Generalised gravitational burst generation with Generative Adversarial Networks

J. McGinn, C. Messenger, I.S. Heng

University of Glasgow, Physics & Astronomy Department, Glasgow G12 8QQ, UK

LIGO-PXXXXXXX

Abstract. The next generation of Gravitational wave detectors will accelerate the number of gravitational wave detection's such that we can gain new in site into the physics behind the sources causing the phenomena. Numerical simulations and matched filtering are the standard for detecting gravitational waves for know sources such as binary-black hole mergers. Other sources of gravitational waves that remain elusive to standard modelling techniques and are expected to be detectable, however, there must be a way to characterise them in order to build a model template. Here we construct a unmodeled burst generation scheme using Generative adversarial networks - a powerful class of machine learning.

1. Introduction

GW! (**GW!**) astronomy is now an established field, starting with the first detection of a binary black hole merger [**?**] in September 2015. Following this, the first and second observations runs (O1 and O2) of Advanced LIGO and Advanced Virgo **CHRIS: need refs for the detectors, see here https://pnp.ligo.org/ppcomm/standard_references.html** reported several more mergers [**?**, **?**, **?**, **?**]. On August 2017 a binary neutron star merger was observed alongside its electromagnetic counterpart for the first time, giving rise to multimessenger gravitational wave astronomy.

GW bursts are transient signals of typically short duration (< 1s) whose waveforms are not accurately modelled or are complex to re-produce. Astrophysical sources for such transients include: Core collapse supernova, Neutron star instabilities, Fallback accretion onto a neutron star, Non axisymetric deformation in magnetars, Pulsar glitches and Neutron star post-mergers. As **GW!** bursts are un-modelled they are not sensitive to template based detection schemes **CHRIS: OK, but logically dodgy. You make it sound like we could use a template scheme but it would be insensitive. The point is that we don't have templates in the first place.** such as matched-filtering [**?**], instead, detection involves distinguishing the signal from detector noise. This is only possible if the detector noise is well characterised and the candidate signal can be

differentiated from systematic or environmental glitches. As such, **GW!** burst searches rely on an astrophysical burst signature appearing in multiple detectors **CHRIS: at similar times?**.

Many **GW!** burst algorithms [**?**, **?**] **CHRIS: many? needs more than 2 refs for many.** are tested and tuned using model waveforms that may or may not have astrophysical significance but have easy to define parameters and share characteristics of real bursts that is enough to simulate coincident non-stationary deviations **CHRIS: fancy sounding but I'm not sure what non-stationary deviations means.** between detectors. Such waveforms may have long-duration, short bandwidth (ringdowns), long-duration, large bandwidth (inspirals) and many algorithms make use of sine-Gaussians: a Gaussian modulated sine wave that is characterised by it's central frequency and narrow bandwidth. **CHRIS: not the time to try to describe the types of burst waveforms. Also, be careful with satying things like ringdowns have narrow bandwidth. I know we specify a single frequency but becuase of the short duration, the signal is broad band. Look at the FFT** This makes it a great tool for diagnosing LIGOs sensitivity to frequency. **CHRIS: strange unfinished sentence.**

CHRIS: Introduce ML techniques in GWs

CHRIS: What this paper does on GANs in 1 paragraph

CHRIS: describe the structure of the paper We aim to explore the use of machine learning in generating and interpreting these mock **GW!** burst signals. Neural networks have shown to replicate the sensitivities of matched filtering in **GW!** detection [**?**] and rapid parameter estimation [**?**], however these methods have primarily been focused on binary black hole signals and have not yet expanded to burst examples. **CHRIS: OK, thanks for the referdndes but you should probably have more refs to other ML GW papers. I think you should dedicate at least one paragrpah to GW ML efforts. The final paragrapgh of the section should introduce the paper structure. The penultimate paragraph should expain what the point of this paper is - in non technical terms. This is all outlined in the bullet points (hidden) at the start of this section.**

2. Generative Adversarial Networks

A subset of deep learning that has seen fruitful development in recent years [**?**] is **GAN!**s (**GAN!**s). These unsupervised algorithms learn patterns in a given training data set using an adversarial process. The generations from **GAN!**s are state-of-the-art in fields such as high quality image fidelity [**?**, **?**], text-to-image translation [**?**] and video prediction [**?**] as well as time series generations [**?**].

GAN!s train two competing neural networks, consisting of a discriminator that is set up to distinguish between real and fake data and a generator that produces synthetic reproductions of the real data. The generator performs a mapping from an input noise vector $\mathbf{z}$ to its representation of the data and the discriminator maps its

input $\mathbf{x}$ to a probability that the input came form either the training data or generator. During training, the discriminator is given a batch of samples that contains one half real data and one half fake data which it then makes predictions on. The loss for the discriminator is calculated by comparing its predictions to the labelled data through the binary cross-entropy function. The training process of a **GAN!** alternatively updates the weights **CHRIS: too much knowledge assumed - what are weights, they haven't been defined** of the discriminator and generator based on information on its competitors loss function. This loss of discriminator is used to update the weights of the generator to produce more realistic samples of the input distribution, the loss of G **CHRIS: what is G?** encourages the discriminator to update its classification abilities. Both networks compete in a minimax game **CHRIS: what is a minimax game? references?** which the generator is trying to minimise and the discriminator is trying to maximise: **CHRIS: nicely written, however, it needs an extra 25% explanation for people not familar with ML or GANs in particular. Try to spell things out more. Also refer to Fig 1 somewhere in this section.**

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

CHRIS: In line with explanding the explanation a bit more, really take the time to state what controls the functions and also deine ALL mathematical variables. What are $D, G, V \ldots$ Why are you taking the logs? What situation so you start with? What do you end up with? This is a nice start but beef it up.

2.1. Auxiliary conditional GANs

In theory, the adversarial process will eventually lead to the local Nash equilibrium [**?**] whereby both neural networks are trained optimally **CHRIS: this statement and this paragraph should be stated in the previous section**. In practice, however, **GAN!**s are notoriously difficult to train. Such difficulties include: Non-convergence, where the model parameters **CHRIS: you haven't stated what the model parameters are yet** oscillate and the loss **CHRIS: the reader soes not know what the loss is and what the point of it is** never converges, mode collapse where G produces a limited diversity of samples **CHRIS: samples of what?**, and diminishing gradients when applying gradient descent to a non-continuous function **CHRIS: not sure about what non-continuous means here but in general you are bringing in too many concepts without explanation e.g., gradient descent, why is that relevant? the reader does not know.**.

To overcome some of these difficulties, a conditional variant of **GAN!**s named **CGAN!**s (**CGAN!**s) [**?**] adds structure to the latent space **CHRIS: what is the latent space - you called it an input noise vector before** by providing the generator with a class or attribute label. The generator learns to segment the latent space by clustering distributions which have similar properties, making a point in latent

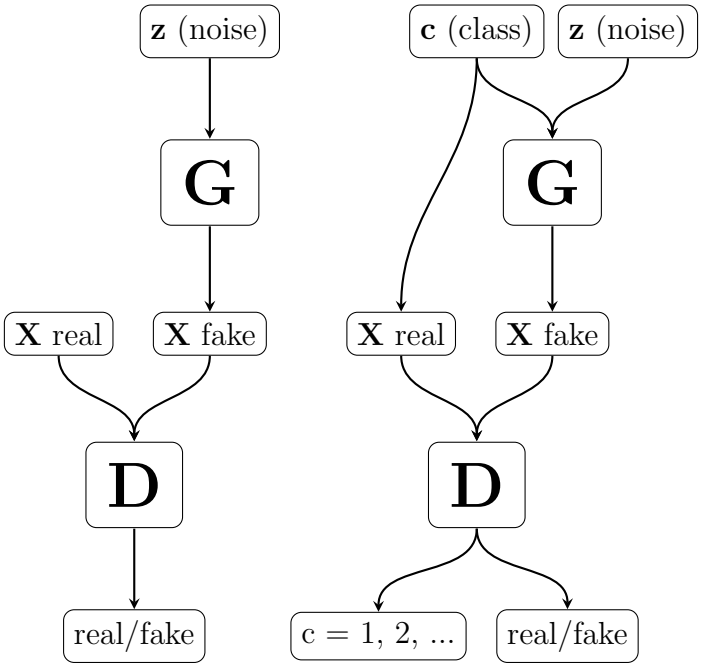

Figure 1. Comparison of the original GAN method and the Auxiliary Conditional-GAN method. For ACGANs the training data requires a label denoting its class that is also fed to the generator which then learns to generate waveforms based on the input label. Additionaly, the discrminator learns to classify which class the signal belongs to.

space conditional on a class **CHRIS: are we sure about this? all classes share the same latent space and I don't think that the space gets segmented into areas corresponding to different classes - I do think that it gets broken down into spaces with different intrinsic physical properties e.g., high frequency, long duration, etc...**. This idea was extended further with **ACGAN!**s (**ACGAN!**s) [**?**] that require the discriminator to output a probability of data belonging to each class. A pictorial representation on the differences between these approaches is shown in Fig. **??**.

CHRIS: this section is supposed to be on ACGANs but it only mentions them in the final sentence. Again, like the previous section, a bit more explanation would be good. Does ths maths change? Could you have a new version of Eq 1 for CGANs and ACGANs? Also, if would be good to have a statement that you will be using one of these methods in this paper.

3. Methodology

GW! burst signals remain an unmodelled phenomenon, as such, current detection algorithms focus on waveform reconstruction reliant on coincident signals within multiple detectors **CHRIS: burst detection algorithms do not need waveform reconstruction for searches - it is an add on feature.**. We propose a signal generation scheme utilizing **CHRIS: utilizing certainly is fancier than using.**

Dont go overboard with the language. **GAN!**s trained on burst-like waveforms. The **GAN!** **CHRIS: is it a GAN or is it a CGAN or an ACGAN? Be clearer** is trained on five signal morphology's spanning a range of prior parameters. The families are: **CHRIS: I know they don't all share the same parameters but it might be sensible to make a table of the prior ranges rather than haviong it all stated in text.**

- Sine-Gaussian:
  CHRIS: don't just start things with an equation

$$h(t) = A \exp\left[-\frac{(t-t_0)^2}{\tau^2}\right] \sin(2\pi f_0(t-t_0)) \tag{2}$$

  where f_0 is the central frequency that takes values between 30 Hz - 50 Hz. τ is the decay parameter chosen to be between 60s^{-1} and 15s^{-1} and the starting epoch chosen between 0.2s - 0.8s. Sine-Gaussian waveforms have a similar form to those produced by the merger of relatively high mass binary black holes for which the inspiral phase occurs at frequencies below ground-based detector sensitivities.

- Ring-down:
  CHRIS: don't just start things with an equation

$$h(t) = A \exp\left[-\frac{(t-t_0)}{\tau}\right] \sin(2\pi f_0(t-t_0)) \tag{3}$$

  For ring-down signals the parameters for f_0, τ, t_0 are chosen between; 30 Hz-50Hz, 0.02-0.1 and 0.1s-0.8s respectfully. These signals aim to replicate the post-merger late stages of binary coalescence.

- White-noise bursts:
  CHRIS: where is the waveform equation? These signals are produced by inserting samples of random Gaussian noise with zero mean and a variance of 0.1 at a random point between 0.2s - 0.8s. The source of this signal type can be attributed to core-collapse supernova **CHRIS: is this correct? these waveofms are approximations so best not to say that they correspond directly to physical phenomena**.

- Gaussian pulse:
  CHRIS: where is the waveform equation? A simple Gasssian pulse waveform with t_0 ranging from 0.2s to 0.8s and τ between 100^{-1} and 20^{-1}

- Binary black holes (BBH):
  BBH! (**BBH!**) signals are simulated using IMRPhenomD waveform routine from LALSuite **CHRIS: references for both the waveofrm and lalsuite** which models the inspiral, merger and ringdown of a **BBH!** waveform. The component masses lie in the range of [5,70]M_0 **CHRIS: not M_0 but $M_\odot$** with zero spins and we fix $\text{m}_1 > \text{m}_2$. The mass distribution is approximated by a power law with index of 1.6 https://arxiv.org/abs/1811.12940 **CHRIS: correct;ly reference this**. The

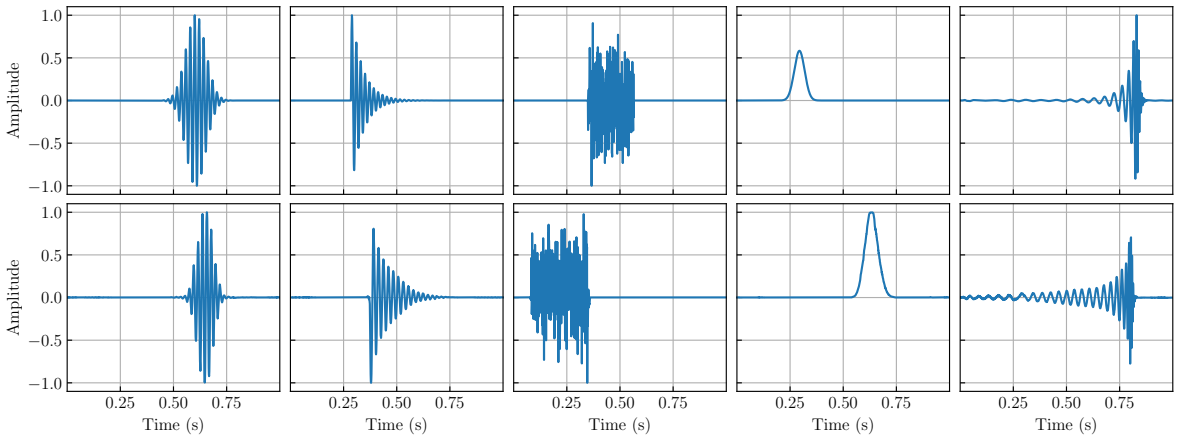

Figure 2. Examples of simulated **GW!** burst signals. Top row shows examples from the training set. From left to right: Sine-Gaussian, Ringdown, White-noise burst, Gaussian pulse, Binary black hole merger. The bottom row shows the conditional generations from the GAN. **CHRIS: slightly more explanation. The reader may be concerned that the top and bottom do not look the same. Should they? They may wonder why is there only one waveform and not one per detector?**

signals are generated using random right ascensions and declinations uniform over the sky and the inclinations are drawn from the cosine of a uniform distribution in the range [-1,1]. The peaks of the waveforms are set to be within [0.75,0.95]s of the 1s time interval. **CHRIS: for all the signal models please check that the prior ranges and distributions are consistent with the presented results.**

CHRIS: you seem to be missing a description of the antenna patterns and time delay that you apply to each waveform. It should go here. You should also include all other technical information like how you normalise the amplitudes and how you select random sky positions, how the data is represented to the network (as an image or as channels) how many detectors do you use? etc...

3.1. Architecture details

Extensions to the original **GAN!** method such as the **DCGAN!** (**DCGAN!**) [**?**] have been widely praised for building **CHRIS: enabling?** a stable **GAN!** architecture. Modern **GAN!** research **CHRIS: isn't all GAN research modern?** favours a fully convolution neural network which replace upsampling procedures like maxpooling to strided convolutions **CHRIS: a couple of things here. Max pooling and striding hasn't been introduced, and why do max-pooling and striding upsample? I thought they downsampled, don't they?** . **CNN!**s (**CNN!**s) are designed to work with grid-like structures that exhibit strong local spatial dependencies **CHRIS: great but that sounds like a computer science paper. Make it easier for the physicists to understand.**. Although most work with **CNN!**s involve image based

data, they can be applied to other spatially adjacent data types such as time-series and text items. Audio synthesis work [**?**] has showed that **GAN!**s can achieve recognisable audio within a few hours **CHRIS: ambiguous. A few hours of training?**.

We adopt the suggestions of these papers **CHRIS: which papers?**, lengthening one-dimensional convolution kernels on both the generator and discriminator. The Generator model is fully convolutional, upsampled using strided transposed convolutions with batch normalisation in the first layer and ReLU activations throughout with the exception of Tanh for the output layer. Each transposed convolutional layer uses a kernel size of 18×1 and stride of 2. The discriminator network mirrors that of the generator without batch normalization, using LeakyReLU activations, SpatialDropout, and a 2-stride convolution for downsampling. The discriminator has two output layers: the first output is a single node activated by a Sigmoid that can be interpreted as the realness of the the signal, the second output is 5 nodes activated using the softmax function predicting the class of the input. This model is trained with binary cross entropy for the first output and sparse categorical cross-entropy for the second output. **CHRIS: OK, very technically tight but too computer sciencey. Try to make it human readbale for a physicist and refer to the table of parameters in the appendix.**

Neural networks and subsequently **GAN!**s 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 comes from the use of trial and error and the initial designs influenced by the available literature. After tuning the multiple hyperparamters (Table **??**), the GAN was trained for 6×10^5 iterations **CHRIS: what is an iteration here? It's not an epoch right?** and takes O(1) day to train **CHRIS: trained on what? Were there any major decisions made based on the trial and error process? Anything that was learned from your mistakes that might be useful to others?**.

3.2. Class labels and embedding

A learned embedding is an efficient way of representing categorical or class based data. In contrast to traditional "one-hot encoding" where each class is represented by a binary sparse vector, embeddings map each class to its own distinct vector of a given size. The elements of these vectors are initialized randomly and tuned during training just like weights **CHRIS: I thought that they weren't trained. Are they definitely trained? Is this not just a single fully connected layer that turns 5 numbers into 120? How is it any different to that?**. This reduces computational cost as the sparse vectors containing mostly zeros are not needed and allows related classes to cluster together. Once the networks are trained the learned embedding vectors can be extracted from the model and used as inputs to the generator. The benefit here is that the former integer class labels are replaced by higher dimensional vector representations allowing for finer experimenting in other applications. The class label is interpreted as an additional channel early in the generator model. This is achieved by projecting the

class inputs as a learned embedding layer, into a fully connected layer or "dense" layer which can then be concatenated channel-wise to $\mathbf{z}$. **CHRIS: very nice but still a bit too technical without consideration for lowly astrophysicists. Can you soften it a bit to make it more accessible. Explain more of the concepts when you introduce them.**

CHRIS: Also, it isn't clear where this comes into your analysis and why it's important. I'm not sure that the definition of "signal class" has been made anywhere.

3.3. Applying a time shift and antenna responses

CHRIS: Why is this not described in the section defining the signal waveforms? There may be a good reason so it may well fit best here. I'm not sure We consider a two detector case **CHRIS: why 2 detector and not 3?** in which the generator is trained to output two identical signals with a physical time shift representing the time of flight between detectors and the suppression **CHRIS: not a supression. It's just the antenna response.** of amplitudes due to interaction with the detector. To achieve this, the generator is trained to output a single waveform that is then put through a non-trainable "response" layer before the output of the generator. This layer creates a copy of its input, shifts it along in time and applies antenna responses to both the input and shifted signal. The time shift is achieved by converting the waveform to the frequency domain, multiplying by $e^{2\pi i f \Delta t}$, where Δt is the time shift and converting **CHRIS: converting? what do you mean?** back to the time domain. After which, both original and shifted signals are multiplied by their respective antenna responses. Both Δt and the antenna responses are calculated using the LALSimulation [**?**] package and we choose Hanford and Livingston as detection points **CHRIS: just say the detectors**. The result is a 2×1024 times series waveform representing an overlay data stream **CHRIS: what the hell is an overlay data stream?** from two detector outputs. This scheme is also used in generating the training set. **CHRIS: OK, first of all you should define some equations here rather than just saying everything in words. Why is it here that you reveal the data sampling rate? Surely that is an important quantity that shouldn't just be stated in this subsection. What about the random sky position that is input to the "box"? How is the sky sampled? What about the wrapping from the tim shift? Don't you apply a window? The "box" or "response" layer should be visible in the network diagram.**

CHRIS: In general I find this section to read like un-connected statements in the sense that they are un-connected to each other and un-connected to the main GW problem. Try to make it more like a recipe for doing this analysis.

4. Results

Given a 100 dimensional vector drawn from a normal distribution, a class label and sky localisation information, the GAN is able to generate burst-like waveforms generalised from the training set. We set out by describing the quality of 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 vector arithmetic that can be used to generate a new breed of signal by merging two or more families together. Finally, we discuss the capacity of the discriminator as a **GW!** burst classifier and the auxiliary component of this work.

4.1. Waveform quality

The generator network is a function $\mathrm{G} : \mathbf{z}, \mathbf{c}, \mathbf{s} \in \mathbb{R}^{100} \rightarrow \mathbb{R}^{1024\times 2}$, where $\mathbf{z}, \mathbf{c}, \mathbf{s}$ are the latent vector, class embedding vector and sky positions respectively. Given a latent vector randomly sampled from a normal distribution with zero mean and unit variance, a class label which is represented by a 120 dimensional vector for each class and antenna responses, the results from the generator can be seen in Figure **??**. Depending on the orientation of the detector with respect to a hypothetical signal in the sky, the waveforms may appear inverted, shifted in time and their strain attenuated. Each plot shows the output of the generator after given randomised $\mathbf{z}, \mathbf{s}$ and one of the five class vectors $\mathbf{c}$.

4.2. Interpolation

Machine learning algorithms are often described as universal function approximators. In the generators case it maps samples drawn from a 100 dimensional Gaussian space to its representation of the training set. As with any function, there should be a one to one mapping from the domain and co-domain to allow for smooth transitions across the latent space. One advantage of using GANs as a waveform generator is that once it is trained, it can perform rapid generations faster than more intricate and computationally expensive algorithms. For complicated data sets, the network architecture must be diverse and dense enough to capture distinct variations from the training set. Most GANs perform well on relatively low resolution image generations, however, higher resolutions demand larger networks and long training times. GANs attempting to replicate complicated structures and do not have the necessary architecture either struggle to produce results at all or fall into the common failure mode know as mode collapse; where the generator produces a small variety of samples or simply memorises the training set. To test this, we perform linear interpolations in the latent and class space.

4.3. Latent space interpolation

In this section we explore the latent space formed by the generator by interpolation. We take two random points in the latent space and linearly interpolate between them

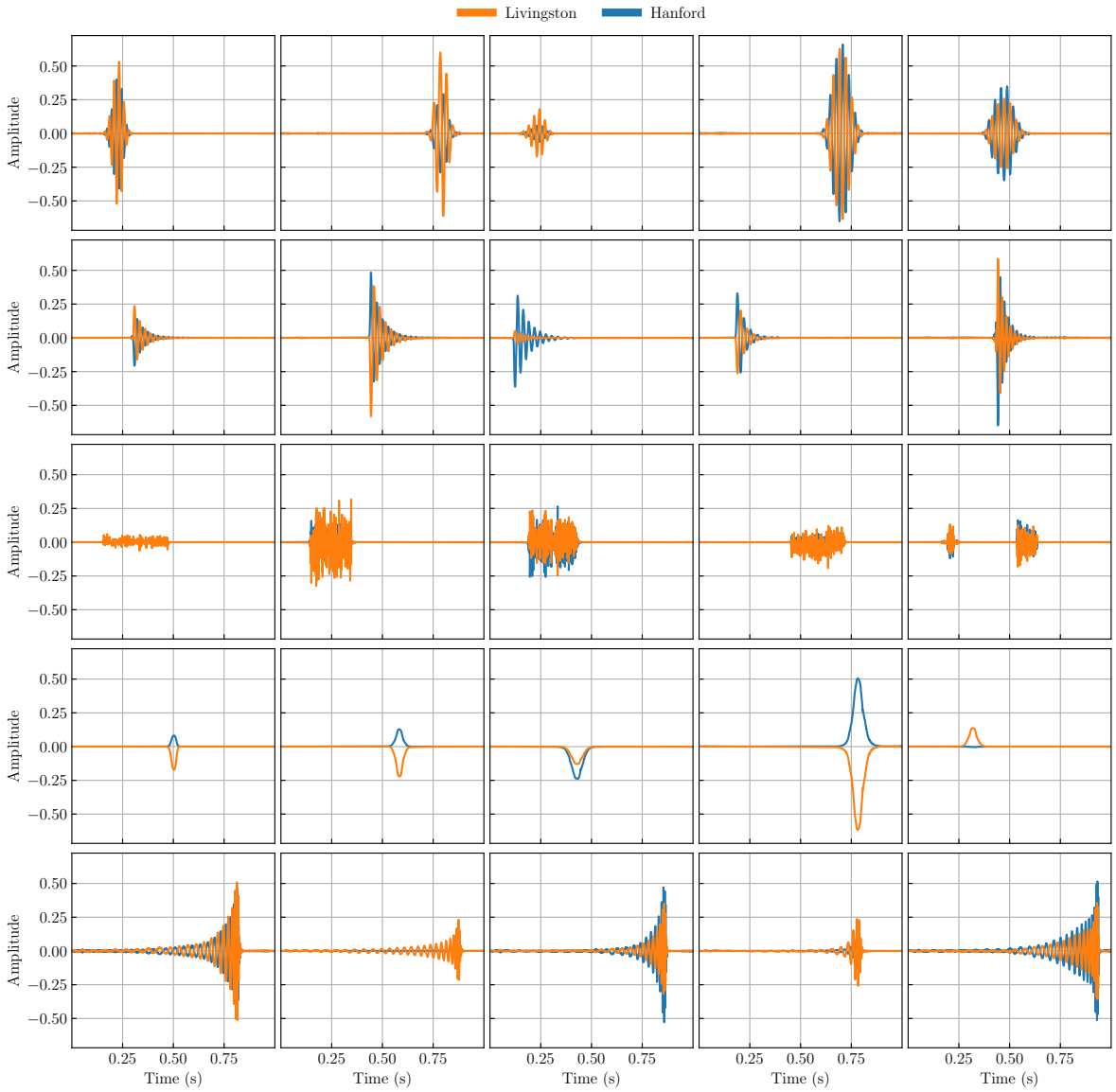

Figure 3. Generated waveforms after training and conditioning on five classes. The generator will output two waveforms as seen by detectors in Hanford (red) and Livingston (blue). The generator is able to capture the characteristics of each waveform and structure the class space to give control over which waveform to generate. Each row shows a random assortment of one of the five classes the GAN is trained on.

(six times inclusive). These new latent space vectors can now be fed into the generator to make predictions on while keeping the class vectors constant. The third input, the response values are kept constant for each class. The full effect shown in Figure **??**. We can see that each plot shows plausible waveforms suggesting that the generator has constructed a smooth space unlike the discrete training case. Additionally as each class is given the same latent points to interpolate over, we can see that the waveforms cluster together with respect to their parameters. Visually, the sine-gaussian and ring-down waveforms share similar frequencies and the other signals show similar decays and

starting epochs. The only exception is BBH waveforms, which is expected as they were trained with more variety of parameters and consistently have their peaks in the last quarter of the time series.

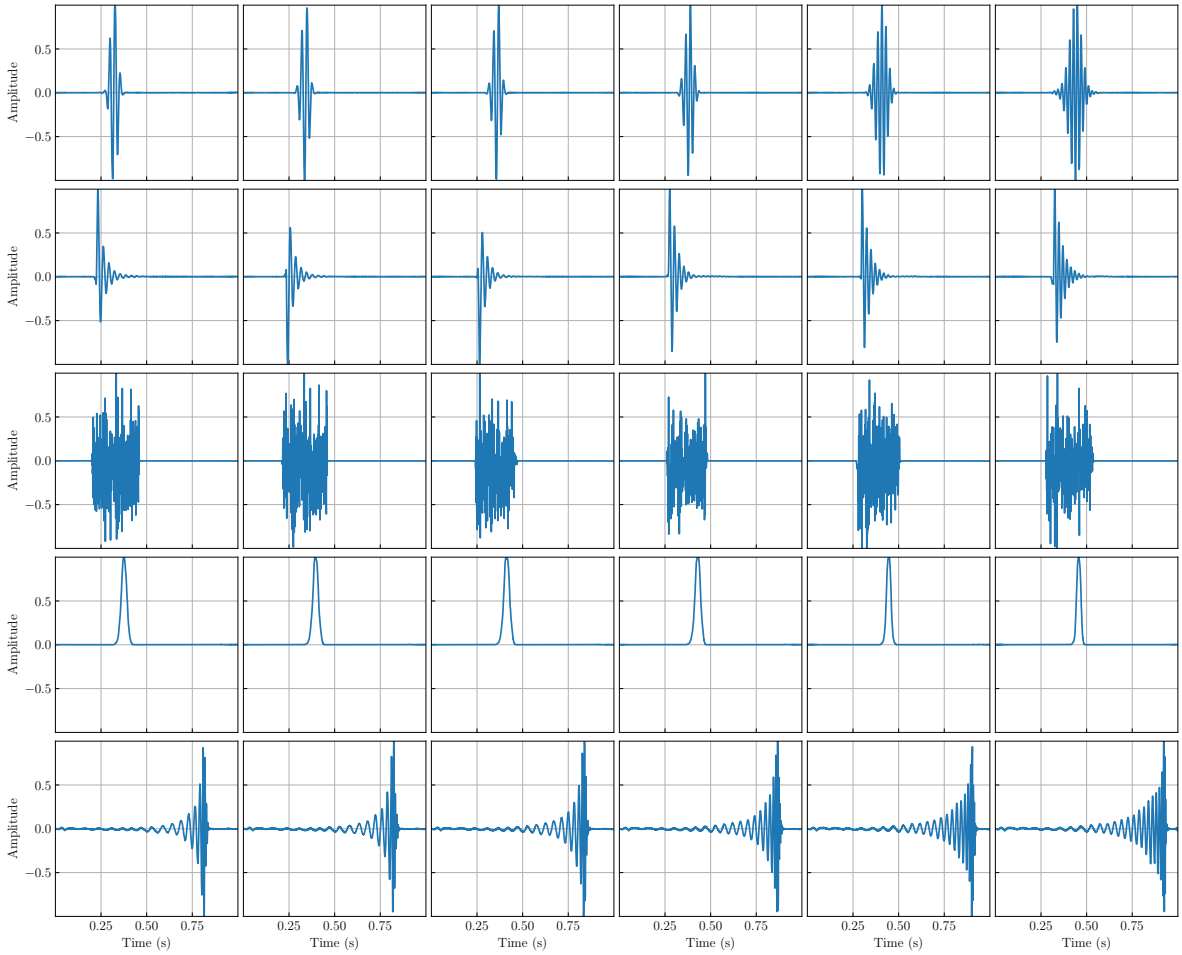

Figure 4. Interpolations between two random latent points in z. Each row uniformly interpolates between two points in **z** keeping the class fixed. Only a single waveform from the generator is plotted and each signal is re-scaled to [-1,1] effectively removing the antenna responses for clarity.

4.4. Class space interpolation

In order to explore the class space we keep the latent vector held constant and interpolate through the 5 classes. We construct a path between the 5 waveforms and show that the space is populated enough to allow for transitions between classes. Sine-Gaussian to ringdown performs well in interpolation with each signal being a plausible burst GW. It is obvious that the embedding layer has clustered these two groups during training as they share many characteristics. The other signals have sharper transitions but still retain plausible looking waveforms.

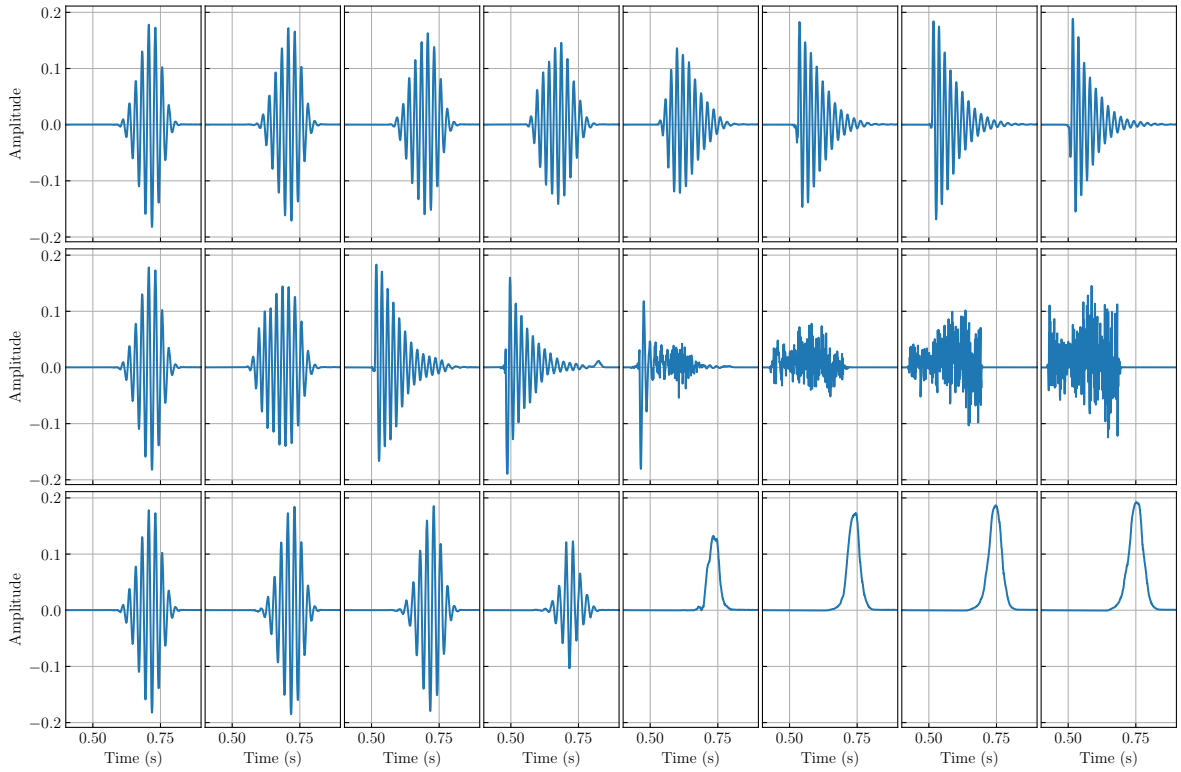

Figure 5. Class space interpolation with latent space held constant throughout. The plots show a zoomed in section of the 1s time interval. Top row: Sine-Gaussian class to ringdown class, Middle Row: Sine-Gaussian class to whitenoise burst class, Bottom row: Sine-Gaussian class to Gaussian class.

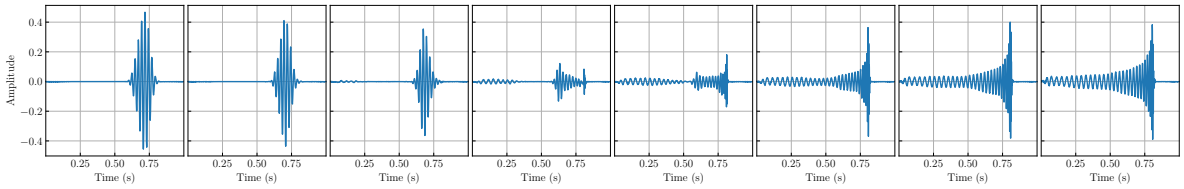

Figure 6. Class space interpolation with latent space held constant throughout. The full 1s interval is plotted for class based interpolations between a Sine-Gaussian and a BBH in spiral.

4.5. Vector Arithmetic

In DCGANs the authors demonstrated unsupervised vector arithmetic with celebrity face generation. They kept points in the latent space and performed simple vector arithmetic to generate new images. This allows for intuitive and targeted generation of images. However, the authors realised that single images vectors were unstable and there was a need to average three vectors before any arithmetic. DCGANs is unconditional, therefore, the vectors used were chosen empirically, generating many faces and choosing the attributes for study. Here, we show that this GAN only needs a single vector representation and due to conditioning we can have more control over which vectors to

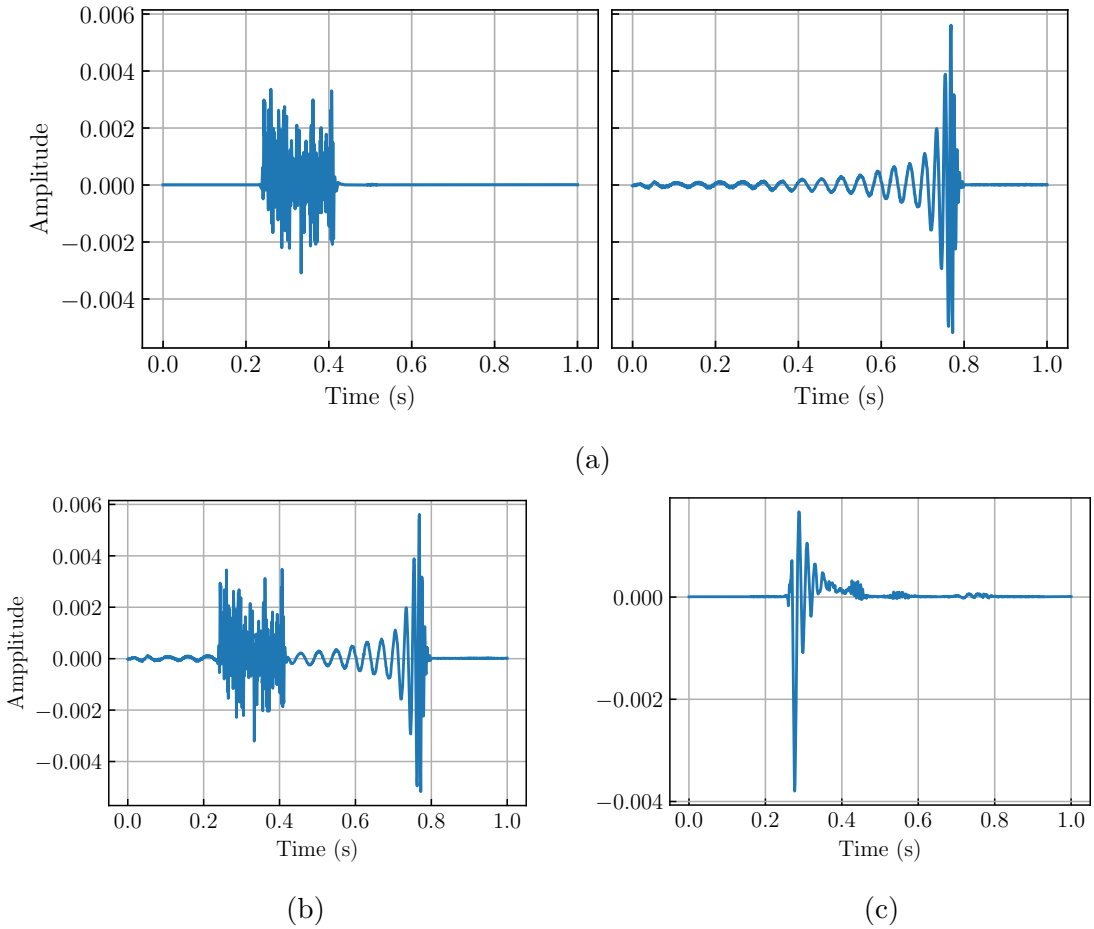

Figure 7. Conditional vector arithmetic. (a) Generated samples of a whitenoise burst and BBH insprial. (c) Effect of naively adding the components from (a) in the time domain. **JORDAN: just adding the two signals post generation** (c) Generation from the combined class vectors.

use in the arithmetic. In Figure **??** we generate a whitenoise burst and a BBH insprial using their respective class labels and use a randomised latent point and response. Keeping the latent point and response, we average the two class vectors and use this as input to the generator. Figure **??** (c) shows the generation based on the combined class vectors which visually looks similar to a supernova waveform. Supernova waveform generations require expensive numerical relativity simulations and various assumptions about the collapse of its parent star. Here, we are able to produce similar waves at a fraction of the expense and we are able to further modify the resultant by using different latent space samples. **JORDAN: this is the adding noise thing but i need to work out the correct way to do it with the directional vector**.

4.6. Classifier

JORDAN: I'm working on this part and how to present the auxiliary part.

5. Conclusions

In this work we present the potential of Generative Adversarial Networks for burst gravitational wave analysis. We have shown that GANs have the ability to generate 5 class varieties of modelled burst **GW!** signals that can be generated at whim. The latent and class spaces were explored through interpolation and suggest that the space provides smooth translations between classes and overall waveform shape. We then showed targeted waveform generation by mixing classes to produce new unmodlled waveform varieties that can be used to test current burst search pipelines. **JORDAN: couple of sentences about classifier**.

In order to extend this work to a viable burst wave generator and classifier a few points require further research. In principle it is trivial to add another detector inside the response layer **JORDAN: response is the wrong thing to say since the time delay isnt really a response, extrinsic layer? just non trainable layer? The box?**, however, as this now 3 dimensional signal is feed to the discriminator this will no doubt require further tweaking of the network. We can add more burst-like wave forms in the training set, like detector glitches which would similarly require further network design. The work presented here is noise free. To havea complete generation and detection package we would like to train the network on signals hidden in additive Gaussian noise and test the ability of the auxiliary classifier.

The approach shown in this work shows promise in generating unmodelled burst waveforms from exotic sources. Having the ability to quickly generate new waveforms is essential to test current detection schemes and their susceptibilty to unmodelled sources. We belive that GANs have the ability to generate high fidelity waveforms at a fraction of the computational expense and do not rely on large prior parameter space. Having banks of these waveforms at hand can aid in our understand of the physics processes behind these non-standard gravitational wave emitters.

JORDAN: I want to include a link to the github etc but also a google collab scrip like the one for BigGANs:
`https://colab.research.google.com/github/tensorflow/hub/blob/master/examples/colab/biggan_generation_with_tf_hub.ipynb`. **It's fun and gives a better feel for the interpolating that static images.**

References

Appendix A. List of hyperparameters

Table A1. ACGAN architecture

Operation	Kernel	Strides	Output Shape	BN	Dropout	Activation
G($\mathbf{z}$): Input $\mathbf{z} \sim$ Normal(0,0.02)	N/A	N/A	(100,)	✗	0	N/A
Dense	N/A	N/A	(32768,)	✗	0	ReLU
Class input c	N/A	N/A	(1,)	✗	0	N/A
Embedding	N/A	N/A	(1, 120)	✗	0	N/A
Dense	N/A	N/A	(1,128)	✗	0	ReLU
Reshape $\mathbf{z}$	N/A	N/A	(128, 256)	✗	0	N/A
Reshape c	N/A	N/A	(128, 1)	✗	0	N/A
Concatenate	N/A	N/A	(128, 257)	✗	0	N/A
Reshape	N/A	N/A	(64, 514)	✗	0	N/A
Transposed Convolution	18x1	2	(256, 256)	✓	0	ReLU
Transposed Convolution	18x1	2	(512, 128)	✗	0	ReLU
Transposed Convolution	18x1	2	(1024, 64)	✗	0	ReLU
Convolution	18x1	1	(1024, 1)	✗	0	Tanh
Sky input	N/A	N/A	(3,)	✗	0	N/A
Concatenate	N/A	N/A	(1027,)	✗	0	N/A
Lambda	N/A	N/A	(1024, 2)	✗	0	N/A
D($\mathbf{x}$): Input $\mathbf{x}$	N/A	N/A	(1024, 2)	✗	0	N/A
Convolution	14x1	2	(512, 64)	✗	0.5	Leaky ReLU
Convolution	14x1	2	(256, 128)	✗	0.5	Leaky ReLU
Convolution	14x1	2	(128, 256)	✗	0.5	Leaky ReLU
Convolution	14x1	2	(64, 512)	✗	0.5	Leaky ReLU
Flatten	N/A	N/A	(32768,)	✗	0	N/A
Dense	N/A	N/A	(1,)	✗	0	Sigmoid
Dense	N/A	N/A	(5,)	✗	0	Softmax
Optimizer	Adam($\alpha = 0.0002$, $\beta_1 = 0.5$)					
Batch size	128					
Iterations	60000					
Leaky ReLU slope	0.2					
Weight initialization	Gaussian($\mu = 0$, $\sigma = 0.02$)					
Generator loss	Binary cross-entropy					
Discriminator loss	Binary cross-entropy & sparse categorical cross-entropy					

Appendix B. Many more generated examples