Exoplanet host star detection using an LSTM RNN

Anand N Warrier

ee16btech11042@iith.ac.in

December 31, 2019

Abstract

The benefit of deep learning is that we need not do any feature extraction. But, at times when the number of inputs to the neural network is very high, the model fails to extract the correct features required for the model to perform well. The Kepler telescope dataset has light intensities of various stars measured at 3198 time instances, which is too high for the number of inputs to the RNN. The frequency components of exoplanet host stars differ from that of non-exoplanet host stars. Applying Fourier transform on the data helps us to transform it into frequency domain. This also helps in reducing the number of inputs to the RNN. Then, we use an LSTM RNN as Recurrent Neural Networks perform well on time-series data.

I. Introduction

If there exists a planet which is revolving around a star, there will be instants when the light intensity of the star (as measured by a telescope) is significantly less compared to other instants. This dip in intensity is caused when the planet eclipses the star w.r.t the telescope.

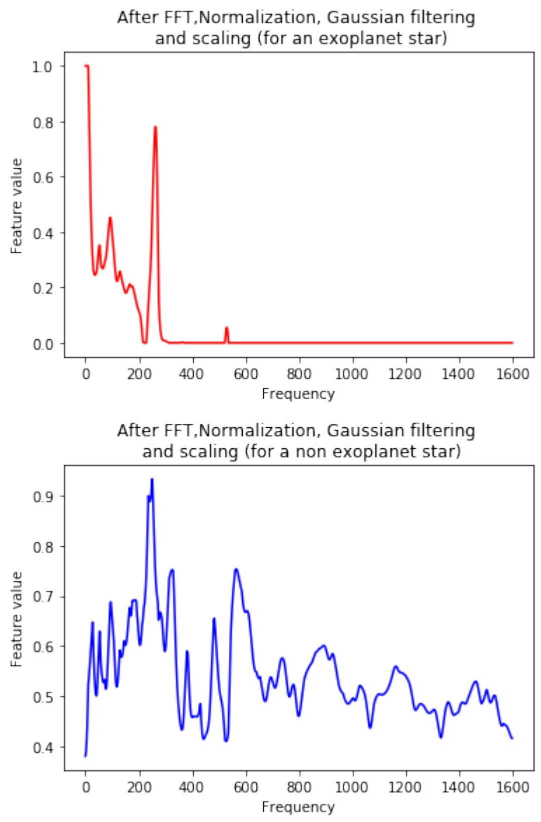
II. Implementation

- The training set has the light intensities of 5087 stars measured at 3198 time instants while the testing data has the the flux sequence for 570 stars.

- After we split the train and test data into features and labels, we apply Fourier transform on the features. Since the resulting sequence will be symmetric w.r.t the middle element, we only consider the first half of the sequence.

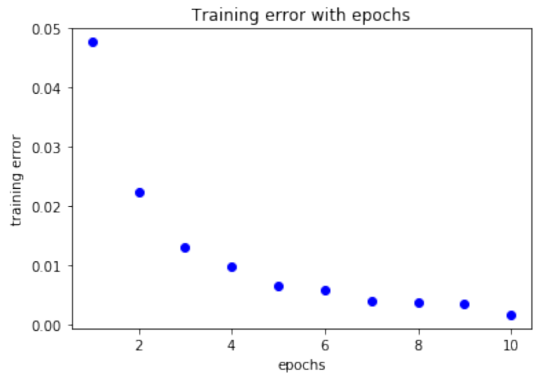
- Then, we normalize the data, apply a Gaussian filter and scale it down to the range (0,1).
- Then, we reshape the features to match the input format of the RNN.

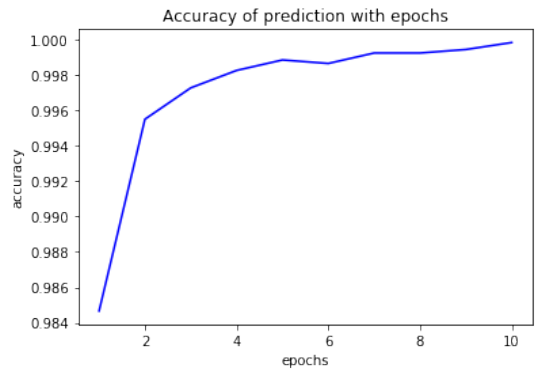
Figure 1: *After Fourier transform, Normalizing, applying Gaussian filter and scaling*



- The features are passed into an LSTM cell layer with 32 LSTM units. We add a drop out of 0.2 to prevent overfitting. Drop out helps in preventing overfitting by neglecting unwanted nodes(parameters) which are generally due to higher number of nodes. Then we have a Dense layer which gives a single output node after sigmoid activation.
- The loss function used is binary cross-entropy as it is a binary classification problem. The optimizer used for gradient descent is Adam (Adaptive moments). We train the model for 20 epochs.
- We plot the training loss and accuracy as a function of epochs to get an understanding.

Figure 2: *Training results:*





- The training set has 37 stars labelled as exoplanet host star while the test data has 5. This is very less compared to the number of non-exoplanet stars in their respective datasets. Due to this imbalance, accuracy alone can't determine the performance of this model. We calculate metrics such as recall and precision as well.

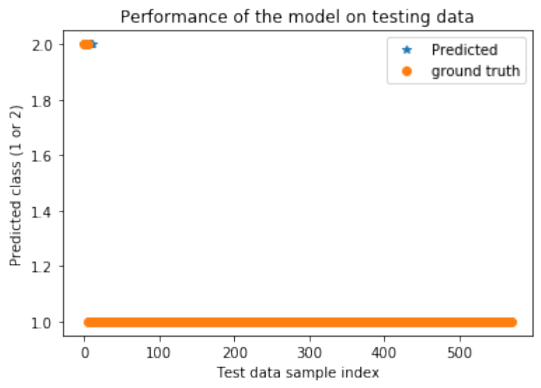
The jupyter notebook file can be found in the project github repository:`https://github.com/anandnwarrier/Exoplane_host_star_detection`

III. Results

From Figure 2, we can say that the model is learning as the training loss is decreasing. Moreover, the accuracy is increasing with epochs. The performance metrics are shown in the table below:

Figure 3: *After training*

	On train data	On Test data
Accuracy	0.9990171024179281	0.9894736842105263
Precision	0.9990108803165183	1.0
Recall	1.0	0.9893805309734514



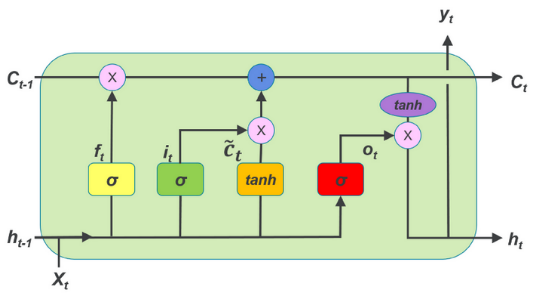
The model has performed very well on test data as the predicted and ground truth points are overlapping. We can verify this by looking at the test accuracy, precision and recall from the above table. From Figure 1, we can also see how Fourier transform has helped in bringing up the differences in frequency components of the light intensity signals of the exoplanet host stars and non-exoplanet host stars.

IV. Appendix

i. LSTM RNN

Long Short-Term Memory (LSTM) networks are an extension of Recurrent Neural networks(RNN) which solves the vanishing gradient problem. In Figure 4, the yellow, green and red colored gates are the forget gate, the input gate and the output gate. A sigmoid activation is used at the gates to decide whether a feature is to be kept (when output is near 1) or is to be discarded(when output is near 0).

Figure 4: *Diagram of an LSTM cell*

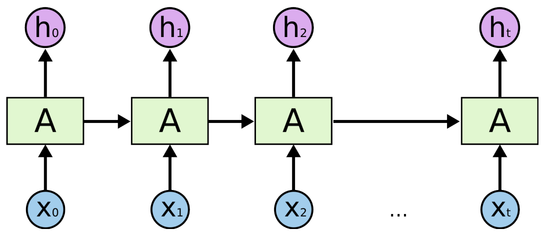


where,
$i_t = \sigma(w_i(h_{t-1}, x_t) + b_i)$
$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$
$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$

$\tilde{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c)$
$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$
$h_t = o_t \times tanh(c_t)$

i_t = output of input gate
f_t = output of forget gate
o_t = output of output gate
c_t = new cell state
h_t = new hidden state
$\tilde{c}_t$= candidate for cell state
x_t = current input
c_{t-1}= previous cell state
h_{t-1}= previous hidden state
w_x, b_x = weights and bias of gate 'x'

Figure 5: *LSTM layer, input nodes and output nodes*



ii. Precision and recall

Precision $=\frac{t_p}{t_p+f_p}$

Recall $=\frac{t_p}{t_p+f_n}$

Accuracy $=\frac{t_p+t_n}{t_p+t_n+f_p+f_n}$

where,
t_p=true positive
t_n=true negative
f_p=false positive
f_n=false negative

V. References

- https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data
- https://www.youtube.com/watch?v=8HyCNIVRbSU
- https://keras.io/layers/recurrent/
- https://medium.com/dvlpr/exoplanet-hunting-in-deep-space-with-machine-learning-4db85d5f7769