# Recovering Sine-Gaussian Signal Parameters From Noise Using Machine Learning

Genevieve Connolly

July 3, 2025

## Contents



# 1 Introduction

At the frontier of modern astrophysics, and in a sense the frontier of the universe itself, gravitational wave (GW) detectors like LIGO (Laser Interferometer Gravitational Wave Observatory) provide a new lens through which to observe the universe. LIGO detects GWs through two main methods: modeled searches and background/foreground comparison. Modeled searches look for GWs with a specific waveform model, while background/foreground searches look for a difference between the background (a timeframe when there is no signal) and the foreground (a timeframe when we want to test if a signal is present). For modeled searches we need to know the equation for a waveform, but for background/foreground comparison we do not need a waveform model.

A particular waveform for which we do not yet have a model is for Gottlieb cooled disk GWs [1]. These GWs are hypothesized to originate from the same suspected source

of long gamma ray bursts: collapsars, or stars that collapse into compact objects (eg. black holes or neutron stars) at the end of their life. Being able to detect and parameterize these GWs will help further our understanding of stellar evolution and perhaps even human evolution, as GWs have recently been identified as potentially a fundamental ingredient in the development of human intelligence [2].

# 2 Methods

Although there is thus far no formal model for Gottlieb GWs, I have access to an example waveform for a 10 $M_\odot$ black hole located 10 Mpc from Earth, provided by Dr. Gottlieb (this same waveform is used in several plots in [1]). Since this is a single waveform, I am unable to produce more with different parameters (eg. black hole, disk mass, black hole-disk separation, etc.), so instead of using this waveform for machine learning, I used a sinusoidal signal to mimic an arbitrary signal, with parameters $a$, $f$, and $\phi$ (amplitude, frequency, and phase, respectively):

$$y(x) = a\sin(2\pi f x + \phi)$$

Ultimately the goal is to calculate the values of $a$, $f$, and $\phi$, which in a real GW signal would relate to source characteristics such as the black hole mass, disk mass, disk radius, separation between the disk and black hole, etc., but first the signal must actually be detected. I tested the ability of both logistic regression and a neural network to detect sinusoidal signals in noise by first generating 100 sine signals with different randomly chosen parameters, then adding random noise scaled by the amplitude (such that the noise cannot be larger than the overall amplitude of the signal). I trained the regression model and neural network with about 80 of these signals and then tested with the remaining ∼20. The training and test sets were chosen by random selection with a 20% chance of being chosen for the test set. As I will explain in the next section, the neural network was much more successful.

After determining whether or not a signal is present, I used logistic regression and MCMC regression to find the parameters of the signal. These models did not involve training and testing separately, but rather feeding the model a single sine signal with randomized parameters and assessing the accuracy of the model's output parameters. The MCMC regression was more successful than the logistic regression model.

# 3 Binary Classification

The logistic regression model and neural network for binary classification test whether or not there is a sine signal present in the data. Since each signal (or pure noise) has the same $x$ values, I treated the $x$ values as the features and the $y$ values as the value for each corresponding feature. The goal is for the model to learn that the $y$ values should increase and decrease consistently in each signal based on the frequency.

## 3.1 Logistic Regression

I used the binary 1 and $-1$ for the presence of a signal and pure noise, respectively. I used the log-likelihood loss function

$$L = \log\left(1 + e^{-c(wy+b)}\right),$$

where $c$ is the binary classification. This leads to the gradient functions

$$\frac{\partial L}{\partial w} = \frac{-cy}{1 + e^{c(wy+b)}}$$

$$\frac{\partial L}{\partial b} = \frac{-c}{1 + e^{c(wy+b)}}.$$

I also regularized the weights with $\frac{\lambda}{2}\sum_i w_i^2$ where $\lambda$ is the L2 regularizer, $w$ is updated as $w - \eta\left(\frac{\partial L}{\partial w} + \lambda w\right)$ and $b$ is updated as $b - \eta\frac{\partial L}{\partial b}$ for learning rate $\eta$. I considered the weights and biases to have converged if the gradient $\sqrt{\left(\frac{\partial L}{\partial w}\right)^2 + \left(\frac{\partial L}{\partial b}\right)^2}$ is less than 0.0001, but this convergence never occurred during the many trials I ran with various $\eta$, $\lambda$, and numbers of iterations. The likelihood of a classification being 1 is given by

$$P = \left(1 + e^{-wy-b}\right)^{-1}.$$

After many trials, the best results I found were with $\eta = 0.1$, $\lambda = 0.001$, and 200 iterations. These results still weren't good, though. I got 61% accuracy and found that the model is very susceptible to false positives. At 61% accuracy I had 11 true positives, one false negative, six false positives, and no true negatives. An example of each is shown in Figure 1. This distribution was fairly consistent throughout the various combinations of hyperparameters I tried.

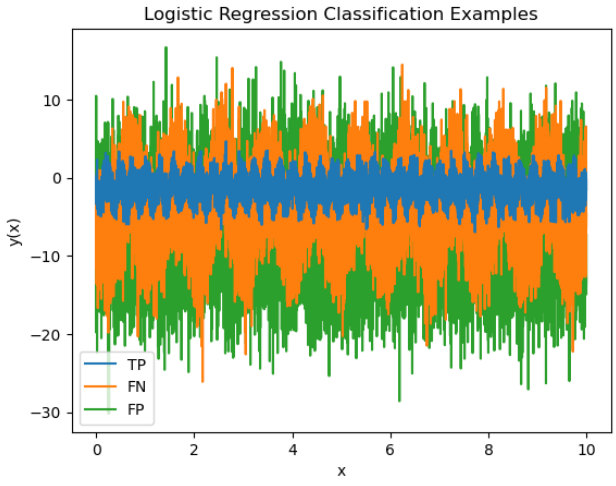


Figure 1: An example of a true positive (TP), false negative (FN), and false positive (FP) classified by logistic regression.

The logistic regression model is good at identifying positives but bad at identifying negatives. The false negative in Figure 1, shown in orange, has a very distinct sinusoidal shape to the human eye, while the false positive shown in green does not. I believe these misclassifications stem from the fact that at different frequencies, the relationship between the "features" and the true classification are very different. For example, at a frequency of 1 Hz, the $y$ value should be about the same at $x = 0$, $x = \frac{1}{2}$, and $x = 1$ for a classification of 1, meaning that a signal is present. But at a frequency of 5 Hz, the $y$ values at $x = 0$, $x = 0.1$, and $x = 0.2$ should be the same. The logistic regression model is unable to capture these characteristics for different frequencies with a linear relationship applied to each $y$ value $(wy + b)$. A different loss function may help mitigate this issue. But thankfully, the neural network performed much better.

## 3.2 Neural Network

I used `flax` to create a five-layer neural network. The first layer is `flax.linen.Dense` which applies a linear transformation with 10 features. The second layer is the ReLU function $\max(0, x)$, followed by another `Dense` layer with 10 features, then another ReLU function, and finally a `Dense` layer with two features as the binary classification output. I chose these particular layers because they generally have good rapport for use in binary classification specifically, especially the ReLU function. The choice of five layers and 10 features in the first two `Dense` layers was more arbitrary, simply as a starting point. I used the `optax.adam` optimizer for the same reason. For the loss function I used `optax.losses.softmax_cross_entropy_with`

`_integer_labels` which returns

$$\sigma_i = \log\left(\frac{\exp(x_{i,c_i})}{\sum_j \exp(x_{i,j})}\right)$$

I chose this function as a starting point because it is similar to the loss function used in logistic regression for binary classification. The output of the neural network is a value reflecting the confidence that the dataset contains a sine signal. A very positive value is confidently a signal, and a very negative value is confidently noise.

With 50 iterations and a learning rate of 0.01, the neural network reached 100% accuracy in training and 93% accuracy in testing. With 13 datasets for testing, the network successfully classified 16 positives and nine negatives, with only two false negatives and no false positives. Examples of each are shown in Figure 2. The false negative in Figure 2 is to the human eye clearly a sine signal with a low frequency.

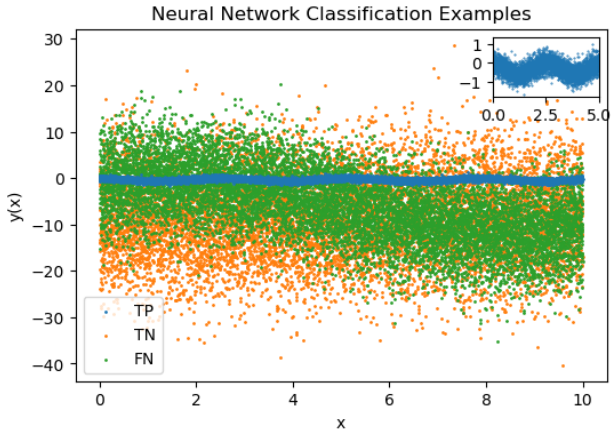


Figure 2: An example of a true positive (TP), a true negative (TN), and a false negative (FN) classified by a neural network.

With a 93% reliable model for detecting a sine signal in noise, now it's time to move onto determining the parameters of sine signals.

# 4 Parameter Estimation

I tested the ability of logistic regression and MCMC regression to estimate the parameters $a$ and $\phi$ of a sine signal in noise. I found that the logistic regression model was very inaccurate at estimating three parameters, so I decided to find $f$ using a Fourier transform instead of machine learning. This process is very simple: I used functions in `scipy.fft` to determine the frequency domain based on the $x$ data I had generated, then performed a Fourier transform on the amplitudes $y(x)$. With a pure sine signal, this results in the correct frequency $f$ having the highest FFT amplitude, but with noise, there is also a spike in FFT amplitude at 0 Hz. To mitigate this, I set $f$ as the frequency with the highest FFT amplitude above 0.1 Hz, which is the lowest frequency allowed by my random frequency generator. This solved the problem in general but occasionally the FFT amplitude of the actual frequency may still be below the height of the 0 Hz-centered peak at 0.1 Hz, so a more robust solution would be needed for the future. With the frequency pre-determined by a Fourier transform, the only parameters left to estimate are $a$ and $\phi$.

## 4.1 Logistic Regression

I used the squared loss function $L = (y(x) - \hat{y}(x))^2$, where $\hat{y}$ is the predicted amplitudes of the sine function using the estimated $a$, $f$, and $\phi$ values. This results in the gradients

$$\frac{\partial L}{\partial a} = -2(y - \hat{y})\sin(2\pi f x + \phi)$$

$$\frac{\partial L}{\partial \phi} = 2a(y - \hat{y})\cos(2\pi f x + \phi).$$

I also used the same convergence rule that if $\sqrt{\frac{\partial L}{\partial a}^2 + \frac{\partial L}{\partial \phi}^2} < 0.0001$ then $a$ and $\phi$ have

converged, but this condition was never met. The parameters were updated in the usual fashion with $a - \eta\frac{\partial L}{\partial a}$ and $c - \eta\frac{\partial L}{\partial a}$. I did not attempt to use weight regularization because I was unsure how to apply them to a sine function rather than a linear function.

I found that changing $\eta$ by a small amount seemed to make much more difference for the parameter estimation cases than it did for the binary classification cases. In particular, the amplitude parameter $a$ responded very strongly to changes in $\eta$. A high value of $\eta$ caused the estimated $a$ to be orders of magnitude above or below (negative) the actual value. The phase parameter $\phi$ did not respond as strongly, and regardless $\phi$ is cyclic so any differences may be more difficult to notice. It is also notable that a negative $a$ value has the same effect as $\phi = \pi$, so even if the estimated parameters are far from the original parameters, they may still produce a sine function somewhat close to the original.

The accuracy of the logistic regression was highly variable. Figures 3 and 4 show examples of bad and good results. Note that these plots show the noise distinct from the original function for clarity, but the original function and noise were passed to the logistic regression function as a sum.

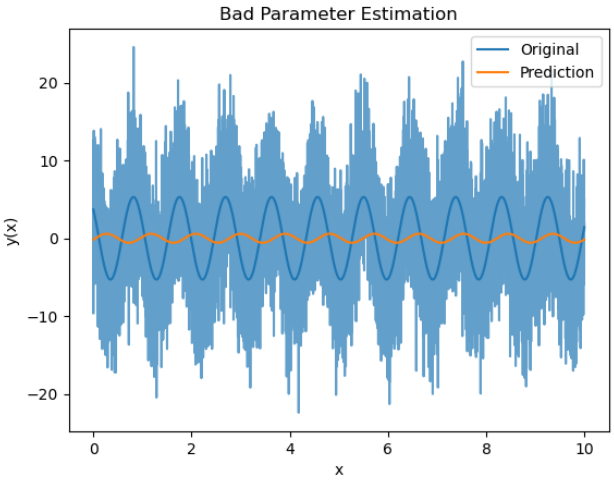


Figure 3: The estimated amplitude is almost a factor of 10 lower than the original amplitude and the phase is off by close to $\phi$, offsetting the crests and troughs.

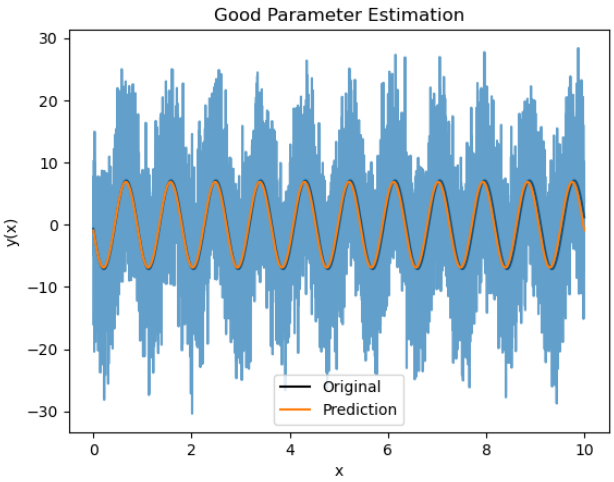


Figure 4: The predicted signal is so close to the original signal that it is almost difficult to see the original.

## 4.2 Regression with MCMC

I used `numpyro` to create another regression model, this time using an MCMC algorithm instead of gradient descent to update the parameters—specifically the `NUTS` algorithm in `numpyro`. This model calls for a prescribed $y$ error, so I simply used the amplitude $a$ as the error so that the error would be similar to the noise. The sine model is

created by choosing $a$ and $\phi$ randomly from uniform distributions, which I assigned the same minimum and maximum values as the $a$ and $\phi$ distributions used to generate the injected sine signal. The sine model also calls for $\sigma$ and $\mu$ where I chose to generate $\sigma$ by a half-normal distribution of scale 5, so that it would be positive-definite and hopefully not cause the parameter estimations to run away. $\mu$ is given by $a\sin(2\pi fx + \phi)$ and represents the estimated $y$ at each MCMC step for $a$ and $\phi$. The likelihood is given by a normal distribution with mean $\mu$ and standard deviation $\sigma$.

I ran two MCMC chains of length 10000 with a warm-up of 5000 samples, 5000 steps are run and then dropped as burn-in before the 10000 steps are run. This results in a list of 20000 $a$, $\phi$, and $\sigma$ values. From these values I chose one $a$ and $\phi$ at random to demonstrate the accuracy, since the 20000 values should be converged after the burn-in. Figure 5 demonstrates the very high accuracy of this method. This example has higher than 99% accuracy for $a$ and 91% accuracy for $\phi$. The Fourier transform estimation for $\phi$ is also highly accurate at over 99%. Notice that the noise does not look like a clear sine function here like it did in the logistic regression plots, yet the accuracy is much higher.

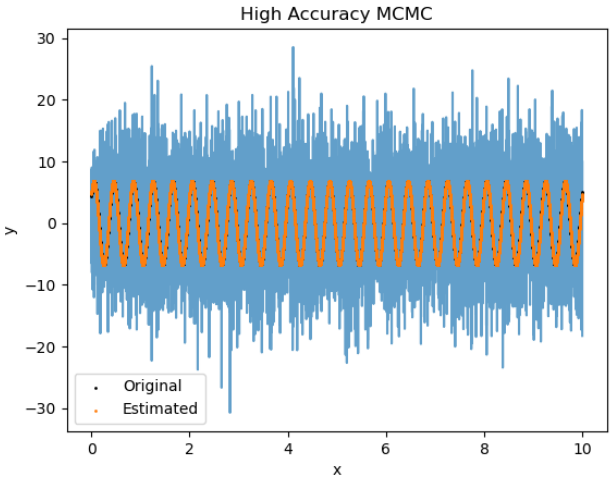


Figure 5: The parameters estimated via MCMC are so close to the original values that the original signal is hardly visible beneath the estimated signal.

# 5 Analysis

The neural network is clearly the more accurate method for binary classification of sine signals in noise, and the MCMC method is clearly more accurate than logistic regression for parameter estimation. The neural network for binary classification allows a much more nuanced model of the signal due to having multiple layers with 10 features, whereas the logistic regression model uses a very structured model with just $w$ and $b$ to characterize the signal. The logistic regression model tends to classify most datasets as positive, resulting in many false positives alongside the true positives. This model could be improved upon by trying different loss functions and by tuning.

The neural network performed well overall with 93% accuracy but the false positive in Figure 2 indicates that even a human may be a better classifier at these noise levels. Unlike the logistic regression model, the neural network seemed more likely to classify false negatives than false positives. Changing the

number of layers, types of layers, or number of features in the layers may improve the accuracy, as increasing these numbers may allow the network to recognize more distinct characteristics to associate with a positive result (i.e. low frequencies such as the false negative in Figure 2).

Logistic regression had highly variable accuracy for parameter estimation. There are many loss function options which may improve this accuracy. The squared difference loss function may not work well because as can be seen in Figures 3 and 4, the noise may be much higher than the actual signal amplitude, so the squared difference between the predicted and injected amplitudes may not be a good choice. Tampering with the learning rate $\eta$, the number of iterations, and adding weight regularization may also help.

MCMC regression had fantastic accuracy in parameter estimation, and may also be a good candidate to adapt for binary classification. Using an MCMC allows the parameter values at each step to be slightly randomized from the previous value, more so than in gradient descent, which seems to allow the model to escape from some pitfalls the gradient descent model fell into such as runaway amplitudes. Altering the burn-in and number of samples could make this even more accurate, as well as choosing different distributions from which to draw $a$, $\phi$, $\mu$, $\sigma$, and the loss function.

# 6 Conclusions

Overall, the MCMC parameter estimation performed the best. If I had an actual gravitational wave signal on which to run this parameter estimation, it could allow me to determine the physical characteristics of the source system with over 99% accuracy, even in noise. The next steps towards using this MCMC model in practice is to test its accuracy with other non-sinusoidal signals, and with more noise. If this model could be adapted for binary classification, perhaps it could be used to determine not only if a signal is present, but also what form that signal takes (eg. sinusoidal, sine gaussian, aperiodic signal, etc.) but also the parameters of the signal based on what form it is classified as. This would be a very powerful tool for not only astrophysical signals but also other applications such as communications, healthcare, and electronics.

# 7 References


[1] Ore Gottlieb, Amir Levinson, and Yuri Levin. *In LIGO's Sight? Vigorous Coherent Gravitational Waves from Cooled Collapsar Disks*. 2024. arXiv: 2406.19452 [astro-ph.HE]. URL: https://arxiv.org/abs/2406.19452.

[2] Bernard F. Schutz, Tsvi Piran, and Patrick J. Sutton. *Evolution of human cognition required Einstein's gravitational waves*. 2025. arXiv: 2503.03604 [gr-qc]. URL: https://arxiv.org/abs/2503.03604.