Generalised gravitational burst generation with Generative Adversarial Networks

J. McGinn, C. Messenger, I.S. Heng, M. J. Williams
University of Glasgow, Physics & Astronomy Department, Glasgow G12 8QQ, UK

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

textwidth in inches: 6.17804in

Gravitational wave astronomy is now an established field, starting with the first detection of a binary black hole merger [1] in September 2015. Following this, the first and second observations runs (O1 and O2) of Advanced LIGO and Advanced Virgo [2, 3, 4, 5] reported several more Compact Binary Coalescence (CBC) mergers [6, 7, 8, 9]. 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.

With these successes and continued upgrades to the detectors, further detections of CBCs are expected to be commonplace in a few short years. Another group of GW signals that has thus far been undetectable is GW "bursts". 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. **MICHAEL: Might be worth citing some sources for this statement**

GW detections so far have used a process called matched filtering, [10], where a large template bank of possible GW waveforms are compared to the detector outputs. In order to increase the chances of detection these template banks must span a large multi dimensional parameter space and the templates must be accurate, requiring significant computational cost. GW bursts are un-modelled; therefore there are no templates available and so these signals are undetectable through this scheme. Instead, detection involves distinguishing the signal from detector noise. Burst searches look 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.

Gravitational wave (GW) burst algorithms [11, 12] 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 a GW passing between detectors. Such waveforms include sine-Gaussians: a Gaussian modulated sine wave that is by it's central frequency and decay parameter. Bandlimited white-noise bursts: white noise that is contained within a certain frequency range and ring-downs which mimic the damped oscillations after a CBC merger.

With the expectation that there will be many more GW detections in the future, researches are starting to realise the need for fast and efficient GW analysis to compete with the rising number of detections. While still in its infancy, Machine Learning (ML) with GW has already shown great potential for overcoming data bottlenecks. GW analysis in areas of detection [13, **?**, **?**], glitch classification [**?**, **?**, **?**] and parameter estimation [14, **?**, **?**] have shown the success of in ML pattern recognition. Applying these algorithms show similar results to matched filtering techniques and have the benefit of being orders of magnitude faster.

In this work we aim to explore the use of machine learning to generate and interpret unmodelled GW burst waveforms. Using the generative machine learning model: Generative Adversarial Networks (GANs), we train on five classes of known burst morphologies in the time domain. Working on the assumption that GANs construct smooth n-dimension vector spaces between it's input and output, we can then explore the space between the five classes to construct new unmodelled waveforms. As all the computationally expensive processes occur during training, the learned model after training is able to produce waveforms much faster than and in fact produce waveforms that are impossible to produce with current techniques. These new varieties of waveforms can then be used to diagnose detection algorithms and gain new insight into sources of GW bursts.

This paper is organised as follows: **JORDAN: unstructured at the moment**

2. Generative Adversarial Networks

2.1. Artificial neural networks

Since the birth of programmable computers, people have wondered if machines can develop true intelligence beyond formal mathematical procedures. Today, ML aims to learn apparent relationships held within the given data or 'training data' in order to

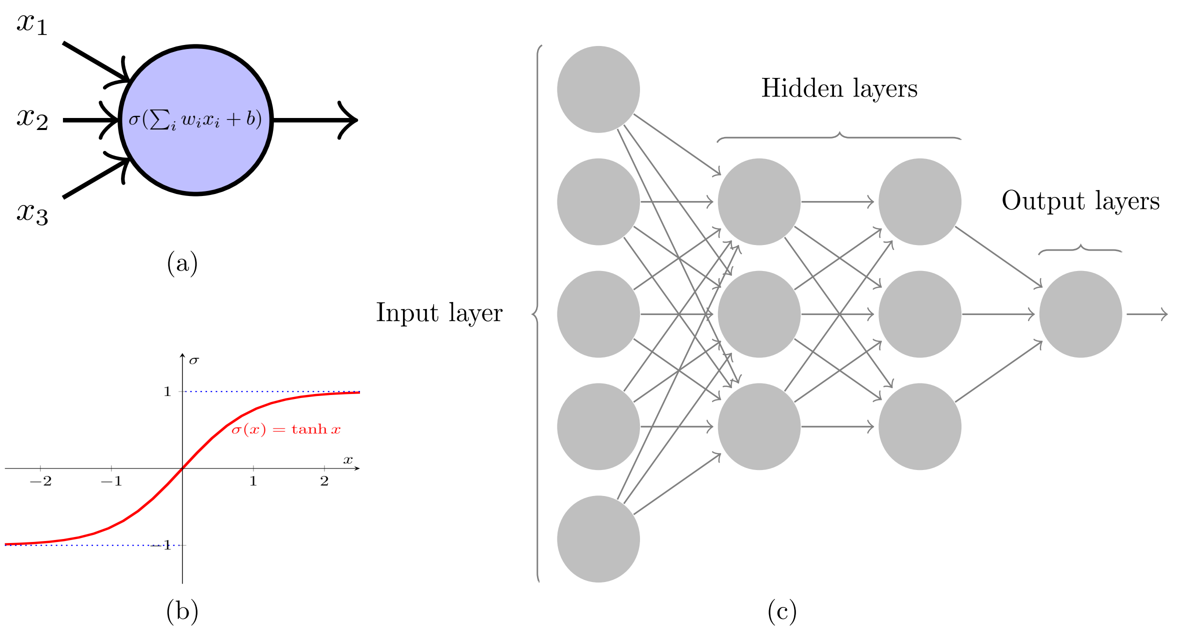


Figure 1: Neural Networks (a) A single neuron taking a vector of inputs and returning a singular output based on the weights, bias and activation function of the network. (b) The hyperbolic tangent used as an activation function. (c) A fully connected neural network containing two hidden layers that performs a mapping of an input vector to a singular output. **MICHAEL: Equation in the neuron looks very small**

make accurate predictions without the need for additional programming . A common approach in ML relies on the computer learning on past experience to make decisions on future events. The hierarchical nature of this approach means that the machine can build complicated concepts from simpler ones. This branch of AI is called deep learning. Neural networks are the quintessential machine learning algorithm that aims to approximate a function. They are built from many single processing units called neurons. The simpliest Neural Network is the perceptron layer Fig. 1a which holds a single neuron that takes several real inputs $x_1, ..., x_i$ and maps them to an output according to the following linear equation:

$$\sigma(\sum_i w_i x_i + b) \tag{1}$$

where w and b are the weights and bias and σ denotes the activation function. The weights are numbers which can be thought as the strength between connections. The output of a neuron is called its activation. Different choices of activation functions allow the user to simulate various models. A common example of an activation function is the hyperbolic tangent Fig. 1b that is used when the outputs of the network must be both positive and negative and re-scaled to $[-1, 1]$. It is often useful to introduce a bias, b, such that the neuron remains inactive above zero but is active when the sum reaches a defined threshold. This bias is added before the activation function and it tells us how high the weighted sum needs to be before the neuron becomes active.

The output of a single neuron is defined by Eq. (1) and gives a prediction $\hat{y}$ that can be compared to the real value y through a cost or loss function. If the loss is non-zero, the network must work to minimise this function by updating the weights in the negative direction of the loss gradient in a process referred to as gradient descent. This loss function plays a critical role in how neural networks learn.

A neural network contains many single neurons connected in a layered structure as shown in Fig. 1c. The activations of the first layer (or input layer) act as the inputs to the second layer and so on until the output layer. Multilayered neural networks have intermediate layers between the input and output stages dubbed the hidden layers as the computations performed are not accessible to the user. The network can be thought of as a function F: $\mathbb{R}^N \rightarrow \mathbb{R}^M$ reliant on how activations from one layer bring activations in the next. The system is analogous to biology, where, some groups of neurons in animal brains cause certain others to fire.

2.2. GANs

A subset of deep learning that has seen fruitful development in recent years is generative adversarial networks (GANs) [15]. These unsupervised algorithms learn patterns in a given training data set using an adversarial process. The generations from GANs are state-of-the-art in fields such as high quality image fidelity [16, 17], text-to-image translation [18] and video prediction [19] as well as time series generations [20].

GANs 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 $\mathbf{z}$, that is usually sampled from a n-dimensional Gaussian space known as the latent space, 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. **JORDAN: note to myself: This is important since eqn 2 is derived from it (thats where the logs come from)** The training process of a GAN alternatively updates the weights 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 the generator encourages the discriminator to update its classification abilities. GANs seek to find an equilibrium between the generator and discriminator in the form of a two-player game that can be summarised by the following:

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

where, V is an objective function that is to be optimized. Objective functions are a general term for either a loss function (to be minimized) or a negative loss function (to be maximised). D is the discriminator, G is the generator and $\mathbf{x} \sim p_{\mathrm{r}}(\mathbf{x})$, $\mathbf{z} \sim p_{\mathrm{z}}(\mathbf{z})$ are samples taken from the training data and generated data respectfully. In practice this approach was found to be unstable. Early on in training the generators weights are initialised randomly meaning that the early generations are noisy and uninformative. This means that the discriminator can easily learn to spot the fake generations and so the discriminator wins and the generator cannot continue to improve. To overcome this the framing of the generator is often changed. Rather than minimizing $\log(1-D(G(\mathbf{z})))$ the objective now is to maximise $\log(D(G(\mathbf{z})))$, that is, to maximise the probability of the generations being predicted as real. The idea being that we always want the loosing side to be able to leverage from its competitor.

In theory, the adversarial process will eventually lead to the local Nash equilibrium [21] whereby both neural networks are trained optimally. In practice, however, GANs are notoriously difficult to train. Such difficulties include: Non-convergence, where the model parameters oscillate and the loss never converges, mode collapse where the generator produces a limited diversity of samples, and diminishing gradients when applying gradient descent to a loss function that is discontinuous.

2.3. 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) [22] was introduced by feeding in extra information into the generator and discriminator such as a class label or attribute label, c. The objective function is then modified:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x},\mathbf{c}\sim p_{\mathrm{r}}(\mathbf{x},\mathbf{c})}[\log D(\mathbf{x},\mathbf{c})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathrm{z}}(\mathbf{z}),\mathbf{c}\sim p_{\mathrm{r}}(\mathbf{c})}[\log(1-D(G(\mathbf{z},\mathbf{c})))] \quad (3)$$

This simple addition has shown to work well in practice, for instance in image-to-image translation [**?**]. We will be using a conditional GAN for this study.

3. Methodology

Table 1: Burst training parameters

Waveform	Central frequency (Hz)	Decay (s)	Central time epoch (s)	Mass range ($\mathrm{M}_\odot$)
Sine-Gaussian	70 - 250	0.004 - 0.03	0.4 - 0.6	N/A
Ringdown	70 - 250	0.004 - 0.03	0.4 - 0.6	N/A
White-noise burst	70 - 250	0.004 - 0.03	0.4 - 0.6	N/A
Gaussian pulse	N/A	0.004 - 0.03	0.4 - 0.6	N/A
BBH	N/A	N/A	N/A	5 - 70

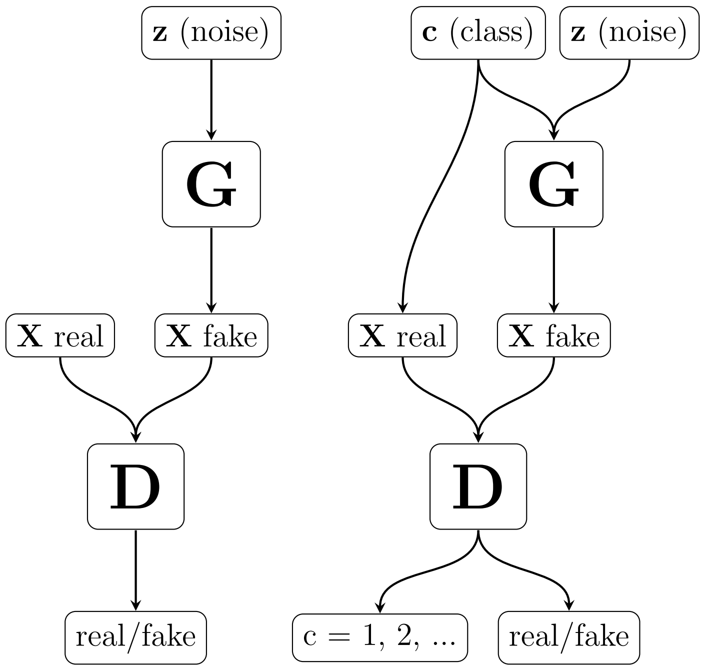


Figure 2: Comparison of the original GAN method and the Auxiliary Conditional-GAN method. For ACGANs the training data requires a label denoting its class that is also fed to the generator which then learns to generate waveforms based on the input label. Additionaly, the discrminator learns to classify which class the signal belongs to. **JORDAN: change to cGAN**

GW burst signals remain an unmodelled phenomenon, as such, current detection algorithms focus on waveforms signals appearing within multiple detectors We propose a signal generation scheme using GANs trained on burst-like waveforms. BurstGAN is a conditional GAN that is trained on five signal morphology's spanning a range of prior parameters. The parameter ranges can be seen in Table 1 The families are: sine-Gaussian $h(t) = A \exp\left[-(t-t_0)^2/\tau^2\right] \sin(2\pi f_0(t-t_0))$, a sinusoidal wave with a Gaussian envelop characterised by a central frequency, f_0; Ring-down, $h(t) = A \exp\left[-(t-t_0)/\tau\right] \sin(2\pi f_0(t-t_0))$, with frequency f_0 and duration τ, white-noise bursts, with bounded frequency bandwidth Δf and duration τ, gaussian pulse $h(t) = \exp(-t^2/\tau^2)$ with duration τ and binary-black hole inspirals which are simulated using IMRPhenomD waveform [**?**] routine from LALSuite [26] which models the inspiral, merger and ringdown of a binary black hole (BBH) waveform. The component masses lie in the range of [5,70] $M_\odot$ with zero spins and we fix $m_1 > m_2$ **MICHAEL: I think the exponentials look neater without using frac{}{}**. The mass distribution is approximated by a power law with index of 1.6 [**?**]. The 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 [0.4,0.6]s of the 1s time interval.

All training waveforms are sampled at 1024 Hz.

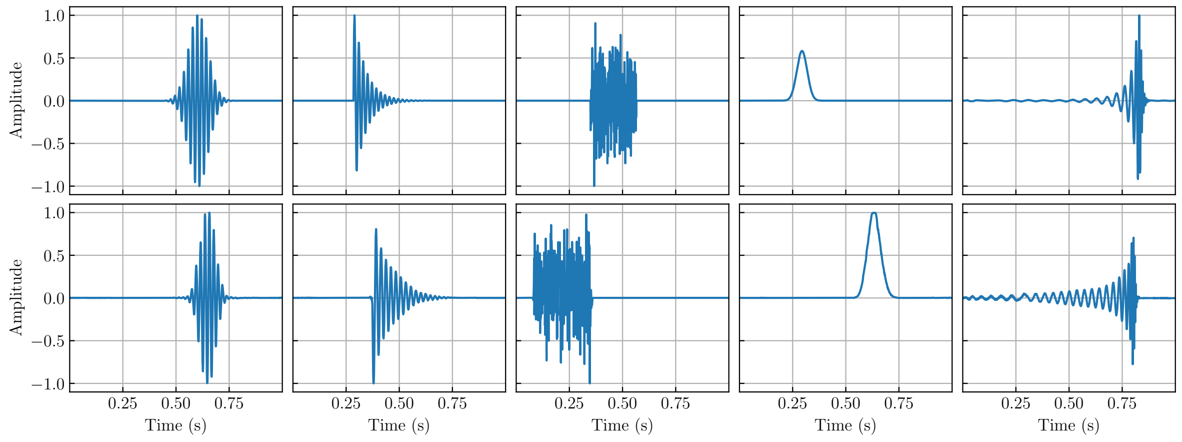


Figure 3: Examples of simulated GW burst signals. Top row shows examples from the training set. From left to right: Sine-Gaussian, Ringdown, White-noise burst, Gaussian pulse, Binary black hole merger. The bottom row shows the conditional generations from the GAN. Each plots contains only one detector output for ease of viewing.

3.1. Applying a time shift and antenna responses

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 H1 and L1 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 takes as input, the generated waveform, time delay and antenna responses. The sky location is chosen by drawing from a uniform distribution across the whole sky and the time delay, and antenna responses generated by LALSimulation [26].

This response layer Fourier transforms the waveform to the frequency domain, multiplies this by $e^{2\pi i f \Delta t}$, where Δt is the time shift and transforming back to the time domain. After which, both original and shifted signals are multiplied by their respective antenna responses. The output is two 1024 Hz times series waveform split into two channels. This scheme is also used in generating the training set.

3.2. Architecture details

Extensions to the original GAN method such as the deep convolutional generative adversarial network (DCGAN) [24] have been widely praised for enabling a stable GAN architecture. Most GANs now replace fully connected layers, where each neuron is connect to all the neurons in the previous and next layer, with convolutional layers. convolutional neural networks (CNN) are designed to work with grid-like structures with close local dependencies like image and text based data, however, there are examples of audio synthesis work [25].

We adopt the suggestions of [24, 25], 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. The full architecture description can be seen in Table A1. **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 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 comes from the use of trial and error and the initial designs influenced by the available literature. After tuning the multiple hyperparamters (Table A1), the GAN was trained for 1000 epochs and takes O(2) days to train. **JORDAN: talk about making D and G 'symmetric' increasing batch size, number of filters powers of 2, spatial dropout,no early stopping. 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?**.

3.3. 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?**. **JORDAN: each class is one number that is turned into 120 numbers, embedding keeps track of the 5x120 numbers and as its training similar classes are grouped together in the class space. This explains it better than i can: https://machinelearningmastery.com/what-are-word-embeddings/** 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.

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 Fig. 4. 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

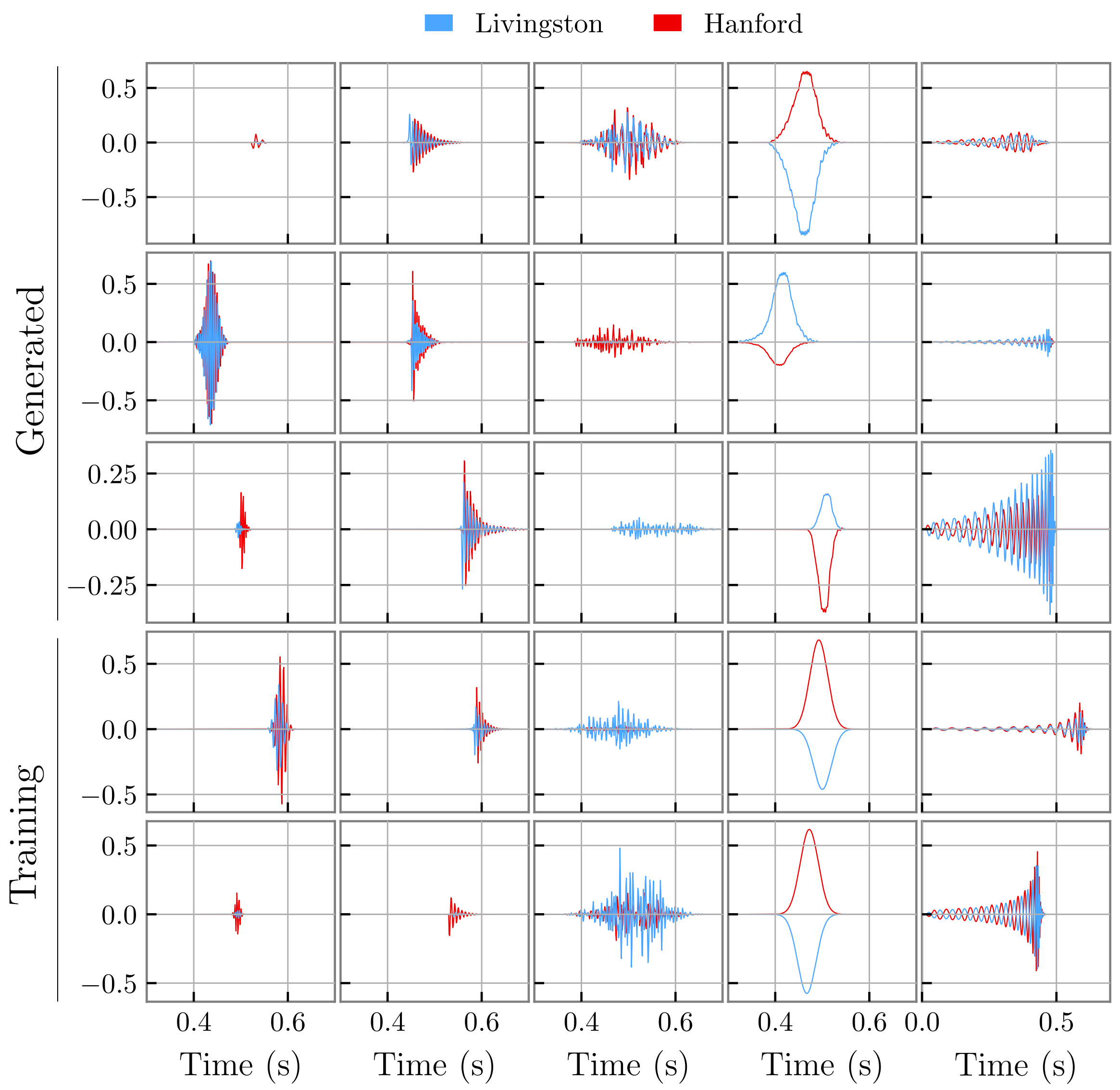


Figure 4: Generated waveforms after training and conditioning on five classes. The generator will output two waveforms as seen by detectors in Hanford (red) and Livingston (blue). The generator is able to capture the characteristics of each waveform and structure the class space to give control over which waveform to generate. Each row shows a random assortment of one of the five classes the GAN is trained on.

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 (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 Fig. 5. 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 ringdown 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.

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.

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 use in the arithmetic. In Fig. 8 we generate a whitenoise burst and a BBH insprial using their respective class labels and use a randomised latent point and response. Keeping

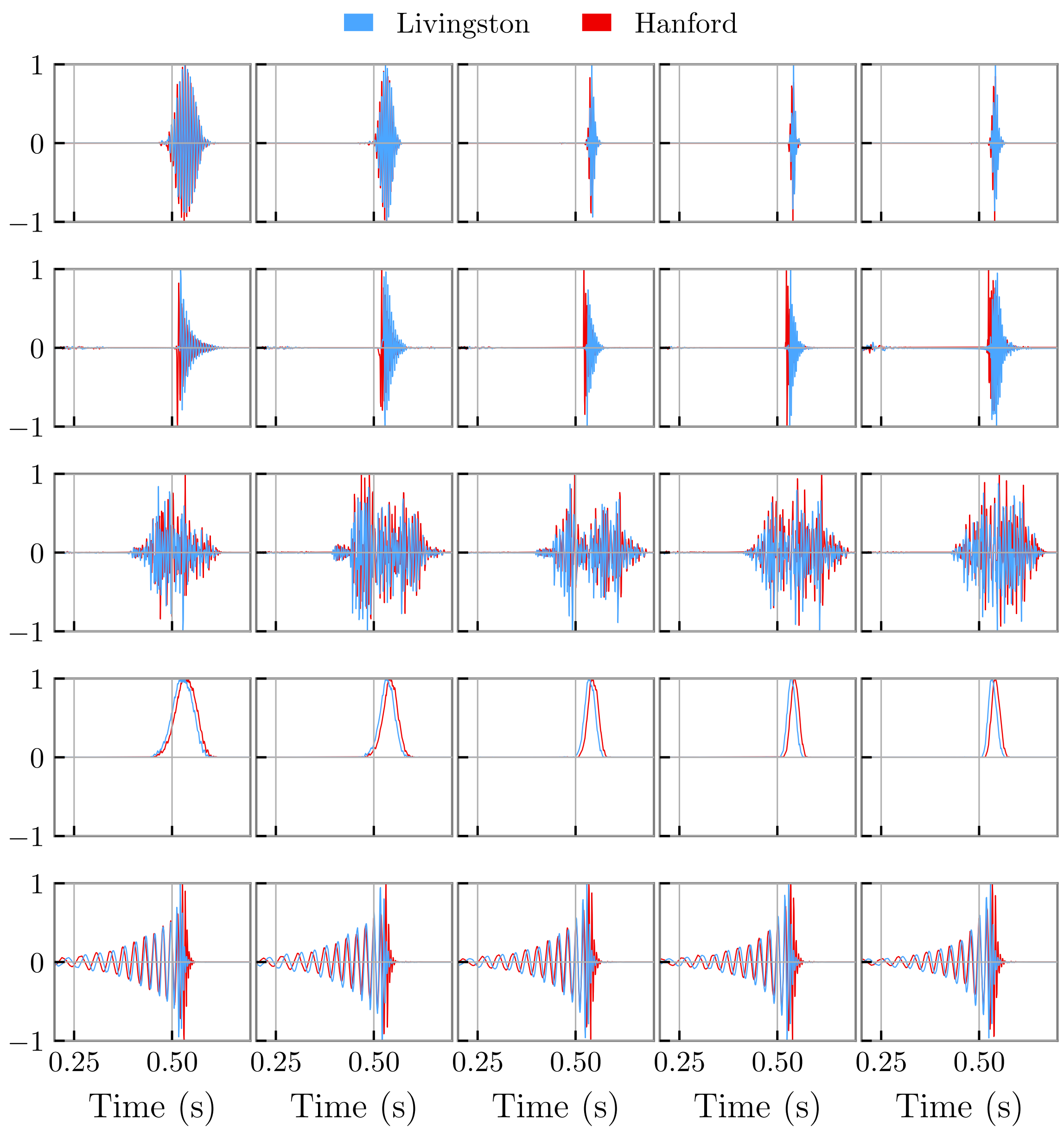


Figure 5: Interpolations between two random latent points in z. Each row uniformly interpolates between two points in **z** keeping the class fixed. Only a single waveform from the generator is plotted and each signal is re-scaled to [-1,1] effectively removing the antenna responses for clarity.

the latent point and response, we average the two class vectors and use this as input to the generator. Figure 8 (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

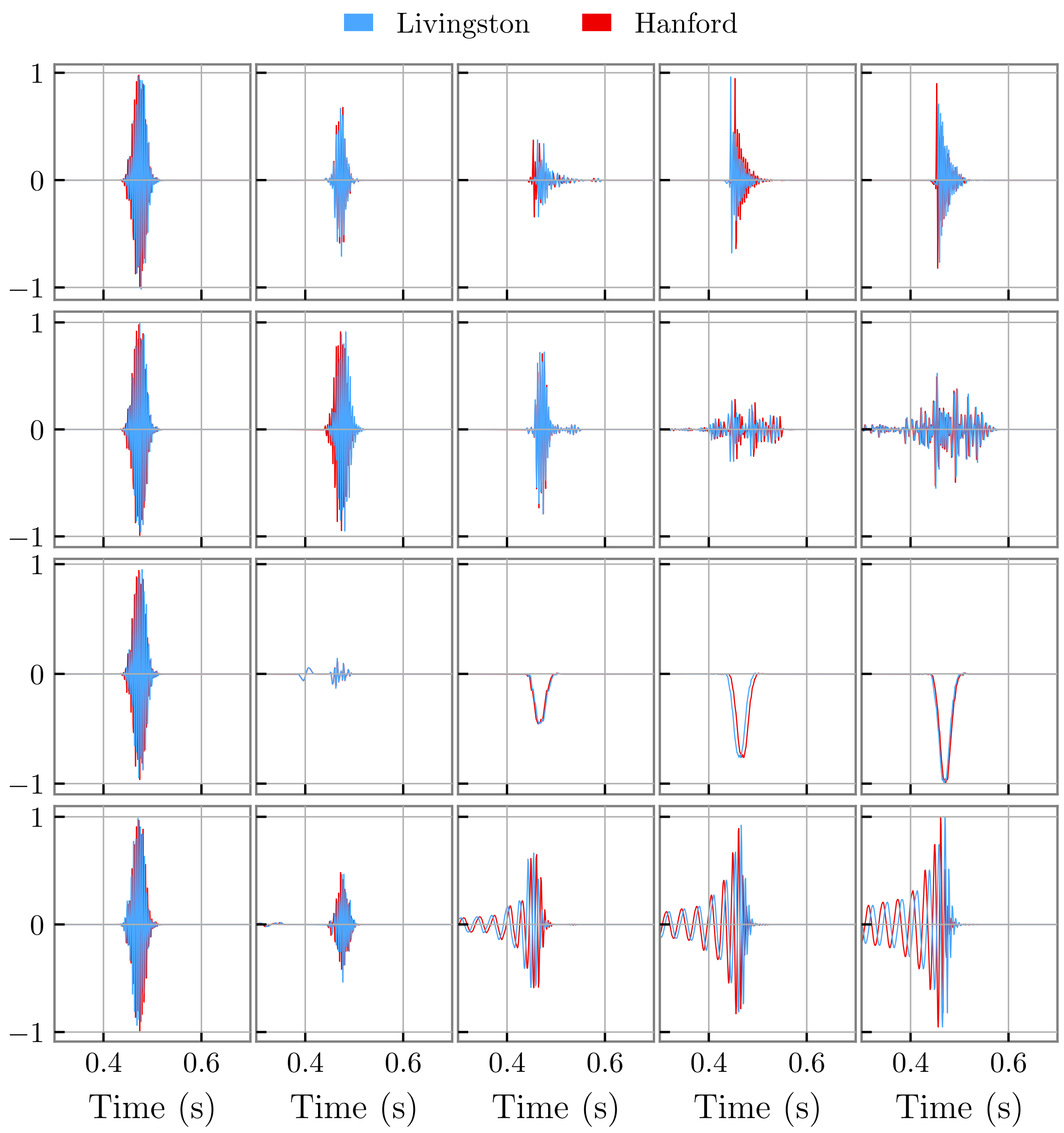
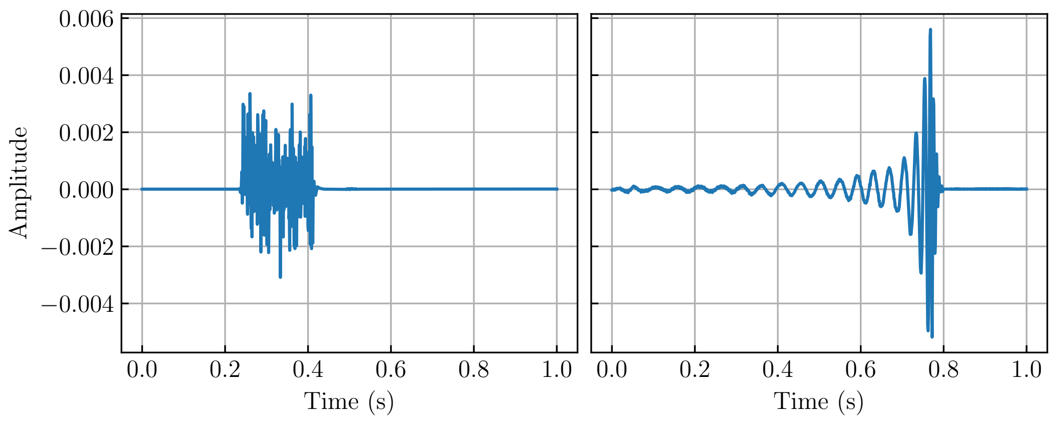
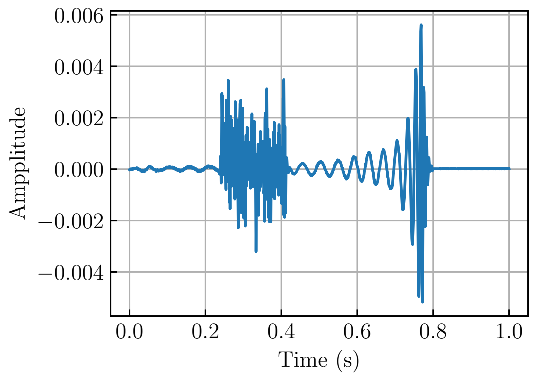


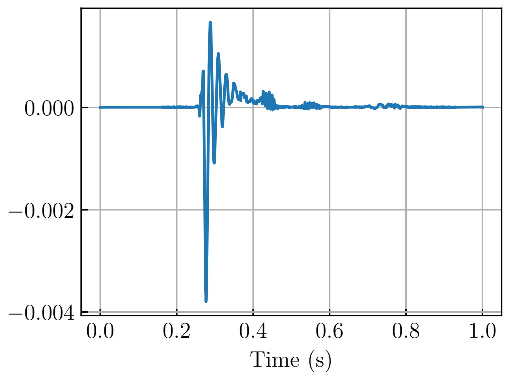
Figure 6: Class space interpolation with latent space held constant throughout. The plots show a zoomed in section of the 1s time interval. First row: Sine-Gaussian class to ringdown class, Second Row: Sine-Gaussian class to whitenoise burst class, Third row: Sine-Gaussian class to Gaussian class, Fourth row: Sine-Gaussian to BBH class.



(a)



(b)



(c)

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.

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

[1] Abbott B *et al.* 2016 *Physical Review Letters* **116** 061102 ISSN 0031-9007
[2] Abbott B *et al.* (KAGRA, LIGO Scientific, VIRGO) 2018 *Living Rev. Rel.* **21** 3 (*Preprint* `1304.0670`)
[3] Aasi J *et al.* (LIGO Scientific) 2015 *Class. Quant. Grav.* **32** 074001 (*Preprint* `1411.4547`)
[4] Harry G M (LIGO Scientific) 2010 *Class. Quant. Grav.* **27** 084006
[5] Acernese F *et al.* (VIRGO) 2015 *Class. Quant. Grav.* **32** 024001 (*Preprint* `1408.3978`)
[6] Abbott B *et al.* 2016 *Physical Review Letters* **116** 241103 ISSN 0031-9007
[7] Abbott B P *et al.* 2017 *The Astrophysical Journal Letters* **851** L35 ISSN 2041-8205
[8] Abbott B *et al.* 2017 *Physical Review Letters* **118** 221101 ISSN 0031-9007
[9] Abbott B *et al.* 2017 *Physical Review Letters* **119** 161101 ISSN 0031-9007
[10] Owen B J and Sathyaprakash B S 1998 *Matched filtering of gravitational waves from inspiraling compact binaries: Computational cost and template placement* (*Preprint* `9808076`)
[11] Klimenko S *et al.* 2008 *Classical and Quantum Gravity* **25** 114029
[12] Aso Y *et al.* 2008 *Classical and Quantum Gravity* **25** 114039
[13] Gabbard H *et al.* 2017 *Matching matched filtering with deep networks in gravitational-wave astronomy* (*Preprint* `1712.06041`)

[14] Gabbard H *et al.* 2019 Bayesian parameter estimation using conditional variational autoencoders for gravitational-wave astronomy (*Preprint* `1909.06296`)

[15] Goodfellow I *et al.* 2014 *Advances in Neural Information Processing Systems 27* ISSN 10495258

[16] Brock A, Donahue J and Simonyan K 2018 Large scale gan training for high fidelity natural image synthesis (*Preprint* `1809.11096`)

[17] Karras T *et al.* 2019 Analyzing and improving the image quality of stylegan (*Preprint* `1912.04958`)

[18] Reed S *et al.* 2016 Generative adversarial text to image synthesis (*Preprint* `1605.05396`)

[19] Liang X *et al.* 2017 Dual motion gan for future-flow embedded video prediction (*Preprint* `1708.00284`)

[20] Esteban C, Hyland S L and Rätsch G 2017 Real-valued (medical) time series generation with recurrent conditional gans (*Preprint* `1706.02633`)

[21] Nash J F 1950 *Proceedings of the National Academy of Sciences* **36** 48–49 ISSN 0027-8424

[22] Mirza M and Osindero S 2014 *CoRR* **abs/1411.1784** (*Preprint* `1411.1784`)

[23] Odena A, Olah C and Shlens J 2016 Conditional image synthesis with auxiliary classifier gans (*Preprint* `1610.09585`)

[24] Radford A, Metz L and Chintala S 2015 *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* (*Preprint* `1511.06434`)

[25] Brock A, Donahue J and Simonyan K 2018 *CoRR* **abs/1809.11096** (*Preprint* `1809.11096`)

[26] LIGO Scientific Collaboration 2018 LIGO Algorithm Library - LALSuite free software (GPL)

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