Draft version August 3, 2018
Typeset using LaTeX **twocolumn** style in AASTeX62

Extending the reach of gravitational-wave detectors with machine learning

Tri Nguyen

ABSTRACT

We apply Long Short-Term Memory (LSTM) Neural Networks as a time-series regression analysis technique to filter instrumental noises from gravitational-wave detectors at LIGO. Unlike traditional neural networks, LSTM networks can store and use information from their past inputs, and thus is robust in handling sequential data like gravitational-wave signals. Once trained on the detector noise data, an LSTM network should be able to learn, predict, and subtract both the linear and non-linear noise coupling mechanisms. This would result in a sensitivity improvement, most greatly at the low-frequency limit 20-100 Hz where noise features are expected to be easier to learn, and allow the detection of gravitational-wave sources currently below the noise floor. In this paper, I discuss our analysis pipeline and current progress.

1. INTRODUCTION

1.1. *Noise Regression Analysis at LIGO*

The Laser Interferometer Gravitational-Wave Observatory (LIGO) is the world's largest gravitational-wave observatory. Adopting a Michelson interferometer design, LIGO detects passing gravitational waves by measuring the induced differential arm length (DARM) between its two perpendicular 4-km arms (Adhikari 2004; Tiwari et al. 2015; LIGO Scientific Collaboration 2007). Since the 2015 sensitivity upgrade (The LIGO Scientific Collaboration 2015), LIGO has observed signals from multiple stellar-mass black-hole and neutron-star mergers (Abbott, B. P. et al 2016). However, the gravitational waves from many of these objects are still lying below the detector sensitivity limit. LIGO's noises are consisted of instrumental and environmental sources, each coupling to the DARM with a different, both linear and non-linear, mechanism. A detailed study of these mechanisms would allow us to filter out these noises, improve LIGO's sensitivity, and make gravitational-wave detections a more routine occurrence.

The current regression method at LIGO is based on the Wiener-Kolmogorov filter, which minimizes the squared error between the DARM channel and the predicted noises from the physical environmental minor (PEM) channels (Tiwari et al. 2015; LIGO Scientific Collaboration 2007). The Wiener filter, however, fails to remove the non-linear contributions. In fact, characterizing and filtering out these non-linear noises has proven to be a challenging process, because the coupling mechanisms are often sophisticated. For example, two or more noise sources may interfere before coupling to the DARM.

1.2. *Neural Networks as a Noise-Filtering Technique*

Despite the complexity of the non-linear coupling mechanisms, a neural network may be able to learn them. Neural Networks is a non-parametric, supervised machine learning algorithm often used in data classification and clustering. Modeled loosely after a human brain, a typical network may consist of up to a few thousands of nodes, each carrying parameters characterizing the features of the data. The network learns by looping over a labeled dataset (called the training set) and minimizing a set loss function via gradient descent. Once sufficiently trained, it is capable of recognizing highly complex patterns, such as language models, stock market, etc.

To filter LIGO's noise sources, we used Long Short-Term Memory (LSTM) networks, a special network architecture that specializes in processing sequential input. A major strength of LSTM networks over traditional networks is their ability to store and use information from their past inputs. As a result, LSTM networks are capable of taking into account the long-term dependencies in the data (Olah 2015). They are frequently used in speech recognition, grammar learning, and even DNA pattern recognition (Karpathy 2015). Given a sufficient amount of training data, an LSTM network should be able to learn, predict, and subtract both linear and non-linear noise coupling mechanisms, leading to an improvement in LIGO's sensitivity.

1.3. *Outline*

This paper is structured as follows. Section 2 discusses the objectives of the network, the metrics to measure its performance, and the data. Section 3 describes the procedures used to build an optimal analysis pipeline.

Section 4 and section 5 discuss the current progress and future plan of the project.

2. OBJECTIVES

2.1. *Noise Coupling Mechanisms*

LIGO's noise sources can be categorized into fundamental and non-fundamental noises (The LIGO Scientific Collaboration 2015). Fundamental noises are those imposed by quantum and statistical mechanics; these include photon shot noise, thermal noise, etc. Because fundamental noises define the baseline sensitivity limit, they can only be reduced by improving the detector design. On the other hand, non-fundamental noises consist of instrumental and environmental effects, such as seismic noise, magnetic noise, beam jitter, etc. They can be removed, given that there exist witness channels monitoring the disturbance.

When a gravitational wave passes through the detector, the measured DARM will be composed of both the gravitational-wave signal and the noises. Consider the simple case of one witness channel $w(t)$, the DARM output is:

$$h_m(t) = h_s(t) + f[w(t)]$$

where h_s is the signal and f is some (usually) non-linear function, called the transfer function, coupling the DARM output and the output of the witness channel. When there are multiple channels, the transfer function f may take in all of them.

The goal of the network is to predict the transfer function f of the non-fundamental noise sources and subsequently subtract them, while keeping the gravitational-wave signals and the fundamental noises intact.

2.2. *Criteria for Success*

Given a witness channel w_i at some time t, the network will predict a transfer function $\tilde{f}$ and the DARM $\tilde{h}(t')$ at some time $t' > t$, where

$$\tilde{h}(t') = \tilde{f}[w_i(t)], \quad t' > t$$

In general, the functional forms of the true function f and the predicted one $\tilde{f}$ are not known. The network is trained by minimizing the mean square error (MSE), the mean absolute error (MAE), or the average spectral density loss (ASD) (refer to 3.4) between the true DARM h and the predicted DARM $\tilde{h}$.

Because both MSE and MAE are sensitive to data preprocessing, they do not provide a meaningful absolute metric to evaluate the network's performance (though they may serve well as a relative metric to compare between different networks). Instead, the performance is evaluated using either the PSD ratio (related to the ASD loss) or the coherence $c(f)$ between the DARM $h(t')$ and the clean DARM $h_c(t') = h(t') - \tilde{h}(t')$. The coherence measures the correlation between these two channels at each frequency, and is computed from their Fourier transforms. It is related to the ideal noise suppression factor $r(f)$ by the relation:

$$r(f) = \frac{1}{\sqrt{1 - c^2(f)}}, \quad 0 \leq c(f) \leq 1 \forall f$$

For example, if the coherence between $h(t')$ and $h_c(t')$ at some frequency is 0.9, the noise reduction factor at that frequency will be approximately 2.3 (M Coughlin 2014). The goal of this project is to achieve a noise reduction factor of 2, corresponding to a coherence of about 0.87.

2.3. *Data*

We perform the analysis on both mock and real gravitational-wave data. Mock data provide a reliable gauge to measure the network capacity, or the ability to learn complex functions. They are generated by coupling white noises with the DARM output by a known, non-linear function. For example, resonant noises are generated using the resonance function:

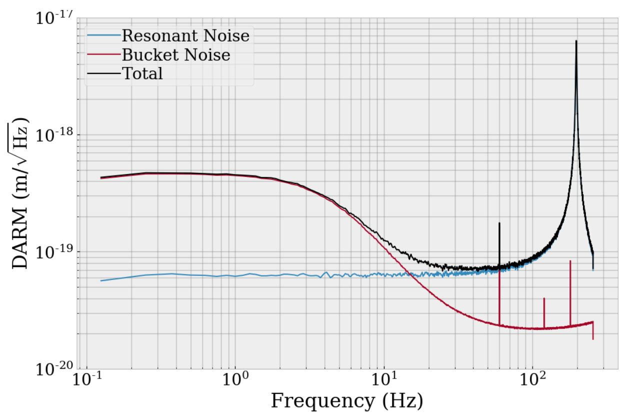
$$y(w) = \frac{x(w)}{w_0^2 - w^2 + i\frac{w_0 w}{Q}}$$

where x is the witness channel, w_0 is the resonant angular frequency and Q is the quality factor describing how under-damped the resonator is. Note though the coupling is in the frequency domain, the analysis is performed on the time domain. Mock data can also be generated from multiple channels. For example, bilinear data are generated by adding the product of angular sensing noise (from ASC channels) and beam spot motion (from beam spot channels) to the DARM. To ensure the network only removes the targeted noises, we inject gravitational-wave signals and other noise sources (called the bucket noises) into the mock, and later check if they are removed or distorted.

Ultimately, however, we want the network to perform well on real data. While we do not know the true coupling mechanisms, we expect the mock data to similar enough to real data and sufficiently capture their complexity. Alternatively, in the DARM power spectral density, there are some known calibration lines. In our data, we check if the network successfully subtracts the calibration line at 7 Hz and the AC power line at 60 Hz.

The analysis in the paper is performed on time series with a duration of 2048 seconds and a sample rate of 512. The length of each time series is then 1,048,576.

3. BUILDING THE NETWORK



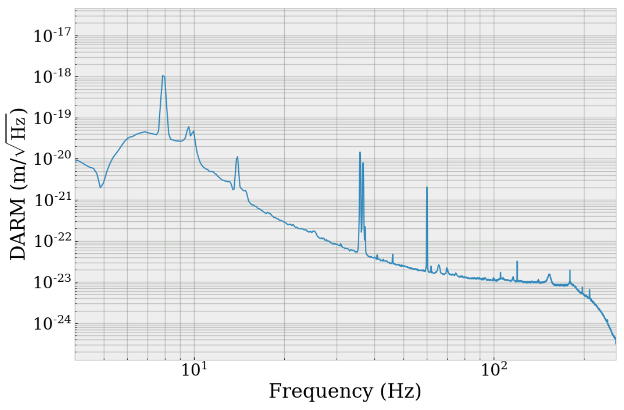


Figure 1. Left: Mock resonant noise spectra with resonant frequency w_0 5 Hz and quality factor Q 100. Right: the DARM spectra on August 14, 2017.

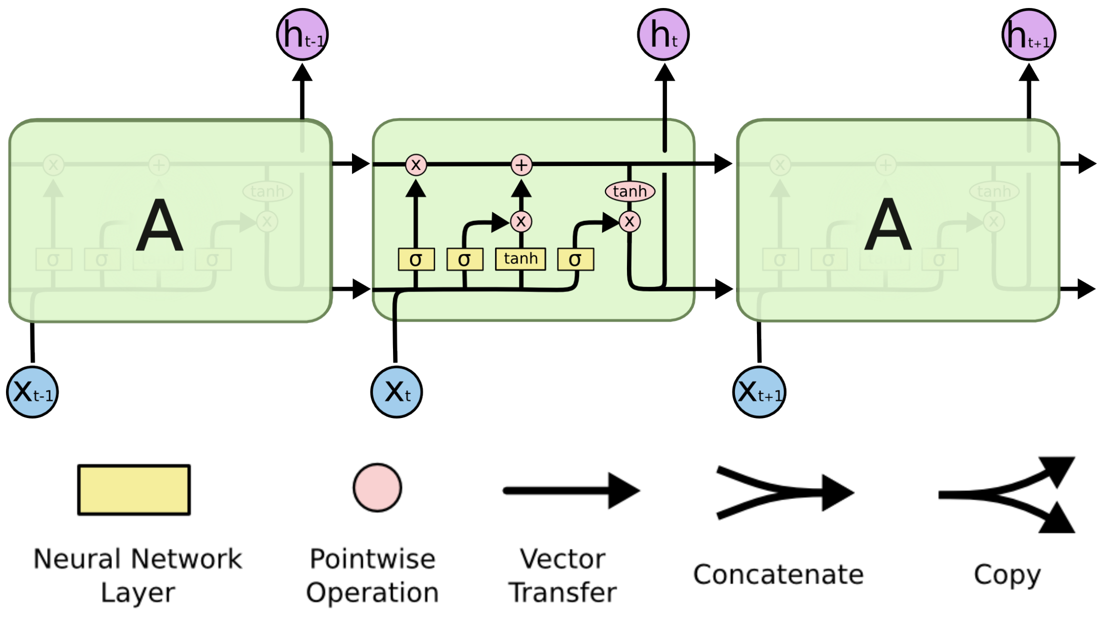


Figure 2. A typical LSTM layer contains four interacting sub-layers. Refer to Olah 2015.

Building an optimal network requires carefully choosing the right architecture and hyperparameters. Hyperparameters are those cannot be learn by the network (e.g. the gradient descent learning rate, the regularization strength, etc.). They govern the learning process, and can greatly affect the optimization and performance of a network. Different networks require different sets of hyperparameters.

Finding the optimal architecture and hyperparameters is an empirical process. In other words, it requires experimenting with different networks and hyperparameters and evaluate their relative performance. In our analysis, we start by picking a simple network architecture (as the training time will be much faster) and increasing the complexity as needed. Initially, we want the network to overfit a small training dataset to make sure it is capable of learning the transfer function. Then, we reduce the overfitting by choosing the optimal hyperparameters via a machine learning process called hyperparameter tuning (or hyperparameter optimization), which will be explained in more details below.

3.1. *Data Preprocessing*

Data preprocessing is one of the most important machine learning processes. It often involves reformatting, cleaning (e.g. removing and/or fixing missing data), resampling and transforming data. In a neural network, data preprocessing helps speed up the learning process, reduces overfitting, and fixes the vanishing/exploding gradient problem in backpropagation.

In our analysis, we down-sample or up-sample the channels so they have a sample rate of 512 Hz. Then, we apply standard scaling and lookback windows.

Standard Scaling—This is the most common form of preprocessing. For each witness channel, we normalize the mean and standard deviation to 0 and 1. Mathematically speaking, for the witness channel j and the data point i is transformed as follows:

$$x_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

where $\mu_j = \frac{1}{N}\sum_k x_{kj}$ and $\sigma_j = \frac{1}{N}\sum_k (x_{kj} - \mu_j)^2$ are the mean and standard deviation over the samples in the witness channel. Here N is the number of samples, which is 1,048,576 as mentioned in section 2.3.

Lookback Windows—Because the length of each channel is large, the network cannot fit all the data into back-propagation memory. Furthermore, the number of parameters is so great that the learning process becomes impossible and computationally expensive. As a solution, we divide each channel into many short and overlapping time series, each consisting of 16 samples. As a result, our dataset has 1,048,546 data examples. In a data example, each witness channel is a time series of length 16.

3.2. *Network Architecture*

LSTM Architecture—As briefly discussed in section 1.2, we use the LSTM network architecture because they can store and use information from past inputs. Figure 2 shows the structure of a typical LSTM layer with four interacting sub-layers. Both the past input and output vectors x_{t-1} and h_{t-1}, along with the current input x_t, are fed into the network to calculate the current output h_t. The input and output x_t and h_t are subsequently used to determine the future output h_{t+1}. For a detailed walk through of the LSTM architecture, refer to Olah 2015.

The hyperparameters of the LSTM architecture are the size of output vector h, the activation function, the bias and weight initializers, etc.

Current Architecture—The current analysis pipeline consists of five separate LSTM networks. Each network uses a different set of hyperparameters and optimization algorithms, and tunes on a different frequency band. The frequency cutoffs are respectively 3-9 Hz, 10-13 Hz, 20-30 Hz, 30-41 Hz, and 57-63 Hz. We employ multiple small networks across multiple frequency bands instead of one big network because they allow us to easily inject and test out new noise sources. For example, injecting a source at 5 Hz only requires us to re-train the first network (3-9 Hz), not the entire pipeline. In addition, each small network is significantly easier and faster to train. Because the networks are trained separately, we may further speed up training via parallel computing on a computer cluster, a process that cannot be done easily if we use one big network.

Each network contains multiple LSTM layers followed by multiple fully-connected (Dense) layers. Between each layer (except for the output layer) is a Batch Normalization layer, which normalizes the mean and standard deviation across each channel to 0 and 1. We found

Layers	Output Shape	# Params
LSTM 1	(N, 16, 16)	1152
Batch Norm 1	(N, 16, 16)	64
LSTM 2	(N, 16, 8)	800
Batch Norm 2	(N, 16, 8)	32
LSTM 3	(N, 16, 8)	544
Batch Norm 3	(N, 16, 8)	32
LSTM 4	(N, 16, 8)	544
Batch Norm 4	(N, 16, 8)	32
LSTM 5	(N, 16, 8)	544
Batch Norm 5	(N, 16, 8)	32
LSTM 6	(N, 16, 8)	544
Batch Norm 6	(N, 16, 8)	32
LSTM 7	(N, 16, 8)	544
Batch Norm 7	(N, 16, 8)	32
LSTM 8	(N, 6)	360
Batch Norm 8	(N, 6)	32
Dense 1	(N, 8)	56
Batch Norm 9	(N, 8)	32
Dense 2	(N, 8)	72
Batch Norm 10	(N, 8)	32
Dense 3	(N, 8)	72
Batch Norm 11	(N, 8)	32
Dense 4	(N, 8)	72
Batch Norm 12	(N, 8)	32
Dense 5	(N, 8)	72
Batch Norm 13	(N, 8)	32
Dense 6	(N, 8)	72
Batch Norm 14	(N, 8)	32
Dense 7	(N, 8)	72
Batch Norm 15	(N, 8)	32
Dense 8	(N, 8)	72
Batch Norm 16	(N, 8)	32
Dense 9	(N, 1)	9

Table 1. Current architecture of each LSTM network, in the case of one witness channel. Here N is the batch size. The lookback window size is 16.

that batch normalization greatly reduces the vanishing gradient problem and makes the network less sensitive to initialization. Each LSTM layer returns a sequence which get fed into the next LSTM layer (with the exception of the last LSTM layer, which returns a vector for the first Dense layer). The full network structure, in the case of one witness channel, is summarized in Table 1. In total, there are 5869 trainable parameters (weights and biases). All networks use the Adaptive Moment Estimation, or adam, optimization algorithm (Kingma & Ba 2014). The biases are initialized as ones, and the initial weights are sampled from a uniform distribution within $(-\sigma, \sigma)$, where

$$\sigma = \sqrt{\frac{6}{n_{in} + n_{out}}}$$

Here n_{in} and n_{out} are the number of features of the input and output vector respectively. This initializer is called the Xavier uniform initializer. Alternatively, the initial weights may be sampled from a Gaussian distribution with mean 0 and standard deviation σ (Glorot & Bengio 2010). This is known as the Xavier normal initializer.

3.3. *Hyperparameter Tuning*

We apply the holdout cross-validation method. In particular, the data are partitioned into a training set and a test set. After each iteration through the training set, the network runs on the test set. As discussed in section 2.2, the test performance is evaluated by computing the MSE, MAE, or ASD between the true DARM and the predicted DARM. This provides an insight on how the network performs on an independent, unknown dataset. We choose the hyperparameters that result in the smallest MSE or MAE. In our analysis, we divide the time series into two equal subsets, with the training set being the first half. The size of the training and test set is 524,288 (as described in section 2.3), which should be sufficient to capture all data features.

To search for the optimal hyperparameters, we apply the algorithms described below:

Random Search—As its name suggested, random search involves randomly sampling the hyperparameters given some prior guesses on them. Scale parameters (e.g. learning rate, learning rate decay, regularization strength, etc.) are sampled from a Jeffreys prior, and location parameters (e.g. dropout, recurrent dropout, etc.) are sampled from a uniform prior. In cases where we also tune discrete parameters like the activation function, the parameter will be chosen with equal probability from a list of possible choices.

At first, the algorithm seems counterintuitive. Why not pick a list of possible value for each parameter and

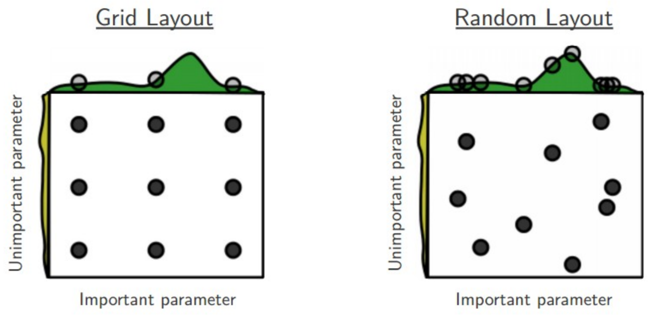


Figure 3. Random search explores the parameter space more efficient than grid search (Bergstra & Bengio 2012).

systematically test out all possible combinations? This method is known as the grid search; it has been shown to be much less efficient in scanning the parameter space. This is because some parameters may greatly affect on the cost function, while others only have a second-order effect. Given the same amount of trials, random search allows the exploration of more values for each parameter (see Figure 3).

Random search is a simple algorithm for hyperparameter tuning. However, there exist more advanced algorithms that employ probability theories to predict the mapping between the parameters and the validation loss. In general, these algorithms decide a set of parameters the network should try based on previous observations. In this analysis, we use the Gaussian Process regression and the Tree-structured Parzen Estimators (TPE).

Gaussian Process Regression—Gaussian process can be thought as a distribution over stochastic (random) functions. For each value of x in the parameter space, the Gaussian process regression assumes the loss function $y = f(x)$, which is a stochastic function whose noises follow a Gaussian distribution with mean μ and standard deviation σ. This is a reasonable assumption for the loss of a neural network because of the many random processes involved in training (e.g. dropout, initialization, etc.). In the case of multiple parameters, the noises follow a multivariate Gaussian distribution. Figure 4 shows an example of the Gaussian process regression for hyperparameter tuning.

Given a prior function distribution and some past evaluations (hyperparameters and losses), the algorithm predicts the loss for each point in the parameter space by constructing the posterior distribution[1]. The network is then tested on the point with the minimum ex-

[1] For a detailed mathematical calculation, please refer to Do & Lee 2008

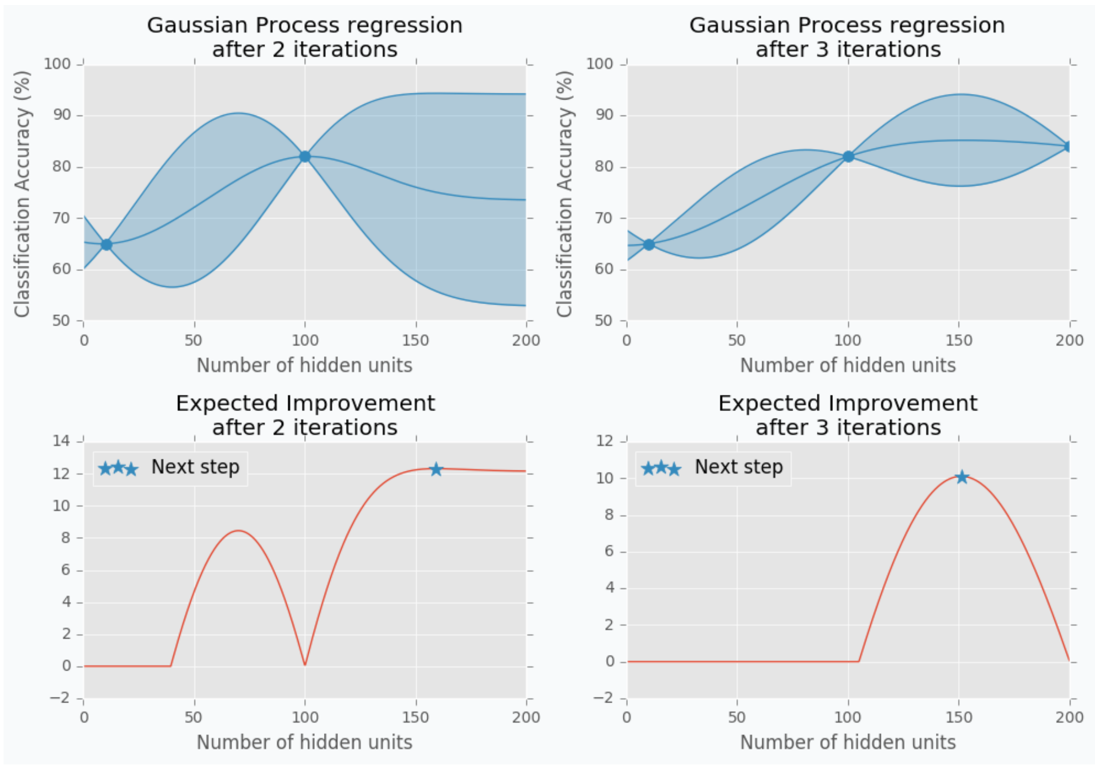


Figure 4. Gaussian process regression for hyperparameter tuning. In this example, the algorithm predicts the number of hidden units that will maximize the validation accuracy of a Dense neural network. Refer to Shevchuk 2016.

pected loss, to within about 2-3 standard deviations of the mean.

The greatest advantage of the Gaussian process regression is the ability to account for the uncertainty of the calculation and the relation between each hyperparameter. However, the algorithm does have some important drawbacks. For example, it is tricky to incorporate categorical parameters (e.g. activation function, initializer, etc.) into the prediction. It is also difficult to select the right hyperparameters for the Gaussian process, such as the prior function distribution. Lastly, when the number of parameters exceed a few dozen, the algorithm becomes computationally expensive (Shevchuk 2016).

Tree-structured Parzen Estimators (TPE)—In TPE, the collected observations are divided into two groups: one with the best validation performance and one with all the others. The algorithm picks the parameter that is most likely belong to the first group and less likely belong to the second group. It does so by constructing the likelihood probability (unlike the Gaussian process, which constructs the posterior probability). In particular, TPE picks the parameter x with the highest expected improvement:

$$r(x) = \frac{l(x|D)}{g(x|D)}$$

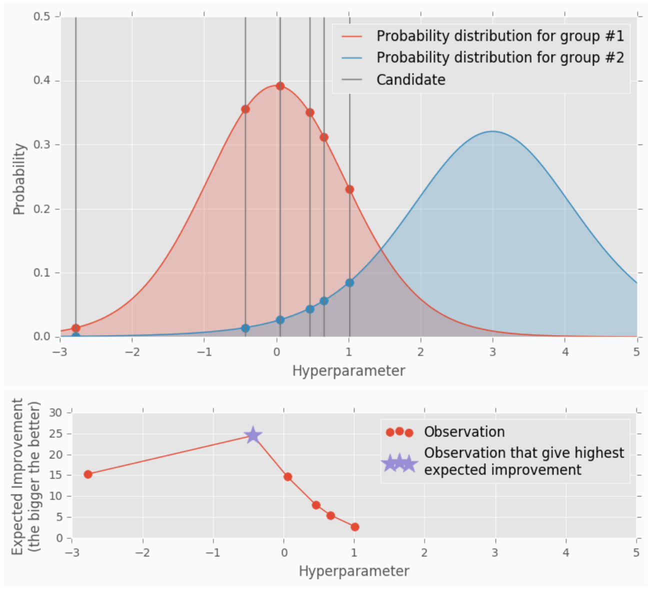


Figure 5. TPE predicts the best parameters by constructing a likelihood ratio of the two groups. Refer to Shevchuk 2016.

where $l(x|D)$ and $g(x|D)$ are the likelihoods of x belong to first and second groups respectively. and $g(x)$ is a probability being in the second group. Figure 5 shows how TPE predicts the next parameters the network should try.

In general, TPE is less sensitive to random noise than the Gaussian process is. It also has fewer hyperparameters, and allows the prediction for categorical parameters more easily. However, the biggest disadvantage of TPE is that it cannot take into account the relation between parameters because it selects the parameters independently.

3.4. *ASD Loss Function*

Currently, the ASD loss function is defined as the mean of the absolute value of the Fourier transform of the difference between the true DARM $h(t)$ and the predicted DARM $\tilde{h}(t)$. Mathematically speaking,

$$\text{loss} = \frac{1}{N}\sum_{i}^{N} |\mathcal{F}[h - \tilde{h}]|(f_i)$$

where N is the number of samples in the series. We may also weight each frequency by the spectra of a compact binary coalescence. In this case,

$$\text{loss} = \frac{1}{N}\sum_{i}^{N} |CBC|(f_i) * |\mathcal{F}[h - \tilde{h}]|(f_i)$$

where $CBC(f)$ is the spectra of the compact binary coalescence.

3.5. *Machine Learning Framework*

Our current machine learning framework for noise regression is DeepClean[2]. DeepClean is written on top of Keras, a high-level Python neural networks API which allows fast and flexible creation and development of neural network models (including Dense, LSTM, Convolution Net, etc.). Keras is a Google Brain's TensorFlow backend. It is able to run across multiple platforms (CPUs, GPUs, TPUs) and thus allows us to speed up training by parallelizing our calculation across GPU or TPU cores.

DeepClean is capable of creating multiple networks, each with a different architecture, to subtract noises across multiple frequency bands. DeepClean trains and tests the networks independently. On each run, it loops over each frequency band and uses the assigned network to perform the subtraction; the final subtraction is a combined result of all networks. Figure 6 shows an example DeepClean output, in which it subtracts the beam jitter noise from LIGO's first observational run O1. DeepClean also has a hyperparameter tuning module, which is still under development. Current progress will be discussed below in section 4.

[2] Available at https://git.ligo.org/rich.ormiston/DeepClean

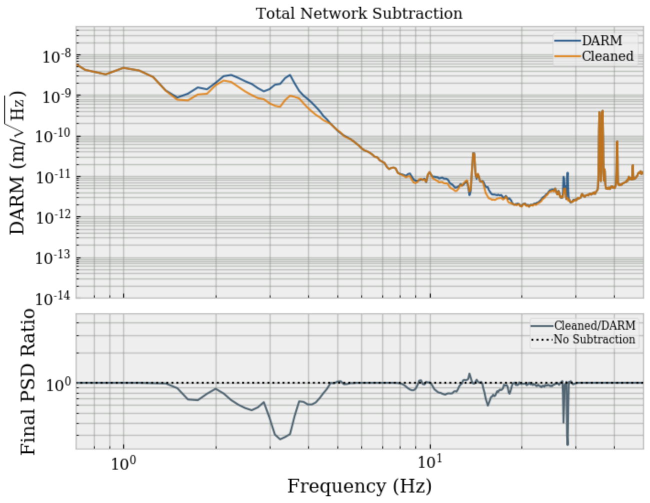


Figure 6. In this example of DeepClean output, the networks subtract the beam jitter noise from LIGO's first observational run O1.

4. CURRENT PROGRESS

4.1. *DeepClean Hyperparameter Tuning*

The majority of my progress has been on developing a hyperparameter tuning module and user interface for DeepClean. The module is at the final stage of completion: the random search and TPE are ready to be used, while the Gaussian process regression is currently under test. It is also worth noting that the Gaussian process currently implemented is not a multivariate Gaussian process; it chooses each parameter independently (like TPE). Future work will be discussed in section 5.

4.1.1. *Progress*

Parameters	Distribution	Values
Activation	Categorical	Tanh, Relu
Bias initializer	Categorical	Zeros, Ones
Weight initializer	Categorical	Xavier uniform, normal
Learning rate	Log-uniform	$x \in (1e-6, 1e-2)$
Learning rate decay	Log-uniform	$x \in (1e-8, 1e-6)$
Dropout	Uniform	$x \in (0.1, 0.9)$

Table 2. Hyperparameter space.

Figure 7 shows the DARM spectral density on August 14, 2017 and the subtraction in the frequency bands 3-9 Hz and 57-63 Hz before and after hyperparameter tuning. The hyperparameter space is summarized in Table 2, and the number of trials is 100. In Figure 7, these frequency bands are highlighted because they include the calibration line at 7 Hz and the AC power line at 60 Hz respectively. In the 3-9 Hz band, tuning results in a bet-

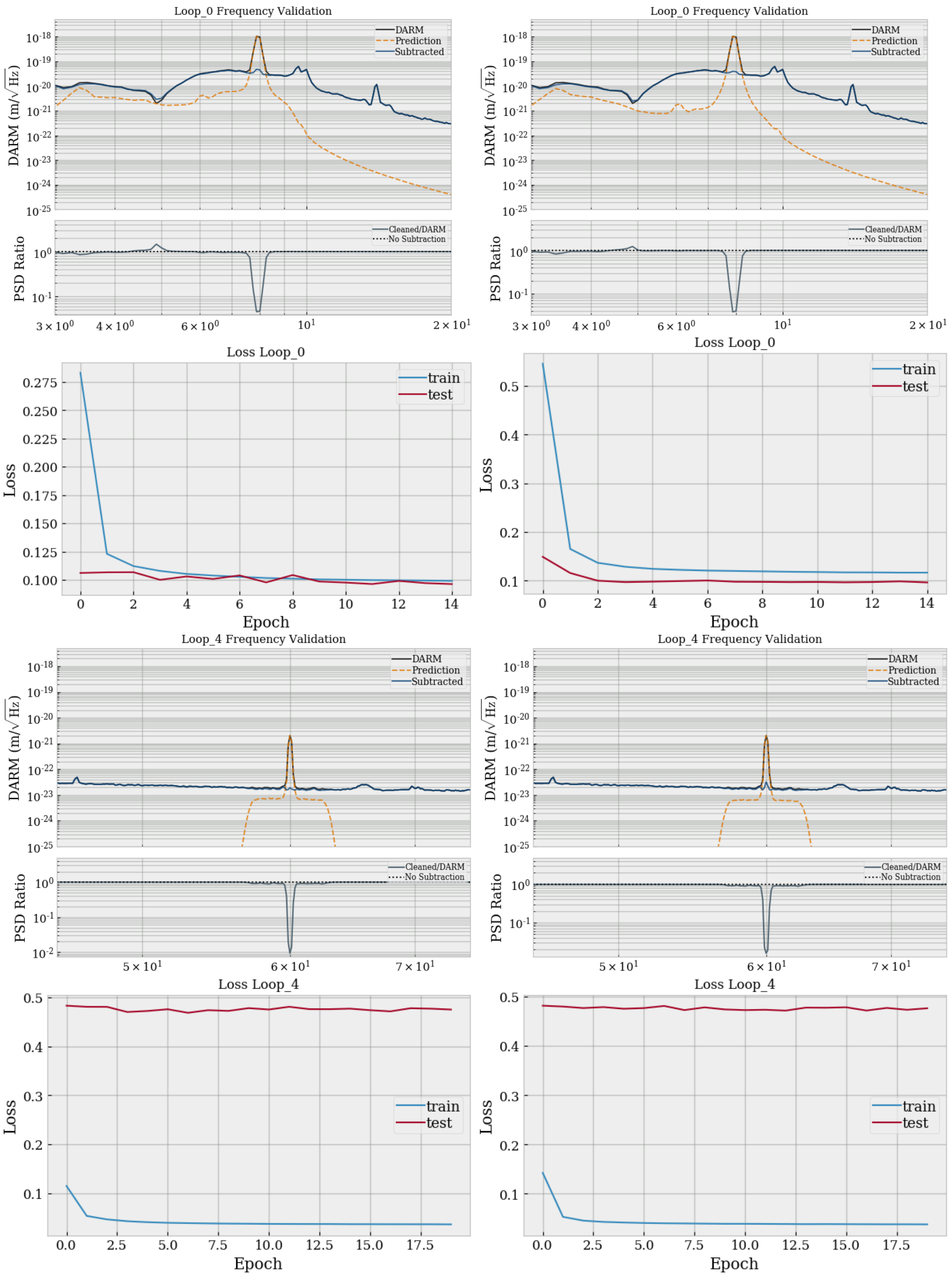


Figure 7. Top: The DARM spectral density on August 14, 2017 before (left) and after (right) hyperparameter tuning. The frequency band is 3-9 Hz. In this case, hyperparameter tuning results in a better subtraction of the calibration line at 7 Hz. Bottom: The DARM spectral density on August 14, 2017 before (left) and after (right) hyperparameter tuning. The frequency band is 57-63 Hz. In this case, hyperparameter tuning does not result in a better subtraction of the AC power line at 60 Hz.

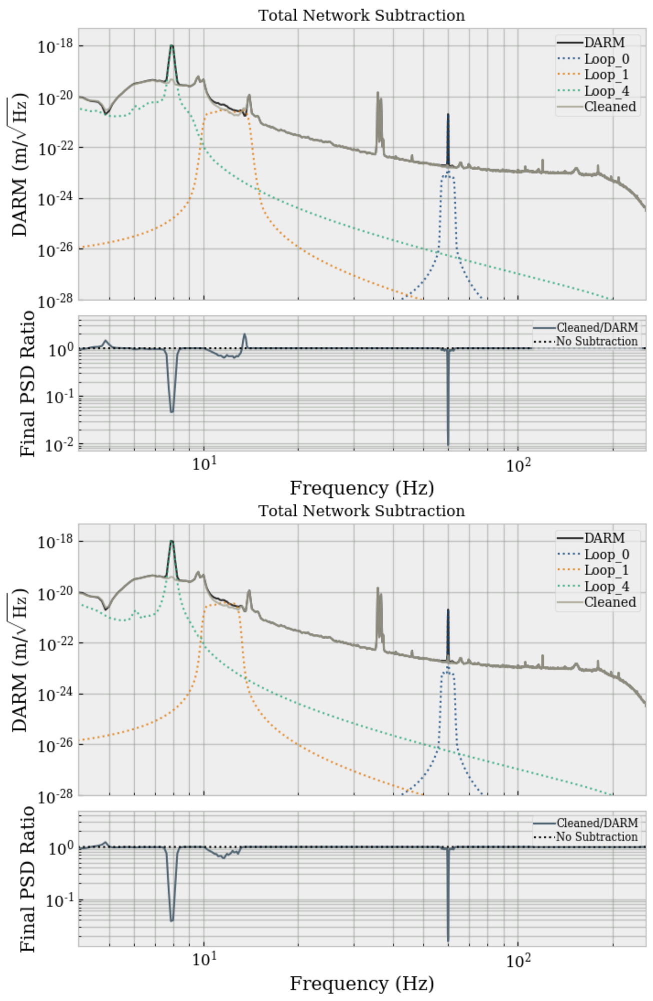


Figure 8. The DARM spectral density on August 14, 2017 and total subtraction before (top) and after (bottom) hyperparameter tuning.

ter subtraction of the calibration line. However, this is not the case for the subtraction of the AC frequency line in the 57-63 Hz band, where the un-tuned network gives a slightly better performance. Furthermore, both networks overfit the training data. The un-tuned network uses the default adam (the gradient descent algorithm) parameters, which are known among the machine learning community to perform well on many problems. In addition, because the parameter space is large, searching over only 100 sets of parameters might not be sufficient. Figure 8 shows the total subtraction before and after tuning. Note that the frequency band of the intermediate network (Loop_1) is changed from 10-14 Hz to 10-13 Hz in the latter case to avoid collision with a spectra line at about 14 Hz (which causes the subtraction to add noises to the DARM).

4.1.2. *Challenges*

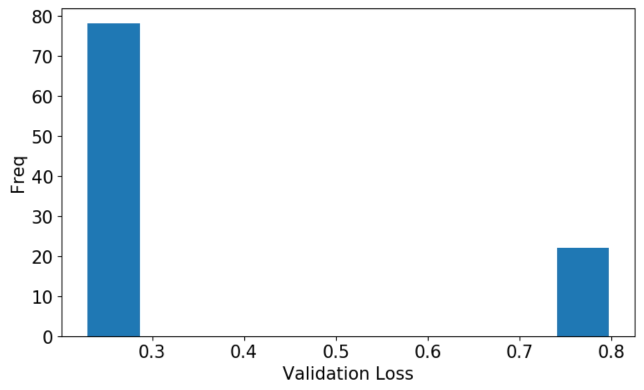


Figure 9. Validation loss distribution of 100 identical runs is bimodal. Approximately 1/5 of the network fails to converge within 10 epochs.

Initially, the hyperparameter module did not share the same data preprocessing procedure with DeepClean running module. In particular, the hyperparameter module did not scale the standard deviation of the time series to one (this process helps training to converge faster). As a result, the optimal parameters suggested by tuning were not the optimal parameters for the running module, and the subtraction failed spectacularly.

As mentioned above, it is possible for the subtraction to add more noise to the DARM. I suspect this is due to the network's attempt to subtract a noise feature which cannot be subtracted. If this is the case, then the frequency bands must be chosen more carefully such that they do not include any non-removable line.

4.2. *Network Instability and Batch Normalization Layers*

4.2.1. *Problems*

The initial architecture of the network does not include batch normalization layers. As a result, the network becomes highly sensitive to the initial weights: given the same dataset and hyperparameters, it may give completely different loss value in two different runs. The loss distribution after 10 epochs is bimodal, as shown in Fig. 9. Moreover, as evidenced from the training and testing loss curve in Fig. 10, once the network starts to get out of a local minimum or saddle point, the gradient descent will always converge. This further shows that the network is highly sensitive to the initial weights. Approximately 1/5 of the network fails to converge within 10 epochs.

Although the underlying problem remains unclear, I suspect the network has run into a vanishing gradient problem as the architecture consists of a large number of LSTM and Dense layers. In other words, the network

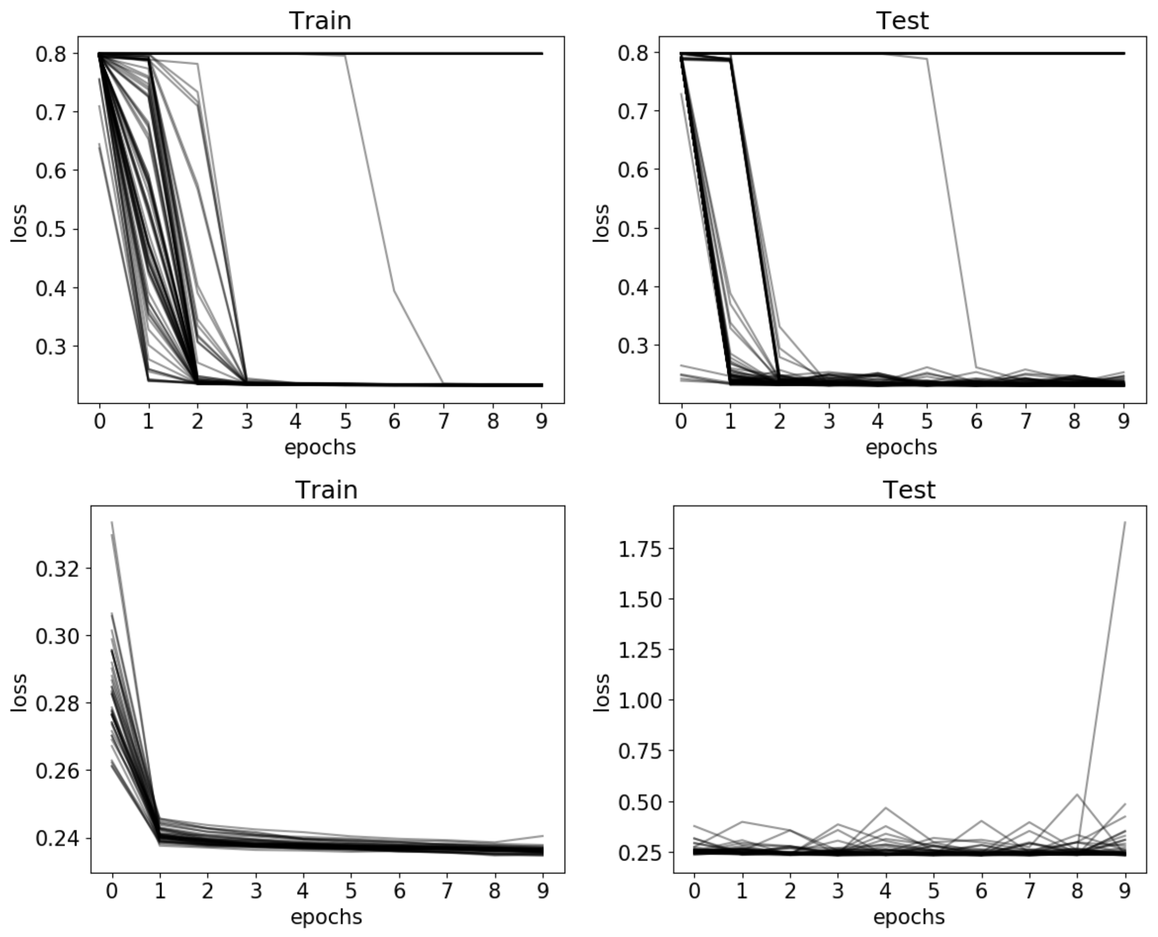


Figure 10. The training loss (left column) and validation loss (right column) per epoch of 100 runs, before (top) and after (bottom) adding batch normalization.

is too deep, so the gradients of the first few layers vanish during backpropagation.

4.3. *Solutions*

Adding a batch normalization layer after each LSTM/Dense layer eliminates the problem. Once the input of each layer is normalized, the data becomes easier to learn because the weights can no longer reach extreme, too small or too high, values (which will result in extreme values of the gradient). Even better, the training process takes much fewer epochs to converge (see Fig. 10). This also shows that the network has indeed run into a vanishing gradient problem.

I added in total 16 batch normalization layers. Each layer has about 16-32 trainable parameters, so the number of parameters increases by about 200-300 parameters. Although this slows down the training time per epoch, the difference is insignificant relative to the total training time.

4.4. *Resonant Noise*

A part of my work is dedicated to generating resonant noises and applying the subtraction algorithm on them.

4.4.1. *Progress*

Figure 11 shows the subtraction in the 10-250 Hz frequency band and the expected outcome. The resonant frequency is about 13.15 Hz, and the quality factor is 1000. The network has successfully subtracted the resonant noises and left out the 60-Hz AC power line and the rest of the bucket noise, which are not supposed to be removed. There is currently no tuning result on this data.

4.4.2. *Challenges*

Initially, the normalization of the resonant noise did not match the scaling of the bucket noise. As the result, either the resonant noise or the bucket noise dominated the spectra. In both cases, the networks failed to produce any meaningful results.

4.5. *Selecting Witness Channels: Ablation Study*

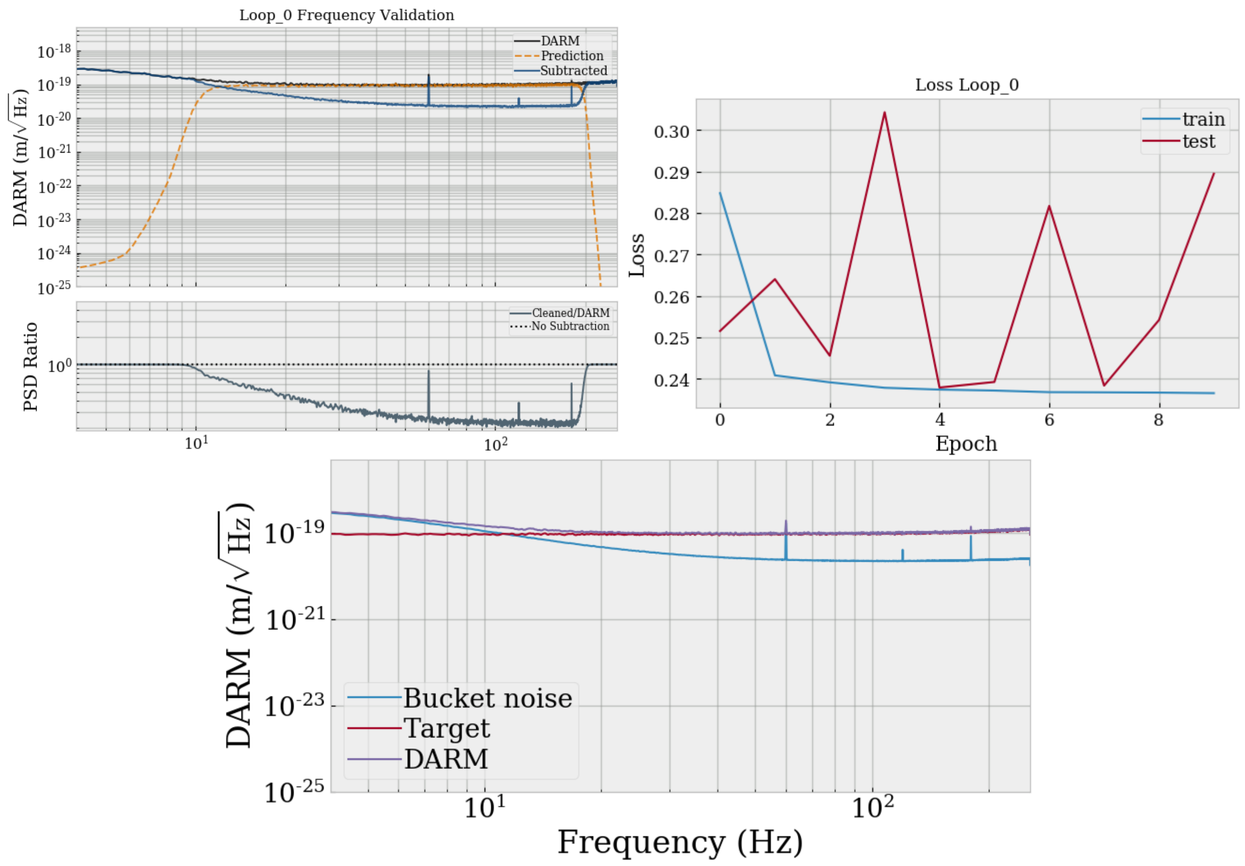


Figure 11. Top: The subtraction in the 10-250 Hz band (left) and the loss curve (right). The subtraction has successfully left out the bucket noise. Bottom: The expected outcome. The spectra is dominated by the resonant white noise. The resonant frequency is about 13.15 Hz, and the quality factor is 1000.

Not all witness channels carry the same weight. Some are more important for the subtraction, while some only have a second-order effect. Furthermore, some channels may share the same, correlated information (degeneracy). Currently, the data include a selected few channels. Ultimately, we would like to understand which channels are important and should be included. We cannot simply use all channels because the networks will become too hard to train (too many features). More importantly, training will become too expensive in both running time and memory. Despite the lack of understanding of the underlying physics, we can still study the importance of each channel via an ablation study. In machine learning, an ablation study is the process of removing some features of the data and seeing how it affects the performance. After the network is trained, I set each witness channel in the validation data to zero and compare the subtraction before and after.

4.5.1. *Progress*

Fig. 12 shows the result of the ablation study on the H1 August data in the 10-15 Hz frequency band. Here, the witness channel "ASC-CHARD_Y_OUT_DQ" is set to zero, and the subtraction fails. This shows that this channel contributes somewhat to the subtraction power of the network on this frequency band.

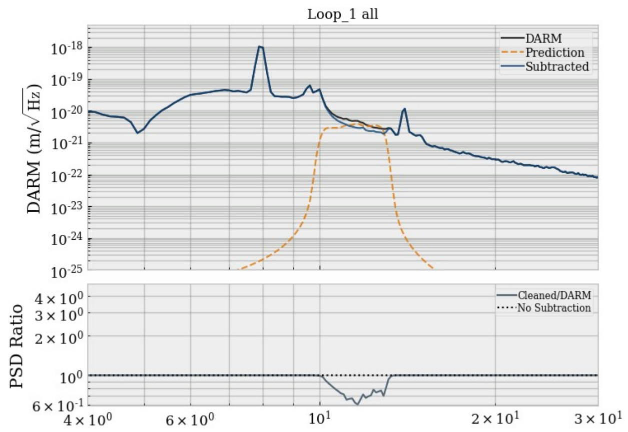
5. FUTURE WORK

5.1. *Multivariate Gaussian Process*

As mentioned in section 4.1, the Gaussian process in DeepClean hyperparameter tuning is a 1-dimensional Gaussian process. This is not ideal, because it eliminates the algorithm's greatest advantage over TPE: the ability to take into account the relation between parameters. My current plan is to develop a multivariate Gaussian process regression.

5.2. *Resonant Noise*

Ultimately, we want to vary the resonant frequency and the quality factor, and study the behavior of the networks. At the moment, the normalization of the resonant noises is done by scaling the time series by an arbitrary factor of 10^{-13}. However, because the amplitude of the power spectra density depends greatly on the resonant frequency and the quality factor, a more mathematical scaling system is needed.



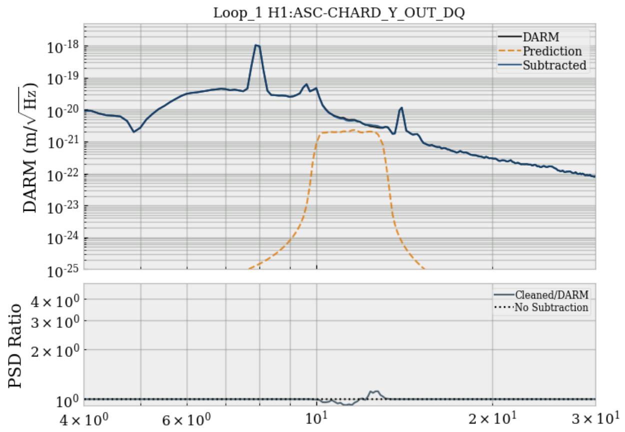


Figure 12. The subtraction result on the H1 August data in the 10-15 Hz frequency band before (left) and after (right) removing the channel "ASC-CHARD_Y_OUT_DQ".

5.3. *Selecting Witness Channels*

5.3.1. *Correlation Study*

As mentioned, ultimately we want to understand which channels are important and should be included in the training. We may also do this with a correlation study, by using machines learning techniques such as principle component analysis (or PCA) or by computing the coherence between the channels. My current plan is to develop these analysis technique.

5.3.2. *Low-frequency Subtraction/Compact Binary Coalescences*

We want to see sets of witness channels would result in the best subtraction in the low-frequency band 20-100 Hz. If we succeed, the network can then be used on O1 data in an attempt to discover more compact binary coalescences.

REFERENCES

Abbott, B. P. et al. 2016, Phys. Rev. Lett., 116, 061102

Adhikari, R. 2004, PhD thesis, Massachusetts Institute of Technology, Massachusetts, USA

Bergstra, J., & Bengio, Y. 2012, JMLR, 13

Do, C. B., & Lee, H. 2008, Lecture notes on Gaussian processes, Stanford CS229

Glorot, X., & Bengio, Y. 2010, in Proceedings of Machine Learning Research, Vol. 9, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, ed. Y. W. Teh & M. Titterington (Chia Laguna Resort, Sardinia, Italy: PMLR), 249–256

Karpathy, A. 2015, The Unreasonable Effectiveness of Recurrent Neural Networks

Kingma, D. P., & Ba, J. 2014, CoRR, abs/1412.6980, arXiv:1412.6980

LIGO Scientific Collaboration. 2007, arXiv, 0711.3041

M Coughlin, J Harms, e. a. 2014, Classical and Quantum Gravity, 31, 215003

Olah, C. 2015, Understanding LSTM Networks

Shevchuk, Y. 2016, Hyperparameter optimization for Neural Networks

The LIGO Scientific Collaboration. 2015, Classical and Quantum Gravity, 32, 074001

Tiwari, V., et al. 2015, Class. Quant. Grav., 32, 165014