Matching Matched Filtering with Deep Networks in Gravitational wave Astronomy

Hunter Gabbard,* Fergus Hayes, Michael Williams, and Chris Messenger
SUPA, School of Physics and Astronomy,
University of Glasgow,
Glasgow G12 8QQ, United Kingdom

(Dated: October 11, 2017)

We report a new method for classifying gravitational-wave (GW) signals from binary black holes (BBH) using a deep convolutional neural network. Using only the raw time series as input, we are able to distinguish GW signals injected in Gaussian noise from purely Gaussian noise time series with (**need figure of merit here**) percent accuracy. We compare our results with the standard method of matched filtering used in Advanced LIGO and find the methods to be comparable.

Usage: Secondary publications and information retrieval purposes.

PACS numbers: May be entered using the `\pacs{#1}` command.

Introduction — The field of gravitational wave astronomy has seen an explosion of binary black hole detections over the past several years [**cite detection papers**]. These detections were made possible by the Advanced Laser Interferometer Gravitational wave Observatory (aLIGO) detectors, as well as the recent joint detection of GW170814 with Advanced Virgo [**citation needed**]. Over the coming years many more such observations, including other more exotic sources such as binary neutron star (BNS), intermediate black hole (IMBH), and neutron star black hole (NSBH) mergers, are likely to be observed on a more frequent basis. As such, the need for more efficient search methods will be more pertinent as the detectors increase in *volume* × *time* sensitivity.

The search pipelines used to make these detections [**cite Usman et al. and gstlal folks**] are computationally expensive to run. Part of the reason being that the methods used by these *search pipelines* are complex processes run over a large parameter space using advanced signal processing techniques. Distinguishing noise from signal in this search pipeline, and others like it, is done using a technique called matched template filtering. Matched template filtering uses a *bank* of template waveforms that spans the astrophysical parameter space [**provide shit-ton of citations for this**] because we do not know *a priori* what the parameters of the gravitational waves in the data are. Because the waveforms of the signals are well modeled, the pipeline uses matched filtering to search for those signals burried in the detector noise. More on how we implement this technique in comparisons with our model will be mentioned later in the methods section of this letter.

We propose that a deep learning algorithm which requires only the raw data time series as input with minimum signal processing would be one alternative. This pipeline could be pretrained and then run on real-time detector data with maximum efficiency and in low-latency.

Deep learning is a subset of machine learning which has gained in popularity over the past several years [**cite various successful nn's**]. A deep learning algorithm is composed of neurons which can be anywhere from one to several layers deep. Deep learning algorithms consists of an input layer, followed by one to several hidden layers and then one neuron that outputs a single value. This value can then either be used to solve classfication, or regression-like problems.

In our model, we use a variant of a deep learning algorithm called a convolutional neural network (CNN) [**citation required**]. CNN layers are composed of five primary variants: input, convolutional, ReLu, pooling, and fully-connected. Where input holds the raw pixel values of the sample image, the convultional layer computes the dot product between the kernel and a local region of the input layer volume, ReLu applies an elementwise activation function leaving the size of the previous layer's output volume unchanged, pooling performs a downsampling operation along the spatial dimensions, and the fully-connected layer computes the class scores.

In the following sections we will discuss our choice of network archetecture and tuning of it's hyperparameters, compare the results of our network with the widely used GW signal classification technique called matched filtering, and comment on future improvements related to this work.

Methods — In this analysis, in order to make the problem simple, we only distinguish between BBH signals injected into a Gaussian noise time series and pure white Gaussian noise time series. The time series for both classes of signals are 1s in duration sampled at 8192 Hz. **say more about how noise and injection are generated using Chris's code.**

For our noise signals we generate a power spectrum density (PSD) that is comparable to aLIGO design sensitivity [**perhaps cite lal folks**]. That PSD is then converted to an amplitude time series where a random phase shift is given to each spectral component. The in-

* Corresponding author: h.gabbard.1@research.gla.ac.uk

verse real fast fourier transform (IFFT) is then applied and returns a Gaussian time series.

Injections are made using the IMRPhenomD type waveform [**cite Phenom wf paper**] where the component masses of the waveform range from $5M_{\odot}$ to $100M_{\odot}$, $m_1 > m_2$, and all with zero spin (**not sure if correct?**). The waveforms are then randomly placed within the time series, where the peak of the waveform is within the last 20% of the time series (**perhaps give reason for why this is done**). Each injection is also given a random sky location. The waveform is normalised using the integrated signal-to-noise ratio (iSNR).

In our runs we used $1,000$ Gaussian noise signals and $1,000$ unique injections with over 25 varrying noise realizations resulting in a total of $50,000$ samples. The samples are then arranged in the form of a 1×8192 pixel sample which is scaled by the GW strain amplitude, $h(t)$, over one color channel (grayscale). (**give ligo definition of optimal SNR**) 70% of these samples are used for training, 15% for validation, and 15% for testing.

A detailed description of the neural network archetecture used can be found in Table 1 below.

In order to achieve the optimal network, multiple sets of hyperparameters are tuned. First, we rescaled the data, but with the existing setup, this did not seem to improve upon the performance. We also attempted applying transfer learning where we used networks trained on successively higher SNR values, though performance benefits were minimal. Network depth was adjusted between 2 to 10 convolutional layers. Our initial data set needed at least 4 convolutional layers. Later data sets with various noise realizations needed fewer convolutional layers to perform comparatively well, but adding more layers still seemed to improved performance. The inclusion of dropout was used within the fully-connected layers as a form of regularization.

For updating our weights and bias parameters (in order to minimize our loss function), we settled on the nesterov momentum optimization function. Nesterov momentum was the ideal choice because of its prescient ability to approximate the next position of the weights and bias parameters which gives a rough approximation of their values (**perhaps this is a bit off topic**). Thus the gradient is calculated not with respect to the current parameters, but with respect to the approximate future positions of those parameters.

Matched filtering bit — The template bank was generated using 8000 randomly sampled mass pairs from the same distribution with no adjustment to assure the parameter space was adequately covered.

Results — In this case, we cyclically adjust the learning rate to oscialte between 5×10^{-4} and 1×10^{-3} at a constant frequency. Studies have shown that this policy of learning rate adjustement (**blah blah blah**)

Conclusions —

TABLE I. The optimal network structure (seen below) was determined through multiple tests and tunnings of hyperparameters by means of trial and error. The network consists of 8 convolutional layers, followed by 2 fully-connected layers. Max-pooling is performed on the first, fifth, and eigth layer, whereas dropout is only performed on the two fully-connected layers. Each layer uses an Elu activation function while the last layer uses a Softmax activation function in order to normalize the output values to be between zero and one so as to give a probability value for each class.

	layer 1	layer 2	layer 3	layer 4	layer 5	layer 6	layer 7	layer 8	layer 9	layer 10
Number of Kernals	8	16	16	32	64	64	128	128	64	2
Filter Size	32	16	16	16	8	8	4	4	n/a	n/a
Max Pooling	8	1	1	1	6	1	1	4	n/a	n/a
Fully Connected	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	yes	yes
Drop out	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5
Activation Function	Elu	Elu	Elu	Elu	Elu	Elu	Elu	Elu	Elu	Softmax

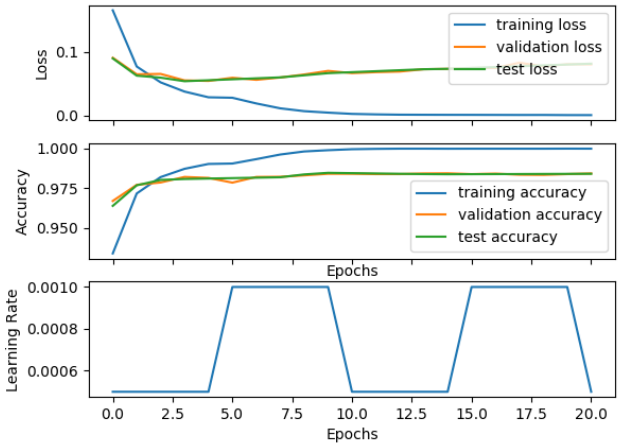

FIG. 1. The loss, accuracy and learning rate plots (shown above) illustrate how the network's performance is defined as a function of the number of training epochs. The goal is to minimize the loss function, which will in turn maximize the accuracy of the classifier. The first initial epochs see an exponential decrease in the loss function and then a slowly falling monotonic curve to follow. This indicates that the longer our network is trained, a limit with respect to the accuracy is approached.

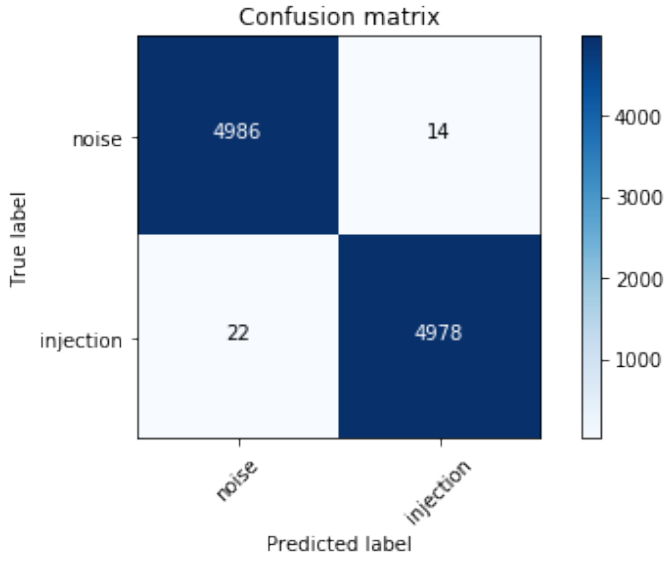

FIG. 2. Say something about confusion matrix here.

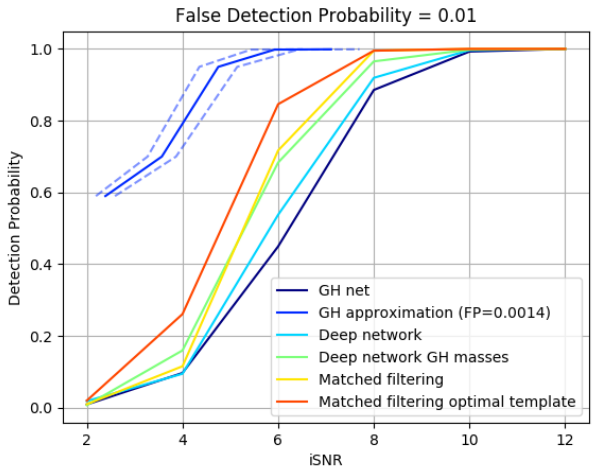

FIG. 3. **Need this plot but without the references to GH.**

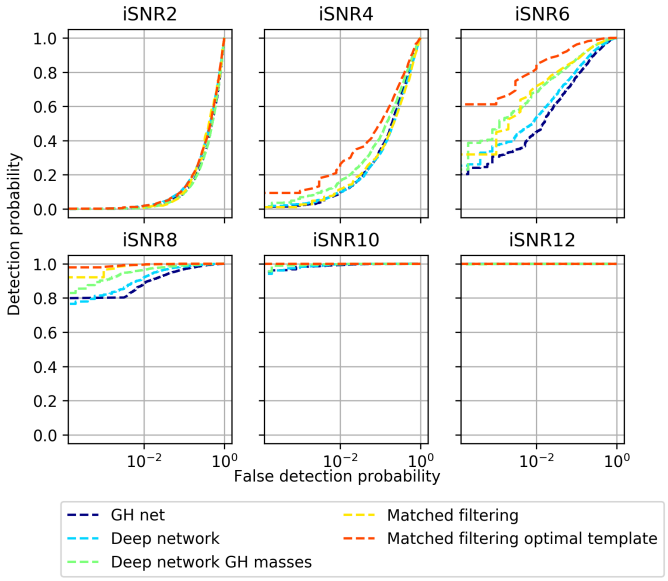

FIG. 4. **Need this plot but without references to GH.**