

Detection of gravitational waves from Ligo Data

Christopher Lawless

August 2021

Introduction

Gravitational Waves are distortions in spacetime, the strongest detectable examples of which come from the collision of black holes. (<https://science.mit.edu/big-stories/detecting-gravitational-waves/>)

The objective of this project was to use Data Analytics techniques to gather insights into black hole collision detection using Ligo Sensor data uploaded to: <https://www.kaggle.com/c/g2net-gravitational-wave-detection/data>

A Github link to my project can be found here: https://github.com/lawlessc/UCDPA_ChristopherLawless

The Data

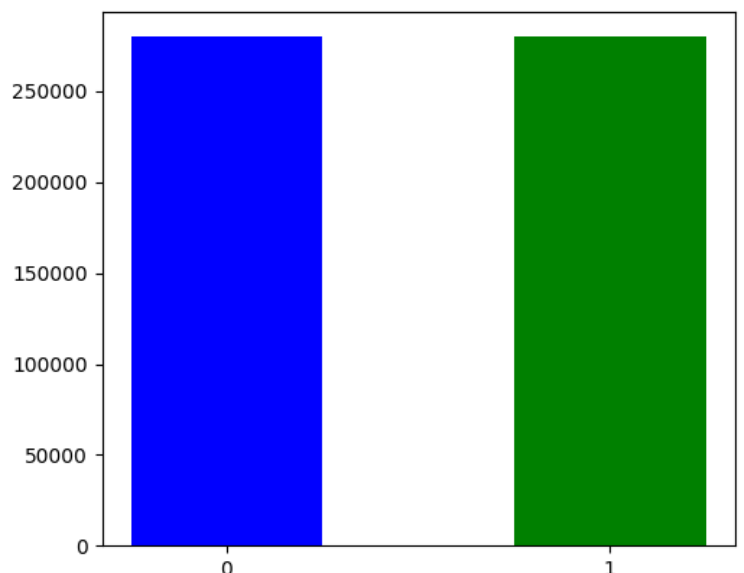
The data consists of vibrations detected by 3 different Gravitational wave sensors in Hanford Washington , Livingston Louisiana, and the Virgo the detector in Pisa Italy.

Each sample consists of 2 seconds of output from all 3 stored as a 2D numpy array(.npz) file sampled at 2048hz, or 4096 array items per a sensor, totalling 12288 .

From reading the dataframe of the data on targets i have found there are 560,000 samples They are evenly split between targets of 1 and 0

```
(560000, 2)
Index(['id', 'target'], dtype='object')
<bound method NDFrame.head of
0      00000e74ad      1
1      00001f4945      0
2      0000661522      0
3      00007a006a      0
4      0000a38978      1
...          ...      ...
559995  ffff9a5645      1
559996  ffffab0c27      0
559997  ffffcf161a      1
559998  fffffd2c403      0
559999  fffff2180b      0
```

Christopher Lawless

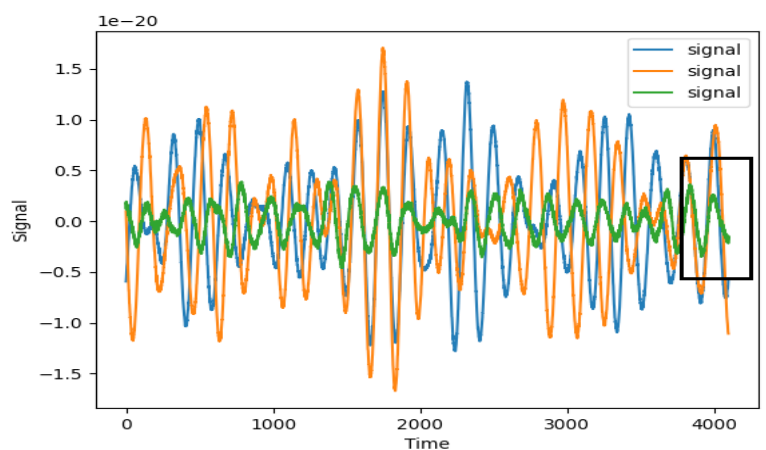
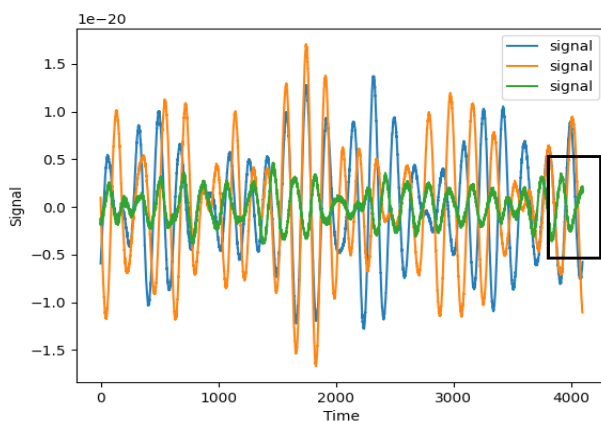


Processing the Data

The data can be downloaded by using within the project using the function in the `kaggle_download.py` file. This can take a while so in case you may just want to run it i have left the folder 0/0/0

While working on the project i went through a lot of scaling methods, one of the more interesting things i came across was the sklearn fast fourier transform, this is an sklearn feature that allows you to view frequencies and even apply a transform to the original signal to remove specific frequencies from the signal.

One thing i did notice from visualization is that one of 3 sensor out samples often appeared to be an inverted version of the signal from the other two detectors. So i inverted this at preprocessing in the hope it will be easier for the neural network to pick up on patterns during training. This is most likely the Italian gravity wave sensor which is located far from the others.



This issue is also mentioned in the Kaggle competition website.

For moving the data from files to training i chose to add them to a dataframe as this made appending the target values and ID's to the data and confirm. I also used a dataframe when i imported target data from **training_labels.csv** for readability.

Christopher Lawless

The Model

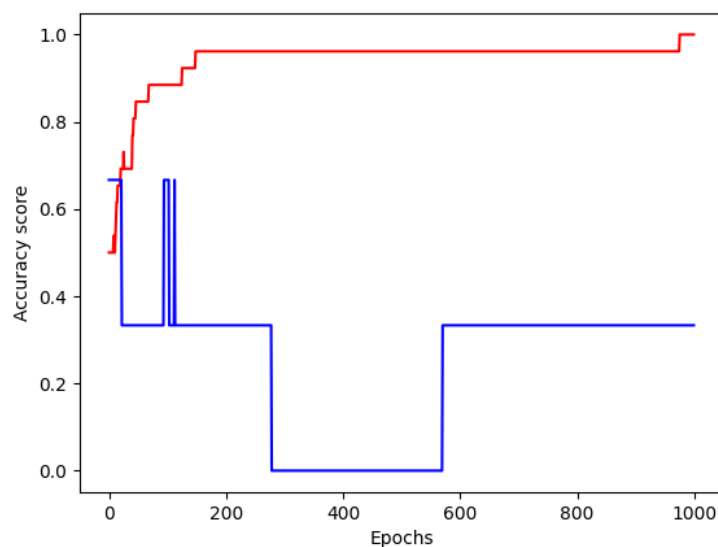
Because of the large number of parameters and the binary targets for the data the noisiness of the data and it's format as entirely small floating point numbers i decided to use a neural network trained via the Keras API.

Training the model

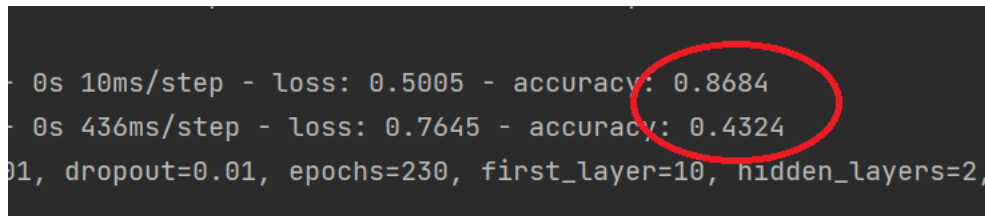
Initially i tried to manually train the model. This however required this required often reloading data everytime i wanted to make changes. I then wrote my own hyperparameter search class called "hyper_parameter_opto", this was useful as i had set it up to save the best models as it was running.

However it was also unsuitably slow.

I also found that loss and accuracy often diverged from another by a large amount or both plateaued in similar places.



I then implemented GridSearchCV in “grid_searcher” class within my project. This worked better in the sense it allowed me to try many more options to experiment with multiple layers, optimization algorithms number of epochs and batch sizes. However the overfitting problem persisted even while using GridSearch.



```
0s 10ms/step - loss: 0.5005 - accuracy: 0.8684  
0s 436ms/step - loss: 0.7645 - accuracy: 0.4324  
01, dropout=0.01, epochs=230, first_layer=10, hidden_layers=2,
```

Figure 1: Image demonstrates overfitting in difference between training and validation accuracy.

I initially started with the Keras basic Dense layers.

Then moved onto LeakyRelu layers.

And later convolutional layers as i experimented.

Issues with training.

When training my models i can overfit to the point of perfect accuracy with a few hundred samples and only several to tens of neurons. This does get harder as i increase the number of data points however accuracy sticks to 50%.

Possible causes

I considered multiple causes for this.

1. Buggy code when importing data causing targets to be matched to the wrong samples.
2. Driver/Cuda setup error
3. The data is too complex and noisy.

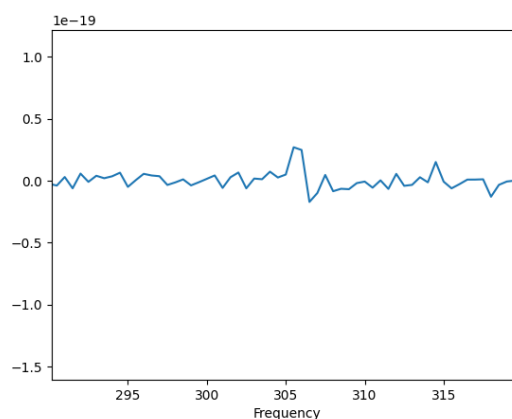
1. I followed up on issue 1 by checking the contents and format of my data at every stage between importing and beginning training, and did find issues with incorrect targets. So i fixed these bugs , but found the issue remained. I also made sure to load the data as 32 bit floats and convert the targets to the same to avoid possible issues Keras might have.

2. I believe the fact the network can over-fit and the lack of error messages from Cuda/Keras discredits this.

3. For this project i understood ,too late, just how noisy the signals are. I believe this is the most likely issue. The models i have tried easily learn the noise in small sets of training data.

Possible Solutions:

1. The data needs to be preprocessed more to remove noise before models train or make predictions. This could maybe be done with a Fast Fourier transform , i did experiment with this via the Sklearn library and seen what could maybe the signal of one in the data labelled as Target 1. That is often absent in the Target 0 samples.



2. Training the networks for longer on larger datasets. I was never able to beyond 60,000 samples from the training sets into my systems memory. However the total number of samples in the training set is 560,000
3. Using Ensembles. I initially considered trying to use an Ensemble of networks, but these would only be useful if the models had an accuracy greater than the 50% i am seeing.
4. Adding dropout layers however i think these may have been futile as the data itself is already very noisy.
5. More analysis of the data in more ways such as how various scaling methods affect it or for example PCA
6. The user of other classification methods like K-means clustering

Imports used in this project:

Sklearn (fft,)

Keras

Matplotlib

Kaggle

This project is also using Python Version 3.7

Because of the size of the dataset i have opted not to include it. It can be download manually from the Kaggle Competition site or via `kaggle_download.py` in my project.

