Using Neural Networks to Remove 60Hz Noise from LIGO Data

Brendan King, Taarak Shah, Sid Suresh, Angelo Rustichini

May 7, 2021

Introduction

With the discovery of gravitational waves came the greatest confirmation of Einstein's Theory of General Relativity and an entirely new way to probe our universe. First confirmed at the Laser Interferometer Gravitational-Wave Observatory (LIGO), a 4000m interferometer, we now have a total of three such observatories scanning the sky for signals. However, new technology often comes with new challenges. One of these challenges is dealing with the near constant influx of data and subsequently processing this data in a timely fashion. Some laboratories are attempting to combat this issue with the also relatively new technology of neural networks (NNs) [1] [2]. By building efficient NNs, one can pipe data directly into the network and clean many sorts of noise from the data in real time so as not to "fall behind" the continuous stream of signal. In the paper below, we will be exploring the use of NNs to clean the ubiquitous 60 Hz noise from real LIGO data. The first section will discuss the basic principles of our method by applying various forms of NNs to simulated signals. Then we import LIGO data and try various networks to subtract the 60 Hz noise, culminating in a Long Short Term Memory (LSTM) network.

Noise Removal and Recovery of Sample Signals

In this report, we will address various methods of noise removal from a given signal on sample data. We are given a specific sinusoidal signal, then we apply various combinations of noise to the true signal. Then, different data augmentation techniques are applied and neural networks attempt to learn the relationship between the applied noise itself and the signal with the applied noise. First, the notation and signals are defined.

$$
\begin{aligned}
s(t) &= \sin(2\pi f_s t) && \text{(true signal)}\\
n_1(t) &= A_1 \sin(2\pi f_1 t + \phi_1) && \text{(noise signal 1)}\\
n_2(t) &= A_2 \sin(2\pi f_2 t + \phi_2) && \text{(noise signal 2)}\\
n_3(t) &= A_3 \sin(2\pi f_3 t + \phi_3) && \text{(noise signal 3)}\\
d_1(t) &= s(t) + n_1(t) && \text{(additive noise)}\\
d_2(t) &= s(t) + n_1(t) + n_2(t) * n_3(t) && \text{(multiplied noise)}
\end{aligned}
$$

We will experiment on additive and multiplied noise to the true signal. A baseline linear neural network will be applied on both types of signals, then the focus lies exclusively on recovering the true signal from the multiplied noise with more advanced neural networks. We experiment with different activation functions and data structures that are fed into the neural network. Initially, we apply a neural network with 1 hidden layer and a linear activation function to $d_1(t)$. This approximates the true signal incredibly well, as it should. Realistically, a neural network is not needed for this approximation, since it is as simple of an additive noise signal as we can get. We see the approximation of this neural network in Figure 1.

For $d_2(t)$, the multiplied noise signal, the same linear neural network performs poorly. The estimated output from the network ends up approximating the multiplied noise rather than the true signal, effectively making it ineffective for detecting true signals when we need to apply it to real data. We experiment with a ReLU activation function, which generally is a good choice for neural networks when dealt with a classification problem. Unfortunately, for time series data, the results were only slightly better than the linear network. We now experiment with some data augmentation and apply different neural network structures to our data.

Rather than feeding in individual data points for the noise, the signals are augmented to be fed in to the network as chunks of 1-second intervals. When we apply the ReLU activation function, our results are fairly poor once again, as expected for time series data. We instead turn to methods that are designed specifically

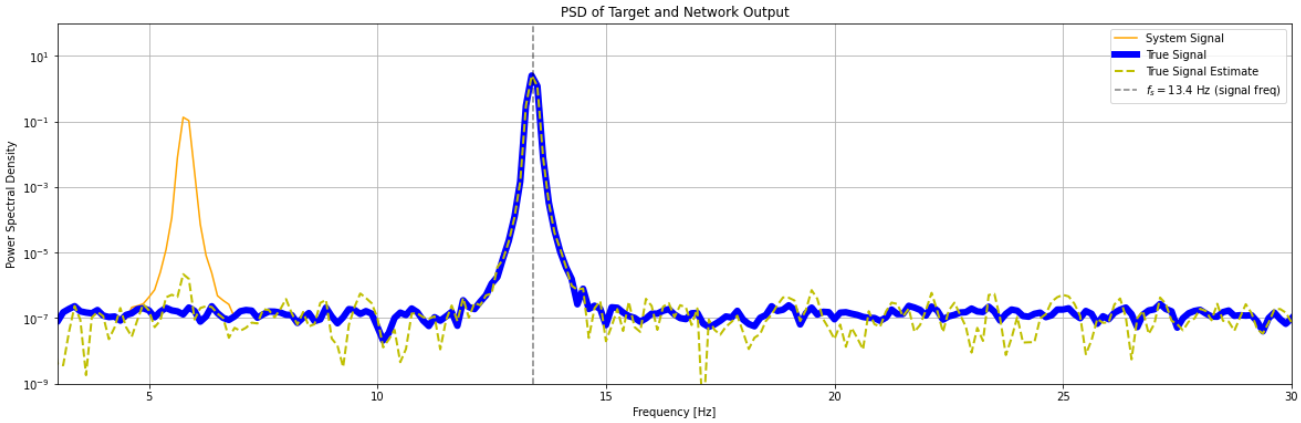

Figure 1: Linear network results with additive noise.

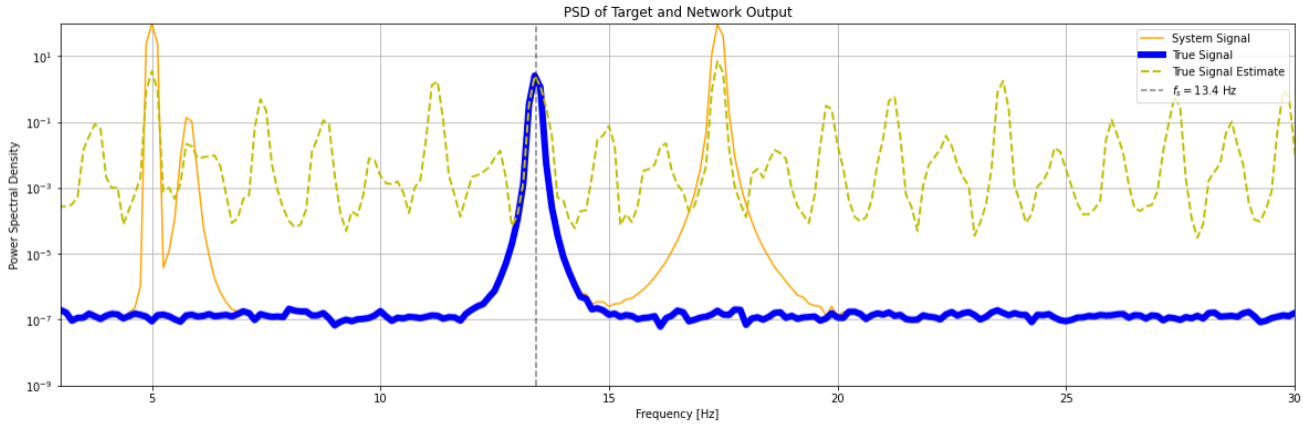

Figure 2: SimpleRNN results with multiplied noise and chunked data augmentation.

to work with time series data, such as recurrent neural networks. We implement a simple recurrent neural network with a hyperbolic tangent activation function. The results of this network are shown below.

We see that the estimate includes peaks that are roughly on the same order of magnitude as our true signal. This makes it very difficult to tell which peaks are the true signal and which are misrepresented by the network. This prompted the question of combining multiple hidden layers in the network and observing performance then. The results of a simple RNN and a ReLU layer afterwards are displayed below.

This does significantly better than the simple RNN alone. We can see that the estimated peaks are about two orders of magnitude below our estimate for the true peak. This indicates that including additional layers will help improve the associations the network can make after implementing a layer intended to deal with time series data.

Moving on to the final, and best performing network, we have a special version of an RNN. The LSTM (long short-term memory) architecture functions by "remembering" important data chunks that approximate our true signal well and using that to aid the performance of the network. This was expected to perform better than our previous attempts, and our visualization of our estimate confirms that idea.

From the LSTM estimate, we see that the multiplied noise peaks are under-approximated by about 2-3 orders of magnitude, a very promising sign. The true signal stands out, making peak detection significantly easier and more efficient.

To recap, we have applied various neural networks to different noise signals to extract a true signal from sample data. The LSTM performed the best, a promising sign for our analysis on real LIGO data. The results from working on the sample data should provide a useful template for working with the true data and removing external noise factors.

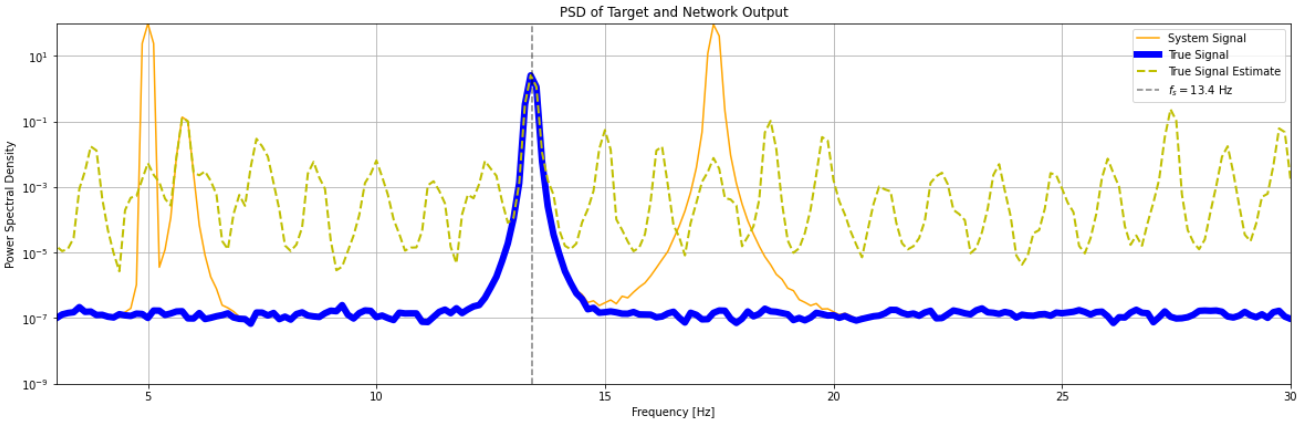

Figure 3: SimpleRNN + ReLU results with multiplied noise and chunked data augmentation.

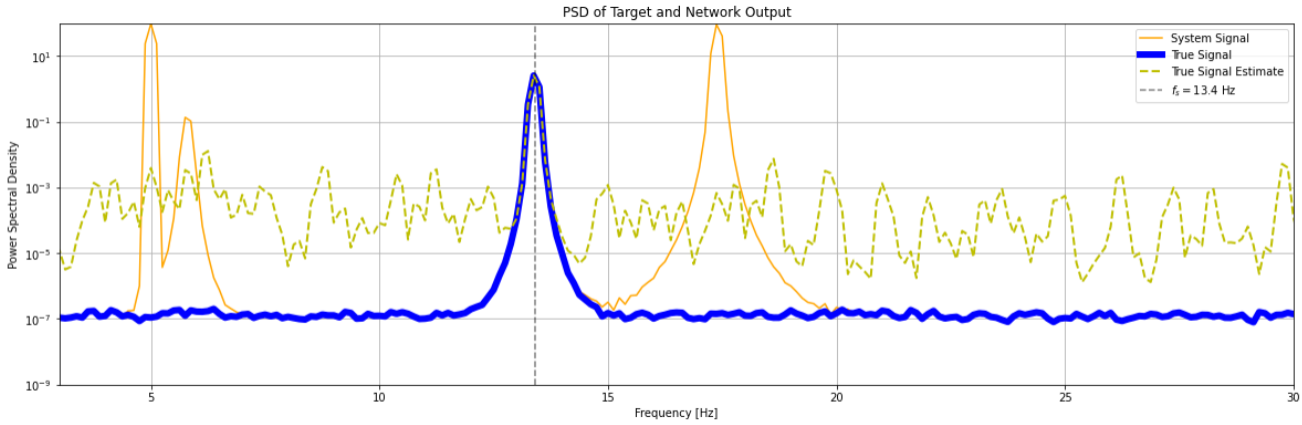

Figure 4: LSTM Results with multiplied noise and chunked data augmentation.

Application of Simpler Linear NNs to LIGO Data

The LIGO data we will be using comes in three forms: DARM, PEM, and ASC (See Figure 5). The main signal in the form of the differential arm length (DARM) contains all the relevant information about the astrophysical objects/events we are hoping to see. We have signal from the powerlines in the peripheral environmental monitor (PEM) and auxiliary interferometric channels (ASC) to monitor the movement of the arm mirrors among other things. Noise from the environment can effect our signal in many complicated (often non-linear) ways. Since analytically solving how noise from the environment would effect our signal is probably impossible and certainly not computationally feasible, the plan is to feed these noise channels (PEM & ASC) to the NN and train it to the data signal (DARM) in hopes of building a network that captures the complicated ways noise interacts with our system. If done correctly, we should be able to subtract the NN output from the signal to remove any undesired noise.

One of the simplest types of neural network is one with only a single layer and a simple linear activation function. As shown above, when the noise is coupled linearly to our signal, we can expect the NN to recognize the noise and be capable of removing it. We begin with the same network structure used to produce Figure 1 from simulated data. We gave the network only the PEM signal for increased simplicity. This network trained quickly but did nothing to remove the 60 Hz noise (See Figure 6). We then tried adding complexity to the networks by adding more layers and more noise channels (four ASC channels), but this also did not reproduce the noise (see Figure 6).

Next we tried running the data through a bandpass filter (56-64 Hz) before training to give more weight to the region containing the powerline noise (See Figure 5). Running this, however, provided no

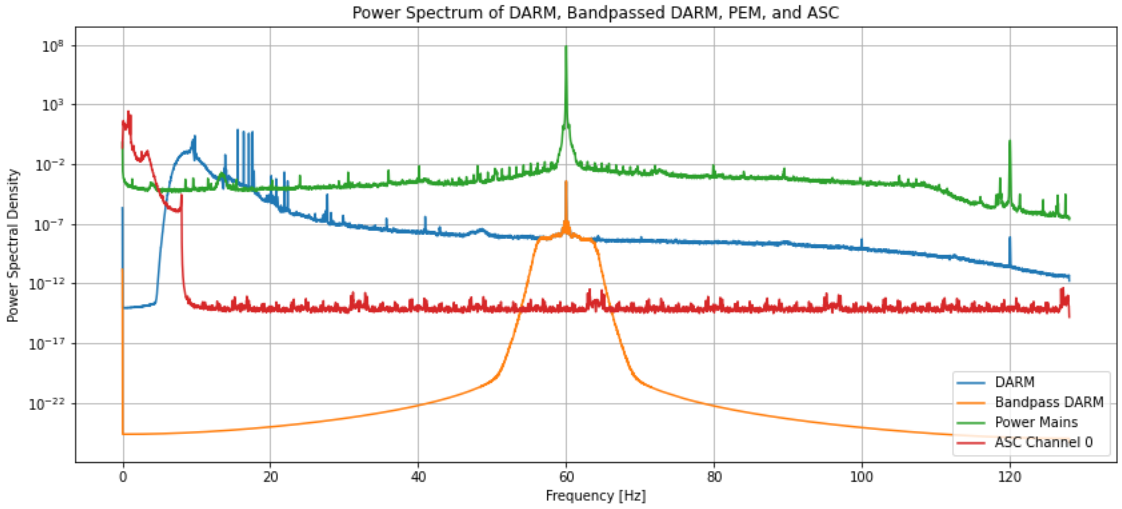

Figure 5: Scaled LIGO data PSD.

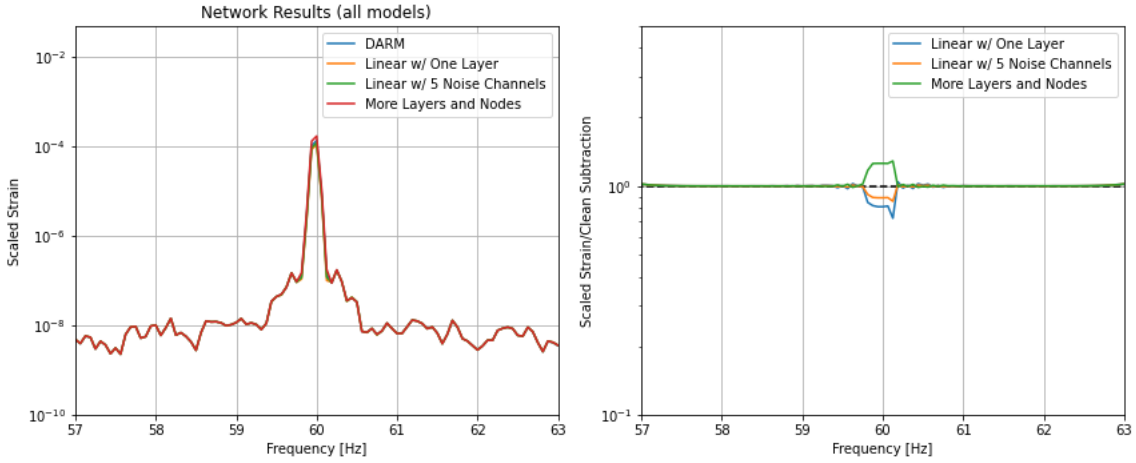

Figure 6: The results of all the simple linear NNs summarized.

improvement, so the results are not shown. Because none of these models worked, it is safe to assume that the noise is non-linearly coupled to our instruments. Also, because we are dealing with time series data, it is hard for a simple neural network to pick up long-range structure. It is due to these two factors we decided to try different types of networks in the proceeding section.

Diversification of Activation Functions

With the rich knowledge we gained from the simulated data, and our experience with linear activation functions, we learned what contributes positively, and what contributes negatively. More importantly however, we saw the importance of diversifying activation functions. In this section we will address the steps we took with activation functions, regarding function type, number of layers, and density.

First, it is useful to understand what an activation function performs. In a neural network such as the one we are building, an activation function defines the output of a node, given an input or set of inputs. Linear activation functions, the ones we have been largely using to this point, are clearly limited in their

potential, as they are similar to a simple on-or-off output. However, by introducing nonlinear activation functions, we can diversify our neural network to better suit the data.

Nevertheless, it may be useful to see exactly how useful (or useless) linear activation functions are as a baseline.

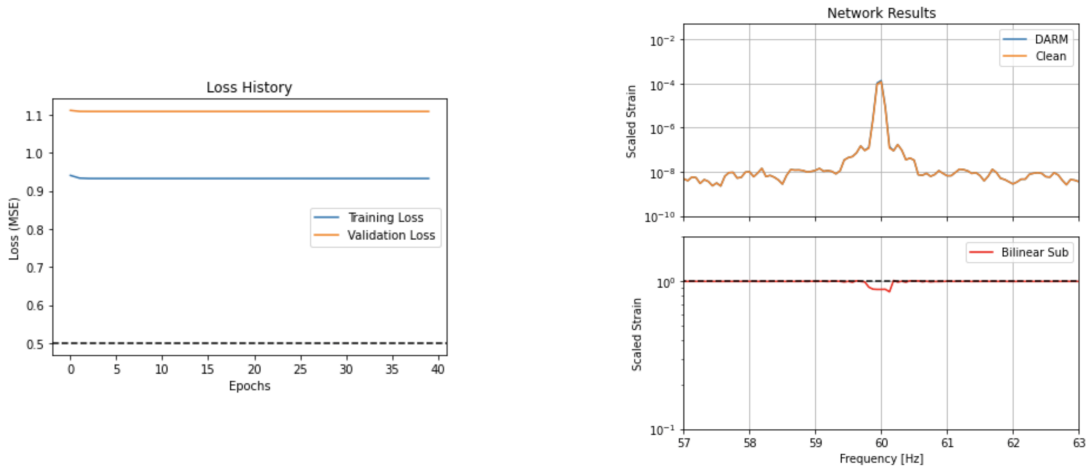

Figure 7: Linear Activation Function

As we see above, a linear activation function alone fails in multiple ways. It still overtrains, leading to a gap between validation and testing loss, and it fails to capture the 60 Hz noise. In fact, by including all sources of noise, these problems became even worse. This alone was a clear indication that other activation functions were needed.

We can thus begin with what is considered the "gold-standard" in neural network classification problems: the "ReLu" activation function. Unlike the linear activation function, this operates by separating non-positive and positive entries, working to better classify inputs. We see the results below after introducing this function in two layers.

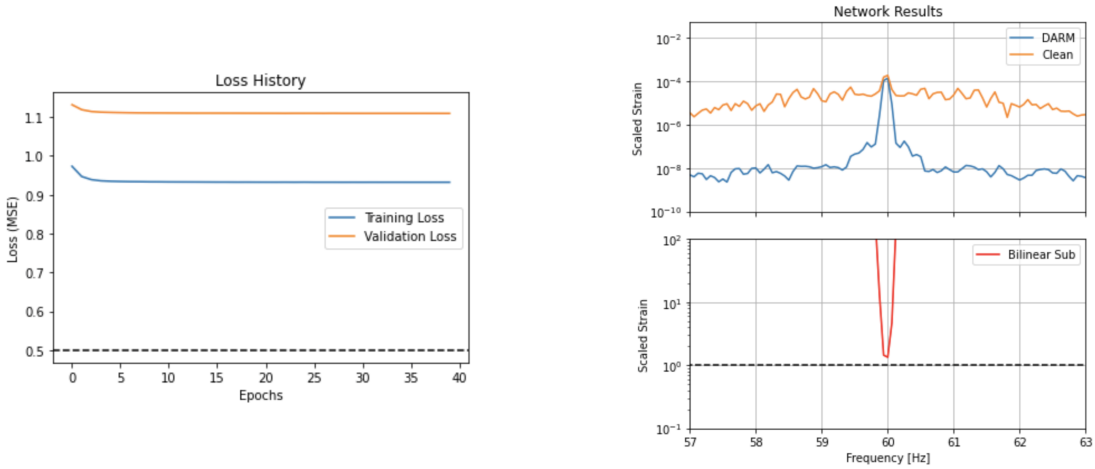

Figure 8: ReLu Activation Function

Clearly, the results here have only worsened. Upon further inspection of the ReLu function, it becomes clear that this kind of function is ill suited to our data. It performs fantastically on binary classification, but does not adjust well to removal of noise, as we are doing here. Thus, we should consider activation functions which are even less linear in nature.

To ensure we are even capable of capturing the noise, we can begin at an extreme by creating a needlessly large model that includes a "tanh", a "sigmoid", an "exponential", and several other activation layers. When doing so, we certainly risk over-fitting, but we can at least identify if there is a ceiling to our fitting problem.

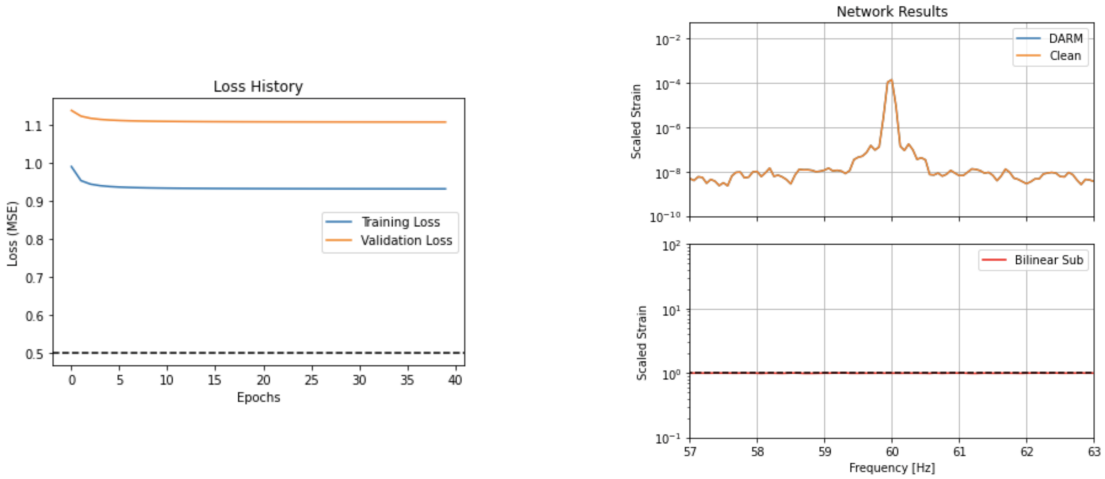

Figure 9: Elaborate Activation Functions

Fortunately, this function is an improvement over the models we trained before. However, it is exceedingly clear that the network over-trained. As we inspect the PSD plot, our estimation of the noise was certainly too precise, and would lead us to problems in the future.

As a final step, we can scale back the model we just created, hoping to find a model that neither over-trains, nor under-trains. Using some intuition, we chose a model with a "tanh" layer, a "sigmoid" layer, an "exponential" layer, and a "sigmoid" output. The results below are encouraging.

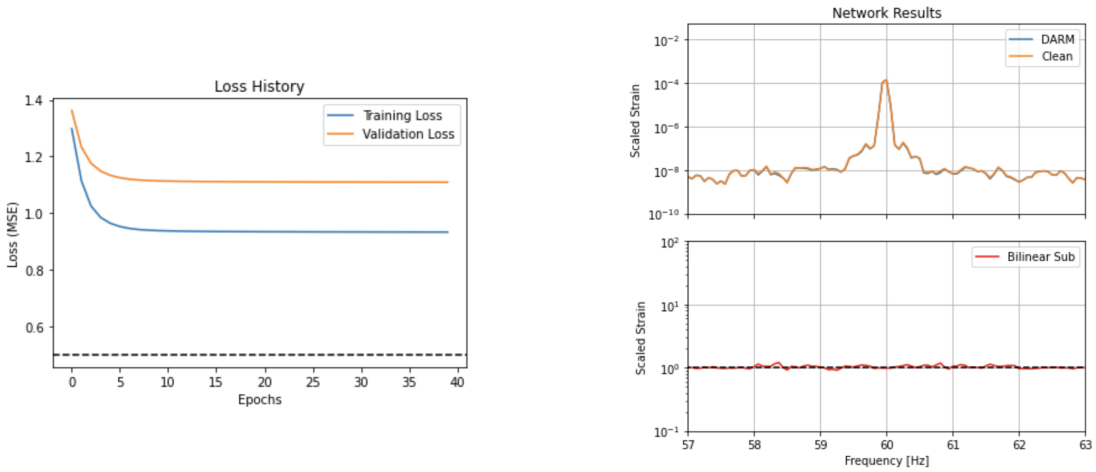

Figure 10: Final Activation Functions

As we see above, this final model neither over-trains, nor fails to fit the data. The difference between validation and testing data is present, but minimal, and the PSD plot reveals a fit that is strong everywhere, but does not hug the data too closely. Therefore, it is this model that proved our intuition right, and showed us that new activation functions would perform well, if mixed properly.

Application of LSTMs to LIGO Data

LSTM stands for Long Short-Term Memory and is a type of recurrent neural network proposed by Sepp Hochreiter and Jürgen Schmidhuber [3] to solve the vanishing gradient problem faced by RNNs, making it really good at learning sequences. LSTMs make use of gates to propagate useful information, determine what to add to the current step, and determine the next hidden state. The algorithm is much more complex than a typical RNN, increasing the runtime and making it less computationally efficient, but the tradeoff in performance is significant as we will show here and in our real data.

However, before data can be passed to an LSTM, there are certain pre-processing steps that need to be done. The first step would be to scale the darm, pem, and asc data by subtracting the mean and dividing it by standard deviation to bring them all to the same scale. Next, as we are focusing specifically on the 60Hz noise in the LIGO data, the signal is passed through a Butterworth filter [4] with a bandpass between 56 and 64 Hz. This brings that region into focus and reduces the impact other frequencies have on the network.

The work done on LSTMs in this project draws heavily from [5], with the focus here being to understand and improve the performance seen there. The original paper implements an LSTM network consisting of an LSTM layer, followed by a Dense layer and is successful in completely eliminating the noise at the 60Hz range.

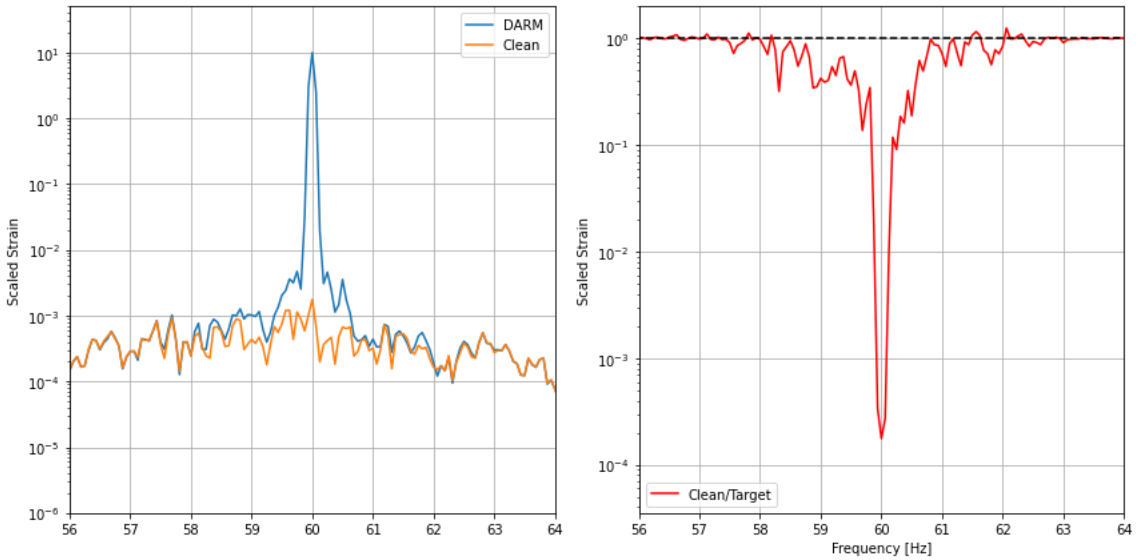

Figure 11: LSTM results overlaid with LIGO data (left) and the difference between the two signals (right).

However, a common issue with LSTM networks is that they can easily overfit data. In order to combat this, we add a Dropout [6] layer which is a regularization method where connections to LSTM units are excluded probabilistically from updates when training the network. Our approach is a network that implements 2 LSTM layers which are followed by Dropouts which help the network not overfit. This network is able to achieve similar performance to the initial model but with Dropouts included (See Figure 12).

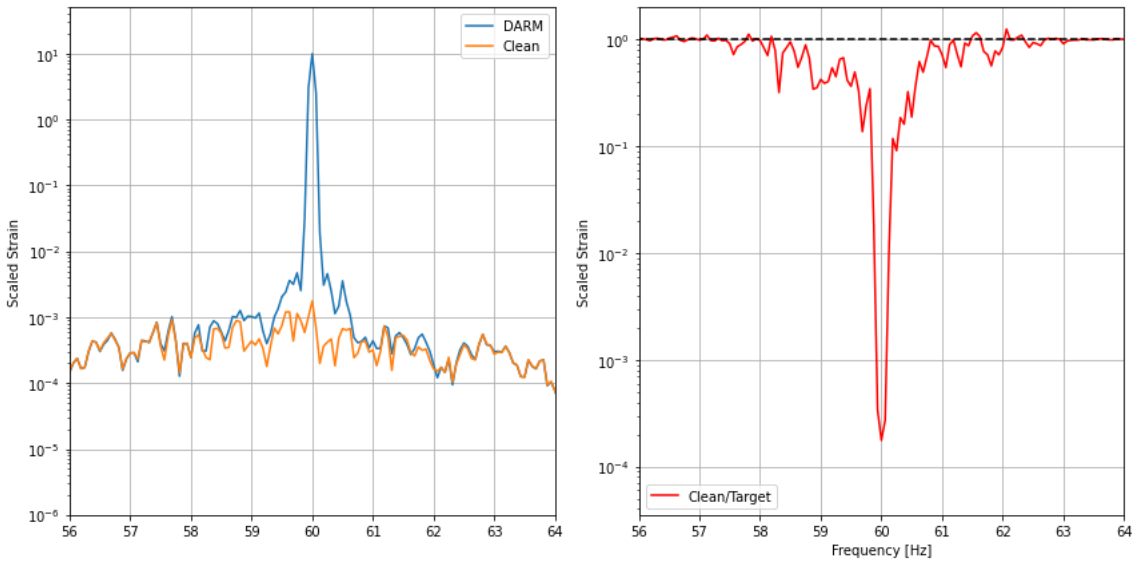

Figure 12: LSTM with dropout results overlaid with LIGO data (left) and the difference between the two signals (right).

Conclusion

In conclusion, we were able to build NNs that were able to clean the ubiquitous 60 HZ noise from real LIGO data. Initially, on the sample data, we observed that RNNs and LSTMs provided the best performance for noise removal. These methods significantly outperformed other common neural network architectures based on linear and ReLU activation functions. This was extended and applied to simple linear neural networks running on real LIGO data. However, these neural networks did not perform well in picking up long-range structure of the time series data of LIGO. To further improve these results, tweaking was done on the activation-functions, as the linear functions were deemed too ineffective. The results with these activation functions were encouraging, as they neither overfit, nor underfit, and matched the intuition built from the sample data. Finally, we used LSTMs with Dropouts and were able to come up with a network that was able to eliminate the 60 Hz noise while also preventing overfitting.

References

[1] Rich Ormiston et al. *Noise Reduction in Gravitational-wave Data via Deep Learning*. https://arxiv.org/abs/2005.06534.

[2] G. Vajente et al. *Machine-learning non-stationary noise out of gravitational-wave detectors*. https://arxiv.org/abs/1911.09

[3] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (1997).

[4] S. Butterworth. "On the Theory of Filter Amplifiers". In: *Experimental Wireless and the Wireless Engineer* 7 (1930).

[5] Rich Ormiston. *sideband_subtraction.ipynb*.

[6] N. Srivastava, G. Hinton, and A. Krizhevsky. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014).