which can be from one to several layers deep. A neuron acts as a filter, whereby it performs a transformation on an array of inputs. This transformation is a linear operation between the input array and the weight and bias parameters assosicated to the neuron. The resulting array is then typically passed to a non-linear activation function to constrain the neuron output to be within a set range. Deep learning algorithms typically consist of an input layer, followed by one to several hidden layers and then one to multiple output neurons. The scalars produced from the output neurons can be used to solve classification problems, where each output neuron corresponds to

the probability that an input sample is of a certain class.

In this letter we investigate the simplest case of establishing whether a signal is present in the data or if the data contains only detector noise. We propose a deep learning procedure requiring only the raw data time series as input with minimal signal pre-processing. We compare the results of our network with the widely used matched-filtering technique and show how a deep learning approach can be pre-trained using simulated data-sets and applied in low-latency to achieve the same sensitivity as established matched-filtering techniques.

4.2 Simulation details

In order to make a clean comparison between deep learning approach and matched-filtering, we distinguish between two cases, BBH merger signals in additive Gaussian noise (signal+noise) and Gaussian noise alone (noise-only). We choose to focus on BBH signals rather than including binary neutron star systems for the reason that BBH systems are higher mass systems and have shorter duration signals once the inspiralling systems have entered the Advanced LIGO frequency band. They typically then merge on the timescale of $\mathcal{O}(1s)$ allowing us to use relatively small datasets for this study.

The input datasets consist of "whitened" simulated gravitational-wave timeseries where the whitening process uses the detector noise PSD to rescale the noise contribution at each frequency to have equal power. Our noise is initially generated from a PSD equivalent to the Advanced LIGO design sensitivity [5].

Signals are simulated using a library of gravitational-wave data analysis routines called `LALSuite`. We use the IMRPhenomD type waveform [64, 72] which models the inspiral, merger and ringdown components of BBH gravitational-wave signals. We simulate systems with component black hole masses in the range from $5M_{\odot}$ to $95M_{\odot}$, $m_1 > m_2$, with zero spin. Training, validation and testing datasets contain signals drawn from an astrophysically motivated distribution where we assume $m_{1,2} \sim \log m_{1,2}$ [7]. Each signal is given a random right ascension and declination assuming an isotropic prior on the sky, the polarization angle and phase are drawn from a uniform prior on the range $[0, 2\pi]$, and the inclination angle is drawn such that the

cosine of inclination is uniform on the range $[-1,1]$. The waveforms are then randomly placed within the time series such that the peak amplitude of each waveform is randomly positioned within the fractional range $[0.75,0.95]$ of the timeseries.

The waveform amplitude is scaled to achieve a predefined optimal SNR defined as

$$\rho^2_{\rm opt} = 4\int_{f_{\rm min}}^{\infty} \frac{|h(\tilde{f})|^2}{S_{\rm n}(f)} df, \tag{4.1}$$

where $h(\tilde{f})$ is the frequency domain representation of the gravitational-wave strain and $S_{\rm n}(f)$ is the single-sided detector noise PSD [20]. The simulated time series were chosen to be 1 s in duration sampled at 8192 Hz. Therefore we consider $f_{\rm min}$ as the frequency of the gravitational-wave signal at the start of the sample timeseries. An example timeseries can be seen in Fig. 4.1.

Due to the requirements of the matched-filtering comparison it was necessary to add padding to each timeseries so as to avoid non-physical boundary artefacts from the whitening procedure. The Gaussianity of the noise and smoothness of the simulated advanced LIGO PSD allows the use of relatively short padding. Therefore each 1 s timeseries has an additional 0.5 s of data prior to and after the signal. The signal itself has a Tukey window ($\alpha = 1/8$) applied to truncate the signal content to the central 1 s. The CNN approach only has access to this central 1 s of data. Similarly, the optimal SNR is computed considering only the central 1 s.

Supervised deep learning requires datasets to be sub-divided into training, validation, and testing sets. Training sets are the data samples that the network learns from, the validation set allows the developer to verify that the network is learning correctly, and the test set is used to quantify the performance of the trained network. In a practical scenario the training and validation sets are used to train the network prior to data taking. This constitutes the vast majority of computational effort and is a procedure that needs to be computed only once. The trained network can then be applied to test data at a vastly reduced cost in comparison to the training stage [54]. Of the dataset generated we use 90% of these samples for training, 5% for validation, and 5% for testing. A dataset was generated for each predefined optimal SNR value ranging from 1–10 in integer steps.

Our training datasets contain 5×10^5 independent timeseries with 50% containing signal+noise

and 50% noise-only. For each simulated gravitational-wave signal (drawn from the signal parameter space) we generate 25 independent noise realizations from which 25 signal+noise samples are produced. This procedure is standard within machine learning classification and allows the network to learn how to identify individual signals under different noise scenarios. Each noise-only sample consists of an independent noise realization and in total we therefore use 10000 unique waveforms in the m_1, m_2 mass space. Each data sample timeseries is then represented in the form of a 1×8192 pixel image with the gray-scale intensity of each pixel proportional to the gravitational-wave amplitude.

We will only use the training set to teach the deep neural network about the signals that we wish to classify. For this reason we have chosen to distribute the masses of the training set according to the density of templates one would use in a matched-filtering approach derived from [24]. The idealized density of templates is expected to be somewhat different [68]. It is logical to place waveforms closer in regions where the waveforms change rapidly and hence we distribute the masses according to the total mass $M \sim M^{10/3}$ and the symmetric mass ratio $\eta \sim \eta^{-3}$ where $M = m_1 + m_2$ and $\eta = m_1 m_2 / M^2$. The validation, and crucially the testing set must be drawn from distributions that one would expect to see in nature and we therefore draw those masses from astrophysically motivated distributions where we assume $m_{1,2} \sim \log m_{1,2}$.

4.3 The Deep Network approach

In our model, we use a variant of a deep learning algorithm called a CNN [78] composed of multiple layers. The input layer holds the raw pixel values of the sample image which, in our case, is a 1-dimensional timeseries vector. The weight and bias parameters of the network are also in 1-dimensional vector form. Each neuron in the convolutional layer computes the convolution between the neuron's weight vector and the outputs from the layer below it, and then the result is summed with the bias vector. Neuron weight vectors are updated through an optimisation algorithm called back-propogation [79]. Activation functions apply an element-wise non-linear operation rescaling their inputs onto a specific range and leaving the size of the previous layer's output unchanged. Pooling layers perform a downsampling operation along the

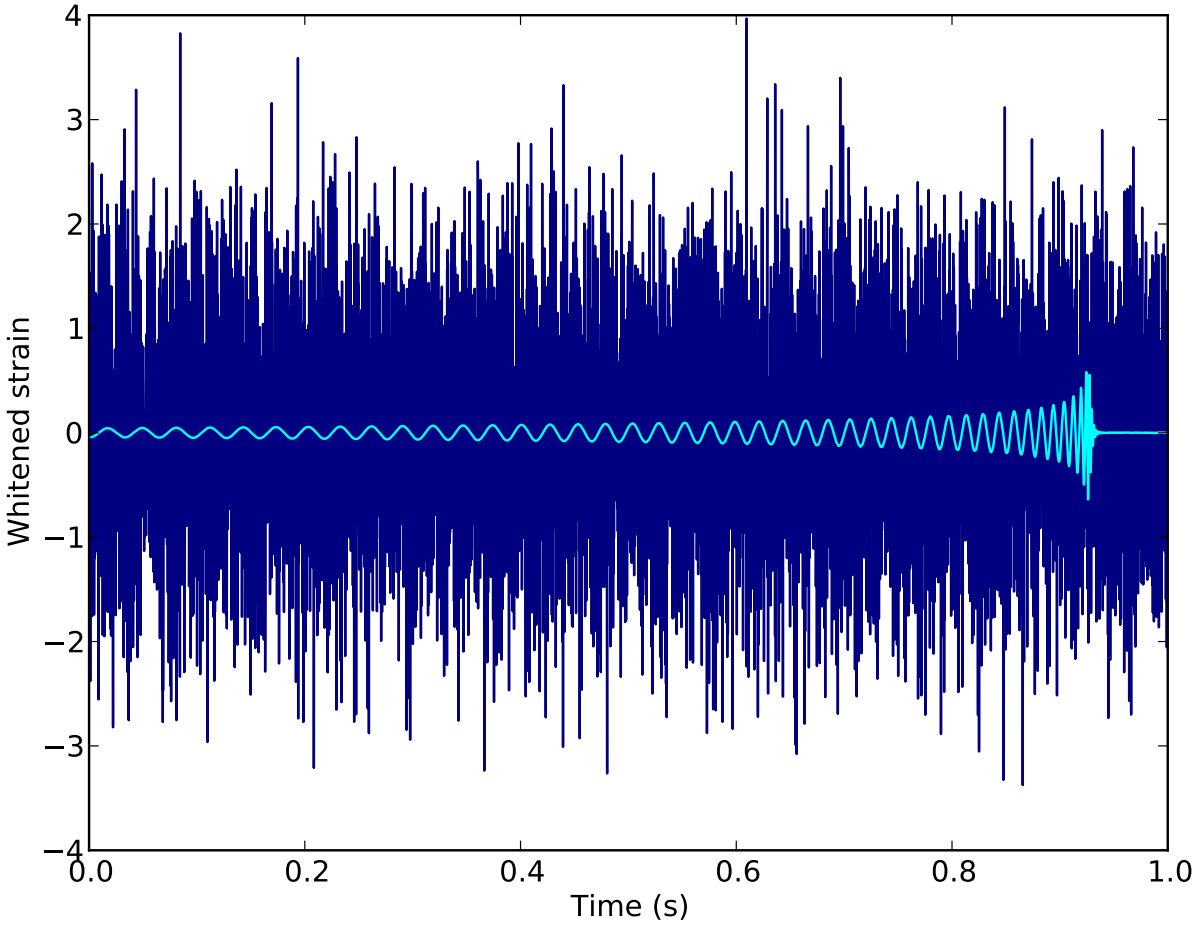


Figure 4.1: A whitened noise-free timeseries of a BBH signal with component masses $m_1 = 41.86\mathrm{M}_\odot$ and $m_2 = 6.65\mathrm{M}_\odot$ with optimal SNR $= 8$ (cyan). The dark blue timeseries shows the same gravitational-wave signal with additive whitened Gaussian noise of unit variance. This latter timeseries is representative of the datasets used to train, validate, and test the deep neural network.

spatial dimensions of their input. Finally we have a hidden layer connected to an output layer which computes the inferred class probabilities. These values are input to a loss function, chosen as the binary cross-entropy [4], defined as

$$f(\theta) = -\sum_{i \in \mathrm{S}} \log(\theta_i^{\mathrm{S}}) - \sum_{i \in \mathrm{N}} \log(\theta_i^{\mathrm{N}}), \tag{4.2}$$

where $\theta_i^{\mathrm{S/N}}$ is the predicted probability of class signal+noise (S) or noise-only (N) for the i'th training sample. The loss function is minimised when input data samples are assigned the correct class with the highest confidence.

In order to optimise a network, multiple hyper-parameters must be tuned. We define hyper-parameters as parameters we are free to choose. Such parameters include the number and type of network layers, the number of neurons within each layer, size of the neuron weight vectors, max-pooling parameters, type of activation functions, preprocessing of input data, learning rate, and the application (or otherwise) of specific deep learning techniques. We begin the process with the simplest network that provides a discernible level of effective classification. In most cases this consists of an input, convolutional, hidden, and logistic output layer. The optimal network structure was determined through multiple tests and tunings of hyperparameters by means of trial and error.

Within our optimization process we experimented with rescaling the input data which we found to have minimal effect on the network performance. The reason for this is that our input data is whitened and our signals are buried beneath the noise. Therefore our data is effectively prescaled on a range $\pm\mathscr{O}(1)$ due to the natural variation of Gaussian noise. We also experimented with using transfer learning [96] where networks pre-trained on high SNR datasets are used as starting points for application to successively lower SNR datasets. We found that there were no performance benefits in using this approach compared to training the network solely on each SNR dataset separately. The network depth was adjusted between 2 and 10 convolutional layers. The inclusion of dropout was used within the final 2 hidden layers as a form of regularization to avoid overfitting.

During the training stage an optimization function (back-propagation) works by computing

the gradient of the loss function (Eq. 4.2), then attempting to minimize that loss function. The errors are then propagated back through the network while also updating the weight and bias terms accordingly. Back propagation is done over multiple iterations called epochs. We use adaptive moment estimation with incorporated Nesterov momentum [39]

$$v_{i+1} = \mu v_i - \varepsilon \nabla f(\theta_i + \mu v_i), \tag{4.3}$$

$$\theta_{i+1} = \theta_i + v_{i+1}, \tag{4.4}$$

where θ_i are the parameters of the neural network from the previous layer, θ_{i+1} are the parameters of the current layer of the network, v_{i+1} is the momentum for the current layer, μ is a scalar constant term which determines the amount of momentum to apply per gradient update (the higher the number, the more momentum), v_i is the momentum term from the previous layer, ε is the learning rate, $\nabla f(\theta_i + \mu v_i)$ is the gradient of the model parameters with respect to the previous layer (including the momentum term from the previous layer (μv_i)).

We also use a learning rate of 0.002, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$ and a momentum schedule of 0.004. We outline the structure of the final neural network architecture in Table 4.1.

The final ranking statistic that we extract from the CNN analysis is taken from the output layer, composed of 2 neurons, where each neuron will produce a probability value between 0 and 1 with their sum being unity. Each neuron gives the inferred probability that the input data belongs to the noise or signal+noise class respectively. The computational time spent on training the network for each SNR is $\mathcal{O}(1)$ hour on a single GPU. This one-time cost can be compared to the $\mathcal{O}(1s)$ spent applying the trained network to all $25,000$ 1 s test data samples also using a single GPU. Therefore at the point of data taking this particular analysis can be run at 10^4 times faster than real-time.

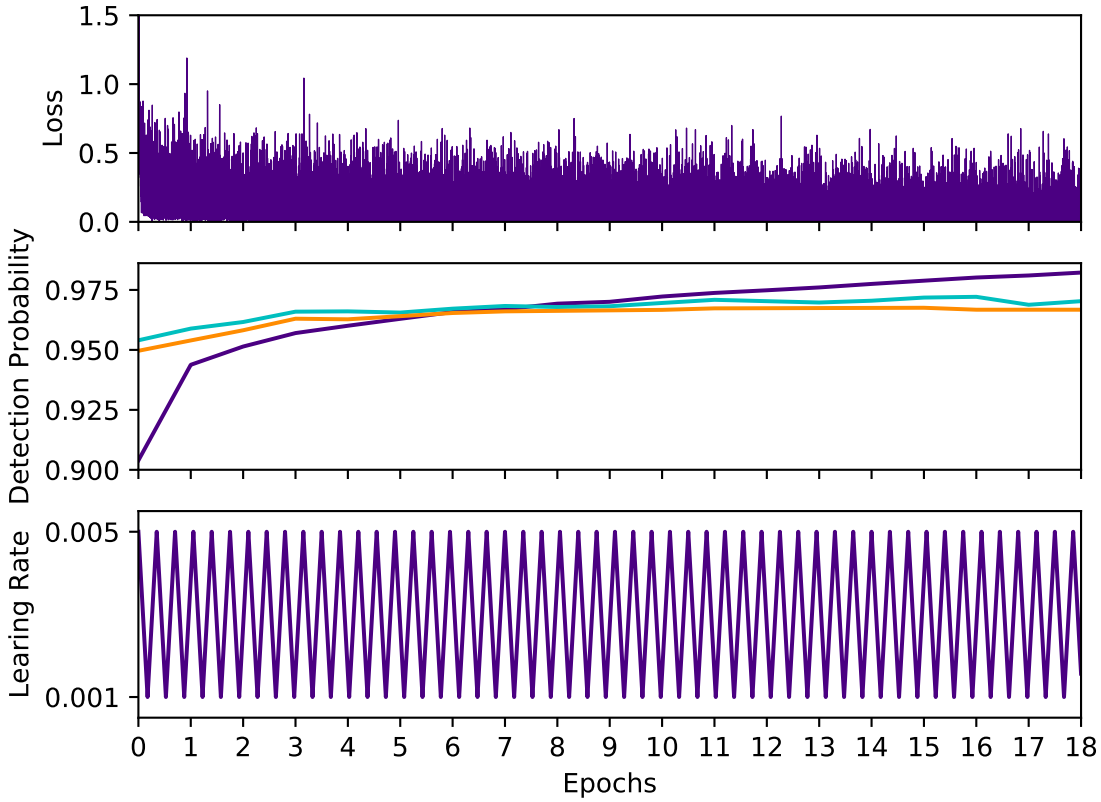


Figure 4.2: The loss, detection probability (accuracy) and learning rate plots (shown above) illustrate how the network's performance is defined as a function of the number of training epochs. The goal is to minimize the loss function, which will in turn maximize the accuracy of the classifier. The first initial epochs see an exponential decrease in the loss function and then a slowly falling monotonic curve to follow. This indicates that the longer our network is trained, a limit with respect to the accuracy is approached. In our case, we cyclically adjust the learning rate to oscillate between 5×10^{-4} and 1×10^{-3} at a constant frequency. The detection probability on training samples is represented by the purple curve, validation detection probability as the orange, and testing detection probability as the blue curve.

Parameter	Layer								
(Option)	1	2	3	4	5	6	7	8	9
Type	C	C	C	C	C	C	H	H	H
No. Neurons	8	8	16	16	32	32	64	64	2
Filter Size	64	32	32	16	16	16	n/a	n/a	n/a
MaxPool Size	n/a	8	n/a	6	n/a	4	n/a	n/a	n/a
Drop out	0	0	0	0	0	0	0.5	0.5	0
Act. Func.	Elu	Elu	Elu	Elu	Elu	Elu	Elu	Elu	SMax

Table 4.1: The optimised network consisting of 6 convolutional layers (C), followed by 3 hidden layers (H). Max-pooling is performed on the first, fifth, and eighth layer, whereas dropout is only performed on the two hidden layers. Each layer uses an exponential linear unit (Elu) activation function (with range $[-1,\infty]$) while the last layer uses a Softmax (SMax) activation function in order to normalize the output values to be between zero and one so as to give a probability value for each class.

4.4 Applying matched-filtering

In order to establish the power of the deep learning approach we must compare our results to the standard matched-filtering process used in the detection of compact binary coalescence signals [20, 25]. The ranking statistic used in this case is the matched-filter SNR numerically maximized over arrival time, phase and distance. By first defining the noise weighted inner product as a function of a time shift Δt between the arrival time of the signal and the template,

$$(a \mid b)[\Delta t] = 4 \int_{f_{\min}}^{\infty} \frac{\tilde{a}(f)\tilde{b}^*(f)}{S_{\rm n}(f)} e^{2\pi i f \Delta t}\, df, \tag{4.5}$$

we can construct the matched-filter SNR as

$$\rho^2[\Delta t] = \frac{(s \mid h)^2[\Delta t] + i(s \mid h)^2[\Delta t]}{(h \mid h)} \tag{4.6}$$

where s is the data containing noise and a potential signal, and h is the noise-free gravitational-wave template [24]. For a given template this quantity is efficiently computed using the FFT and the SNR timeseries maximised over Δt. The subsequent step is to further numerically maximize this quantity over a collection of component mass combinations. In this analysis a comprehensive template bank is generated in the m_1, m_2 mass space covering our predefined range of masses. We use a maximum mismatch of 3% and a lower frequency cutoff of 20 Hz using the

PyCBC geometric non-spinning template bank generation tool [91, 124]. This template bank contained 8056 individual templates.

When generating an SNR timeseries for an input dataset we select f_{min} according to the conservative case (lowest f_{min}) in which the signal merger occurs at the 0.95 fraction of 1 s timeseries. We therefore select only maximised SNR timeseries values recovered from within the $[0.75, 0.95]$ fractional range since this is the parameter space on which the CNN has been trained. For the practical computation of the matched-filtering analysis we take each of the data samples from the testing dataset to compute the matched-filter ranking statistic.

4.5 Results

After tuning the multiple hyper-parameters (Table 4.1) and training the neural network, we present the results of our CNN classifier on a noise versus signal+noise sample set. With values of statistics now assigned to each test data sample from both the CNN and matched-filtering approaches, and having knowledge of the true class associated with each sample, we may now construct receiver operator characteristic (ROC) curves.

A standard method for displaying the accuracy of a classifier is through a confusion matrix in which the number of samples of each true class identified as every possible class are listed in a square matrix. A fully diagonal matrix would imply no incorrectly classified samples and a uniform matrix would imply no classification power. In Figure 4.3 we show results for the CNN approach from which we highlight the overall accuracy (the ratio of incorrectly identified samples to the total number of samples) of % at $\rho_{\text{opt}} = 8$.

In Fig. 4.4 we compare our CNN results to that of matched-filtering. Given the ranking statistic from a particular analysis and defining a parametric threshold value on that statistic we are able to plot the fraction of noise samples incorrectly identified as signals (false alarm probability) versus the fraction of signal samples correctly identified (true alarm probability). These curves are defined as ROC curves and a ranking statistic is deemed superior to another if at a given false alarm probability it achieves a higher detection probability. Our results show that the CNN approach closely matches the sensitivity of matched-filtering for all test datasets

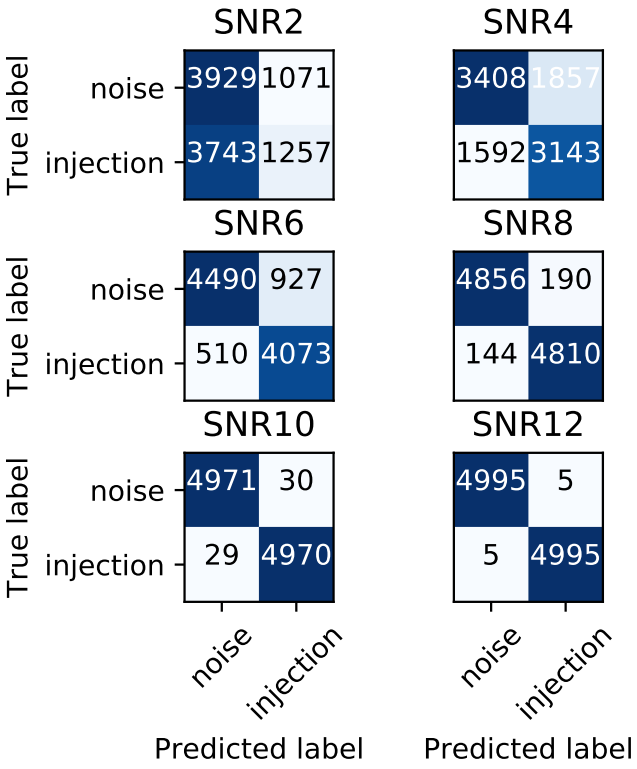


Figure 4.3: Confusion matrices for testing datasets containing signals with optimal SNR $\rho_{\text{opt}} = 2, 4, 6, 8, 10, 12$. Numbers superimposed within matrix elements are the number of samples corresponding to samples that were of true class indicated by the y-axis label but identified as the corresponding x-axis label. For our 2 class system these are equivalent to the numbers of true alarm, true dismissal, false dismissal, or false alarm. The accuracy percentages for all injection SNR values are listed as follows: 51.86% at $= 2$, 65.51% at $= 4$, 85.63% at $= 6$, 96.66% at $= 8$, 99.41% at $= 10$ and 99.99% at $= 12$.

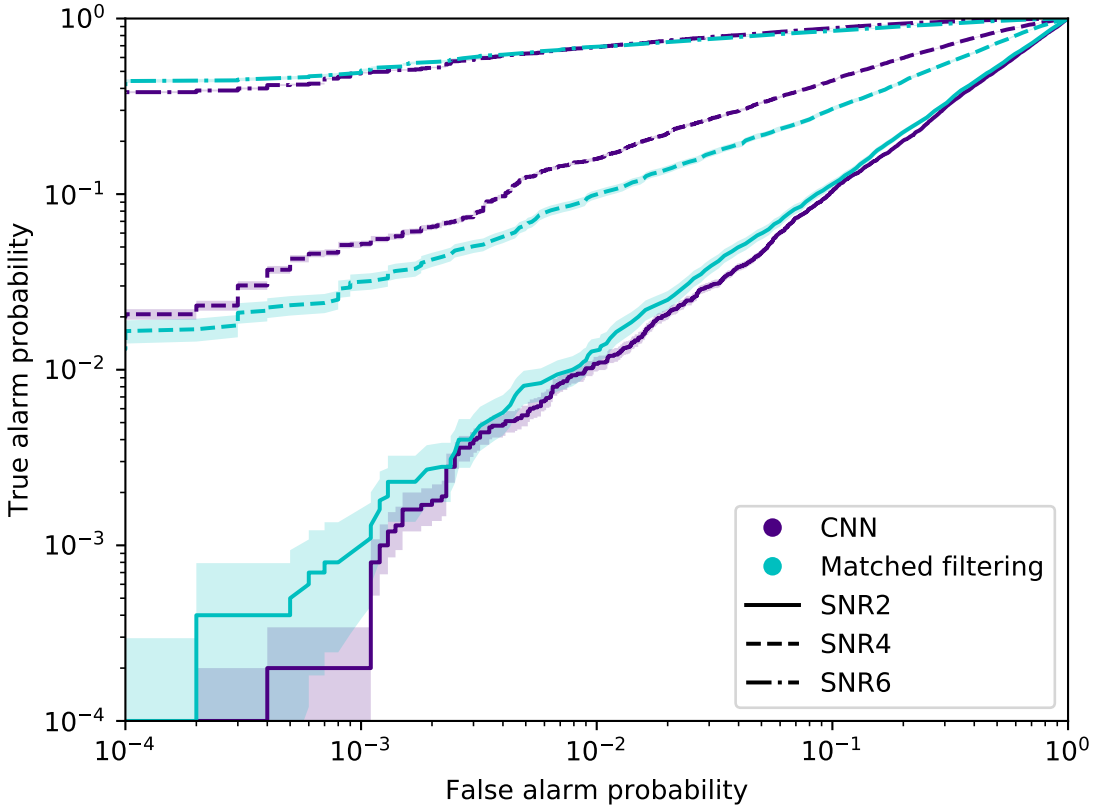


Figure 4.4: The ROC curves for test datasets containing signals with optimal SNR, $\rho_{\text{opt}} = 2, 4, 6$. We plot the true alarm probability versus the false alarm probability estimated from the output of the CNN (purple) and matched-filtering (cyan) approaches. Uncertainties in the true alarm probability correspond to 1-σ bounds assuming a binomial distribution.

across the range of false alarm probabilities explored in this analysis[1]. It is not surprising to see that the matched-filtering method using the optimal template consistently performs better than both the nominal match-filtering method and our deep learning classifier. However, what is considerable is the comparison between the nominal matched filtering and the deep learning classifier detection probability curves. It can clearly be seen that our classifier exceeds the performance of the nominal matched-filtering method at SNR 2, 4 and 6. This is a promising result and certainly merits further investigation.

We can make an additional direct comparison between approaches by fixing a false alarm probability and plotting the corresponding true alarm probability versus the optimal SNR of the

[1]We are limited to a minimal false alarm probability of $\sim 10^{-4}$ due to the limited number of testing samples used.

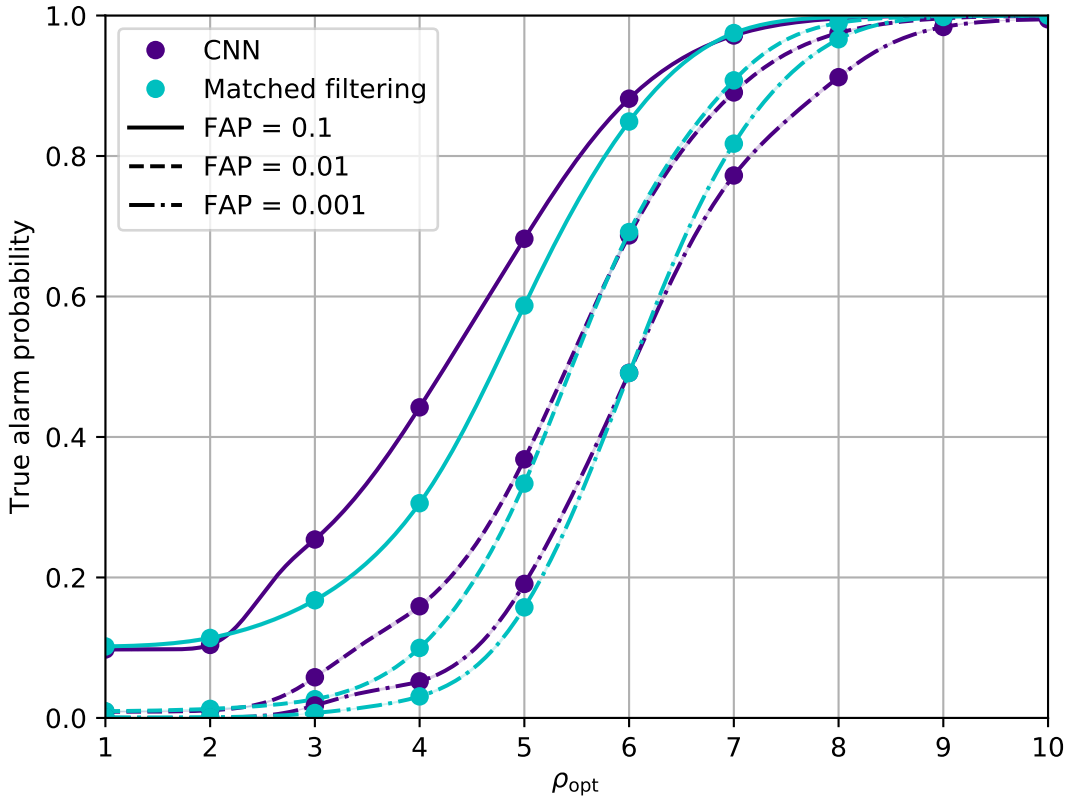


Figure 4.5: Efficiency curves comparing the performance of the CNN and matched-filtering approaches for false alarm probabilities 10^{-1} (solid), 10^{-2} (dashed), 10^{-3} (dot-dashed). The true alarm probability is plotted as a function of the optimal SNR for the CNN (purple) and the matched-filtering (cyan) analyses. Solid dots indicate at which SNR values analyses were performed and line thicknesses are indicative of the statistical uncertainties in the curves.

signals in each test dataset. We show these efficiency curves in Fig. 4.5 at false alarm probabilities $10^{-1}, 10^{-2}, 10^{-3}$ for both the CNN and matched-filtering approaches. We again see very good agreement between the approaches at all false alarm probabilities with the CNN sensitivity exceeding that of the matched-filter approach at low SNR and high false alarm probability. Conversely we see the matched-filter sensitivity marginally exceeds the CNN at high SNR and low false alarm probability. This latter discrepancy could be mitigated by increasing the number of training samples.

4.6 Conclusions

We have demonstrated that deep learning, when applied to gravitational-wave timeseries data, is able to closely reproduce the results of a matched-filtering analysis in Gaussian noise. We employ a deep convolutional neural network with rigorously tuned hyperparameters and produce an output that returns a ranking statistic equivalent to the inferred probability that data contains a signal. Matched-filtering analyses are often described as the optimal approach for signal detection in Gaussian noise. By building a neural network that is capable of reproducing this optimality we answer a fundamental question regarding the applicability of neural networks for gravitational-wave data analysis.

In practice, searches for transient signals in gravitational-wave data are strongly affected by non-Gaussian noise artefacts. To account for this, standard matched-filtering approaches are modified to include carefully chosen changes to the ranking statistic [19, 92] together with the excision of poor quality data [8, 36]. Our analysis represents a starting point from which a deep network can be trained on realistic non-Gaussian data. Since the claim of matched-filtering optimality is applicable only in the Gaussian noise case, there exists the potential for deep networks to exceed the sensitivity of existing matched-filtering approaches in real data.

In this work we have presented results for BBH mergers, however, this method could be applied to other merger types, such as binary neutron star and neutron star-black hole signals. This supervised learning approach can also be extended to other well modelled gravitational-wave targets such as the continuous emission from rapidly rotating non-axisymmetric neutron stars [35]. Moreover, unsupervised approaches have the potential to be powerful detection tools in searches for unmodelled burst-like gravitational-wave signals. Finally we mention the possibilities for parameter estimation [**?**] where in the simplest cases an output regression layer can return point estimates of parameter values. As was exemplified in the case of GW170817, rapid detection confidence coupled with robust and equally rapid parameter estimates is critical for gravitational-wave multi-messenger astronomy.

Chapter 5

Variational Inference for GW Parameter Estimation

5.1 Introduction

This is a cut and paste of our soon-to-be published paper (with appropriate modifications to format). Please see the following reference for our paper [45].

GW detection is now commonplace [7, 14] and as the sensitivity of the global network of GW detectors improves, we will observe $\mathcal{O}(100)$s of transient GW events per year [15]. The current methods used to estimate their source parameters employ optimally sensitive [109] but computationally costly Bayesian inference approaches [126] where typical analyses have taken between 6 hours and 5 days [2]. For BNS and NSBH systems prompt counterpart EM signatures are expected on timescales of 1 second – 1 minute and the current fastest method for alerting EM follow-up observers [111], can provide estimates in $\mathcal{O}(1)$ minute, on a limited range of key source parameters. Here we show that a CVAE [95, 122] pre-trained on BBH signals can return Bayesian posterior probability estimates. The training procedure need only be performed once for a given prior parameter space and the resulting trained machine can then generate samples describing the posterior distribution ~ 6 orders of magnitude faster than existing techniques.

With the overwhelmingly successful observation runs of O1 and O2 now complete, LIGO and Virgo have produced a large catalogue of GW data covering both BBH and BNS sig-

nals [119]. Over the next five years we expect the number of detections to increase to be upwards of ~ 180 BNS and ~ 400 BBH events per year [118, 119]. This large influx in the number of detections will put an increased amount of pressure on the current GW inference methods used for parameter estimation.

The problem of detecting GWs has largely been solved through the use of template based matched-filtering, a process recently replicated using machine learning techniques [46, 48, 50]. Once a GW has been identified through this process, Bayesian inference, known to be the optimal approach [109], is used to extract information about the source parameters of the detected GW signal.

In the standard Bayesian GW inference approach, we assume a signal and noise model and both may have unknown parameters that we are either interested in inferring or prefer to marginalise away. Each parameter is given a prior astrophysically motivated probability distribution and in the GW case, we typically assume a Gaussian additive noise model (in reality, the data is not truly Gaussian). Given a noisy GW waveform, we would like to find an optimal procedure for inferring some set of the unknown GW parameters. Such a procedure should be able to give us an accurate estimate of the parameters of our observed signal, whilst accounting for the uncertainty arising from the noise in the data.

According to Bayes' Theorem, a posterior probability distribution on a set of parameters, conditional on the measured data, can be represented as

$$p(x|y) \propto p(y|x)p(x), \tag{5.1}$$

where x are the parameters, y is the observed data, $p(x|y)$ is the posterior, $p(y|x)$ is the likelihood, and $p(x)$ is the prior on the parameters. The constant of proportionality, which we omit here, is $p(y)$, the probability of our data, known as the Bayesian evidence or the marginal likelihood. We typically ignore $p(y)$ since it is a constant and for parameter estimation purposes we are only interested in the shape of the posterior.

Due to the size of the parameter space typically encountered in GW parameter estimation and the volume of data analysed, we must stochastically sample the parameter space in order to

estimate the posterior. Sampling is done using a variety of techniques including Nested Sampling [112, 114, 125] and Markov chain Monte Carlo methods [43, 128]. The primary software tools used by the LIGO parameter estimation analysis are `LALInference` and `Bilby` [22, 126], which offer multiple sampling methods.

Machine learning has featured prominently in many areas of GW research over the last few years. These techniques have shown to be particularly promising in signal detection [46, 48, 50], glitch classification [136] and earthquake prediction [37]. We also highlight a recent development in GW parameter estimation (published in parallel and independent to this work) where 1- and 2-dimensional marginalised Bayesian posteriors are produced rapidly using neural networks [34]. This is done without the need to compute the likelihood or posterior during training which is also a characteristic of the approach described in this letter.

Recently, a type of neural network known as CVAE was shown to perform exceptionally well when applied towards computational imaging inference [113,122], text to image inference [134], high-resolution synthetic image generation [89] and the fitting of incomplete heterogeneous data [87]. It is this type of machine learning network that we apply in the GW case to accurately approximate the Bayesian posterior $p(x|y)$.

The construction of a CVAE begins with the definition of a quantity to be minimised (referred to as a cost function). We can relate that aim to that of approximating the posterior distribution by minimising the cross entropy, defined as

$$H(p,r) = -\int dx\, p(x|y) \log r_\theta(x|y) \tag{5.2}$$

between the true posterior $p(x|y)$ and $r_\theta(x|y)$, the parametric distribution that we will use neural networks to construct and which we aim to be equal to the true posterior. In this case θ represents a set of trainable neural network parameters. Starting from this point it is possible to derive a computable form for the cross-entropy that is reliant on a set of unknown functions that can be modelled by variational encoder and decoder neural networks. The details of the derivation are described in the methods section and in [122]. The final form of the cross-entropy loss function

is given by the bound

$$H \lesssim -\frac{1}{N}\sum_{n=1}^{N}\Big[\log r_{\theta_2}(x_n|z_n,y_n) - \mathrm{KL}\left[q_\phi(z|x_n,y_n)||r_{\theta_1}(z|y_n)\right]\Big], \tag{5.3}$$

and requires three fully-connected networks; two encoder networks (labelled E_1, E_2 in Fig. 5.1) representing the functions $r_{\theta_1}(z|y)$ and $q_\phi(z|x,y)$ respectively, and one decoder network (D) representing the function $r_{\theta_2}(x|z,y)$. The function $\mathrm{KL}(\cdot||\cdot)$ denotes the KL divergence and the variable z represents locations within the *latent space*. This latter object is typically a lower dimensional space within which the encoder networks attempt to represent their input data. In practice, during the training procedure the various integrations that are part of the derivation are approximated by a sum over a batch of N training data samples (indexed by n above) at each stage of training. Training is performed via a series of steps detailed in the methods section.

5.2 Results

We present results on 256 single detector GW test BBH waveforms in simulated advanced detector noise [1] and compare between variants of the existing Bayesian approaches and the CVAE. Posteriors produced by the `Bilby` inference library [22] are used as a benchmark in order to assess the efficiency and quality of our machine learning approach with the existing method for posterior sampling.

For the benchmark analysis we assume that 5 parameters are unknown: the component masses m_1, m_2, the luminosity distance d_L, the time of coalescence t_0, and the phase at coalescence ϕ_0. For each parameter we use a uniform prior with ranges and fixed values defined in Table 5.2. We use a sampling frequency of 256 Hz, a timeseries duration of 1 second, and the waveform model used is `IMRPhenomPv2` [71] with a minimum cutoff frequency of 20Hz. For each input test waveform we run the benchmark analysis using multiple sampling algorithms available within `Bilby`. For each run and sampler we extract 3000 samples from the posterior on the 5 physical parameters.

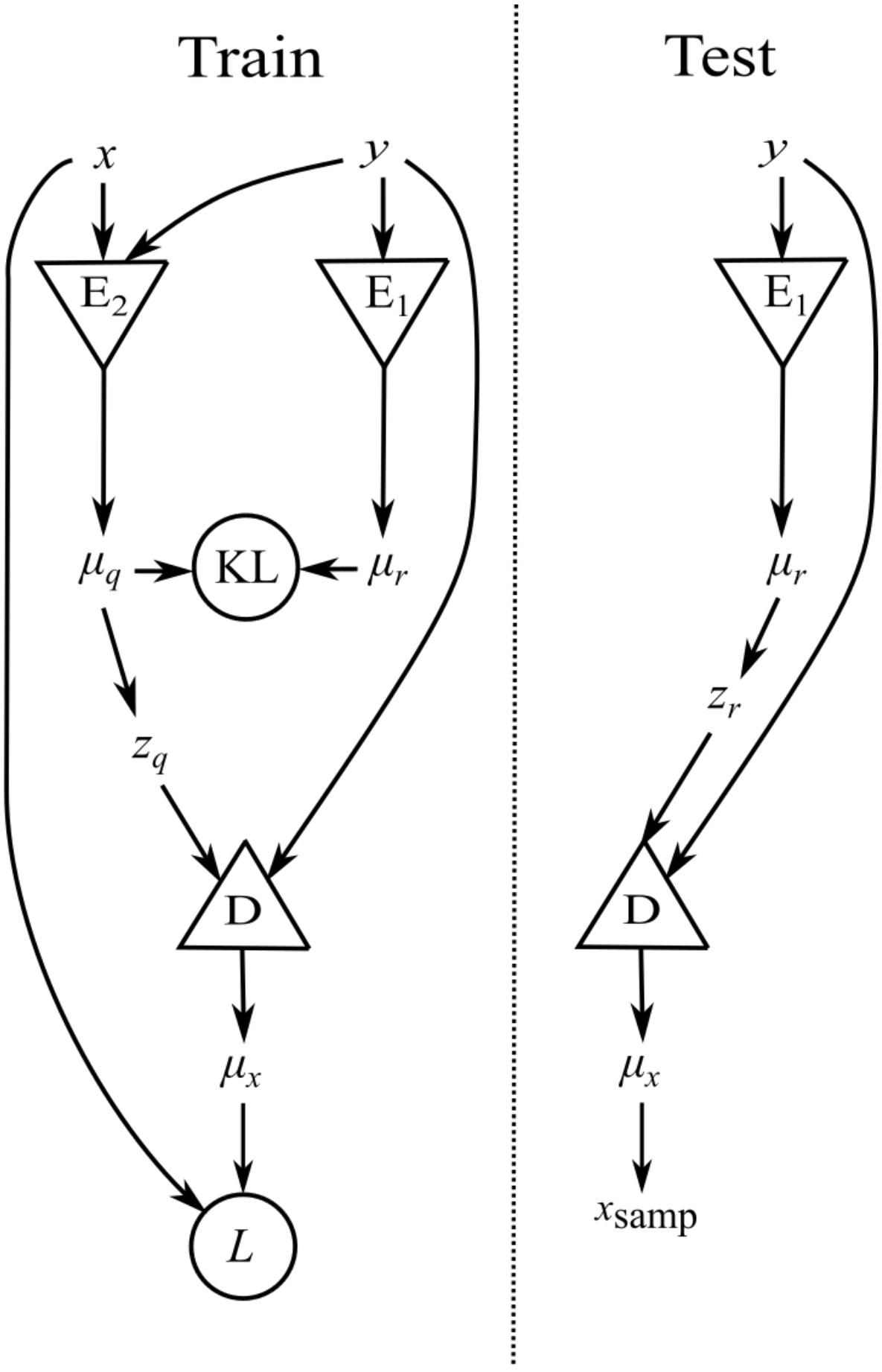


Figure 5.1: The configuration of the CVAE neural network. During training (left-hand side), a training set of noisy GW signals (y) and their corresponding true parameters (x) are given as input to encoder network E_2, while only y is given to E_1. The K-L divergence (Eq. 5.7) is com-

The CVAE training process used as input 10^6 waveforms corresponding to parameters drawn from the same priors as assumed for the benchmark analysis. The waveforms are also of identical duration, sampling frequency, and waveform model as used in the benchmark analysis. When each waveform is placed within a training batch it is given a unique detector noise realisation after which the data is whitened using the same advanced detector PSD [1] from which the simulated noise is generated[1]. The CVAE posterior results are produced by passing our 256 whitened noisy testing set of GW waveforms as input into the testing path of the pre-trained CVAE 5.1. For each input waveform we sample until we have generated 3000 posterior samples on 4 physical parameters ($x = (m_1, m_2, d_{\rm L}, t_0)$). We choose to output a subset of the full 5-dimensional space to demonstrate that parameters (such as ϕ_0 in this case) can (if desired) be marginalized out within the CVAE procedure itself, rather than after training.

We can immediately illustrate the accuracy of our machine learning predictions by directly plotting 2 and 1-dimensional marginalised posteriors generated using the output samples from our CVAE and `Bilby` approaches superimposed on each other. We show this for one example test dataset in Fig. 5.2 where the strong agreement between both `Bilby` (blue) and the CVAE (red) is clear.

A standard test used within the GW parameter estimation community is the production of so-called p-p plots which we show for our analysis in Fig. 5.4. The plot is constructed by computing a p-value for each output test posterior on a particular parameter evaluated at the true simulation parameter value (the fraction of posterior samples $>$ the simulation value). We then plot the cumulative distribution of these values [126]. Curves consistent with the black dashed diagonal line indicate that the 1-dimensional Bayesian probability distributions are consistent with the frequentist interpretation - that the truth will lie within an interval containing $X\%$ of the posterior probability with a frequency of $X\%$ of the time. It is clear to see that our new approach shows deviations from the diagonal that are entirely consistent with those observed in all benchmark samplers. In Fig. 5.5 (see Sec. 5.4) we also show distributions of the KL divergence statistic between all samplers on the joint posterior distributions. It is also clear that the KL divergences

[1] Although we whiten the data as input to our network the whitening is simply to scale the input to a level more suitable to neural networks and need not be performed with the true PSD.

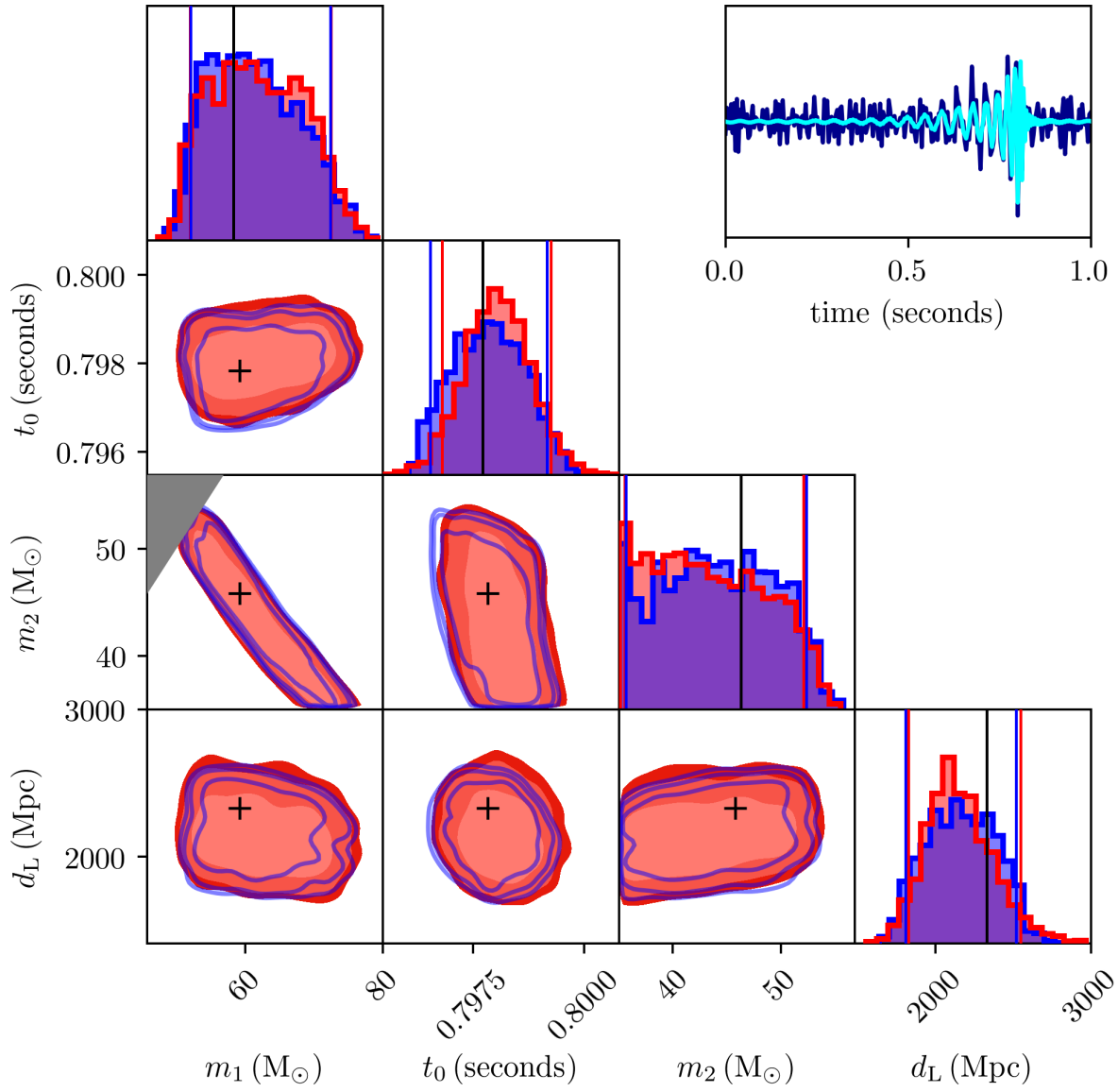


Figure 5.2: Corner plot showing 2 and 1-dimensional marginalised posterior distributions for one example test dataset. Filled (red) contours represent the posteriors obtained from the CVAE approach and solid (blue) contours are the posteriors output from our baseline analysis (`Bilby` using the dynesty sampler). In each case, the contour boundaries enclose 68, 90 and 95% probability. One dimensional histograms of the posterior distribution for each parameter from both methods are plotted along the diagonal. Blue and red vertical lines represent the 5—95% symmetric confidence bounds for `Bilby` and variational inference respectively. Black crosses and vertical black lines denote the true parameter values of the simulated signal. The original whitened noisy timeseries y and the noise-free signal are plotted in blue and cyan respectively in the upper right hand panel. The test signal was simulated with optimal signal-to-noise ratio of 13.9.

Table 5.1: Durations required to produce samples from each of the different posterior sampling approaches.

sampler	run time (seconds)			ratio $\frac{\tau_{\text{VItamin}}}{\tau_X}$
	min	max	median	
Dynesty[2]	602	1538	774[3]	2.6×10^{-6}
Emcee	2005	11927	4351	4.6×10^{-7}
Ptemcee	3354	12771	4982	4.0×10^{-7}
Cpnest	1431	5405	2287	8.8×10^{-7}
`VItamin`[4]		$\mathbf{2 \times 10^{-3}}$		1

between `VItamin` and any other sampler are consistent with the distributions between any 2 existing benchmark samplers.

The dominating computational cost of running `VItamin` lies in the

We stress that once trained, there is no need to retrain the network unless the user wishes to use different priors $p(x)$ or assume different noise characteristics. The speed at which posterior samples are generated for all samplers used, including `VItamin`, is shown in Table 5.1. Run-time for the benchmark samplers is defined as the time to complete their analyses when configured using their default parameters [22]. For `VItamin` this time is defined as the total time to produce 3000 samples. For our test case of BBH signals `VItamin` produces samples from the posterior at a rate which is $\sim$ 6—7 orders of magnitude faster than our benchmark analysis using current inference techniques, representing a dramatic speed-up in performance.

5.3 Conclusions

There are currently a variety of challenges within the field of GW parameter estimation including, and not limited to, accounting for detector artefacts (non-Gaussian transient "glitches") in the data, time-dependent variations (non-stationarity) of the detector noise power spectrum density, systematic uncertainties in the waveform models themselves and decreasing the latency with which parameter estimates are produced [126]. In this letter we have demonstrated that we are able to reproduce, to a high degree of accuracy, Bayesian posterior probability distributions generated through machine learning. This is accomplished using CVAEs trained on simulated GW signals and does not require the input of precomputed posterior estimates. We have demon-

strated that our neural network model which, when trained, can reproduce complete and accurate posterior estimates in $\mathcal{O}(1)$ millisecond, can achieve the same quality of results as the trusted benchmark analyses used within the LIGO-Virgo Collaboration.

The significance of our results is most evident in the orders of magnitude increase in speed over existing approaches. This will help the LIGO-Virgo collaboration alert EM follow-up partners with minimum latency, enabling tightly coupled, closed-loop control of sensing resources, for maximum information gain. Improved low-latency alerts will be especially pertinent for signals from BNS mergers (e.g. GW170817 [14]) and NSBH signals where parameter estimation speed will no longer be limiting factor[5] in observing the prompt EM emission expected on shorter time scales than is achievable with existing LVC analysis tools such as Bayestar [111].

The predicted number of future detections of BNS mergers (~ 180 [15]) will severely strain the GW community's current computational resources using existing Bayesian methods. Future iterations of our approach will provide full-parameter estimation on CBC signals in < 1 second on a single graphics processing unit (GPU). Our trained network is also modular, and can be shared and used easily by any user to produce results. The specific analysis described in the letter assumes a uniform prior on the signal parameters. However, this is a choice and the network can be trained with any prior the user demands, or users can cheaply resample accordingly from the output of the network trained on the uniform prior. We also note that our method will be invaluable for population studies since populations may now be generated and analysed in a full-Bayesian manner on a vastly reduced time scale.

For BBH signals, GW data is usually sampled at 1—4 kHz dependent upon the mass of binary. We have chosen to use the noticeably low sampling rate of 256Hz and a single detector configuration largely in order to decrease the computational time required to develop our approach. We do not anticipate any problems in extending our analysis to higher sampling frequencies other than an increase in training time and a larger burden on the GPU memory. Our lower sampling rate naturally limited the chosen BBH mass parameter space to high mass sig-

[5]The complete low-latency pipeline includes a number of steps. The process of GW data acquisition is followed by the transfer of data. There is then the corresponding analysis and the subsequent communication of results to the EM astronomy community after which there are physical aspects such as slewing observing instruments to the correct pointing.

nals. We similarly do not anticipate that extending the parameter space to lower masses will lead to problems but do expect that a larger number of training samples may be required. Future work will incorporate a multi-detector configuration at which point parameter estimation will be extended to sky localisation.

In reality, GW detectors are affected by non-Gaussian noise artefacts. To account for this noise within our scheme, we would train our network using samples of real detector noise (preferably recent examples to best reflect the state of the detectors). Our method has the added advantage of not being dependent on the choice of PSD used for whitening, unlike current Bayesian methods. The whitening procedure applied to the training data for the CVAE is simply to rescale the input to a scale more suitable to neural networks whereas for existing methods assumptions on the PSD directly affect the posterior results. Our work can naturally be expanded to include the full range of CBC signal types but also to any and all other parameterised GW signals and to analyses of GW data beyond that of ground based experiments. Given the abundant benefits of this method, we hope that a variant of this of approach will form the basis for all GW parameter estimation over the next several decades to come.

5.4 Methods

Conditional variational autoencoders are a form of variational autoencoder which are conditioned on an observation, where in our case the observation is a 1-dimensional GW time series signal y. The autoencoders from which variational autoencoders are derived are typically used for problems involving image reconstruction and/or dimensionality reduction. They perform a regression task whereby the autoencoder attempts to predict its own given input (model the identity function) through a "bottleneck layer", a limited and therefore distilled representation of the input parameter space. An autoencoder is composed of two neural networks, an encoder and a decoder [47]. The encoder network takes as input a vector, where the number of dimensions is a fixed number predefined by the user. The encoder converts the input vector into a (typically) lower dimensional space, referred to as the *latent space*. A representation of the data in the latent space is passed to the decoder network which generates a reconstruction of the original

input data to the encoder network. Through training, the two sub-networks learn how to efficiently represent a dataset within a lower dimensional latent space which will take on the most important properties of the input training data. In this way, the data can be compressed with little loss of fidelity. Additionally, the decoder simultaneously learns to decode the latent space representation and reconstruct that data back to its original form (the input data).

The primary difference between a variational autoencoder [95] and an autoencoder concerns the method by which locations within the latent space are produced. In our variant of the variational autoencoder, the output of the encoder is interpreted as a set of parameters governing statistical distributions (in our case the means and variances of multivariate Gaussians). In proceeding to the decoder network, samples from the latent space (z) are randomly drawn from these distributions and fed into the decoder, therefore adding an element of variation into the process. A particular input can then have a range of possible outputs. In both the decoder and the encoder networks we use fully-connected layers (although this is not a constraint and any trainable network architecture may be used).

5.4.1 Cost function derivation

We will now derive the cost function and the corresponding network structure and we begin with the statement defining the aim of the analysis. We wish to obtain a function that reproduces the posterior distribution (the probability of our physical parameters x given some measured data y). The cross entropy between 2 distributions is defined in Eq. 5.2 where we have made the distributions explicitly conditional on y (our measurement). In this case $p(x|y)$ is the target distribution (the true posterior) and $r_\theta(x|y)$ is the parametric distribution that we will use neural networks to construct. The variable θ represents the trainable neural network parameters.

The cross-entropy is minimised when $p(x|y) = r_\theta(x|y)$ and so by minimising

$$H = -\mathrm{E}_{p(y)}\left[\int dx\, p(x|y) \log r_\theta(x|y)\right], \tag{5.4}$$

where $\mathrm{E}_{p(y)}[\cdot]$ indicates the expectation value over the distribution of measurements y, we therefore make the parametric distribution as similar as possible to the target for all possible mea-

surements y.

Converting the expectation value into an integral over y weighted by $p(y)$ and applying Bayes' theorem we obtain

$$H = -\int dx\, p(x) \int dy\, p(y|x) \log r_\theta(x|y) \tag{5.5}$$

where $p(x)$ is the prior distribution on the physical parameters x.

The CVAE network outlined in Fig. 5.1 makes use of a conditional latent variable model and our parametric model is constructed from the product of 2 separate distributions marginalised over the latent space

$$r_\theta(x|y) = \int dz\, r_{\theta_1}(z|y) r_{\theta_2}(x|z,y). \tag{5.6}$$

We have used θ_1 and θ_2 to indicate that the 2 separate networks modelling these distributions will be trained on these parameter sets respectively. Both new conditional distributions are modelled as n_z dimensional multivariate uncorrelated Gaussian distributions (governed by their means and variances). However, this still allows $r_\theta(x|y)$ to take a general form (although it does limit it to be unimodal).

One could be forgiven in thinking that by setting up networks that simply aim to minimise H over the θ_1 and θ_2 would be enough to solve this problem. However, as shown in [113] this is an intractable problem and a network cannot be trained directly to do this. Instead we define a recognition function $q_\phi(z|x,y)$ that will be used to derive an ELBO. Here we use ϕ to represent the trainable parameters of an encoder network (E_2).

Let us first define the KL divergence between 2 of our distributions as

$$\begin{aligned} &\mathrm{KL}\left[q_\phi(z|x,y)||r_{\theta_2}(z|x,y)\right] = \\ &\int dz\, q_\phi(z|x,y) \log\left(\frac{q_\phi(z|x,y)}{r_{\theta_2}(z|x,y)}\right). \end{aligned} \tag{5.7}$$

It can be shown, after some manipulation, that

$$\log r_\theta(x|y) = L + \mathrm{KL}\left[q_\phi(z|x,y)||r_{\theta,}(z|x,y)\right], \tag{5.8}$$

where the ELBO L is given by

$$L = \int dz\, q_\phi(z|x,y) \log\left(\frac{r_{\theta_2}(x|z,y) r_{\theta_1}(z|y)}{q_\phi(z|x,y)}\right) \tag{5.9}$$

and is so-named since KL cannot be negative and has a minimum of zero. Therefore, if we were to find a $q_\phi(z|x,y)$ function (optimised on ϕ) that minimised the KL-divergence then we can state that

$$\log r_\theta(x|y) \geq L. \tag{5.10}$$

After some further manipulation of Eq. 5.9 we find that

$$\begin{aligned}\log r_\theta(x|y) \geq & \mathrm{E}_{q_\phi(z|x,y)}\left[\log r_{\theta_2}(x|z,y)\right] \\ & - \mathrm{KL}\left[q_\phi(z|x,y)||r_{\theta_1}(z|y)\right].\end{aligned} \tag{5.11}$$

We can now substitute this inequality into Eq. 5.5 (our cost function) to obtain

$$\begin{aligned}H \leq -\int dx\, p(x) \int dy\, p(y|x) \Big[& \mathrm{E}_{q_\phi(z|x,y)}\left[\log r_{\theta_2}(x|z,y)\right] \\ & - \mathrm{KL}\left[q_\phi(z|x,y)||r_{\theta_1}(z|y)\right]\Big],\end{aligned} \tag{5.12}$$

which can in practice be approximated as a stochastic integral over draws of x from the prior, y from the likelihood function $p(y|x)$, and from the recognition function, giving us Eq. 5.3, the actual function evaluated within the training procedure.

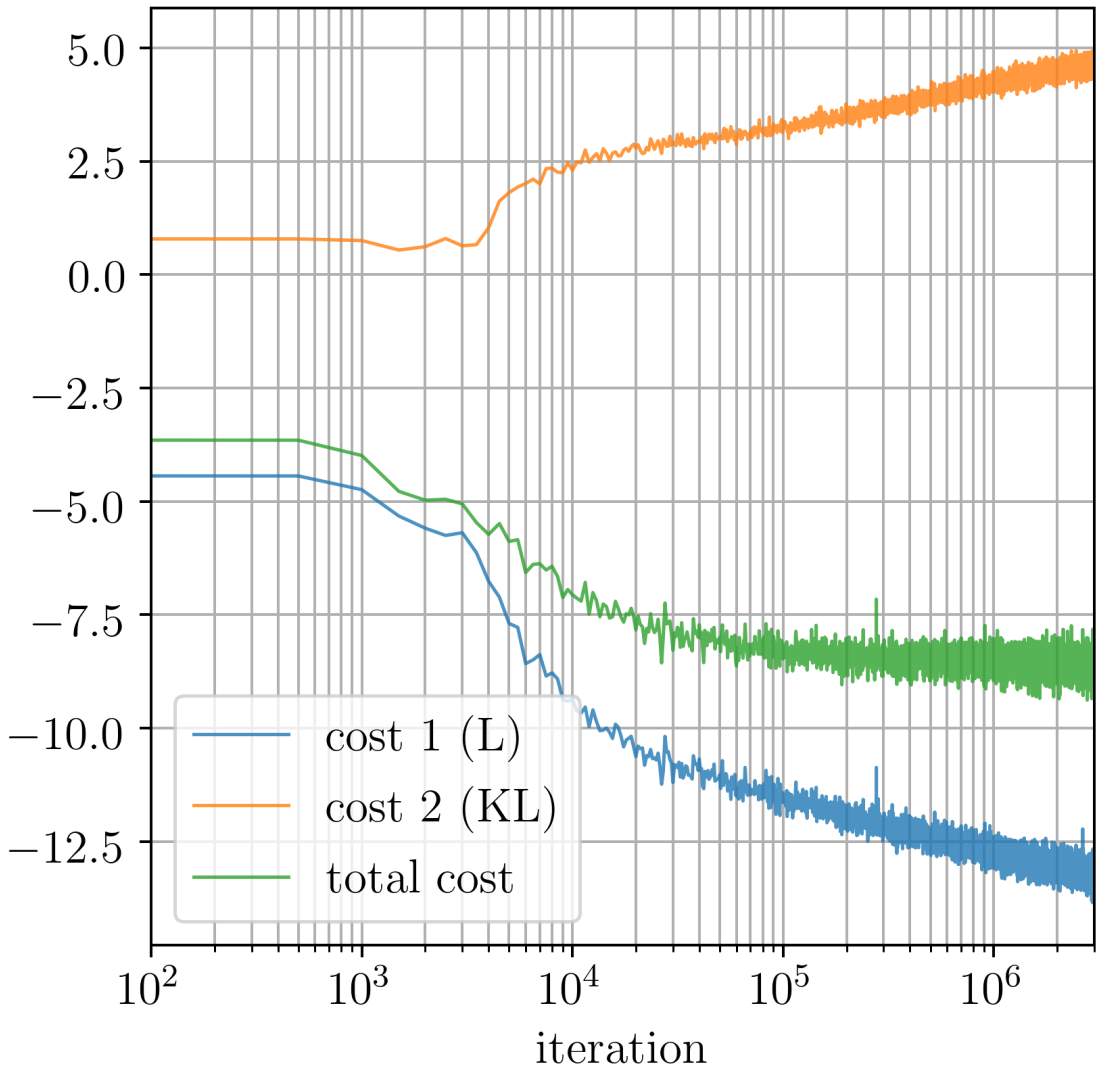


Figure 5.3: The cost as a function of training iteration. We show the ELBO cost function component (blue), the KL divergence component (orange) and total cost (green). The total cost is simply a summation of the 2 components and one training iteration is defined as training over one batch of signals.

Table 5.2: The uniform prior boundaries and fixed parameter values used on the BBH signal parameters for the benchmark and the CVAE analyses.

Parameter name	symbol	min	max	units
mass 1	m_1	35	80	solar masses
mass 2	m_2[6]	35	80	solar masses
luminosity distance	d_{L}	1	3	Gpc
time of coalescence	t_0	0.65	0.85	seconds
phase at coalescence	ϕ_0	0	2π	radians
right ascension	α	1.375		radians
declination	δ	-1.2108		radians
inclination	ι	0		radians
polarisation	ψ	0		radians
spins	-	0		-
epoch	-	1126259642		GPS time
detector	-	LIGO Hanford		-

5.4.2 The training procedure

Having set up a cost function composed of 3 probability functions that have well defined inputs and outputs where the mapping of those inputs to outputs is governed by the parameter sets θ_1, θ_2, ϕ. These parameters are the weights and biases of 3 neural networks acting as (variational) encoder, decoder, and encoder respectively. To train such a network one must connect the inputs and outputs appropriately to compute the cost function H and back-propagate cost function derivatives to update the network parameters. The network structure shown schematically in Fig. 5.1 shows how for a batch of N sets of x and corresponding y values, the cost function is computed during each iteration of training.

Training is performed via a series of steps illustrated in Fig. 5.1.

- The encoder E_1 is given a set of training GW signals (y) and encodes y into a set of variables μ_q defining a distribution in the latent space. In this case $\mu_q = (\mu_{q0}, \sigma_q^2)$ describes the first 2 central moments (mean and variance) for each dimension of a uncorrolated (diagonal covariance) multivariate Gaussian distribution.

- The encoder E_2 takes a combination of both the data y and the true parameters x defining the GW signal and encodes this into parameters defining another uncorrelated multivariate Gaussian distribution in the same latent space. These parameters we denote by $\mu_r = (\mu_{r0}, \sigma_r^2)$ again representing the means and variances.

- We then sample from the distribution described by μ_q giving us samples z_q within the latent space.

- These samples, along with their corresponding y data, then go to the decoder D which outputs $\mu_x = (\mu_{x0}, \sigma_q^2)$, a set of parameters (much like μ_q, μ_r) that define the moments of an uncorrelated multivariate Gaussian distribution in the physical x space.

- The first term of the loss function (Eq. 5.3) is then computed by evaluating the probability density defined by μ_x at the true x training values. The component of the loss allows the network to learn how to predict accurate values of x but to also learn the intrinsic variation due to the noise properties of the data y. It is important to highlight that the GW parameter predictions from the decoder D do describe a multivariate Gaussian, but as is shown in our results (see Fig. 5.2), this does *not* imply that our final output posterior estimates will also be multivariate Gaussians.

- Finally the loss component described by the KL divergence between the distributions described by μ^q and μ^r is computed using

$$\begin{aligned} &\mathrm{KL}\left[q_\phi(z|x_n, y_n)||r_{\theta_1}(z|y_n)\right] = \\ &\frac{1}{2}\sum_{j=1}^{n_z}\left[\frac{\sigma_{q,j}^2}{\sigma_{r,j}^2} + \frac{(\mu_{r0,j} - \mu_{q0,j})^2}{\sigma_{r,j}^2} + \log\left(\frac{\sigma_{r,j}^2}{\sigma_{q,j}^2}\right)\right] - \frac{n_z}{2}. \end{aligned} \tag{5.13}$$

  Here we highlight that we do not desire that the network tries to make these 2 distributions equal to each other. Rather, we want the ensemble network to minimise the total cost (of which this is a component).

As is standard practice in machine learning applications, the cost is computed over a batch of training samples and repeated for a pre-defined number of iterations. Completion of training is determined by comparing output posteriors on test samples with those of `Bilby` iteratively during training. This comparison is done using standard figures of merit such as the KL divergence and the P-P plot. We also assess training completion based on whether the cost function and its component parts (Fig. 5.3) have converged. We use a single Nvidia Tesla V100 GPUs

with 16/32 Gb of RAM although consumer grade “gaming" GPU cards are equally fast for this application.

5.4.3 Network and Training parameters

For our purposes, we found that $\sim 3 \times 10^6$ training iterations, a batch size of 512 training samples and a learning rate of 10^{-4} was sufficient. We used a total of 10^6 training samples in order to adequately cover the BBH parameter space. We additionally ensure that an (effectively) infinite number of noise realizations are employed by making sure that every time a training sample is used it is given a unique noise realisation despite only having a finite number of waveforms. Each neural network (E_1, E_2, D) is composed of 3 fully connected layers and has 2048 neurons in each layer with ReLU [86] activation functions between layers. We use a latent space dimension of 8 and we consider training complete when both components to the loss function have converged to approximately constant values or when comparisons with benchmark test posteriors indicate no significant changes in the output posterior.

5.4.4 The testing procedure

After training has completed and we wish to use the network for inference we follow the procedure described in the right hand panel of Fig. 5.1. Given a new y data sample (not taken from the training set) we simply input this into the encoder E_1 from which we obtain a single value of μ_r describing a distribution (conditional on the data y) in the latent space. We then repeat the following steps:

- We randomly draw a latent space sample z_r from the latent space distribution defined by μ_r.
- Our z_r sample and the corresponding original y data are fed as input to our pre-trained decoder network (D). The decoder network returns a set of moments μ_x which describe a multivariate Gaussian distribution in the physical parameter space.
- We then draw a random x realisation from that distribution.

A comprehensive representation in the form of samples drawn from the entire joint posterior distribution can then be obtained by simply repeating this procedure with the same input data (see Eq. 5.6).

5.4.5 Additional tests

A standard test used within the GW parameter estimation community is the production of so-called p-p plots which we show for our analysis in Fig. 5.4. The plot is constructed by computing a p-value for each output test posterior on a particular parameter evaluated at the true simulation parameter value (the fraction of posterior samples $>$ the simulation value). We then plot the cumulative distribution of these values [126]. Curves consistent with the black dashed diagonal line indicate that the 1-dimensional Bayesian probability distributions are consistent with the frequentist interpretation - that the truth will lie within an interval containing $X\%$ of the posterior probability with a frequency of $X\%$ of the time. It is clear to see that our new approach shows deviations from the diagonal that are entirely consistent with those observed in all benchmark samplers.

The KL divergence between 2 distributions is a measure of their similarity and we use this to compare the output posterior estimates between samplers for the same input test data. To do this we run each independent sampler (including the CVAE) on the same test data to produce samples from the corresponding posterior. We then compute the KL-divergence between output distributions from each sampler with itself and each sampler with all other samplers. For distributions that are identical the KL-divergence is equal to zero but since we are representing our posterior distributions using finite numbers of samples, identical distributions should result in KL-divergence values < 1. In Fig. 5.5 we show the distributions of these KL-divergences for the 256 test GW samples where we see that the CVAE approach when compared to the benchmark samplers have distributions consistent with those produced when comparing between 2 different benchmark samplers.

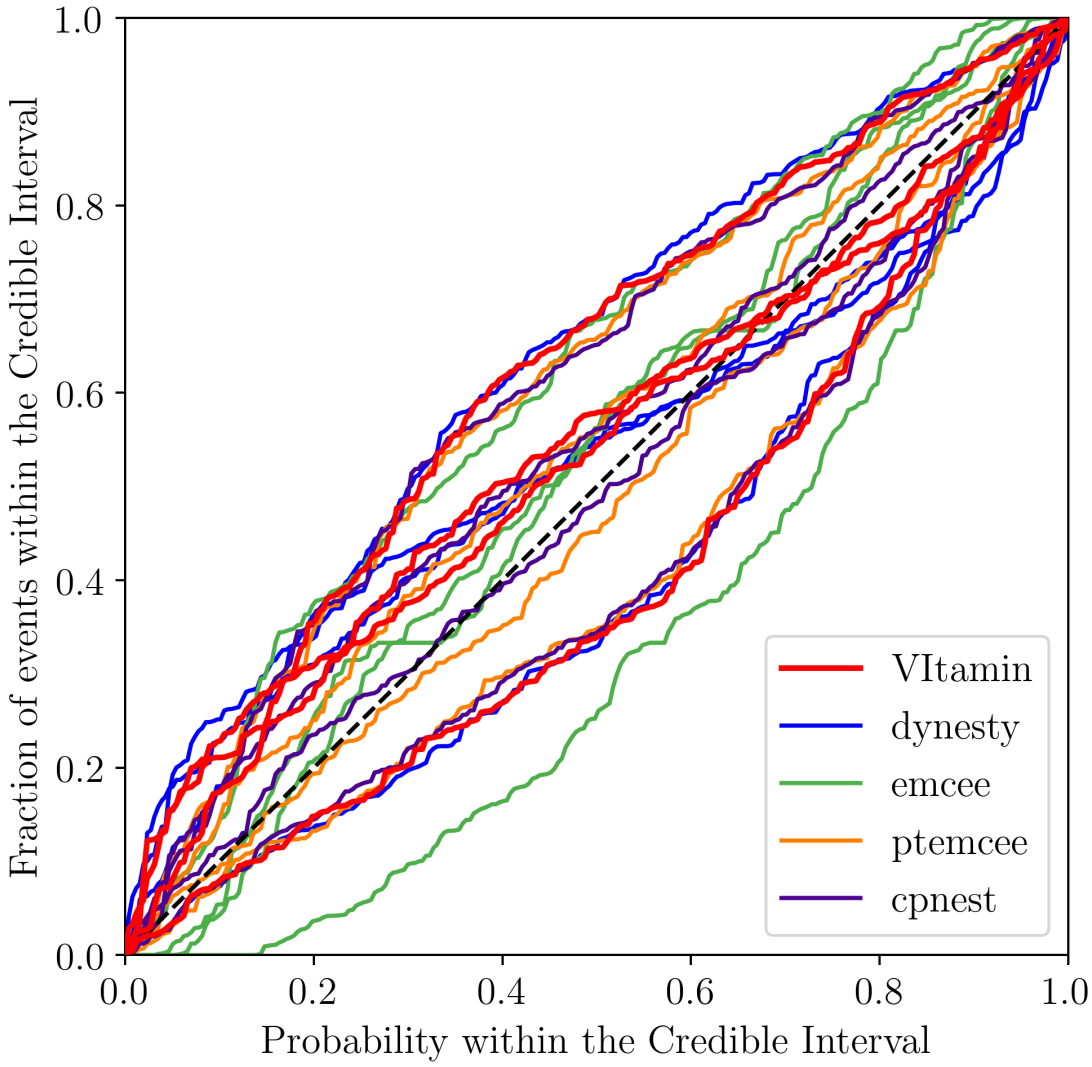


Figure 5.4: One-dimensional p-p plots for each parameter and each benchmark sampler and `VItamin`. The curves were constructed using the 256 test datasets. The dashed black diagonal line indicates the ideal result.

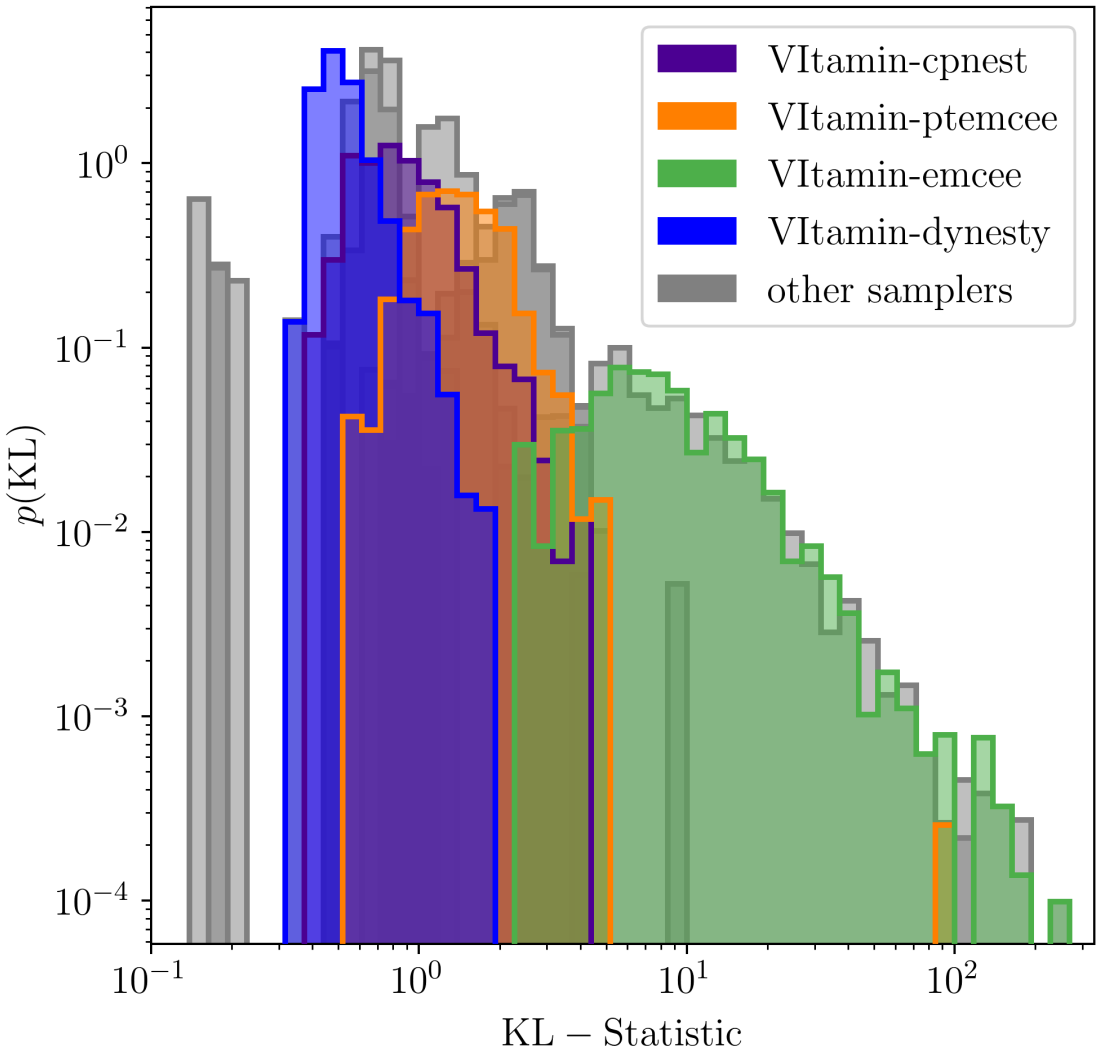


Figure 5.5: Distributions of KL-divergence values between posteriors produced by different samplers. The KL divergence is computed between all samplers with every other sampler over all 256 GW test cases. The distributions of the resulting KL divergence values are then plotted, with each color representing a different sampler combination including `VItamin` as one of the sampler pairs. The grey distributions represent the results from all benchmark sampler pairs for comparison. Both the x and y axes are scaled logarithmically for readability.

5.5 Additional analysis

In the following section we will discuss additional analysis of the results from `VItamin` including: analysis of the JS divergence using individual source parameter values, training set SNR distribution and it's effect on the performance of `VItamin`, the structure and behavior of the CVAE latent space and it's effect on the predicted posterior distributions, ...

5.5.1 JS divergence for individual source parameters

We provide additional figures of merit in Fig. 5.7, 5.6, 5.8, 5.9 where we plot the JS divergence values for each GW source parameter across all Bayesian sampler approaches. Each sampler method vs. another sampler method are denoted as different colors. The lower and upper end of boxes represent the 25th and 75th credibility regions respectively. The lower and upper end of the whiskers represent the 5th and 95th percent credibility regions. The dashed red line represents optimal JS divergence value according to sampler experts (~ 0.002).

In Fig. 5.6, it can be seen that `Dynesty` vs. `VItamin` JS values closely matches results from `Dynesty` vs. `Ptemcee` for nearly all parameters, with the exception of t_0, θ_{jn}, ϕ_{jl}, α and δ. `VItamin` predictions have a slightly higher mismatch across all source parameters except for the spin parameters. `Dynesty` vs. CPNest seems to generally have similar JS values to `Dynesty` vs. `Ptemcee` with the exception of having broader credibility intervals on t_0, θ_{jn}, ϕ_{jl}, α and δ. `Dynesty` vs. `Emcee` generally has higher JS values than all other methods, which is expected given the difficulty of `Emcee` convergence.

In Fig. 5.7, `CPNest` is highlighted against all other sampler approaches (including `VItamin`). It can be seen that both `CPnest` vs. `Dynesty` and `CPnest` vs. `Ptemcee` are in strong agreement with each other (although with broad credibility regions on t_0, θ_{jn}, α, δ). `CPNest` vs. `VItamin` is also in strong agreement with `CPnest` vs. `Dynesty` and `CPnest` vs. `Ptemcee`, but it does show some disagreement on m_1, m_2 and ψ. `CPNest` vs. `Emcee` is generally in disagreement with all other approaches.

Fig. 5.8 highlights the `Emcee` sampler vs. all other approaches including `VItamin`. In Fig. 5.8 it can be seen that `Emcee` has equal overlap against all other sampler approaches. Given the

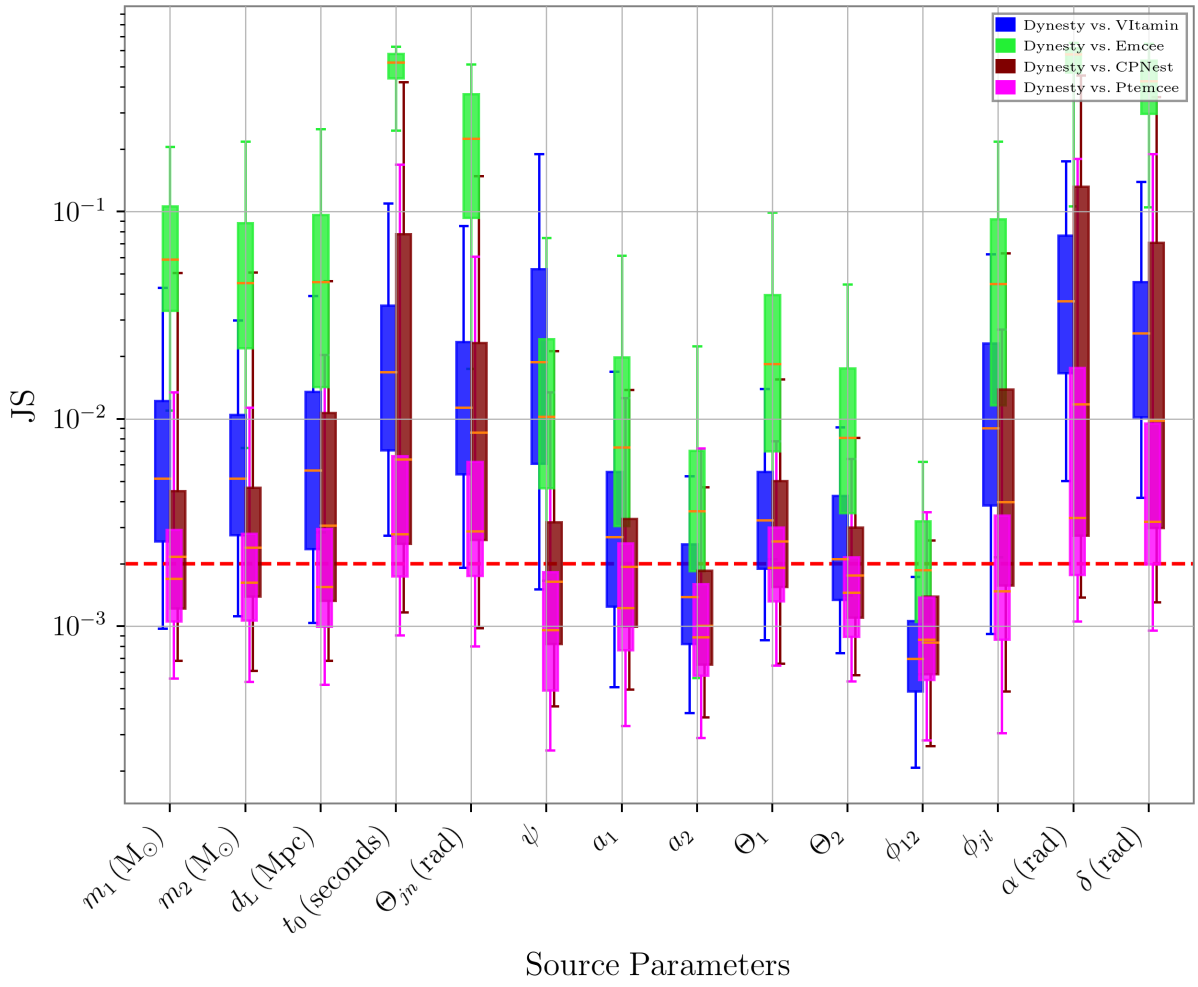


Figure 5.6: Shown are JS divergence values for all test samples as a function of test sample source parameter for `Dynesty` against every other sampling approach. Each JS value is a 1-dimensional JS statistic, rather than the full 14-dimensional JS statistics shown earlier. Each sampler method vs. another sampler method are denoted as different colors. The lower and upper end of boxes represent the 25th and 75th credibility regions respectively. The lower and upper end of the whiskers represent the 5th and 95th percent credibility regions. The dashed red line represents optimal JS divergence value according to sampler experts (~ 0.002). The orange line is representative of the median JS value. We see here that `VItamin` performs to within the same degree of accuracy as other Bayesian samplers when looking at predictions on an individual source parameter basis.

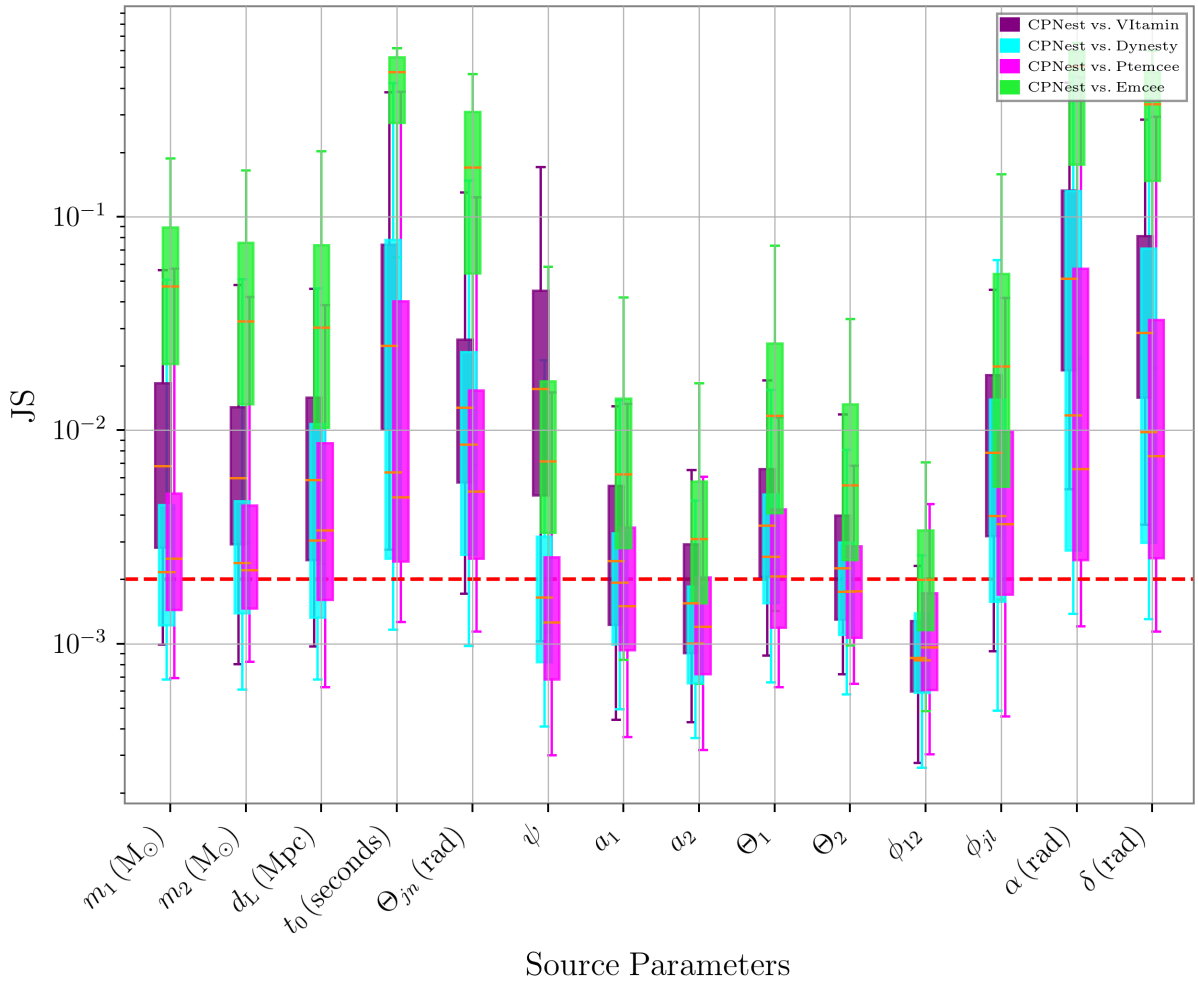


Figure 5.7: Shown are JS divergence values for all test samples as a function of test sample source parameter for `CPNest` against every other sampling approach. Each JS value is a 1-dimensional JS statistic, rather than the full 14-dimensional JS statistics shown earlier. Each sampler method vs. another sampler method are denoted as different colors. The lower and upper end of boxes represent the 25th and 75th credibility regions respectively. The lower and upper end of the whiskers represent the 5th and 95th percent credibility regions. The dashed red line represents optimal JS divergence value according to sampler experts (~ 0.002). The orange line is representative of the median JS value. We see here that `VItamin` performs to within the same degree of accuracy as other Bayesian samplers when looking at predictions on an individual source parameter basis.

high JS values of Emcee vs. all other approaches, this indicates that Emcee has found difficulty converging on many of the test sample cases. This is expected given that it is well known that Emcee generally difficult to tune for proper convergence.

Fig. 5.9 highlights Ptemcee vs. all other sampler approaches. In Fig. 5.9 we see that Ptemcee vs. Dynesty and Ptemcee vs. CPNest closely match each other with the exception of slight disagreement on t_0, θ_{jn} and ϕ_{jl} (with broader credibility regions on Ptemcee vs. CPNest). Ptemcee vs. VItamin in dark green also closely matches the Ptemcee vs. Dynesty and Ptemcee vs. CPNest results with slightl higher JS values accross all source parameter values other than the spin parameters. Emcee is in strong misalignment with all other approaches.

It is evident in Fig's 5.7, 5.6, 5.8, 5.9 that JS values for individual source parameters of VItamin against all other sampler is generally consistent with all other samplers against themselves. This is an important point because it indicates that our machine learning approach is able to produce Bayesian posteriors to within an accuracy which is at a similar level of other Bayesian samplers, using the same individual source parameter figure of merit used for other comparison studies in the Bayesian GW parameter estimation literature [22, 57, 58]. What is also interesting to note is the level of disagreement of other Bayesian samplers with themselves. This disagreement is especially prominent with regards to source parameters t_0, θ_{jn}, ϕ_{jl}, α, and δ. Although Emcee results are fairly poor in comparison with other approaches, they are a useful benchmark for indicating underperformance.

5.5.2 JS divergence as a function of SNR

Over the course of this project there was some discussion on the possibility that SNR might be a limiting factor with regards to the performance of the neural network model. It was originally hypothesized that due to the low number of high SNR signals in our training set (as seen in Fig. 5.10), that the network would perform worse on high SNR signals. This hypothesis is supported by the well known understanding in machine learning literature that less available data in specific regions of the parameter space can cause a neural network model to underfit to those regions [41]. As a test, we have plotted in Fig. 5.11 the JS divergence for individual test

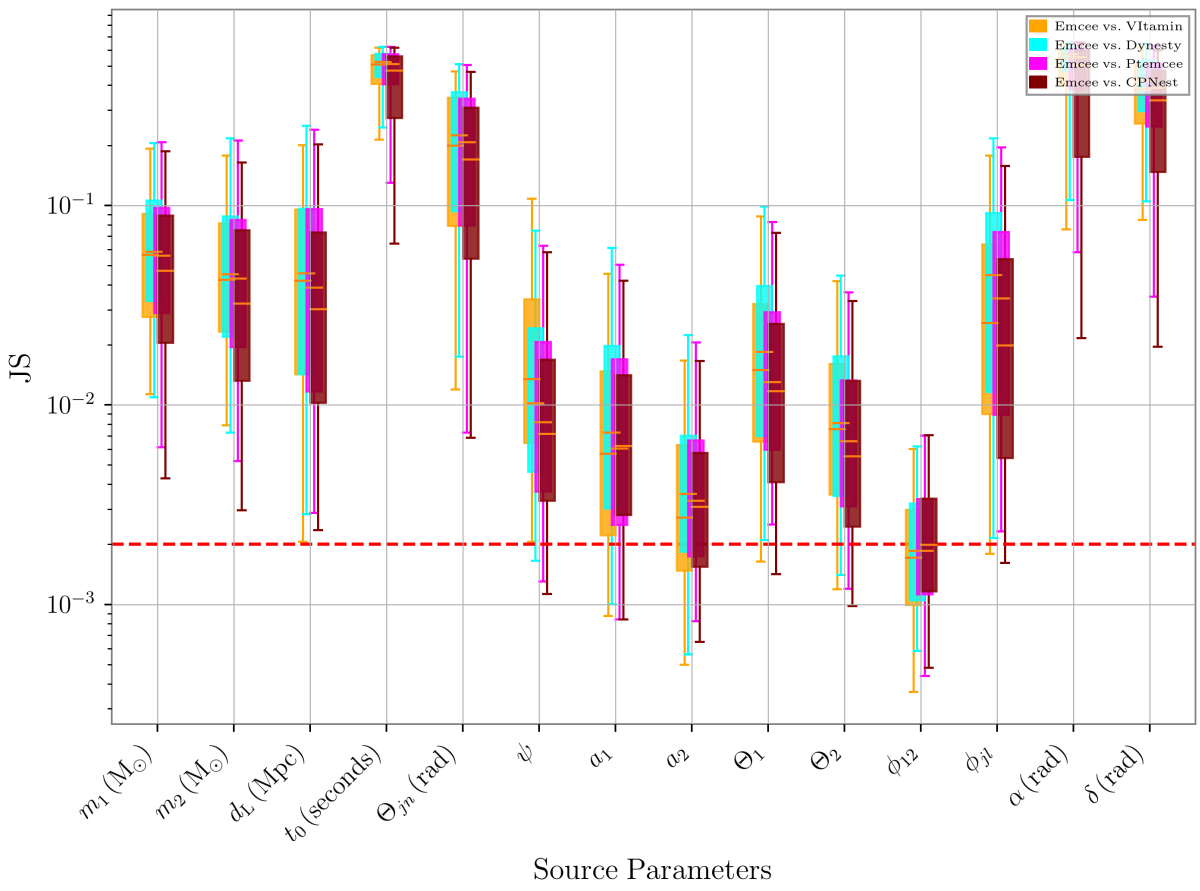


Figure 5.8: Shown are JS divergence values for all test samples as a function of test sample source parameter for `Emcee` against every other sampling approach. Each JS value is a 1-dimensional JS statistic, rather than the full 14-dimensional JS statistics shown earlier. Each sampler method vs. another sampler method are denoted as different colors. The lower and upper end of boxes represent the 25th and 75th credibility regions respectively. The lower and upper end of the whiskers represent the 5th and 95th percent credibility regions. The dashed red line represents optimal JS divergence value according to sampler experts (~ 0.002). The orange line is representative of the median JS value. We see here that `VItamin` performs to within the same degree of accuracy as other Bayesian samplers when looking at predictions on an individual source parameter basis.

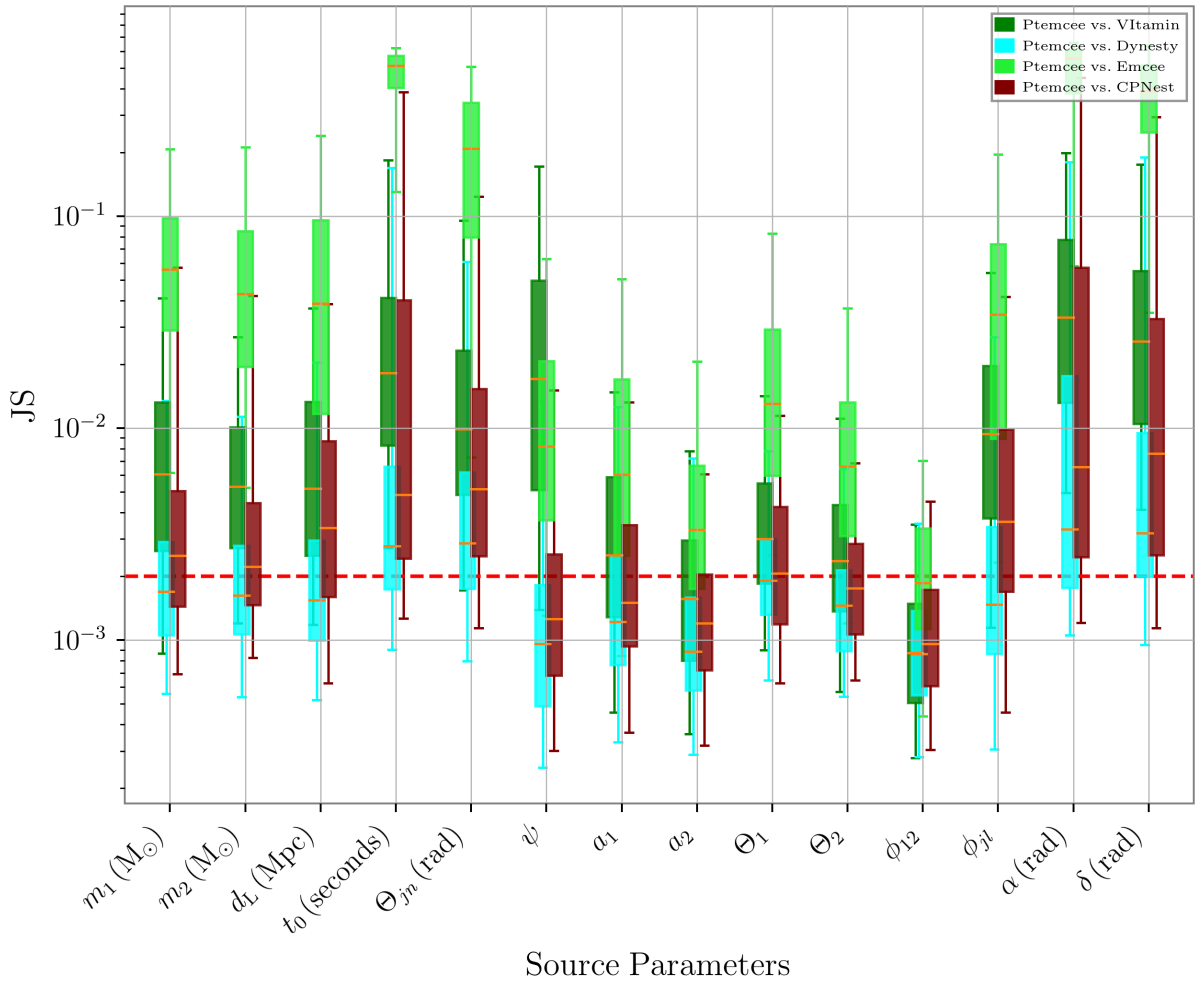


Figure 5.9: Shown are JS divergence values for all test samples as a function of test sample source parameter for `Ptemcee` against every other sampling approach. Each JS value is a 1-dimensional JS statistic, rather than the full 14-dimensional JS statistics shown earlier. Each sampler method vs. another sampler method are denoted as different colors. The lower and upper end of boxes represent the 25th and 75th credibility regions respectively. The lower and upper end of the whiskers represent the 5th and 95th percent credibility regions. The dashed red line represents optimal JS divergence value according to sampler experts (~ 0.002). The orange line is representative of the median JS value. We see here that `VItamin` performs to within the same degree of accuracy as other Bayesian samplers when looking at predictions on an individual source parameter basis.

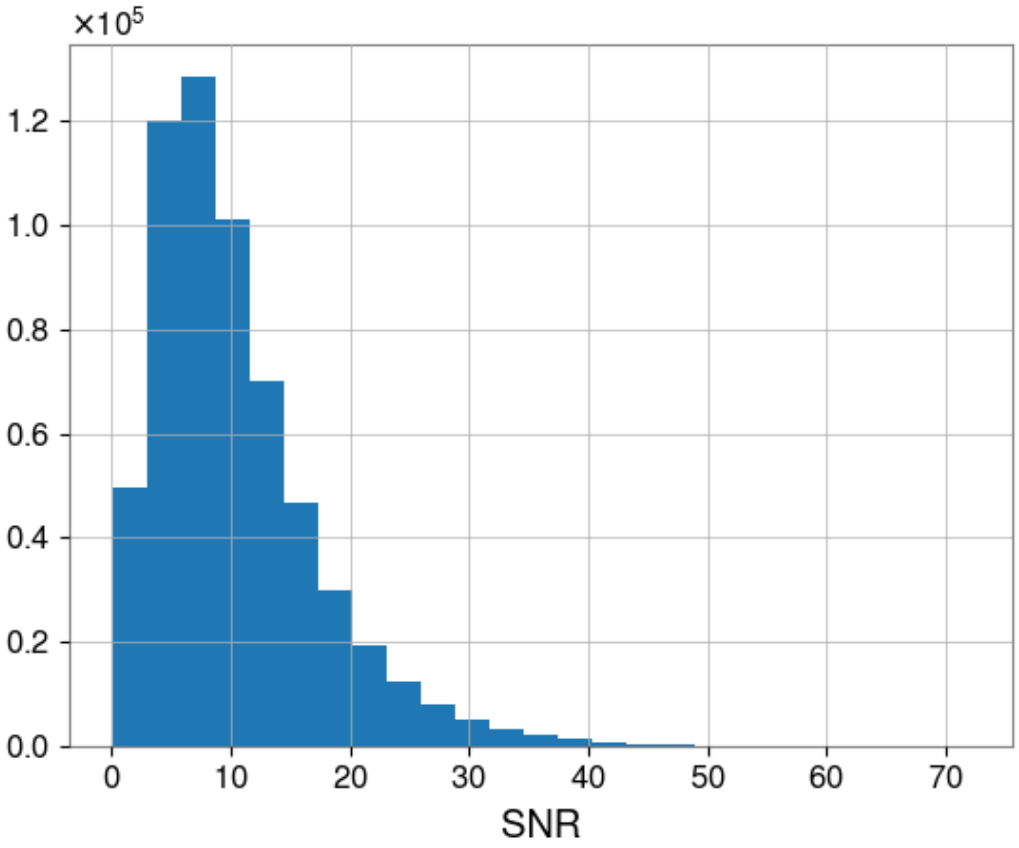


Figure 5.10: Shown is a histogram of the SNR values of the `VItamin` training set. There is a peak of signals centered around an SNR value of 8 which drops off quickly to zero on the left-hand side. There is a tail on the right-hand side which drops off more gradually up to a maximum SNR value of ~ 70. The peak location and general distribution of the SNR values is heavily dependent on both the chosen source parameter priors and the PSD.

sample cases of `VItamin` vs. `Dynesty` as a function of SNR. Each plus sign is representative of indivudal test cases and different colors correspond to each interferometer.

We see in Fig. 5.11 that there is little to no positive correlation between SNR and JS divergence. In fact, there may be a slight negative correlation. This shows that there would be marginal benefit gained from augmenting the training set to more strongly emphasize high SNR signals. Even if there were a positive correlation it is unlikely that simply including more high SNR signals would be beneficially to the statistical results of the network model as a whole because there are fewer test signals at high SNR anyways due to the priors used.

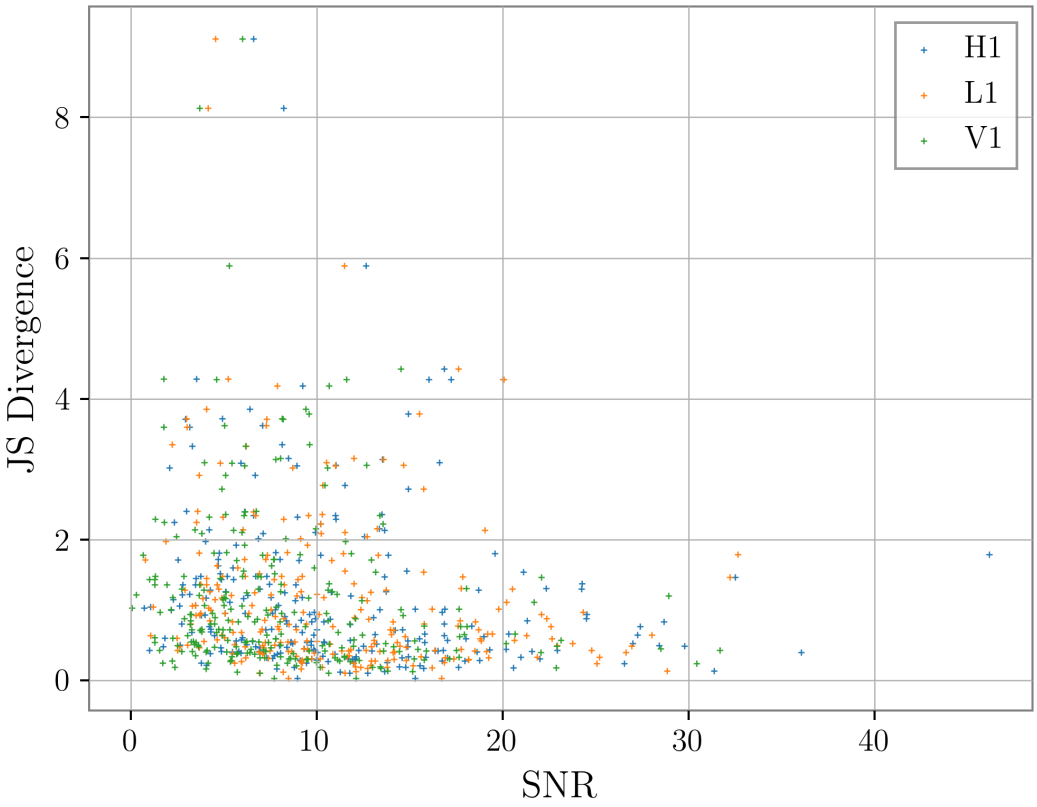


Figure 5.11: Shown are JS divergence values of `Dynesty` vs. `VItamin` as a function of SNR. Different colors are representative of each of the 3 detectors (H1, L1, V1) used in the analysis. Each "plus" symbol represents a different test GW case. As can be seen, there does not appear to be a strong positive correlation with SNR. One could argue there is in fact a small negative correlation with network performance and SNR value, but this is not clear.

5.5.3 VItamin latent space analysis

In this section I will introduce diagnostic plots we have used in order to gauge the performance of our neural network model with respect to the latent space. I will also provide the posterior predictions for context for each test sample discussed.

For context, we will first examine a test case at a median network SNR value of ~ 14.33. As seen in Fig. 5.12, this median SNR event exhibits complex multi-modal behavior in across several dimensions including: t_0, ψ, α and δ. There also appears to some disagreement between Bayesian samplers, particularly on the ϕ_{jl}, ϕ_{12} and a_1 parameters. The sky plot in the upper right-hand corner of Fig. 5.12 is also multi-modal and spans a large region of the sky. We will discuss in the following paragraphs how latent space predictions translate to posterior predctions.

There is no hard rule for determining the number of latent space dimensions to use when deciding on the architecture for a CVAE. That being said, our primary motivation for choosing a latent space size of 15 is directly related to the total number of source parameter posteriors we are trying to get the neural network model to learn. Our hope was that each dimension of the latent space would encode information corresponding to each dimension of the posterior space. However, as can be seen in Fig. 5.14, there are some indications that this may not be the case of what's actually happening underneath the hood.

In Fig. 5.14 we plot 8000 latent space samples drawn from predictions made by both the q (blue) and r_1 (red) encoder networks. If a dimension of the latent space is being used, we would expect that the corresponding 1-D histogram for that latent space dimension along the diagonal of the corner plot to show some level of disagreement between the predictions from both encoder networks. The reasoning behind this statement is that the q network is given a different level of information with respect to the r_1 network. Specifically, the q network is given not only the GW time series, but also the true values of the source parameters themselves, whereas the r_1 network is given the GW time series by itself. If a latent space dimension is not being used it means that there is no additional amount of information that can be gleamed from that dimension, so both encoder networks will simply return mean zero unit variate Gaussian distributions, which contribute nothing to the loss function. Nothing is contributed to the loss function during training because the KL divergence between two Gaussian distributions is zero. Since the KL component

of the loss is zero for this dimension, weights will not be updated during the backpropogation process which encourage this dimension to be used.

From Fig. 5.14 we see that 6 of the 15 latent space dimensions are not being used for this test case. It also appears that the r_1 encoder network (which is capable of producing multi-modal distributions) is in fact choosing to, on the whole, produce latent space samples which are representative of uni-modal distributions. What this implies is that the multi-modality clearly seen in the final posteriors of Fig. 5.12 is entirely handled by the r_2 decoder network which is apprently choosing what level of likelihood to assign to each mode in the posterior space.

This could indicate that the decoder network could have a higher level of capacity than what may be needed and is just bypassing the Gaussian Mixture Model in the r_1 encoder network.

5.5.4 Dynesty vs. Dynesty JS Divergence

Here we provide some discussion on what the absolute best JS values we could hope for from `VItamin` by comparing two independent Dynesty runs on all 250 test sample cases for both the full 14-dimensional JS divergence and the 1-dimensional JS divergence.

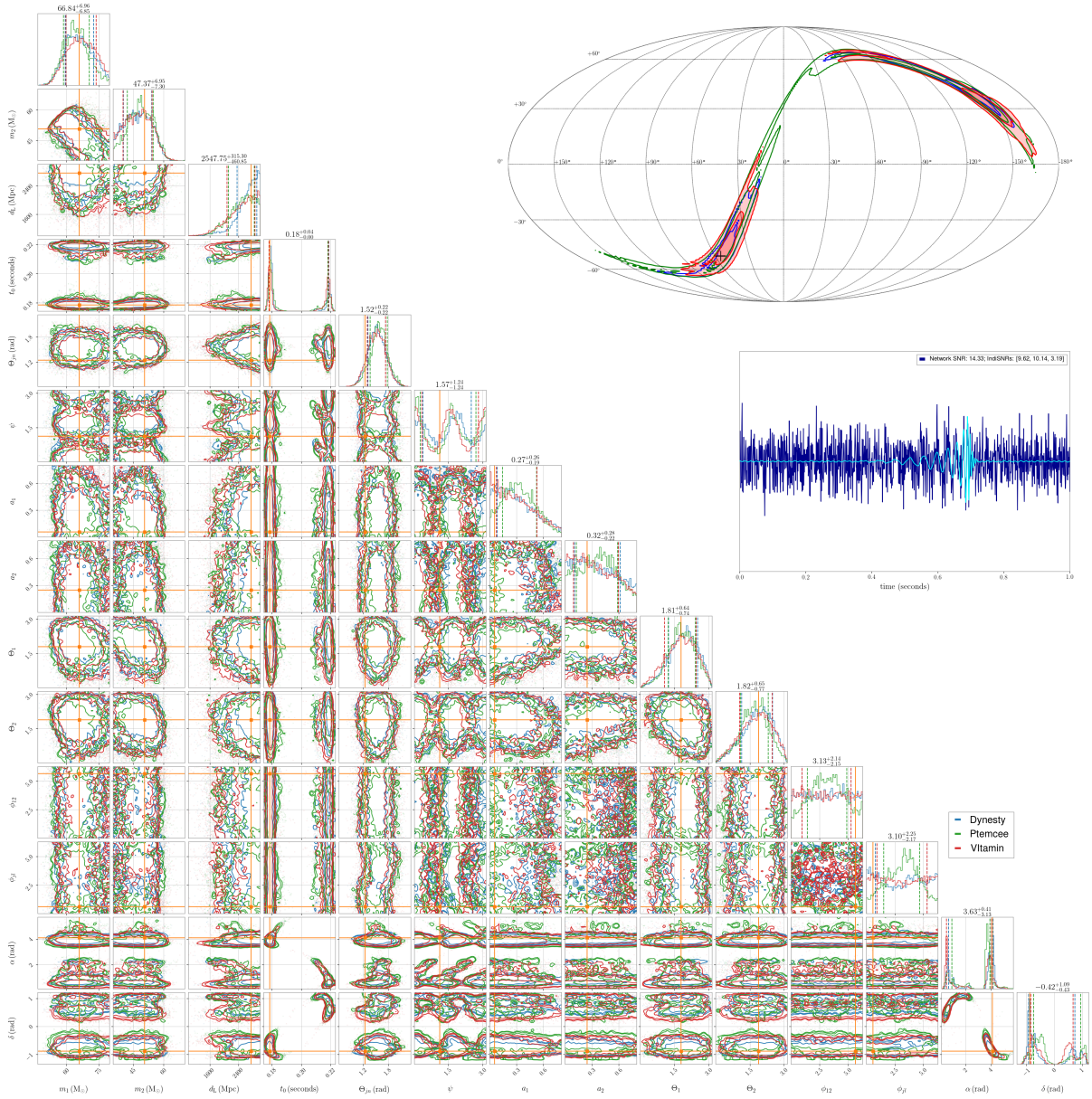


Figure 5.12: Corner plot showing 2 and 1-dimensional marginalised posterior distributions for 184th test dataset sample. Filled (red) contours represent the posteriors obtained from the CVAE approach and solid (blue) contours are the posteriors output from our baseline analysis (`Bilby` using the `Dynesty` sampler). In each case, the contour boundaries enclose 68, 90 and 95% probability. One dimensional histograms of the posterior distribution for each parameter from both methods are plotted along the diagonal. Blue and red vertical lines represent the 5—95% symmetric confidence bounds for `Bilby` and variational inference respectively. Black crosses and vertical black lines denote the true parameter values of the simulated signal. The original whitened noisy timeseries y and the noise-free signal are plotted in blue and cyan respectively in the upper right hand panel. The test signal was simulated with network optimal signal-to-noise ratio of 14.33.

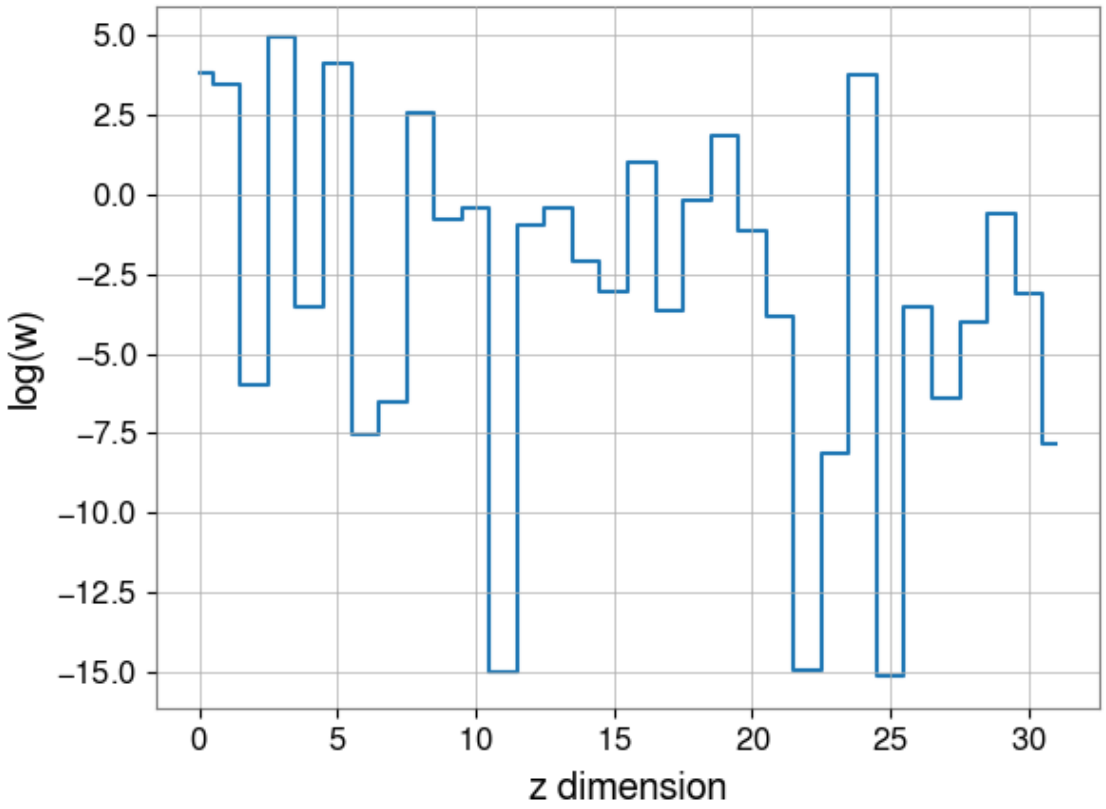


Figure 5.13: Plotted are the predicted log'd weight values for the zeroth posterior sample of the 184th test sample as a function of mode latent space dimension number. Mode weights are predicted for each posterior sample by the r_1 encoder network.

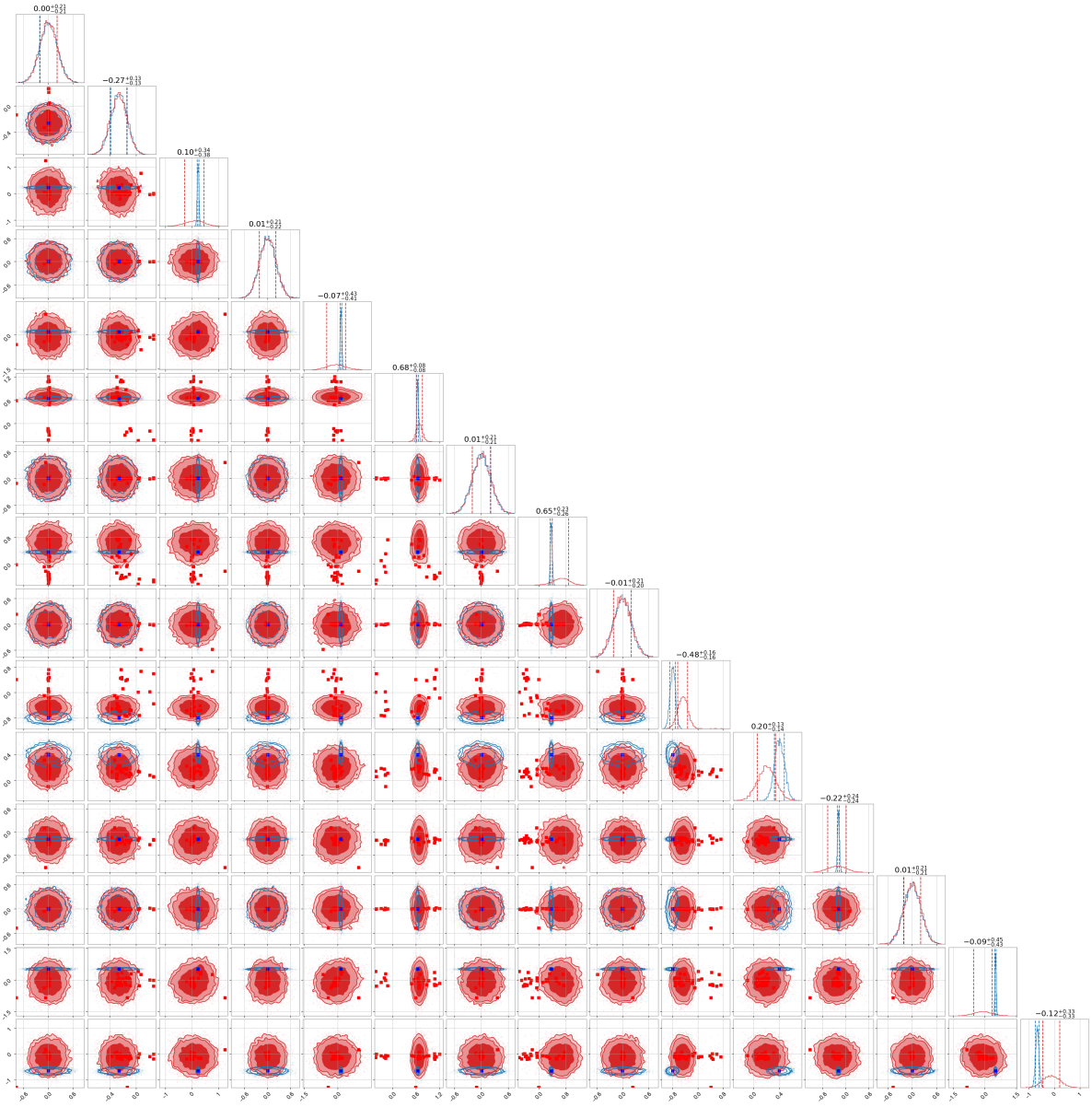


Figure 5.14: Shown are 8000 latent space samples from all latent space dimensions of both the q encoder network (blue) and the r_1 encoder network (red). Each square point is representative of the predicted mean values for each latent space dimension (15). Each dimension on the x and y axis represents a different latent space dimension. 1-d histograms of latent space samples for each dimension are plotted along the diagonal. Contours represent the $68, 90, 95\%$ credibility intervals. Blue and red vertical lines denote the $5-95\%$ symmetric confidence bounds for the q and r_1 networks respectively.

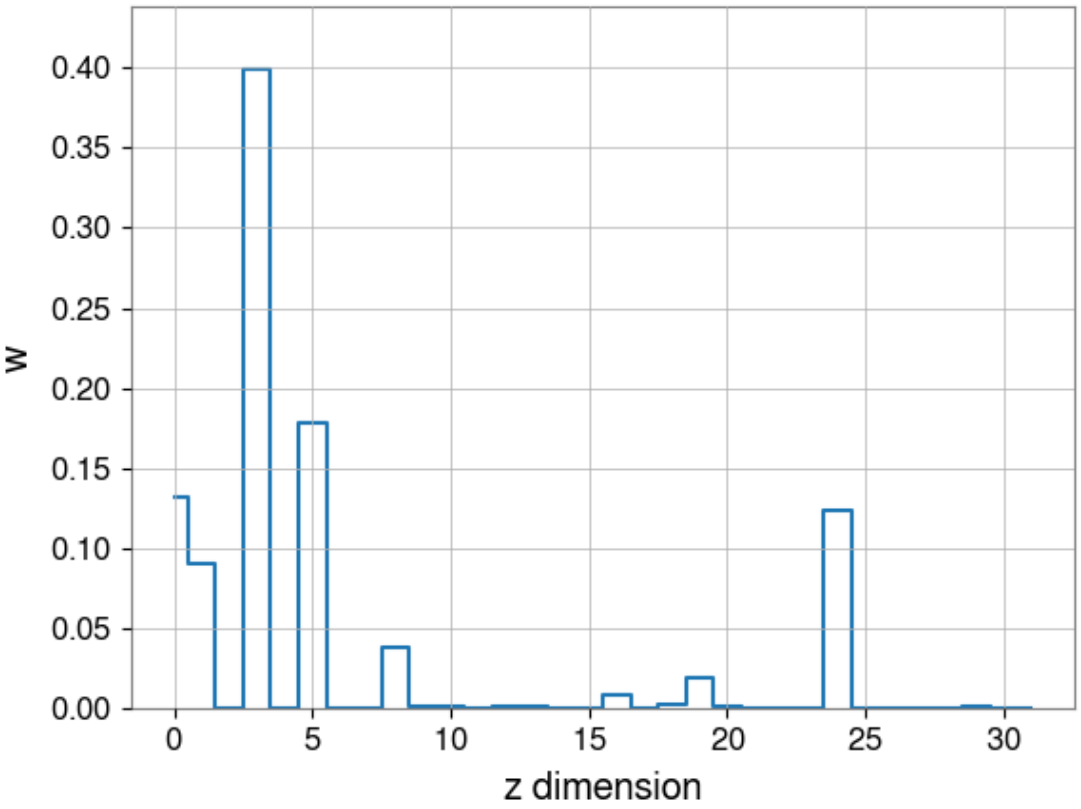


Figure 5.15: Plotted are the unlogged predicted weight values from the r_1 encoder network for the zeroth posterior sample of the 184th test case as a function of latent space mode dimension number. Each value is normalised to be betweeen 0 and 1 where 1 is representative of the network model assigning a high likelihood of sampling from that particular mode.

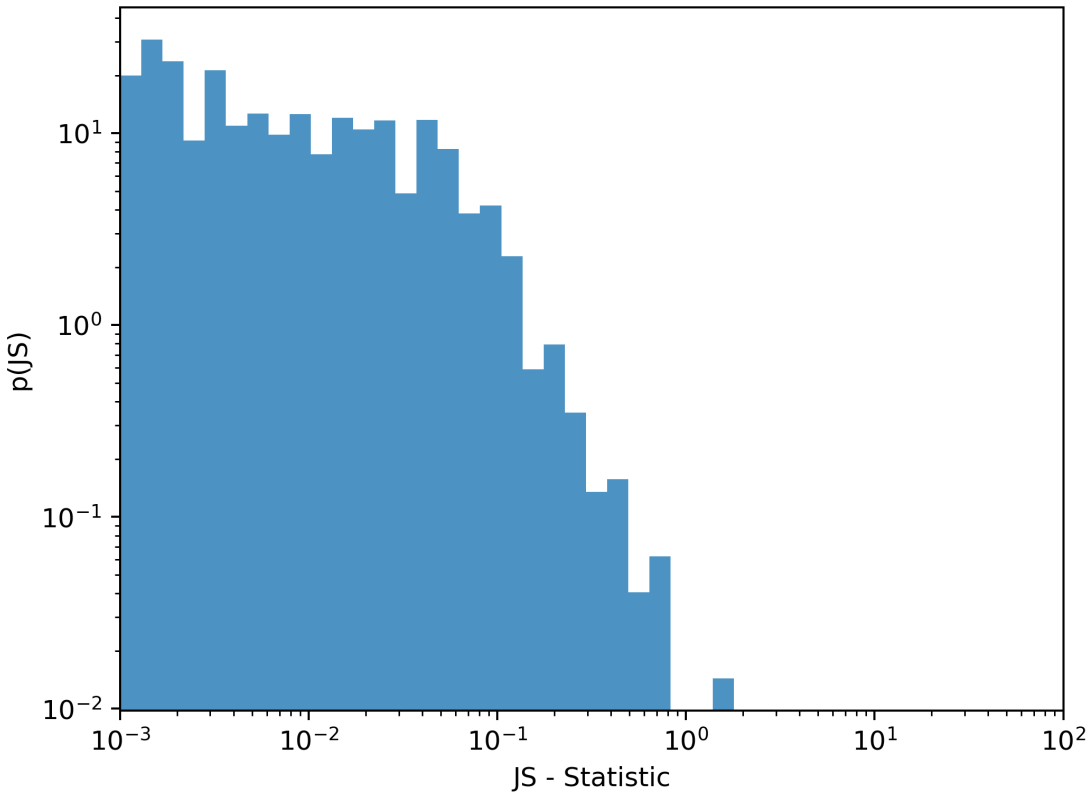


Figure 5.16: A histogram of JS divergence values for `Dynesty` vs. another independent run of `Dynesty`. The mean JS divergence has a value of 0.05. We do note that since since the JS statistic is calculated using the `universal-divergence` code-base, that there are some negative values which are not shown here.

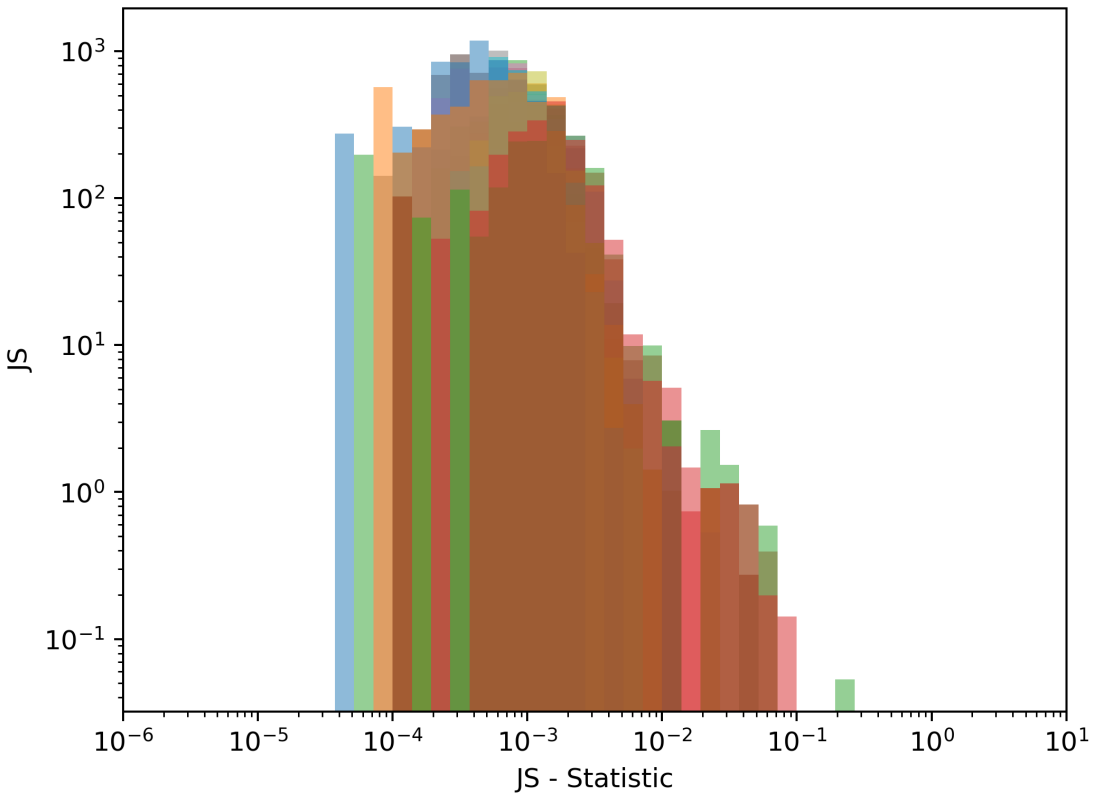


Figure 5.17: Shown are histograms of 1-dimensional JS divergence values for `Dynesty` vs. another independent run of `Dynesty`. Different colors represent JS values with respect to individual source parameter posterior `Dynesty` predictions. Mean JS values for each source parameter are given as: $m_1 \sim 0.00139$, $m_2 \sim 0.00143$, $d_l \sim 0.00142$, $t_0 \sim 0.00358$, $\theta_{jn} \sim 0.00192$, $\psi \sim 0.00134$, $a_1 \sim 0.00101$, $a_2 \sim 0.00088$, $\Theta_1 \sim 0.00143$, $\Theta_2 \sim 0.00123$, $\phi_{12} \sim 0.00080$, $\phi_{jl} \sim 0.00136$, $\alpha \sim 0.00538$, $\delta \sim 0.00401$. JS values are calculated using the `scipy` JS divergence code-base.

Chapter 6

Conditional variational autoencoders for detecting real GW signals

In this chapter I will discuss some recent soon to be published work on using the `VItamin` machine learning for gravitational wave parameter estimation pipeline to do Bayesian PE on real GW signals. In the first section I will introduce the problem, in the second I will discuss data pre-processing and generation, in the third section I will discuss results in both real and Gaussian noise cases on a variety of real GW signals.

6.1 Introduction

6.2 Signal pre-processing/generation

6.3 Training tricks

6.3.1 Data augmentation

In addition to allowing the network to see multiple noise realizations of the same signal multiple times, every time we give the network a new chunk of signals, we also randomize the phase, time of arrival and distance of the new loaded in training samples.

For distance, we first choose values uniformly at random from 0 to 1 for each distance

training sample parameter. These values are then converted to units of Mpc by

$$d_{\text{new}} = d_{\text{min}} + d_{\text{uni}} * (d_{\text{max}} - d_{\text{min}}), \tag{6.1}$$

where d_{uni} is a uniform set of numbers between 0 and 1, d_{max} represents the maximum allowed distance according to the prior and d_{min} is the minimum allowed distance according to the prior. We then determine the factor by which the distance has changed from its old value for each training sample by dividing the old distance by the new distance value.

$$d_{\text{corr}} = \frac{d_{\text{old}}}{d_{\text{new}}} \tag{6.2}$$

where d_{old} is the original distance value for the sample and d_{new} is the new value.

In order to get the phase augmentation scale term we do a similar process as the distance augmentation above. We again choose a uniform value between 0 and 1 and apply the same rescaling as that in equation 6.1, but instead with bounds relavent to phase (0 to π). Bounds are only up to a maximum of π due an applied reparameterization trick discussed below in subsection 6.3.2. A phase correction term is then calculated

$$\phi_{\text{corr}} = -1.0 \times (\cos(\phi_{\text{new}} - \phi_{\text{old}}) + \sin(\phi_{\text{new}} - \phi_{\text{old}}) \times i), \tag{6.3}$$

where ϕ_{corr} is the phase correction factor we will use to randomize the phase, ϕ_{new} is the new randomized phase value and ϕ_{old} is the original training sample phase value.

The time scale term is then computed by again choosing a random value between 0 and 1 for all training samples and rescaling those random values to be in units of seconds which are between the allowable prior bounds of time. We then convert the new randomized time to radians and find the difference between the new and old times

$$t_{\text{corr}} = -2.0\pi(t_{\text{new}} - t_{\text{old}}), \tag{6.4}$$

where t_{new} is the new time of coalescence, t_{old} is the old time of coalescence and t_{corr} is the time of coalesence scale factor. The complex value for the time of coalesence scale factor t_{corr}

is computed

$$t_{\text{corr}} = -1.0 \times (\cos(t_{\text{corr}}) + \sin(t_{\text{corr}}) \times i). \tag{6.5}$$

Finally, given all the correction factors for time t_{corr}, distance d_{corr} and phase ϕ_{corr} have been calculated, we need only simply multiply the phase correction term and the time correction term by the real FFT of the training sample time series

$$y_{\phi\text{corr}} = \mathscr{F}(y) \times \phi_{\text{corr}} \times t_{\text{corr}}, \tag{6.6}$$

where $\mathscr{F}$ is the real FFT. To apply the distance term apply the inverse real FFT and multiply by the distance correction scale factor

$$y_{\phi\text{corr}} = \mathscr{F}^{-1}(y_{\phi\text{corr}}) \times d_{\text{corr}}. \tag{6.7}$$

Adding these randomized elements to the existing training workflow was absolutely key in ensuring that the neural network model did not overfit the training set.

6.3.2 Phase and Psi reparameterization trick

One of the single biggest issues we have faced while training the neural network has been dealing with the multi-modal nature of the phase (ϕ) and psi (ψ) parameters. Along with the addition of the Gaussian mixture model component of the network mentioned previously, we have also implemented a reparameterization of both phase and psi in order to simplify the search space for the neural network partly inspired by the work of Jones in [67].

In order to go from ψ and ϕ to a new representation ψ and X, we first take the remainder of $\psi + \phi$ divided by π which then becomes a new parameter denoted as X. We also take the remainder of ψ divided by $\frac{\pi}{2}$.

$$X = (\psi + \phi) \bmod \pi. \tag{6.8}$$

$$\psi = \psi \bmod \frac{\pi}{2}. \tag{6.9}$$

In order to get back to the original ψ, ϕ representation, we choose a set of two random integers between zero and 2 (D_1) multiplied by π, as well as a set of two random integers between 0 and $\frac{1}{2}$ (D_2) multiplied by π for each ψ value and ensure both ψ and X are in radians.

$$D_1 = \{x \in \mathbb{Z} | 0 \leq x \leq 2\} \times \pi \tag{6.10}$$

$$D_2 = \left\{x \in \mathbb{Z} | 0 \leq x \leq \frac{1}{2}\right\} \times \pi \tag{6.11}$$

We then subtract off ψ from X, add both random radian integers and take the modulus of the whole expression with respect to 2π in order to get back ϕ.

$$\phi = ((X - \psi) + D_1 + D_2) \bmod 2\pi \tag{6.12}$$

For ψ we add set D_2 and take the modulus with respect to π

$$\psi = (\psi + D_2) \bmod \pi. \tag{6.13}$$

This essentially tessellates the $X - \psi$ and ψ parameters across four quadrants of the parameter space while maintaining the same number of samples and general distribution shape.

The reparameterization process is visually illustrated in Figure 6.1. It can be clearly seen that the 2D representation X, ψ in both the upper left and lower right subplots, is vastly simpler than the original 2D ϕ, ψ representation. The transformation also is able to maintain the property of being fully-reversible. Although, we do note that if one considers GW template waveforms with higher order modes this degeneracy is broken and the above reparameterization would not be necessary [67].

6.4 Results

6.5 Conclusions

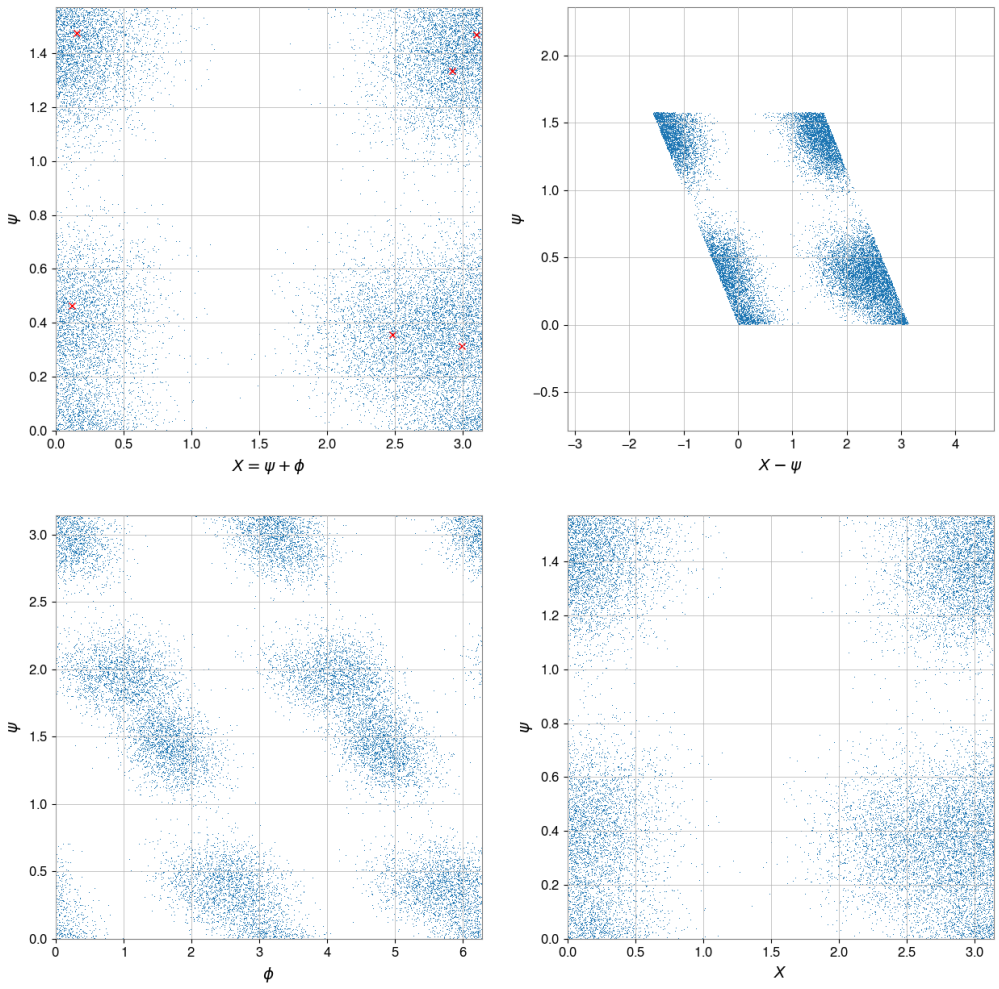


Figure 6.1: An illustration of the ψ and ϕ reparameterization trick. Upper left plot: a random set of samples drawn from randomly generated multivariate Gaussian distributions for both ψ and ϕ. Red cross hairs denote Each Gaussian's mean and X represents a reparameterization of ψ and ϕ. Upper right: Same plot as the upper right, but with ψ subtracted off from the reparameterization in order to form a trapezoidal-like shape for tessellation purposes. Lower left: We then apply a tessellation of the samples in the upper right figure in order to convert the reparameterization back to the original units of ψ and ϕ. Lower right: We can get back to the reparameterized version by applying our reparameterization trick again without any change to the original in the upper left.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

7.2 Future Work

Bibliography

[1] Advanced LIGO sensitivity design curve. https://dcc.ligo.org/LIGO-T1800044/public. Accessed: 2019-06-01.

[2] Gracedb — gravitational-wave candidate event database (ligo/virgo o3 public alerts). https://gracedb.ligo.org/superevents/public/O3/. Accessed: 2019-09-16.

[3] Finding the origin of noise transients in ligo data with machine learning. *Communications in Computational Physics*, 25(4):963–987, 2018.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] B. P. Abbott, R. Abbott, T. D. Abbott, M. R. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, and et al. Prospects for Observing and Localizing Gravitational-Wave Transients with Advanced LIGO and Advanced Virgo. *Living Reviews in Relativity*, 19:1, Feb. 2016.

[6] B. P. Abbott et al. Search for gravitational-wave bursts in LIGO data from the fourth science run. *Classical and Quantum Gravity*, 24(22):5343–5369, oct 2007.

[7] B. P. Abbott et al. Binary black hole mergers in the first advanced LIGO observing run. *Phys. Rev. X*, 6:041015, Oct 2016.

[8] B. P. Abbott et al. Characterization of transient noise in advanced ligo relevant to gravitational wave signal gw150914. *Classical and Quantum Gravity*, 33(13):134001, 2016.

[9] B. P. Abbott et al. Gw151226: Observation of gravitational waves from a 22-solar-mass binary black hole coalescence. *Phys. Rev. Lett.*, 116:241103, Jun 2016.

[10] B. P. Abbott et al. Observation of gravitational waves from a binary black hole merger. *Phys. Rev. Lett.*, 116:061102, Feb 2016.

[11] B. P. Abbott et al. Tests of general relativity with gw150914. *Phys. Rev. Lett.*, 116:221101, May 2016.

[12] B. P. Abbott et al. Gw170104: Observation of a 50-solar-mass binary black hole coalescence at redshift 0.2. *Phys. Rev. Lett.*, 118:221101, Jun 2017.

[13] B. P. Abbott et al. Gw170814: A three-detector observation of gravitational waves from a binary black hole coalescence. *Phys. Rev. Lett.*, 119:141101, Oct 2017.

[14] B. P. Abbott et al. Gw170817: Observation of gravitational waves from a binary neutron star inspiral. *Phys. Rev. Lett.*, 119:161101, Oct 2017.

[15] B. P. Abbott et al. Prospects for observing and localizing gravitational-wave transients with Advanced LIGO, Advanced Virgo and KAGRA. *Living Reviews in Relativity*, 21(1):3, Apr 2018.

[16] B. P. Abbott et al. Binary black hole population properties inferred from the first and second observing runs of advanced LIGO and advanced virgo. *The Astrophysical Journal*, 882(2):L24, sep 2019.

[17] O. D. Aguiar. The past, present and future of the resonant-mass gravitational wave detectors. 2010.

[18] P. Ajith, N. Fotopoulos, S. Privitera, A. Neunzert, N. Mazumder, and A. J. Weinstein. Effectual template bank for the detection of gravitational waves from inspiralling compact binaries with generic spins. *Phys. Rev. D*, 89:084041, Apr 2014.

[19] B. Allen. χ^2. *Phys. Rev. D*, 71:062001, Mar 2005.

[20] B. Allen, W. G. Anderson, P. R. Brady, D. A. Brown, and J. D. E. Creighton. Findchirp: An algorithm for detection of gravitational waves from inspiraling compact binaries. *Phys. Rev. D*, 85:122006, Jun 2012.

[21] K. G. Arun, A. Buonanno, G. Faye, and E. Ochsner. Erratum: Higher-order spin effects in the amplitude and phase of gravitational waveforms emitted by inspiraling compact binaries: Ready-to-use gravitational waveforms [phys. rev. d 79, 104023 (2009)]. *Phys. Rev. D*, 84:049901, Aug 2011.

[22] G. Ashton, M. Huebner, P. D. Lasky, C. Talbot, K. Ackley, S. Biscoveanu, Q. Chu, A. Divarkala, P. J. Easter, B. Goncharov, F. H. Vivanco, J. Harms, M. E. Lower, G. D. Meadors, D. Melchor, E. Payne, M. D. Pitkin, J. Powell, N. Sarin, R. J. E. Smith, and E. Thrane. Bilby: A user-friendly bayesian inference library for gravitational-wave astronomy. *Astrophysical Journal Supplement Series*, 2018.

[23] P. Astone, A. Colla, S. D'Antonio, S. Frasca, and C. Palomba. Method for all-sky searches of continuous gravitational wave signals using the frequency-hough transform. *Phys. Rev. D*, 90:042002, Aug 2014.

[24] S. Babak, R. Balasubramanian, D. Churches, T. Cokelaer, and B. S. Sathyaprakash. A template bank to search for gravitational waves from inspiralling compact binaries: I. physical models. *Classical and Quantum Gravity*, 23(18):5477, 2006.

[25] S. Babak, R. Biswas, P. R. Brady, D. A. Brown, K. Cannon, C. D. Capano, J. H. Clayton, T. Cokelaer, J. D. E. Creighton, T. Dent, A. Dietz, S. Fairhurst, N. Fotopoulos, G. González, C. Hanna, I. W. Harry, G. Jones, D. Keppel, D. J. A. McKechan, L. Pekowsky, S. Privitera, C. Robinson, A. C. Rodriguez, B. S. Sathyaprakash, A. S. Sengupta, M. Vallisneri, R. Vaulin, and A. J. Weinstein. Searching for gravitational waves from binary coalescence. , 87(2):024033, Jan. 2013.

[26] L. Blanchet. Gravitational radiation from post-newtonian sources and inspiralling compact binaries. *Living Reviews in Relativity*, 17(1):2, Feb 2014.

[27] D. A. Brown, I. Harry, A. Lundgren, and A. H. Nitz. Detecting binary neutron star systems with spin in advanced gravitational-wave detectors. *Phys. Rev. D*, 86:084017, Oct 2012.

[28] A. Buonanno and T. Damour. Effective one-body approach to general relativistic two-body dynamics. *Phys. Rev. D*, 59:084006, Mar 1999.

[29] A. Buonanno, B. R. Iyer, E. Ochsner, Y. Pan, and B. S. Sathyaprakash. Comparison of post-newtonian templates for compact binary inspiral signals in gravitational-wave detectors. *Phys. Rev. D*, 80:084043, Oct 2009.

[30] K. Cannon, R. Cariou, A. Chapman, M. Crispin-Ortuzar, N. Fotopoulos, M. Frei, C. Hanna, E. Kara, D. Keppel, L. Liao, S. Privitera, A. Searle, L. Singer, and A. Weinstein. Toward early-warning detection of gravitational waves from compact binary coalescence. *The Astrophysical Journal*, 748(2):136, 2012.

[31] T. D. Canton and I. W. Harry. Designing a template bank to observe compact binary coalescences in advanced ligo's second observing run, 2017.

[32] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs, 2014.

[33] A. J. K. Chua, C. R. Galley, and M. Vallisneri. Reduced-order modeling with artificial neurons for gravitational-wave inference. *Phys. Rev. Lett.*, 122:211101, May 2019.

[34] A. J. K. Chua and M. Vallisneri. Learning Bayes' theorem with a neural network for gravitational-wave inference. *arXiv e-prints*, page arXiv:1909.05966, Sep 2019.

[35] T. L. S. Collaboration, the Virgo Collaboration, B. P. Abbott, R. Abbott, T. D. Abbott, et al. First low-frequency einstein@home all-sky search for continuous gravitational waves in advanced ligo data, 2017.

[36] T. L. S. Collaboration, the Virgo Collaboration, B. P. Abbott, et al. Effects of data quality vetoes on a search for compact binary coalescences in advanced ligo's first observing run, 2017.

[37] M. Coughlin, P. Earle, J. Harms, S. Biscans, C. Buchanan, E. Coughlin, F. Donovan, J. Fee, H. Gabbard, M. Guy, N. Mukund, and M. Perry. Limiting the effects of earthquakes on gravitational-wave interferometers. *Classical and Quantum Gravity*, 34(4):044004, feb 2017.

[38] T. Dal Canton, A. H. Nitz, A. P. Lundgren, A. B. Nielsen, D. A. Brown, T. Dent, I. W. Harry, B. Krishnan, A. J. Miller, K. Wette, K. Wiesner, and J. L. Willis. Implementing a search for aligned-spin neutron star-black hole systems with advanced ground based gravitational wave detectors. *Phys. Rev. D*, 90:082004, Oct 2014.

[39] T. Dozat. Incorporating nesterov momentum into adam, 2016.

[40] A. Einstein. Die grundlage der allgemeinen relativitätstheorie [adp 49, 769 (1916)]. *Annalen der Physik*, 14(S1):517–571, 2005.

[41] B. Everitt and A. Skrondal. *The Cambridge Dictionary of Statistics*. Cambridge University Press, 2010.

[42] É. É. Flanagan and S. A. Hughes. The basics of gravitational wave theory. *New Journal of Physics*, 7:204–204, sep 2005.

[43] D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman. emcee: The mcmc hammer. *PASP*, 125:306–312, 2013.

[44] P. I. Frazier. A tutorial on bayesian optimization, 2018.

[45] H. Gabbard, C. Messenger, I. S. Heng, F. Tonolini, and R. Murray-Smith. Bayesian parameter estimation using conditional variational autoencoders for gravitational-wave astronomy, 2019.

[46] H. Gabbard, M. Williams, F. Hayes, and C. Messenger. Matching matched filtering with deep networks for gravitational-wave astronomy. *Phys. Rev. Lett.*, 120:141103, Apr 2018.

[47] P. Gallinari, Y. LeCun, S. Thiria, and F. F. Soulie. Mémoires associatives distribuées: une comparaison (distributed associative memories: a comparison). In *Proceedings of COGNITIVA 87, Paris, La Villette, May 1987*. Cesta-Afcet, 1987.

[48] T. Gebhard, N. Kilbertus, G. Parascandolo, I. Harry, and B. Schölkopf. Convwave: Searching for gravitational waves with fully convolutional neural nets. In *Workshop on Deep Learning for Physical Sciences (DLPS) at the 31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

[49] T. D. Gebhard, N. Kilbertus, I. Harry, and B. Schölkopf. Convolutional neural networks: a magic bullet for gravitational-wave detection?, 2019.

[50] D. George and E. Huerta. Deep learning for real-time gravitational wave detection and parameter estimation: Results with advanced ligo data. *Physics Letters B*, 778:64 – 70, 2018.

[51] D. George and E. A. Huerta. Deep neural networks to enable real-time multimessenger astrophysics. *Phys. Rev. D*, 97:044039, Feb 2018.

[52] D. George, H. Shen, and E. A. Huerta. Deep transfer learning: A new deep learning glitch classification method for advanced LIGO, 2017.

[53] A. Goldstein, P. Veres, E. Burns, M. S. Briggs, R. Hamburg, D. Kocevski, C. A. Wilson-Hodge, R. D. Preece, S. Poolakkil, O. J. Roberts, C. M. Hui, V. Connaughton, J. Racusin, A. von Kienlin, T. Dal Canton, N. Christensen, T. B. Littenberg, K. Siellez, L. Blackburn, J. Broida, E. Bissaldi, W. H. Cleveland, M. H. Gibby, M. M. Giles, R. M. Kippen, S. McBreen, J. McEnery, C. A. Meegan, W. S. Paciesas, and M. Stanbro. An Ordinary Short Gamma-Ray Burst with Extraordinary Implications: Fermi-GBM Detection of GRB 170817A. *ArXiv e-prints*, Oct. 2017.

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

[55] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[56] P. Graff, F. Feroz, M. P. Hobson, and A. Lasenby. Bambi: blind accelerated multimodal bayesian inference. *Monthly Notices of the Royal Astronomical Society*, 421(1):169–180, 03 2012.

[57] S. R. Green and J. Gair. Complete parameter inference for gw150914 using deep learning, 2020.

[58] S. R. Green, C. Simpson, and J. Gair. Gravitational-wave parameter estimation with autoregressive neural network flows. *Phys. Rev. D*, 102:104057, Nov 2020.

[59] I. W. Harry, B. Allen, and B. S. Sathyaprakash. Stochastic template placement algorithm for gravitational wave data analysis. *Phys. Rev. D*, 80:104014, Nov 2009.

[60] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. 2014.

[61] S. Hild. *A Basic Introduction to Quantum Noise and Quantum-Non-Demolition Techniques*, pages 291–314. Springer International Publishing, Cham, 2014.

[62] C. J. Hogan and P. L. Bender. Estimating stochastic gravitational wave backgrounds with the sagnac calibration. *Phys. Rev. D*, 64:062002, Aug 2001.

[63] R. A. Hulse and J. H. Taylor. Discovery of a pulsar in a binary system. , 195:L51–L53, Jan. 1975.

[64] S. Husa, S. Khan, M. Hannam, M. Pürrer, F. Ohme, X. J. Forteza, and A. Bohé. Frequency-domain gravitational waves from nonprecessing black-hole binaries. i. new numerical waveforms and anatomy of the signal. *Phys. Rev. D*, 93:044006, Feb 2016.

[65] C. W. Jameson Rollins and R. Adhikari. pygwinc. https://git.ligo.org/gwinc/pygwinc.git, July 2019.

[66] K. Janocha and W. M. Czarnecki. On loss functions for deep neural networks in classification, 2017.

[67] D. I. Jones. Parameter choices and ranges for continuous gravitational wave searches for steadily spinning neutron stars. *Monthly Notices of the Royal Astronomical Society*, 453(1):53–66, 08 2015.

[68] C. Kalaghatgi, P. Ajith, and K. G. Arun. Template-space metric for searches for gravitational waves from the inspiral, merger, and ringdown of binary black holes. *Phys. Rev. D*, 91:124042, Jun 2015.

[69] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions, 2014.

[70] V. Kazemi and J. Sullivan. One millisecond face alignment with an ensemble of regression trees. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.

[71] S. Khan, K. Chatziioannou, M. Hannam, and F. Ohme. Phenomenological model for the gravitational-wave signal from precessing binary black holes with two-spin effects, 2018.

[72] S. Khan, S. Husa, M. Hannam, F. Ohme, M. Pürrer, X. J. Forteza, and A. Bohé. Frequency-domain gravitational waves from nonprecessing black-hole binaries. ii. a phenomenological model for the advanced detector era. *Phys. Rev. D*, 93:044007, Feb 2016.

[73] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013.

[74] I. Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine*, 23(1):89 – 109, 2001.

[75] P. G. Krastev. Real-time detection of gravitational waves from binary neutron stars using artificial neural networks. *Physics Letters B*, 803:135330, 2020.

[76] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[77] S. K. Kumar. On weight initialization in deep neural networks, 2017.

[78] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[79] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[80] LIGO Scientific Collaboration, Virgo Collaboration, F. Gamma-Ray Burst Monitor, and INTEGRAL. Gravitational Waves and Gamma-rays from a Binary Neutron Star Merger: GW170817 and GRB 170817A. *ArXiv e-prints*, Oct. 2017.

[81] LIGO Scientific Collaboration, Virgo Collaboration, F. GBM, INTEGRAL, IceCube Collaboration, AstroSat Cadmium Zinc Telluride Imager Team, IPN Collaboration, The Insight-Hxmt Collaboration, ANTARES Collaboration, The Swift Collaboration, AGILE Team, The 1M2H Team, The Dark Energy Camera GW-EM Collaboration, the DES Collaboration, The DLT40 Collaboration, GRAWITA, :, GRAvitational Wave Inaf TeAm, The Fermi Large Area Telescope Collaboration, ATCA, :, A. Telescope Compact Array, ASKAP, :, A. SKA Pathfinder, Las Cumbres Observatory Group, OzGrav, DWF, AST3, CAASTRO Collaborations, The VINROUGE Collaboration, MASTER Collaboration, J-GEM, GROWTH, JAGWAR, C. NRAO, TTU-NRAO, NuSTAR Collaborations, Pan-STARRS, The MAXI Team, T. Consortium, KU Collaboration, N. Optical Telescope, ePESSTO, GROND, T. Tech University, SALT Group, TOROS, :, Transient Robotic Observatory of the South Collaboration, The BOOTES Collaboration, MWA, :, M. Widefield Array, The CALET Collaboration, IKI-GW Follow-up Collaboration, H. E. S. S. Collaboration, LOFAR Collaboration, LWA, :, L. Wavelength Array, HAWC Collaboration, The Pierre Auger Collaboration, ALMA Collaboration, Euro VLBI Team, Pi of the Sky Collaboration, The Chandra Team at McGill University, DFN, :, D. Fireball Network, ATLAS, H. Time Resolution Universe Survey, RIMAS, RATIR, and S. South Africa/MeerKAT. Multi-messenger Observations of a Binary Neutron Star Merger. *ArXiv e-prints*, Oct. 2017.

[82] Q. Meng, D. Catchpoole, D. Skillicom, and P. J. Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371, 2017.

[83] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[84] C. K. Mishra, A. Kela, K. G. Arun, and G. Faye. Ready-to-use post-newtonian gravitational waveforms for binary black holes with nonprecessing spins: An update. *Phys. Rev. D*, 93:084054, Apr 2016.

[85] N. Mukund, S. Abraham, S. Kandhasamy, S. Mitra, and N. S. Philip. Transient classification in ligo data using difference boosting neural network. *Phys. Rev. D*, 95:104059, May 2017.

[86] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[87] A. Nazabal, P. M. Olmos, Z. Ghahramani, and I. Valera. Handling incomplete heterogeneous data using VAEs, 2018.

[88] J. Neyman. Outline of a theory of statistical estimation based on the classical theory of probability. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 236(767):333–380, 1937.

[89] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug and play generative networks: Conditional iterative generation of images in latent space, 2016.

[90] M. Nishio, C. Nagashima, S. Hirabayashi, A. Ohnishi, K. Sasaki, T. Sagawa, M. Hamada, and T. Yamashita. Convolutional auto-encoder for image denoising of ultra-low-dose ct. *Heliyon*, 3(8):e00393, 2017.

[91] A. Nitz et al. Pycbc software, Sept. 2017.

[92] A. H. Nitz, T. Dent, T. D. Canton, S. Fairhurst, and D. A. Brown. Detecting binary compact-object mergers with gravitational waves: Understanding and improving the sensitivity of the pycbc search. *The Astrophysical Journal*, 849(2):118, 2017.

[93] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[94] S. Odaibo. Tutorial: Deriving the standard variational autoencoder (vae) loss function, 2019.

[95] A. Pagnoni, K. Liu, and S. Li. Conditional variational autoencoder for neural machine translation, 2018.

[96] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[97] G. Papamakarios. Neural density estimation and likelihood-free inference, 2019.

[98] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks, 2012.

[99] M. Pirooznia, J. Y. Yang, M. Q. Yang, and Y. Deng. A comparative study of different machine learning methods on microarray gene expression data. *BMC Genomics*, 9(1):S13, Mar 2008.

[100] F. Pretorius. Evolution of binary black-hole spacetimes. *Phys. Rev. Lett.*, 95:121101, Sep 2005.

[101] S. Privitera, S. R. P. Mohapatra, P. Ajith, K. Cannon, N. Fotopoulos, M. A. Frei, C. Hanna, A. J. Weinstein, and J. T. Whelan. Improving the sensitivity of a search for coalescing binary black holes with nonprecessing spins in gravitational wave data. *Phys. Rev. D*, 89:024003, Jan 2014.

[102] K. Riles. Recent searches for continuous gravitational waves. 2017.

[103] J. D. Romano and N. J. Cornish. Detection methods for stochastic gravitational-wave backgrounds: A unified treatment. *Living Reviews in Relativity*, 20(1):1–223, 2017.

[104] S. Ruder. An overview of gradient descent optimization algorithms, 2016.

[105] B. S. Sathyaprakash and S. V. Dhurandhar. Choice of filters for the detection of gravitational waves from coalescing binaries. *Phys. Rev. D*, 44:3819–3834, Dec 1991.

[106] B. S. Sathyaprakash and B. F. Schutz. Physics, Astrophysics and Cosmology with Gravitational Waves. *Living Reviews in Relativity*, 12(1):2, 2009.

[107] V. Savchenko, C. Ferrigno, E. Kuulkers, A. Bazzano, E. Bozzo, S. Brandt, J. Chenevez, T. J.-L. Courvoisier, R. Diehl, A. Domingo, L. Hanlon, E. Jourdain, A. von Kienlin, P. Laurent, F. Lebrun, A. Lutovinov, A. Martin-Carrillo, S. Mereghetti, L. Natalucci, J. Rodi, J.-P. Roques, R. Sunyaev, and P. Ubertini. INTEGRAL Detection of the First Prompt Gamma-Ray Signal Coincident with the Gravitational Wave Event GW170817. *ArXiv e-prints*, Oct. 2017.

[108] M. B. Schäfer, F. Ohme, and A. H. Nitz. Detection of gravitational-wave signals from binary neutron star mergers using machine learning. *Phys. Rev. D*, 102:063015, Sep 2020.

[109] A. C. Searle, P. J. Sutton, and M. Tinto. Bayesian detection of unmodeled bursts of gravitational waves. *Classical and Quantum Gravity*, 26(15):155017, Aug 2009.

[110] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[111] L. P. Singer and L. R. Price. Rapid Bayesian position reconstruction for gravitational-wave transients. , 93(2):024013, Jan 2016.

[112] J. Skilling. Nested sampling for general bayesian computation. *Bayesian Anal.*, 1(4):833–859, 12 2006.

[113] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama,

and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3483–3491. Curran Associates, Inc., 2015.

[114] J. S. Speagle. dynesty: A dynamic nested sampling package for estimating Bayesian posteriors and evidences, 2019.

[115] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014.

[116] A. Taracchini, A. Buonanno, Y. Pan, T. Hinderer, M. Boyle, D. A. Hemberger, L. E. Kidder, G. Lovelace, A. H. Mroué, H. P. Pfeiffer, M. A. Scheel, B. Szilágyi, N. W. Taylor, and A. Zenginoglu. Effective-one-body model for black-hole binaries with generic mass ratios and spins. *Phys. Rev. D*, 89:061502, Mar 2014.

[117] S. R. Taylor and D. Gerosa. Mining gravitational-wave catalogs to understand binary stellar evolution: A new hierarchical bayesian framework. *Phys. Rev. D*, 98:083017, Oct 2018.

[118] The KAGRA Collaboration, the LIGO Scientific Collaboration, and the Virgo Collaboration. Prospects for observing and localizing gravitational-wave transients with advanced LIGO, advanced Virgo and KAGRA. *Living Reviews in Relativty*, 2013.

[119] The LIGO Scientific Collaboration and the Virgo Collaboration. GWTC-1: A gravitational-wave transient catalog of compact binary mergers observed by LIGO and Virgo during the first and second observing runs, 2018.

[120] The LIGO Scientific Collaboration and the Virgo Collaboration. Gwtc-2: Compact binary coalescences observed by ligo and virgo during the first half of the third observing run, 2020.

[121] C. Tian, L. Fei, W. Zheng, Y. Xu, W. Zuo, and C.-W. Lin. Deep learning on image denoising: An overview, 2019.

[122] F. Tonolini, A. Lyons, P. Caramazza, D. Faccio, and R. Murray-Smith. Variational inference for computational imaging inverse problems, 2019.

[123] A. Toshniwal, K. Mahesh, and R. Jayashree. Overview of anomaly detection techniques in machine learning. In *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 808–815, 2020.

[124] S. A. Usman, A. H. Nitz, I. W. Harry, C. M. Biwer, D. A. Brown, M. Cabero, C. D. Capano, T. D. Canton, T. Dent, S. Fairhurst, M. S. Kehl, D. Keppel, B. Krishnan, A. Lenon, A. Lundgren, A. B. Nielsen, L. P. Pekowsky, H. P. Pfeiffer, P. R. Saulson, M. West, and J. L. Willis. The pycbc search for gravitational waves from compact binary coalescence. *Classical and Quantum Gravity*, 33(21):215004, 2016.

[125] J. Veitch, W. D. Pozzo, C. Messick, and M. Pitkin. Cpnest. Jul 2017.

[126] J. Veitch, V. Raymond, B. Farr, W. M. Farr, P. Graff, S. Vitale, B. Aylott, K. Blackburn, N. Christensen, M. Coughlin, W. D. Pozzo, F. Feroz, J. Gair, C.-J. Haster, V. Kalogera, T. Littenberg, I. Mandel, R. O'Shaughnessy, M. Pitkin, C. Rodriguez, C. Röver, T. Sidery, R. Smith, M. V. D. Sluys, A. Vecchio, W. Vousden, and L. Wade. Robust parameter estimation for compact binaries with ground-based gravitational-wave observations using the lalinference software library. *Physical Review D*, 2014.

[127] S. Vitale, D. Gerosa, C.-J. Haster, K. Chatziioannou, and A. Zimmerman. Impact of bayesian priors on the characterization of binary black hole coalescences. *Phys. Rev. Lett.*, 119:251103, Dec 2017.

[128] W. Vousden, W. M. Farr, and I. Mandel. Dynamic temperature selection for parallel-tempering in Markov chain Monte Carlo simulations. 2015.

[129] S. Walsh, M. Pitkin, M. Oliver, S. D'Antonio, V. Dergachev, A. Królak, P. Astone, M. Bejger, M. Di Giovanni, O. Dorosh, S. Frasca, P. Leaci, S. Mastrogiovanni, A. Miller, C. Palomba, M. A. Papa, O. J. Piccinni, K. Riles, O. Sauter, and A. M. Sintes. Comparison of methods for the detection of gravitational waves from unknown neutron stars. *Phys. Rev. D*, 94:124010, Dec 2016.

[130] W. Wang, Y. Huang, Y. Wang, and L. Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 496–503, 2014.

[131] J. Weber. Gravitational radiation. *Phys. Rev. Lett.*, 18:498–501, Mar 1967.

[132] K. W. K. Wong and D. Gerosa. Machine-learning interpolation of population-synthesis simulations to interpret gravitational-wave observations: A case study. *Phys. Rev. D*, 100:083015, Oct 2019.

[133] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[134] X. Yan, J. Yang, K. Sohn, and H. Lee. Attribute2image: Conditional image generation from visual attributes, 2015.

[135] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks, 2013.

[136] M. Zevin, S. Coughlin, S. Bahaadini, E. Besler, N. Rohani, S. Allen, M. Cabero, K. Crowston, A. K. Katsaggelos, S. L. Larson, T. K. Lee, C. Lintott, T. B. Littenberg, A. Lundgren, C. Østerlund, J. R. Smith, L. Trouille, and V. Kalogera. Gravity spy: integrating advanced ligo detector characterization, machine learning, and citizen science. *Classical and Quantum Gravity*, 34(6):064003, 2017.

[137] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization, 2016.