Classical and Quantum Gravity

PAPER • **OPEN ACCESS**

Generalised gravitational wave burst generation with generative adversarial networks

To cite this article: J McGinn *et al* 2021 *Class. Quantum Grav.* **38** 155005

OPEN ACCESS
IOP Publishing
Classical and Quantum Gravity
Class. Quantum Grav. **38** (2021) 155005 (22pp)
https://doi.org/10.1088/1361-6382/ac09cc

Generalised gravitational wave burst generation with generative adversarial networks

J McGinn*, C Messenger, M J Williams and I S Heng

SUPA, School of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, United Kingdom

E-mail: Jordan.mcginn@glasgow.ac.uk

Received 4 March 2021, revised 25 May 2021
Accepted for publication 9 June 2021
Published 30 June 2021

Abstract

We introduce the use of conditional generative adversarial networks (CGANs) for generalised gravitational wave (GW) burst generation in the time domain. Generative adversarial networks are generative machine learning models that produce new data based on the features of the training data set. We condition the network on five classes of time-series signals that are often used to characterise GW burst searches: sine-Gaussian, ringdown, white noise burst, Gaussian pulse and binary black hole merger. We show that the model can replicate the features of these standard signal classes and, in addition, produce generalised burst signals through interpolation and class mixing. We also present an example application where a convolutional neural network (CNN) classifier is trained on burst signals generated by our CGAN. We show that a CNN classifier trained only on the standard five signal classes has a poorer detection efficiency than a CNN classifier trained on a population of generalised burst signals drawn from the combined signal class space.

Keywords: gravitational waves, machine learning, generative adversarial networks, gravitational wave bursts

(Some figures may appear in colour only in the online journal)

1. Introduction

Gravitational wave (GW) astronomy is now an established field that began with the first detection of a binary black hole (BBH) merger [1] in September 2015. Following this, the first and

*Author to whom any correspondence should be addressed.

second observation runs (O1 and O2) of advanced LIGO and advanced Virgo [2–5] reported several more compact binary coalescence (CBC) mergers [6–8]. On 17th August 2017 a binary neutron star merger was observed alongside its electromagnetic counterpart for the first time, giving rise to multi-messenger GW astronomy [9]. The most recent search for CBC, 03a, took place between 1st April 2019 and 1st October 2019 with 39 candidate events reported [10].

With these successes and continued upgrades to the detectors [11, 12], further detections of CBCs are expected to be common place in future advanced detector observation runs. Another group of GW signals that has thus far been undetected is GW 'bursts'. GW bursts are classed as transient signals of typically short duration (<1 s) whose waveforms are not accurately modelled or are complex to reproduce. Astrophysical sources for such transients include: core collapse supernova [13], pulsar glitches [14], neutron star post-mergers [15] and other as-yet unexplained astrophysical phenomena.

GW searches for modelled signals use a process called matched-filtering, [16–18], where a large template bank of possible GW waveforms are compared to the detector outputs. For GW bursts that remain unmodelled; there are no templates available and so matched-filtering is unsuitable for the detection of these signals. Instead, detection algorithms like coherent WaveBurst [19] distinguish the signal from detector noise by looking for excess power contained in the time-frequency domain and rely on the astrophysical burst waveform appearing in multiple detectors at similar times. This is only possible if the detector noise is well characterised and the candidate signal can be differentiated from systematic or environmental glitches. The presence of environmental glitches is often associated with transient signals detected by various sensors within the LIGO and Virgo detectors. The detectors record data from many auxiliary channels that are used to monitor instrument and environmental backgrounds. This allows some of the glitches to be catalogued and removed from the GW data, thereby improving data quality. See [20, 21] for a more detailed explanation of transient noise and detector characterisation.

GW burst detection algorithms [19, 22, 23] are tested and tuned using modelled waveforms that have easy to define parameters and share characteristics of real bursts that aim to simulate a GW passing between detectors. Such waveforms include sine-Gaussians: a Gaussian modulated sine wave that is characterised by its central frequency and decay parameter. Band limited white noise bursts: white noise that is contained within a certain frequency range. Ringdowns: which mimic the damped oscillations after a CBC merger. A Gaussian pulse: a short exponential increase then decrease in amplitude and a BBH inspiral. With the expectation that there will be many more GW detections in the future, there is a growing need for fast and efficient GW analysis methods to match the rising number of detections. While still in its infancy, the application of machine learning (ML) to GW analyses has already shown great potential in areas of detection [24–26], where these techniques have matched the sensitivity of matched filtering for advanced LIGO and advanced Virgo GW searches. Similarly, for unmodelled burst searches the flexibility of ML algorithms has been shown to be a natural and sensitive approach to detection [27, 28]. Progress has also been made in identifying and classifying detector noise transients or 'glitches' [29–32] as well as glitch classification using a combination of ML and citizen science [33]. In Bayesian parameter estimation [34–36] where ML techniques can recover parameters of a GW signal significantly faster than standard methods. Long duration signals like continuous GW require long observing times and therefore have large amounts of data needing to be processed. Current ML approaches [37–39] are particularly well suited to dealing with this as once trained, searches can be performed quickly.

In this work, we aim to explore the use of ML to generate and interpret unmodelled GW burst waveforms. Using the generative ML model, generative adversarial networks (GANs),

we train on five classes of waveforms in the time domain. Working on the assumption that GANs construct smooth high dimensional vector spaces between their input and output, we can then explore the space between the five classes to construct new hybrid waveforms. As all the computationally expensive processes occur during training, once trained, the model is able to generate waveforms in fractions of a second and produce waveforms that are difficult to generate with current techniques. These new varieties of waveforms can then be used to evaluate detection algorithms, gain new insight into sources of GW bursts and allow us to better train our algorithms on a broader range of possible signals and therefore enhance our detection ability.

This paper is organised as follows. In section 2 we introduce the basic ideas of ML and discuss the choice of algorithm we used. In section 3 we describe the training data and the details of the model. We present the results of the GAN in section 4 and show how unmodeled signals can be produced by interpolating and sampling within latent and class spaces. In section 5 we show that a convolutional neural network (CNN) classifier can be trained to distinguish between sets of our GAN generated waveforms from noise only cases. We conclude with a summary of the presented work in section 6.

2. Generative adversarial networks

ML algorithms aim to learn apparent relationships held within given data or 'training data' in order to make accurate predictions without the need for additional programing. For a general introduction to ML see [40]. A subset of deep learning that has seen fruitful development in recent years are GANs [41]. These unsupervised algorithms learn patterns in a given training dataset using an adversarial process. The generations from GANs are currently state-of-the-art in fields such as high quality image fidelity [42, 43], text-to-image translation [44], and video prediction [45] as well as time-series generations [46]. GANs train two competing neural networks, consisting of a discriminator network that is set up to distinguish between real data (those which come from the training set) and fake data which are synthetically generated from the generator network. The generator model performs a mapping from a fixed length vector $\mathbf{z}$ to its representation of the data. The input vector is drawn randomly from a Gaussian distribution, which is referred to as a latent space comprised of latent variables. The latent space is a compressed representation of a data distribution which the generator applies meaning to during training. Sampling points in this space allows the generator to produce a variety of different generations, with different points corresponding to different features in the generations. The discriminator maps its input $\mathbf{x}$ to a probability that the input comes from either the training (real) data or generator (fake). During training, the discriminator and generator are updated using batches of data. Random latent vectors are given to the generator to produce a batch of fake samples and an equal batch of real samples is taken from the training data. The discriminator makes predictions on the real and fake samples and the model is updated through minimising the binary cross-entropy function [40]

$$L = y\ \log(\hat{y}) + (1 - y)\log(1 - \hat{y}), \tag{1}$$

where $\hat{y}$ is the network prediction and y is the true output. While training the discriminator, D, on real data, we set $y = 1$ and $\hat{y} = D(\mathbf{x})$ which from equation (1) gives $L(D(\mathbf{x}), 1) = \log(D(\mathbf{x}))$. While training on fake data produced by the generator, G, $y = 0$ and $\hat{y} = D(G(\mathbf{z}))$ and so, $L(D(G(\mathbf{z})), 0) = \log(1 - (D(G(\mathbf{z}))))$. Since the objective of the discriminator is to correctly classify fake and real data these equations should be maximised, while the goal of the generator should be to minimize these equations. This gives what is know as the GAN value function as

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log\, D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))], \tag{2}$$

where $p_{\text{data}}(\mathbf{x})$ is the distribution of real data and $p_{\text{z}}(\mathbf{z})$ is the latent distribution.

2.1. Training stages

Training a GAN involves updating both the discriminator and generator in stages. First, the discriminator is updated using real instances from the training set. We set the true label y equal to 1 and calculate the loss with respect to the predictions $\hat{y}$ via equation (1). Stochastic gradient descent is used to maximise the discriminators loss on real data, $L_D(\text{real}) = \log(\hat{y})$. The discriminator is then trained on fake instances taken from the generator where we set $y = 0$ and maximise $L_D(\text{fake}) = \log(1 - \hat{y})$.

To train the generator, we use a composite model of the generator and discriminator and allow the gradients to flow through this entire model. Following on from what was described before, to train the generator we set $y = 0$ and minimise $L_G(\text{fake}) = \log(1 - \hat{y})$. During early stages of training the generator produces poor generations and so D can easily determine them as fake. This leads L_G to tend to 0 and we encounter the *vanishing gradient problem*, where the gradients become so small that the weights can no longer be updated. A solution to this problem involves changing the generator loss to maximise $L_G(\text{fake}) = \log(\hat{y})$ or equivalently continue to minimise $L_{\text{G}}(\text{fake}) = \log(1 - \hat{y})$ and simply switch the y label to 1. This tweak to the generator loss is called non-saturating generator loss and was reported in the original GAN paper [41]. It was also shown in that paper that if the generator and discriminator can no longer improve, then the discriminator can no longer distinguish between real and fake samples and will continue to output $D(x) = \frac{1}{2}$ in this optimal training scenario.

2.2. Conditional GANs

To gain more control over what a GAN is able to generate, a conditional variant of GANs named conditional generative adversarial networks (CGANs) [47] was introduced by feeding in extra information into the generator and discriminator such as a class label or attribute label, $\mathbf{c}$. This simple addition has been shown to work well in practice, for instance in image-to-image translation [48]. We use one-hot encoding to define the classes, that is, each class resides at the corner points of a five-dimensional hyper-cube. For example $\mathbf{c} = [0, 1, 0, 0, 0]$ represents the ringdown signal class. The training data and labels are drawn from a joint distribution $p_{\text{data}}(\mathbf{x}, \mathbf{c})$, whereas when generating fake data we sample from $\mathbf{c}$ and $p_{\mathbf{z}}(\mathbf{z})$ independently. Equation (2) is modified to include the class labels

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log\, D(\mathbf{x}|\mathbf{c})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z}|\mathbf{c})))]. \tag{3}$$

Figure 1 shows the differences in inputs and output of a GAN compared with a CGAN. We will be using a conditional GAN for this study.

2.3. Convolutional neural networks

CNNs are designed to work with grid-like input structures that exhibit strong local spatial dependencies. Although most work with CNNs involves image-based data, they can be applied to other spatially adjacent data types such as time-series [49] and text items [50]. CNNs are defined by the use of a convolution operation, a mathematical operation that expresses the amount overlap between the data. Much like traditional neural networks, the convolution operation in this context involves multiplying the input by an array of weights, called a filter or a kernel, which is typically smaller in size than the input. The convolution is applied by shifting

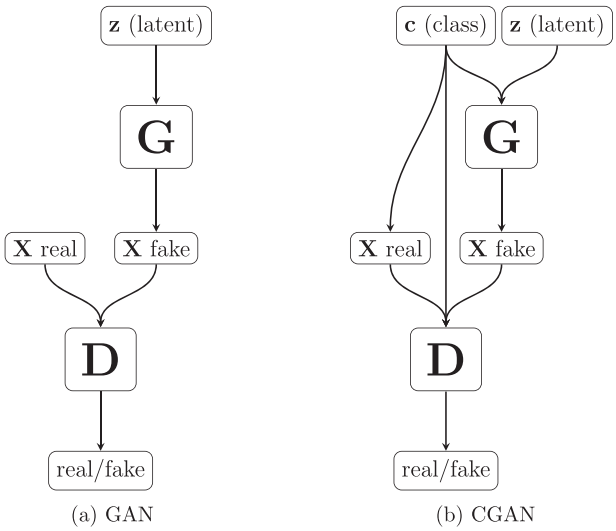


Figure 1. Comparison of the original GAN method and the CGAN method. **G** and **D** denote the generator and discriminator neural networks respectively while **X** real and **X** fake represent samples drawn from the training set and the generated set. For CGANs the training data requires a label denoting its class that is also fed to the generator which then learns to generate waveforms based on the input label.

the kernel over the input, drawing out spatially important features between the two. The distance by which the grid is shifted is known as the stride and increasing it reduces the dimension of the output in a process know as downsampling. Alternatively, upsampling the inputs can be achieved using a transposed convolution [51]. The output of the convolutional layer is then passed to an activation function and through the next layer. For deep neural networks, techniques like BatchNormalisation [52] which standardise the inputs to a layer and SpatialDropout [53] which sever connections between neurons can both help to stabilise learning.

3. Training data and architecture

We propose a signal generation scheme using a CGAN trained on burst-like waveforms which we call `McGANn`[1]. `McGANn` is trained on five signal classes which are used to characterise the sensitivity of GW burst searches (see for example [54]).

- **Sine-Gaussian**: $h_{\rm SG}(t) = A\,\exp\left[-(t-t_0)^2/\tau^2\right]\sin(2\pi f_0(t-t_0)+\phi)$, a sine wave with a Gaussian envelope characterised by a central frequency f_0, amplitude A, time of arrival t_0 and phase ϕ which is uniformly sampled between $[0, 2\pi]$.
- **Ringdown**: $h_{\rm RD}(t) = A\,\exp\left[-(t-t_0)/\tau\right]\sin(2\pi f_0(t-t_0)+\phi)$, with frequency f_0 and duration τ, amplitude A, time of arrival t_0 and phase ϕ which is uniformly sampled between $[0, 2\pi]$.
- **White noise bursts**: $h_{\rm WN}(t_j) = Ag_j\,\exp\left[-(t-t_0)^2/\tau^2\right]$ where g_j are drawn from a zero mean unit variance Gaussian distribution with a Gaussian envelope of duration τ.

[1] https://github.com/jmcginn/McGANn

Table 1. The parameters used in generating the training data. Each parameter is drawn uniformly in the below ranges.

Waveform	Central frequency (Hz)	Decay (s)	Central epoch (s)	Component mass ($M_\odot$)
Sine-Gaussian	70–250	0.004–0.03	0.4–0.6	—
Ringdown	70–250	0.004–0.03	0.4–0.6	—
White noise burst	70–250	0.004–0.03	0.4–0.6	—
Gaussian pulse	—	0.004–0.03	0.4–0.6	—
BBH	—	—	—	30–70

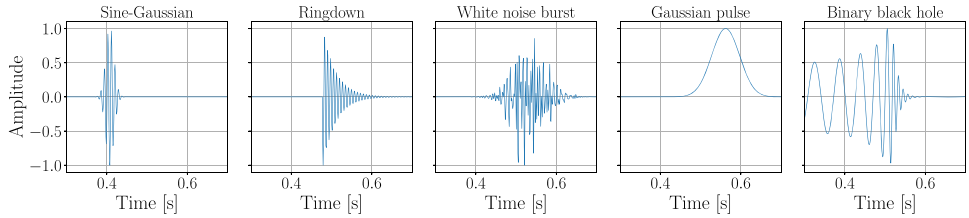


Figure 2. Examples of the five different waveforms that were used in training the GAN for this study. Values of the parameters were selected randomly from uniform distributions from table 1.

- **Gaussian pulse**: $h_{\mathrm{GP}}(t) = \exp(-t^2/\tau^2)$ with duration parameter τ.
- **BBH**: simulated using the IMRPhenomD waveform [55] routine from LALSuite [56] which models the inspiral, merger, and ringdown of a BBH waveform. The component masses lie in the range of [30, 70] $M_\odot$ with zero spins and we fix $m_1 > m_2$. The mass distribution is approximated by a power law with index of 1.6 [57]. The inclinations are drawn such that the cosine of the angles lies uniformly in the range $[-1, 1]$ and we only use the plus polarisation.

The location of the peak amplitude of the waveforms (corresponding to the mid-points of all but the ringdown and BBH classes) are randomly drawn from a uniform distribution to be within $[0.4, 0.6]$ s from the start of the 1 s time interval and all training waveforms are sampled at 1024 Hz. While in principle GANs are able to generate higher dimensional spaces, we choose 1 s segments sampled at 1024 Hz to reduce development/training time. The advantages of going to a higher sampling rate would be to model higher-frequency/broader-band signals and/or longer duration signals. The disadvantages would be increased training time, possible encroachment on the memory limitations of the GPU, and the potential requirement for further development of the GAN architecture. The parameter prior ranges are defined in table 1 and a sample of training waveforms are also shown in figure 2. These priors are based on LIGO–VIRGO injections with quality factors, Q, calculated from choosing frequency as the limiting factor. The ranges of frequency chosen are based on the parameters of the standard injections/simulations in used in [58]. The other parameters can then be evaluated based on the Q relation. For the BBH waveforms we restrict the mass to be above 30 as lower mass systems would produce longer duration waveforms than typical bursts. All training data is rescaled such that their amplitudes peak at one.

With the exception of the BBH waveforms, the signal classes described above are analytic proxy waveforms of GW signals expected from various burst GW sources. The classes of signals and their suitability considered in this work are also described in [54].

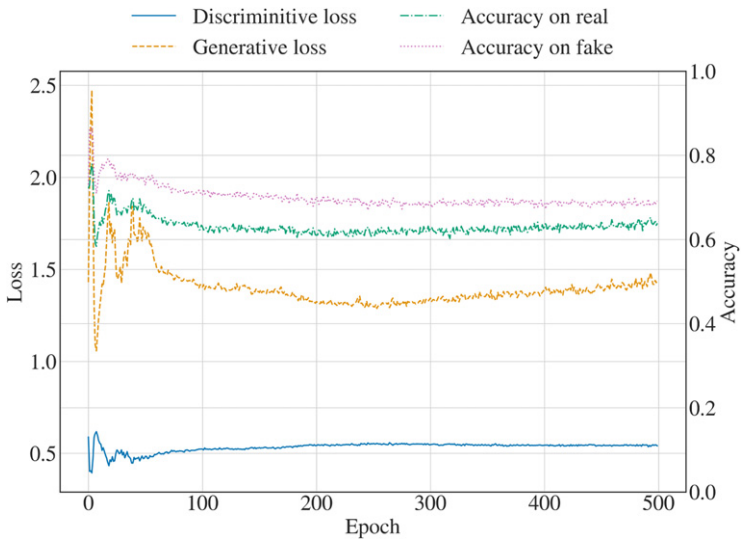


Figure 3. Plot of the discriminator and generator loss and accuracy as a function of epochs. Early in training the losses oscillate as both models attempt to find their equilibrium, after which, both losses vary around a point which signifies stable training. Accuracies on the real and fake data are similar, showing that neither model is stronger than the other.

3.1. Architecture details

Neural networks and subsequently GANs have multiple parameters a developer can tune when designing the model and these are referred to as hyperparameters. The final network design used in this work was developed through trial and error and the initial designs were influenced by the available literature. We found that the GAN performed better with both networks having the same number of layers and neurons, which encourages even competition between the generator and discriminator. After tuning the multiple hyperparameters (see table A1), the GAN was trained on 10^5 signals drawn from a categorical distribution with equal probabilities for each class of sine-Gaussian, ringdown, white noise bursts, Gaussian pulse and BBHs.

The design of the networks is influenced by [59] in which they use a deep convolutional generative adversarial network architecture. The generator model is fully convolutional, upsampled using strided transposed convolutions with BatchNormalisation in the first layer and ReLU activations throughout with the exception of a linear activation for the output layer. The use of a linear activation guarantees the output can have negative and positive outputs. Each transposed convolutional layer uses a kernel size of 18 and stride of 2. The discriminator network mirrors that of the generator without batch normalization, using LeakyReLU activations, SpatialDropout, and a two-stride convolution for downsampling. The discriminator output is a single node with sigmoid activation that can be interpreted as a probability of the signal being real and both models are trained with binary cross entropy equation (1). The full architecture description can be seen in table A1.

As GANs are trained by updating one model at the expense of the other, they can be hard to train. GANs attempting to replicate complicated structures that do not have the necessary architecture either struggle to produce results at all or fall into the common failure mode known as

mode collapse; where the generator produces a small variety of samples or simply memorises the training set. The goal of GAN training is to find an equilibrium between the two models, if this cannot be found then it is said that the GAN has failed to converge. One way to diagnose problems, such as mode collapse, when training GANs is to keep track of the loss and accuracy over time. These loss plots can help to identify common failure modes or to check if the GAN has indeed converged. Accuracy is another metric that may be used to monitor convergence and is defined as the number of correct predictions made divided by total number of predictions. There is currently no notion of early stopping in GANs, instead, training is halted after convergence and by visually inspecting the generations. Figure 3 shows the loss and accuracy during the training process for the GAN used in this work.

All models were designed with the Python Keras library [60] and TensorFlow [61] and trained on a GeForce RTX 2080 Ti GPU. We train the network for 500 epochs which takes $\mathcal{O}(10)$ hours and save the model at each epoch. We choose an appropriate model by visually inspecting the generations at a point of convergence on the loss plot.

4. Results

Given a 100-dimensional vector drawn from a normally distributed latent space and a one-hot encoded class label, the GAN is able to generate burst-like waveforms generalised from the training set. We set out by describing the quality of the generated waveforms and how they compare to the training set. We then explore the structure of the latent and class spaces by interpolating between points in these spaces. We test three methods of sampling from the class space that can be used to generate new signals composed of weighted elements of each training class.

4.1. Known class signal generation

In figure 4 we show conditional signal generations using our generator network. We can see the generations capture the main aspects of each signal class and appear as though they could have plausibly come from the training set. We can also see that the model has learned the overall characteristics of the five training classes and is able to disentangle each class and associate them with the conditional input. Additionally, as the latent variable changes we see indirect evidence of variation within the parameter space for a given class. For instance figure 4 shows how signals vary in frequency, central epoch, decay timescale, and phase. The GANs ability to generate a variety of signals for various latent space input indicates stable training and no mode collapse.

4.2. Interpolation within the latent space

We have shown that the generator produces quality signals and that the model responds well to randomly sampled Gaussian latent vectors. We now assume that during training the generator has learned a mapping from a Gaussian latent space to the signal space and that this mapping is a smooth function of the underlying latent space. To verify this, we fix the class vector input and linearly interpolate between two randomly chosen points in the latent space (different for each class). In figure 5 we show the generated waveforms, with the class vectors held constant along each row. We can see that each plot shows plausible waveforms suggesting that the generator has constructed a smooth traversable, space. We note that the relationship between the latent space location and the physical signal parameters is intractable, and hence the initial and final latent space locations (moving left to right in figure 5) simply represent random possible signals learned from the training set prior. During training the network should have learned how to

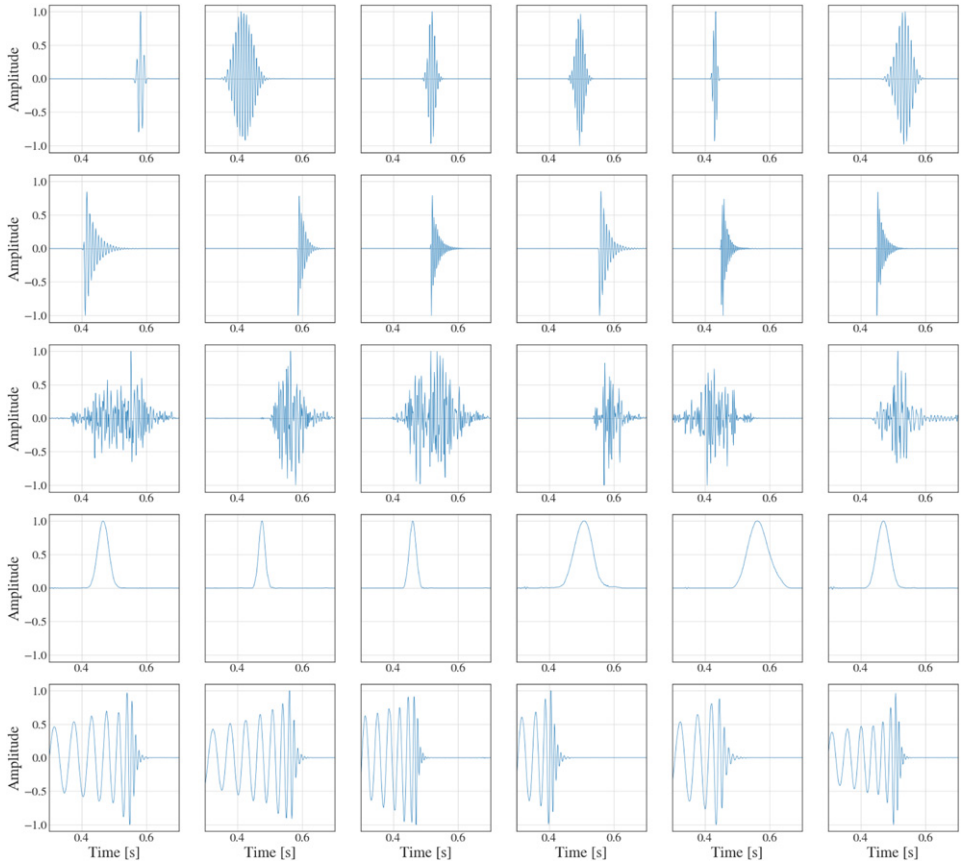


Figure 4. GAN Generated waveforms plotted as a function of time. The latent space inputs for each panel are randomised and each row is assigned one of the five class vectors. By row: sine-Gaussian, ringdown, white noise burst, Gaussian pulse, BBH merger. For ease of viewing, the x-axis for all panels spans the mid 50% of the output range.

smoothly represent the underlying features of a signal as a function of latent space location. For example, the linearly interpolated transition through the latent space for the Gaussian pulse signal shows a shift to an earlier epoch and a larger decay timescale. In contrast, the transition for the ringdown signal appears to pass through a localised region of the latent space consistent with a higher central frequency.

4.2.1. Interpolation between pairs of classes. While the GAN is trained on distinct one-hot encoded classes, we may test arbitrary points in the five-dimensional class space to produce indistinct or hybrid waveforms. In order to explore the class space, in figure 6 we show results where the latent vector is held constant, but we instead linearly interpolate within the one-hot encoded class space between pairs of well-defined training class locations. In this scenario, we highlight that the GAN has not yet probed this intermediate class space during its training and therefore we are reliant on the generator having learned any underlying class space relationships between the five training classes. The results show that for each case that the generated signals show distinct characteristics of the respective class pairs at most stages of the transition. We note that transitions in some cases appear to be rather abrupt, e.g., between the Gaussian

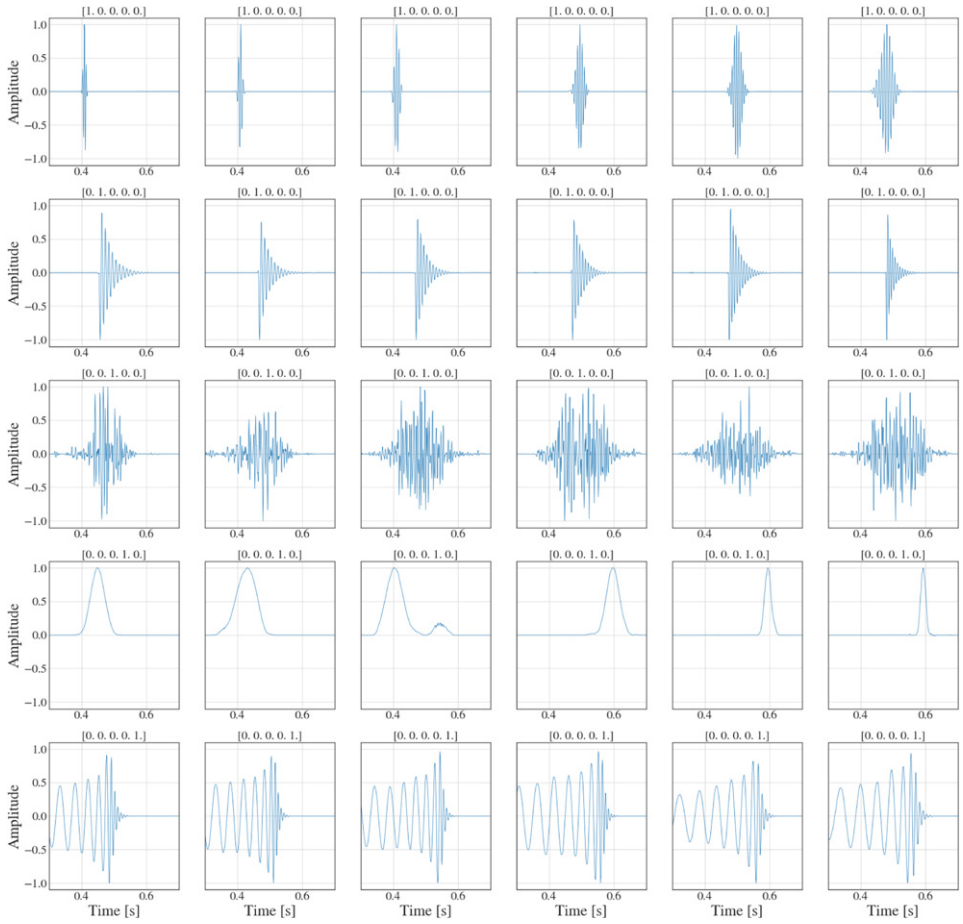


Figure 5. GAN generated interpolated waveforms plotted as a function of time showing latent space interpolations. For each interpolation two different points were randomly chosen in the latent space and represent the first and last panels in each row. The panels between represent signals generated using linearly interpolated vectors between these two points. Each row keeps its class vector constant throughout the latent space interpolation. By row: sine-Gaussian, ringdown, white noise burst, Gaussian pulse, BBH merger. For ease of viewing, the x-axis for all panels spans the mid 50% of the output range.

pulse and the BBH, and that this feature, while not uncommon, is a strong function of the random latent space location.

4.3. General points within the class space

We have shown that the GANs latent space and class space have a structure that can be navigated via interpolation between pairs of locations within each respective space. Taking a step further, we can sample from the class space in novel ways to create new inputs for the generator. These new points are categorised by the method used to sample from the class space. The methods we use are divided into the following:

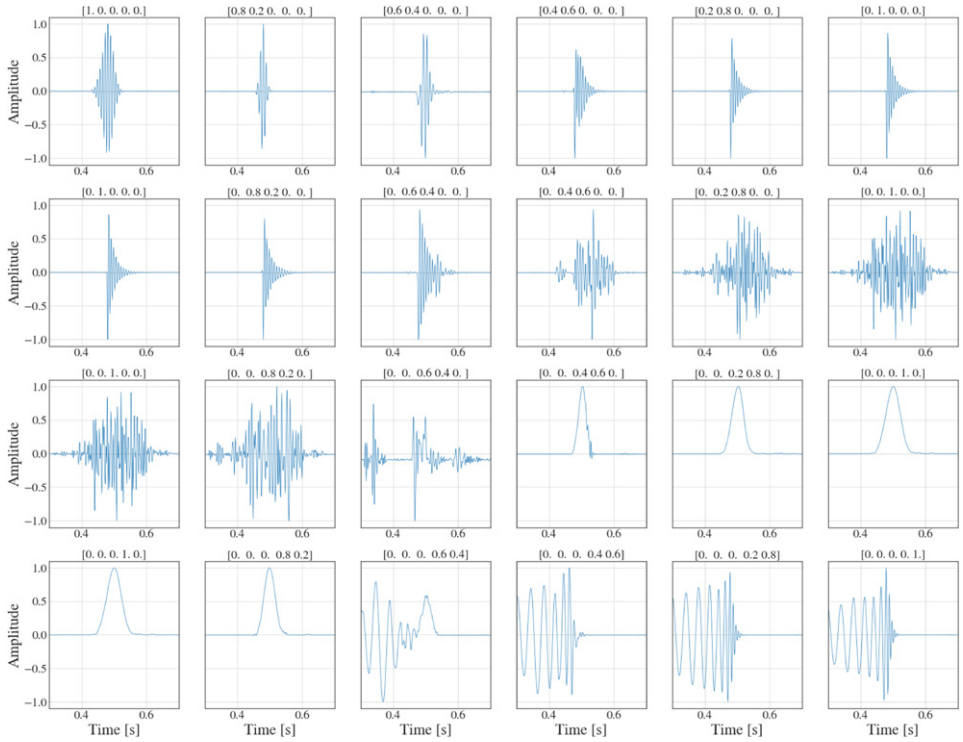


Figure 6. GAN generated class interpolated waveforms as a function of time showing class space interpolations. A single latent space vector is used for all generations and is chosen randomly in the latent space. Each row shows generations using linearly interpolated classes as inputs to the generator. By row top to bottom: sine-Gaussian to ringdown, ringdown to white noise burst, white noise burst to Gaussian pulse, Gaussian pulse to BBH.

- **Vertex**: points that lie at the corners of the five-dimensional class space. These class space locations are equivalent to the training set locations and are our closest generated representation of the training set.
- **Simplex**: this class vector we define as uniformly sampled points on a simplex, which is a generalization of a triangle in k-dimensions. We sample uniformly on the $k = 4$ simplex that is embedded in the five-dimensional class hyper-cube. In practice we use the equivalent of sampling points from a $k = 4$ Dirichlet distribution. It is useful to think of the simplex as the hyper-plane that intersects all five training classes. It is a subspace of the uniform method.
- **Uniform**: each of the entries in the class vector is sampled from a uniform distribution U[0, 1]. This is equivalent to sampling uniformly within the fine-dimensional one-hot encoding hyper-cube.

The vertex points are the most straightforward where one element of the class vector contains one and the other elements are zero. These points are equivalent to the class vectors that the GAN is trained on e.g., $\mathbf{c} = [1, 0, 0, 0, 0]$ would correspond to a sine-Gaussian generation. Uniform class vectors with each element sampled from a uniform distribution are equivalent to a random draw from a five-dimensional hyper-cube. Uniformly sampling generates class

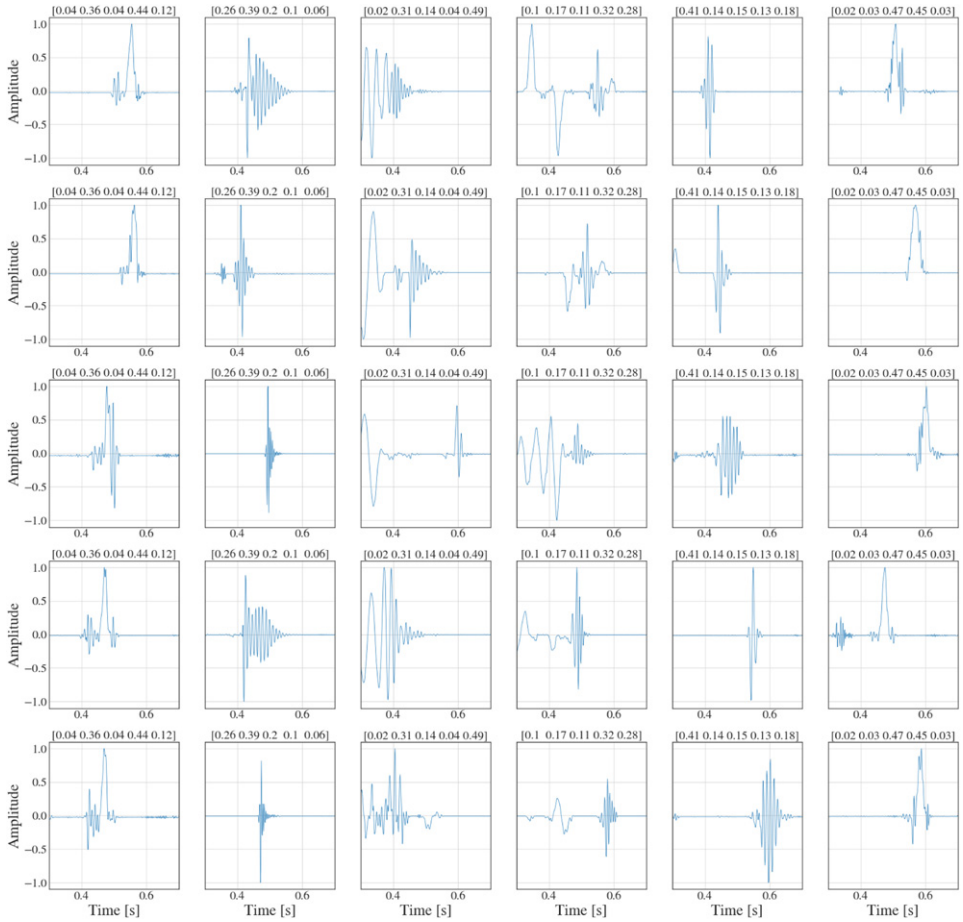


Figure 7. GAN generations where the class vectors are sampled from the four-dimensional plane (simplex) intersecting all training classes. Latent space locations for all signals are drawn randomly from a 100-dimensional Gaussian distribution and the signals are then re-scaled such that they have maximum absolute amplitude at unity. The class label for each generation is shown above each panel.

space locations up to a maximum distance of unity from the closest class e.g. $[0, 0, 0, 0, 0]$ is of distance one away from all classes. For simplex class vectors, we sample from the simplest hyper-surface that intersects all the classes and has a symmetry such that no training class location (any vertex) is favoured over any other. For our five-dimensional case this corresponds to a four-simplex manifold. Sampling from the simplex can be seen as sampling from the simplest space that spans between the training classes.

In figure 7 we show generations conditioned on class vectors drawn randomly from the four-simplex. There are large variations in the signals with some having characteristics strongly resembling the training classes, although this can be partially explained through the random draws from the simplex as there is finite probability that one class entry will dominate over the others (i.e., the class space location is close to a vertex). For instance the generations that look more like sine-Gaussians than hybrid waveforms generally have a larger value placed in the first class space element than others. Similarly figure 8 shows generations conditioned on class

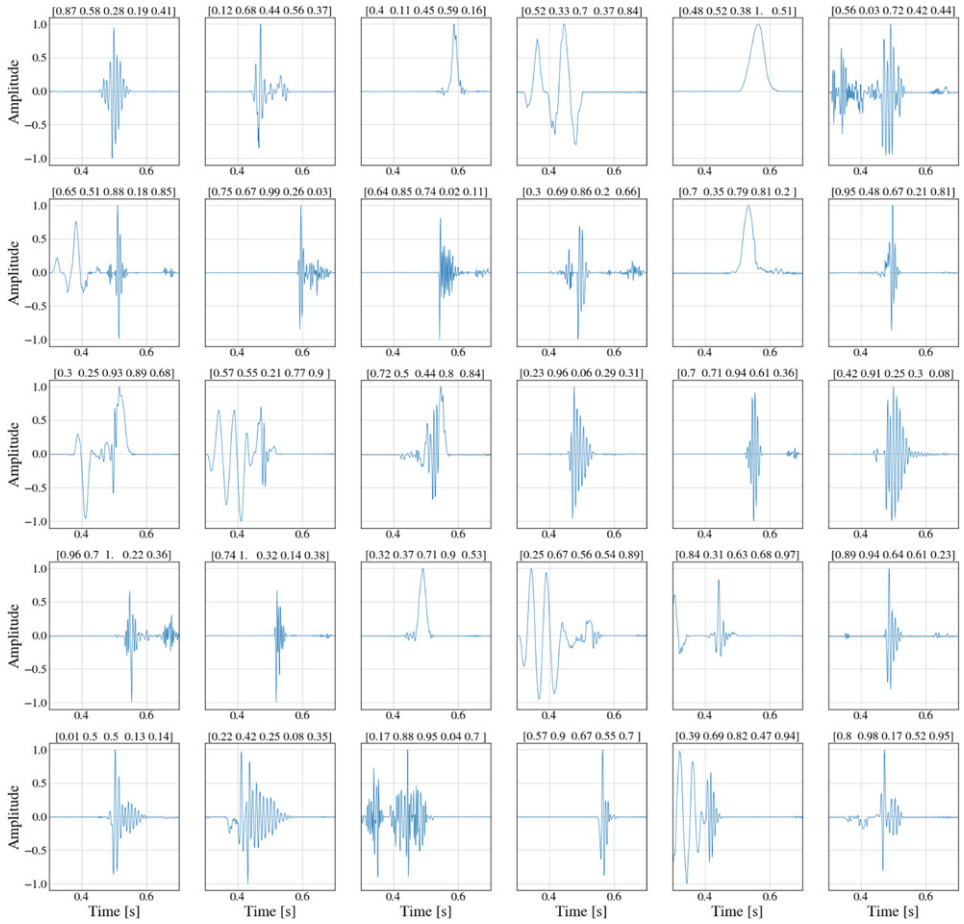


Figure 8. GAN generations where the class vectors are sampled uniformly in the hyper-cube class space. Latent space locations for all signals are drawn randomly from a 100-dimensional Gaussian distribution and the signals are then re-scaled such that they have maximum absolute amplitude at unity.

vectors drawn uniformly in the unit hyper-cube. These types of generations tend to exhibit more noise and some tend to be generated with very low amplitude prior to being rescaled to have a maximum amplitude of unity. Both methods of generating hybrid waveforms, however, do produce signals that appear to share characteristics with the training set but still are distinct in signal morphology. Upon inspection of a larger collection of waveform generations from both methods, we do see a tendency for the uniform hyper-cube approach to generate a wider variety of hybrid waveforms that are more visually distinct from the training set. This is to be expected given that the simplex class space is a subset of the hyper-cube and does not explore regions of the class space as far from the training set vertices.

5. CNN burst classifier

In this section, we develop a basic search analysis using a CNN in order to compare the sensitivity of such a search using different GAN generated waveforms in additive noise. We train

a CNN to perform simple classification and to distinguish between two classes: signals in additive Gaussian noise and Gaussian noise only. We are primarily interested in the relative sensitivity as a function of the types of waveforms used for training the network. We are also interested in how these differently trained networks perform when applied to data from waveform generations not used in the training process.

5.1. Noisy datasets

We use three classes of waveforms: vertex, uniform, and simplex cases generated using our GAN method. We then construct noisy time-series data from each waveform representing measurements from the two LIGO detector sites, Hanford (H1) and Livingston (L1). For each training set we generate 2×10^5 signals and apply antenna responses and sky location dependent relative time delays using routines provided within LALsuite [56]. The generated waveforms are used to represent the plus-polarisation component of signal only and the polarisation angles are drawn uniformly in the range $[0, 2\pi]$ and sky positions are sampled isotropically. Time delays between detectors are computed relative to the Earth's centre. All of the training data used is whitened using the advanced LIGO design sensitivity power spectral density (PSD) [62, 63], such that there is equal noise power at each frequency. To generate signals at a chosen optimal (network) signal-to-noise ratio (SNR) $\rho_{\rm opt}$, we first compute $\rho_{\rm opt}$ for each generated waveform, defined in [64] as

$$\rho^2_{\rm opt} = \sum_{i={\rm H1,L1}} 4 \int_{f_{\rm min}}^{f_{\rm max}} \frac{|\tilde{h}^{(i)}(f)|^2}{S_n^{(i)}(f)}\, {\rm d}f, \tag{4}$$

where $\tilde{h}^{(i)}(f)$ and $S_n^{(i)}(f)$ are the Fourier transform of the GW strain in the frequency domain and the detector noise PSD for the ith detector respectively. We then sample the target optimal SNR uniformly on U [1, 16] and rescale the waveform amplitudes in order to achieve the desired value.

Each 1 s duration time-series input to the CNN is represented by a one-dimensional 1024 sample vector with two channels representing each detector. Example time-series from each detector for a single signal are shown in figure 9. The network is trained to be able to identify whether or not a measurement contains a signal and therefore 50% of the training data have time-series containing signals and 50% have only noise. We randomly divide the data into the three standard sets (training, validation, and test data) where 40% is used for training, 10% used for validation, and 50% is used for testing in order to achieve suitably low false-alarm probability of 10^{-3}. For the uniform and simplex datasets samples are drawn uniformly from their respective spaces. For the vertex dataset the five different vertex locations in class space are sampled with equal probability.

5.2. CNN architecture

In this approach the inputs to the CNN are 1024 sample time-series (with two channels representing each detector output) which are passed through a series of four convolutional layers, onto two fully connected or 'dense' layers and finally to a single output neuron which represents the probability that a signal is present within the noise. We used dropout in the final dense layer and used a selection of different activation functions including the swish activation [65] which improved overall performance, and a sigmoid activation for the output layer. We used binary cross-entropy equation (1) as the loss function and Adam [66] as an optimizer with learning rate set to 10^{-3}. In total we train three separate CNNs on the vertex, uniform

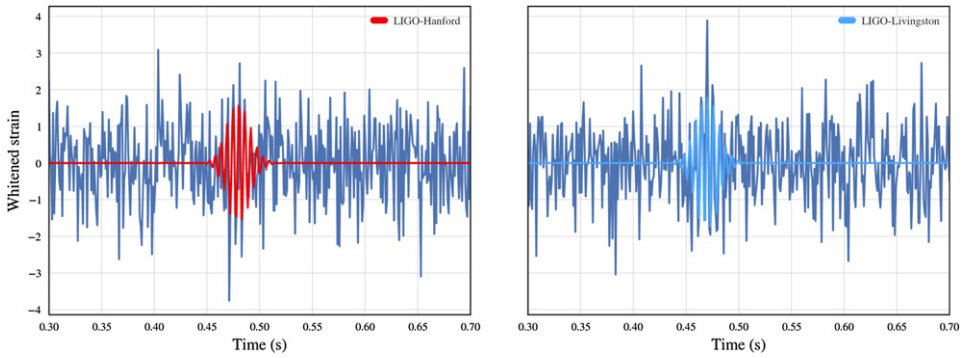


Figure 9. Example of CNN training data showing a whitened noisy (dark blue) and noise-free (red, light blue) sine-Gaussian time-series as seen by Hanford (left) and Livingston (right) detectors. This signal has network SNR $= 8$.

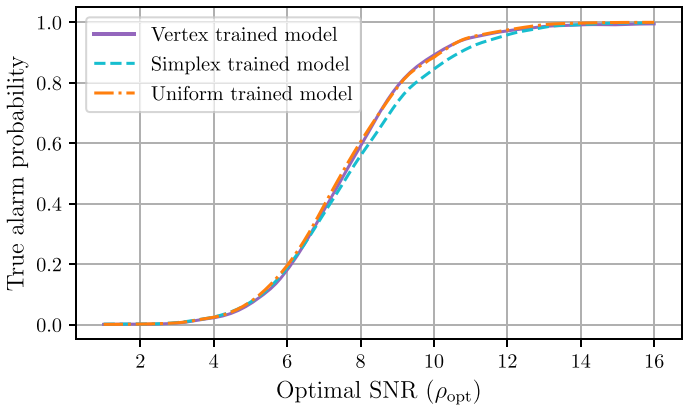
and simplex datasets respectively. In each case the networks share the same architecture and hyperparameters which are defined in table A2.

5.3. CNN results

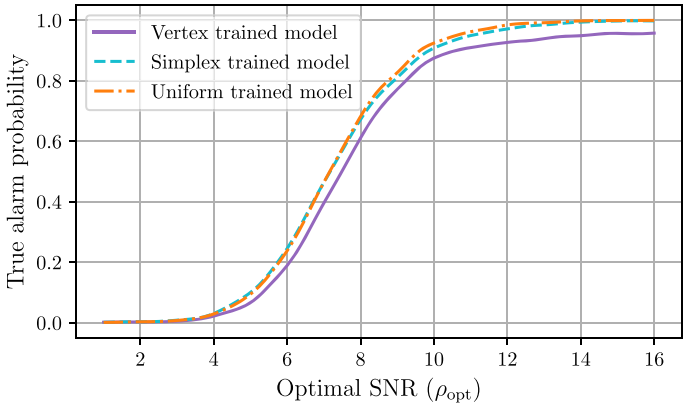
We now compare the CNN results between the datasets by first training three CNNs on the vertex, simplex, and uniform datasets and then using these models to make predictions on the other testing data that is unseen during the network training process. We compare results for the different permutations using efficiency curves as shown in figure 10. The efficiency curves are computed by fixing the false alarm probability (FAP) and plotting the true alarm probability (TAP) against optimal SNR for a direct comparison between CNN approaches. FAP is the fraction of noise-only samples that are incorrectly identified as signals whereas TAP is the fraction of signals correctly identified as signals. However, since the distribution of SNRs is continuous, we cannot compute the TAP using the method described previously in [24]. Instead, we compute the TAP at specific SNRs by considering all of the signals weighted by a Gaussian window with standard deviation 0.3 centred on each SNR. A classifier model performs better than another when it achieves a higher TAP for a fixed FAP at a given optimal SNR. In figure 10, the top panel presents results for the three different networks tested on the vertex data and shows that each model confidently detects all the signals with SNRs > 13. At lower SNRs the vertex and uniform datasets perform similarly, however, at $\rho_{\text{opt}} \sim 10$ the simplex trained model has slightly worse performance, dropping in TAP by a few percent.

We would expect that when the vertex trained model is tested on vertex data that it outperforms the alternatively trained networks. This is because the vertex data is a subset of each of the other two datasets and the network is not required to classify any samples unlike those it has trained on. We also expect that all vertex testing signals should be correctly classified at high SNR since the vertex data is a subset of the uniform and simplex training tests. The weaker performance of the simplex trained model could be attributed to the lower density of training signal locations in close proximity to the vertices.

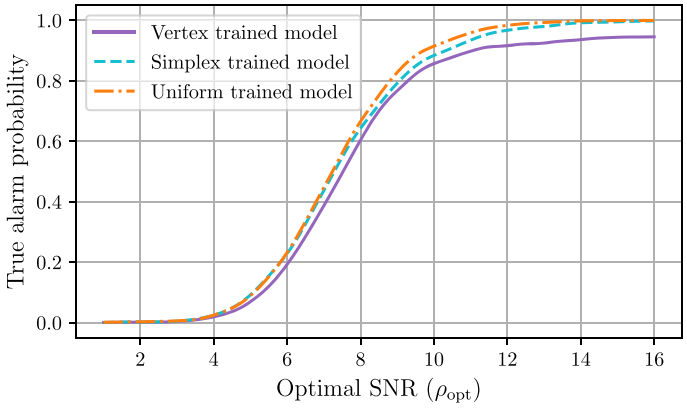
The second panel of figure 10 shows the results of the differently trained CNNs tested on simplex data. As expected the simplex and uniform models detect 100% of the signals at higher SNRs. However, the vertex trained model fails to detect all the simplex signals, achieving only



(a) Tested on vertex



(b) Tested on simplex



(c) Tested on uniform

Figure 10. Efficiency curves comparing the performance of the CNNs. The TAP is plotted as a function of the optimal SNR of the signals for a FAP of 10^{-3}. Each plot shows the performance of a CNN trained on vertex, simplex and uniform datasets tested on vertex (a), simplex (b), uniform (c).

96% TAP at the highest simulated SNR $\rho_{\text{opt}} = 16$. This is explained when we consider that the simplex data is a subset of uniform data while the vertex data is not. It is interesting to note that the simplex and uniform trained models perform identically (within statistical uncertainty). The uniform model has a larger signal parameter space volume and we might expect it to be more susceptible to misidentifying instances of the Gaussian noise model as signals from the uniform dataset.

The final panel of figure 10 tests the models on uniform data and again shows that at high SNRs both simplex and uniform trained models are result in 100% TAP. One might not expect this since the simplex training data is only a subset of the uniform testing data parameter space. The simplex trained CNN in the high SNR limit is able to confidently generalise to be able to identify signals from the uniform testing dataset. This is not the case for the vertex trained model which achieves only a 95% TAP in the high SNR limit. The vertex trained CNN is not able to fully generalise and identify signal from noise for signals within the class space hyper-cube, nor from within the class space simplex hyper-surface. We also note that specifically in the $\rho_{\text{opt}} \sim 10$ region we see marginally more sensitive results for the uniform trained model when applied to the uniform testing data in comparison to the simplex trained model. This is expected since again the simplex data space is a subset of the uniform data space and the uniform trained model will have explicitly learned how to identify signals in regions distant from the simplex hyper-surface. The simplex trained model performs well despite having to extrapolate away from its training space.

The tests discussed above show that the CNN trained on the vertex model only manages full detection when tested on vertex model data. The uniform model performs best in all cases and since it contains signals from the vertex and simplex samples and does not appear to suffer from an increased FAP due to its larger parameter space volume. This suggests that the uniform method of sampling the class space for training or characterising a search algorithm is the most robust and sensitive approach given the intrinsically unknown nature of GW burst signals. Furthermore, since the uniform trained model performs equally as well as the vertex trained model when applied to vertex test data, we can conclude that the inclusion of the unmodelled signals does not negatively affect the model's performance on the modelled signals.

6. Conclusions

In this work we present the potential of GANs for burst GW generation. We have shown that GANs have the ability to generate plausible time-series burst data and present a novel approach to generating unmodelled waveforms. We have shown that our implementation of a CGAN is able to generate five distinct classes of burst like signals through conditional training which can then be utilised for specified signal generations. The CGANs allows us to map the parameter space of each signal class into a common abstract latent space in which common signal characteristics are grouped into smoothly connected regions. We are then able to sample from this space as input to the generator network and produce high fidelity random examples of any of our trained signal classes.

While we have trained our CGAN on five discrete signal classes, each having its own signal parameter space, we have shown that we can subsequently sample from the continuous class space to generate hybrid burst waveforms. This novel aspect of our analysis takes advantage of the learned mapping between individual discrete signal classes. When coupled with the latent space, we are then able to generate hybrid waveforms that span the variation between signal classes and the variation within each class. The resultant hybrid waveforms then represent a

generalised set of potential GW burst waveforms that are vastly different from the limited training set. Such waveforms are in demand in GW astronomy as they allow burst search pipeline developers to test and enhance their detection schemes.

To provide a practical example of the usage of these waveforms, we have concluded our analysis with a simple search for signals in additive Gaussian noise. We have suggested three variations of how to sample from the CGAN signal class space and have trained a basic CNN separately on those data in order to classify whether a signal was present in the noisy data versus only Gaussian noise. The resulting trained networks were then tested on independent datasets from each of the three signal hybrid classes. The resulting efficiency curves compare the detection sensitivities of the CNN as a function of SNR and allow us to conclude that in this simple analysis, training the search using the most general set of hybrid waveforms (our 'uniform' set) provides the most sensitive overall result.

In contrast to typical approaches in signal generation this is the first time a GAN has been used for generating GW burst data. Our approach allows us to explicitly control the mixing of different signal training classes but the variation within the space of signal properties is determined randomly through sampling of the abstract latent space. In the future, as development in GANs and generative ML advances it is expected that we will gain greater control over targeted generation of signal features. It will also be important to extend our models to train on, and generate, longer duration waveforms, higher sampling rates, and to be conditioned on additional classes. One such set of additional classes of interest would be the population of detector 'glitches'. These are typically high-amplitude short-duration events in the output of GW detectors that represent sources of terrestrial detector noise rather than that of astrophysical origin. Using a GAN to model these would provide us with a tool to simulate an unlimited set of glitches which could be used to better understand their origin and guide us towards more effective methods of mitigation and removal from the data stream.

Having the ability to quickly generate new waveforms is essential to test current GW burst detection schemes [19, 22, 23]. They can be used to truly assess their sensitivity to unmodeled sources and identify signal features to which they are susceptible.

Acknowledgments

The authors would like to acknowledge valuable input from Sarah Caudill and Mellisa Lopez and gratefully acknowledge the Science and Technology Facilities Council of the United Kingdom. JM is supported by the Science and Technology Facilities Council Newton-Bhabha ST/R001928/1 and the Gravitational-wave Excellence through Alliance Training (GrEAT) Network with China ST/R002770/1 funds. CM and ISH are supported by the Science and Technology Research Council (Grant No. ST/L000946/1) and acknowledge the European Cooperation in Science and Technology (COST) action CA17137. MJW is supported by the Science and Technology Facilities Council [2285031].

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/jmcginn/McGANn.

Appendix A. List of hyperparameters

See tables A1 and A2

Table A1. The architecture and hyperparameters describing our GAN consisting of discriminator and generator convolution neural networks. The discriminator casts the class input through a fully connected layer such that its dimensions match the signals input which it then concatenates channel-wise. This is then downsampled through four convolutional layers all activated by Leaky ReLU functions and drops half of the connections at the end of each of these layers. The vector is then flattened to one dimension before fully connecting to a single neuron and its output activated by sigmoid to represent the probability the signal came from the training set. The generator concatenates the latent and class input vectors which is fed to a fully connected layer. This layer is then upsampled by four transposed convolutions. Batch normalisation is applied to the output of the first layer and all convolutional layers are activated by ReLU with the exception of the final layer which is linear. Finally, the extra dimension introduced for the convolution is removed.

Discriminator

Operation	Output shape	Kernel size	Stride	Dropout	Activation
Class input	(5)	—	—	0	—
Dense	(1024)	—	—	0	—
Signal input	(1024)	—	—	0	—
Concatenate	(1024, 2)	—	—	0	—
Convolutional	(512, 64)	14	2	0.5	Leaky ReLU
Convolutional	(256, 128)	14	2	0.5	Leaky ReLU
Convolutional	(128, 256)	14	2	0.5	Leaky ReLU
Convolutional	(64, 512)	14	2	0.5	Leaky ReLU
Flatten	(32 768)	—	—	0	—
Dense	(1)	—	—	0	Sigmoid

Generator

Operation	Output shape	Kernel size	Stride	BN	Activation
Class input	(5)	—	—	✗	—
Latent input	(100)	—	—	✗	—
Concatenate	(105)	—	—	✗	—
Dense	(32 768)	—	—	✗	ReLU
Reshape	(64, 512)	—	—	✗	—
Transposed conv	(128, 256)	18	2	✓	ReLU
Transposed conv	(256, 128)	18	2	✗	ReLU
Transposed conv	(512, 264)	18	2	✗	ReLU
Transposed conv	(1024, 1)	18	2	✗	Linear
Reshape	(1024)	—	—	✗	—

Optimizer	Adam ($\alpha = 0.0002$, $\beta_1 = 0.5$)
Batch size	512
Epochs	500
Loss	Binary cross-entropy

Table A2. The architecture and hyperparameters describing our CNN consists of four convolutional layers followed by two dense layers. The convolutional and dense layers are activated by the swish function [65] and dropout is applied, while the final layer uses the sigmoid activation. The network is trained by minimising the binary cross entropy and optimised with Adam with learning rate 10^{-3}. We train for 100 epochs with a batch size of 1000.

Operation	Output shape	Kernel size	Stride	Dropout	Activation
Input	(1024, 2)	—	—	—	—
Convolutional	(512, 8)	5	2	0	Swish
Convolutional	(256, 8)	5	2	0	Swish
Convolutional	(128, 8)	5	2	0	Swish
Convolutional	(64, 8)	5	2	0	Swish
Dense	(100)	100	—	0.2	Swish
Dense	(1)	1	—	0	Sigmoid
Optimizer	Adam ($\alpha = 0.001$, $\beta_1 = 0.5$)				
Batch size	1000				
Epochs	100				
Loss	Binary cross-entropy				

ORCID iDs

J McGinn https://orcid.org/0000-0001-8910-0881
C Messenger https://orcid.org/0000-0001-7488-5022
M J Williams https://orcid.org/0000-0003-2198-2974
I S Heng https://orcid.org/0000-0002-1977-0019

References

[1] Abbott B *et al* 2016 *Phys. Rev. Lett.* **116** 061102
[2] Abbott B *et al* (KAGRA) (LIGO Scientific) (VIRGO) 2018 *Living Rev. Relativ.* **21** 3
[3] Aasi J *et al* (LIGO Scientific) 2015 *Class. Quantum Grav.* **32** 074001
[4] Harry G M (LIGO Scientific) 2010 *Class. Quantum Grav.* **27** 084006
[5] Acernese F *et al* (VIRGO) 2015 *Class. Quantum Grav.* **32** 024001
[6] Abbott B *et al* 2016 *Phys. Rev. Lett.* **116** 241103
[7] Abbott B P *et al* 2017 *Astrophys. J. Lett.* **851** L35
[8] Abbott B *et al* 2017 *Phys. Rev. Lett.* **118** 221101
[9] Abbott B *et al* 2017 *Phys. Rev. Lett.* **119** 161101
[10] Abbott R *et al* 2020 arXiv:2010.14527
[11] Buikema A *et al* 2020 *Phys. Rev.* D **102** 062003
[12] Tse M *et al* 2019 *Phys. Rev. Lett.* **123** 231107
[13] Fryer C L and New K C B 2003 *Living Rev. Relativ.* **6** 2
[14] Andersson N and Comer G L 2001 *Phys. Rev. Lett.* **87** 241101
[15] Baiotti L, Hawke I and Rezzolla L 2007 *Class. Quantum Grav.* **24** S187–S206
[16] Owen B J and Sathyaprakash B S 1998 arXiv:9808076
[17] Usman S A *et al* 2016 *Class. Quantum Grav.* **33** 215004
[18] Sachdev S *et al* 2019 The GstLAL search analysis methods for compact binary mergers in advanced Ligo's second and advanced Virgo's first observing runs (arXiv:1901.08580)
[19] Drago M *et al* 2020 Coherent waveburst, a pipeline for unmodeled gravitational-wave data analysis (arXiv:2006.12604)

[20] Abbott B P *et al* 2016 *Class. Quantum Grav.* **33** 134001
[21] Abbott B P *et al* 2020 *Class. Quantum Grav.* **37** 055002
[22] Klimenko S, Yakushin I, Mercer A and Mitselmakher G 2008 *Class. Quantum Grav.* **25** 114029
[23] Aso Y, Márka Z, Finley C, Dwyer J, Kotake K and Márka S 2008 *Class. Quantum Grav.* **25** 114039
[24] Gabbard H, Williams M, Hayes F and Messenger C 2018 *Phys. Rev. Lett.* **120** 141103
[25] Gebhard T D *et al* 2019 *Phys. Rev.* D **100** 063015
[26] Krastev P G 2020 *Phys. Lett.* B **803** 135330
[27] Skliris V, Norman M R K and Sutton P J 2020 Real-time detection of unmodeled gravitational-wave transients using convolutional neural networks (arXiv:2009.14611)
[28] López M *et al* 2021 *Phys. Rev.* D **103** 063011
[29] Bahaadini S *et al* 2017 *2017 IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)* pp 2931–5
[30] George D, Shen H and Huerta E 2018 *Phys. Rev.* D **97** 101501
[31] Razzano M and Cuoco E 2018 *Class. Quantum Grav.* **35** 095016
[32] Gayathri V *et al* 2020 Enhancing the sensitivity of transient gravitational wave searches with gaussian mixture models (arXiv:2008.01262)
[33] Zevin M *et al* 2017 *Class. Quantum Grav.* **34** 064003
[34] Gabbard H *et al* 2019 Bayesian parameter estimation using conditional variational autoencoders for gravitational-wave astronomy (arXiv:1909.06296)
[35] Chua A J K and Vallisneri M 2020 *Phys. Rev. Lett.* **124** 041102
[36] Green S R, Simpson C and Gair J 2020 Gravitational-wave parameter estimation with autoregressive neural network flows (arXiv:2002.07656)
[37] Dreissigacker C and Prix R 2020 *Phys. Rev.* D **102** 022005
[38] Dreissigacker C *et al* 2019 *Phys. Rev.* D **100** 044009
[39] Bayley J, Messenger C and Woan G 2020 A robust machine learning algorithm to search for continuous gravitational waves arXiv:2007.08207
[40] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (Cambridge, MA: MIT Press) http://deeplearningbook.org
[41] Goodfellow I *et al* 2014 Generative Adversarial Networks (arXiv:1406.2661)
[42] Brock A, Donahue J and Simonyan K 2018 Large scale Gan training for high fidelity natural image synthesis (arXiv:1809.11096)
[43] Karras T *et al* 2019 Analyzing and improving the image quality of StyleGan (arXiv:1912.04958)
[44] Reed S *et al* 2016 Generative adversarial text to image synthesis (arXiv:1605.05396)
[45] Liang X *et al* 2017 Dual motion Gan for future-flow embedded video prediction (arXiv:1708.00284)
[46] Esteban C, Hyland S L and Rätsch G 2017 Real-valued (medical) time series generation with recurrent conditional Gans (arXiv:1706.02633)
[47] Mirza M and Osindero S 2014 Conditional generative adversarial nets arXiv:1411.1784
[48] Isola P *et al* 2016 Image-to-image translation with conditional adversarial networks (arXiv:1611.07004)
[49] Ismail Fawaz H *et al* 2018 Deep learning for time series classification: a review arXiv:1809.04356
[50] Minaee S *et al* 2020 Deep learning based text classification: a comprehensive review arXiv:2004.03705
[51] Dumoulin V and Visin F 2016 A guide to convolution arithmetic for deep learning (arXiv:1603.07285)
[52] Ioffe S and Szegedy C 2015 Batch normalization: accelerating deep network training by reducing internal covariate shift (arXiv:1502.03167)
[53] Tompson J *et al* 2014 Efficient object localization using convolutional networks (arXiv:1411.4280)
[54] Abbott B P *et al* (LIGO Scientific Collaboration) (Virgo Collaboration) 2019 *Phys. Rev.* D **100** 024017
[55] Khan S *et al* 2016 *Phys. Rev.* D **93** 044007
[56] LIGO Scientific Collaboration 2018 LIGO algorithm library *LALSuite Free Software (GPL)* https://lscsoft.docs.ligo.org/lalsuite/
[57] Abbott B P *et al* 2019 *Astrophys. J.* **882** L24
[58] Abadie J *et al* (The LIGO Scientific Collaboration) (The Virgo Collaboration) 2012 *Phys. Rev.* D **85** 122007
[59] Radford A, Metz L and Chintala S 2015 Unsupervised representation learning with deep convolutional generative adversarial networks (arXiv:1511.06434)
[60] Chollet F *et al* 2015 Keras https://keras.io

[61] Abadi M *et al* 2015 TensorFlow: large-scale machine learning on heterogeneous systems https://www.tensorflow.org/
[62] Barsotti L *et al* 2018 Updated advanced Ligo sensitivity design curve *Technical Report T1800044-v5 LIGO* Salt Lake City, UT
[63] Abbott B P *et al* 2020 *Living Rev. Relativ.* **23** 3
[64] Babak S *et al* 2013 *Phys. Rev.* D **87** 024033
[65] Ramachandran P, Zoph B and Le Q V 2017 Searching for activation functions (arXiv:1710.05941)
[66] Kingma D P and Ba J 2014 Adam: A method for stochastic optimization arXiv:1412.6980