

**Exoplanet Hunting:  
Kepler Labeled Time Series Data Analysis  
--Individual Report**

**Group 8**

Reported by  
Zixuan Huang

May 1, 2020

# Contents

## **1. Introduction**

- 1.1. Overview of project
- 1.2. Shared work

## **2. Individual work**

- 2.1. Background
- 2.2. Preprocessing
- 2.3. Model
  - 2.3.1. Model Structure
  - 2.3.2. Experimental Setup
  - 2.3.3. Results

## **3. Summary**

- 3.1. Contribution
- 3.2. Future works

## **4. Calculation**

## **5. References**

# **1. Introduction**

## **1.1.Overview of project**

The goal of this project is to develop neural networks to successfully model data for the detection of exoplanets. We used Keras and created three models. In order to improve our models' performance, we tried different preprocessing methods and focused on addressing overfitting problems, which allowed us to evaluate our models from several perspectives.

The dataset was provided by Kaggle, but originally derived from observations made by NASA Kepler Space Telescope. The data describes the change in flux (light intensity) of several thousand stars. Each star has a binary label of "2" or "1". Label "2" indicates that the star is confirmed to have at least one exoplanet in orbit.

## **1.2.Shared work**

We picked dataset, wrote proposal and did data preprocessing together. And then we separately train a model. We also shared our thoughts when we trained our own model, especially when we found some of the preprocessing might not benefit model performance and then we adjusted those preprocessing parts.

We basically meet once or twice a week online. For the slides and final report, we shared one Google docs and finished it together.

# **2. Individual work**

## **2.1.Overview**

My job was doing part of data preprocessing and training a RNN model. For the final report, I wrote math background, RNN and summary parts.

## **2.2.Data Preprocessing**

In this part, we didn't separate our work very detailed. Once we look at those kernel on Kaggle discussed what kind of data preprocessing we can have a try, then we started to try different preprocessing method. What we finally adopted is data scaling, outlier removing, data sampling and data relabeling. We also tried smoothing as well, but found it actually didn't improve the performance of our models. Data balancing is the biggest issue of our project, I tried random sampling and SMOTE for both oversampling and undersampling, but the effects were not good.

## 2.3.RNN Model

### 2.3.1. Model Structure

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 3197, 1)	0	
conv1d_1 (Conv1D)	(None, 3187, 16)	192	input_1[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 797, 16)	0	conv1d_1[0][0]
batch_normalization_1 (BatchNor	(None, 797, 16)	64	max_pooling1d_1[0][0]
conv1d_2 (Conv1D)	(None, 787, 32)	5664	batch_normalization_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 197, 32)	0	conv1d_2[0][0]
batch_normalization_2 (BatchNor	(None, 197, 32)	128	max_pooling1d_2[0][0]
conv1d_3 (Conv1D)	(None, 187, 64)	22592	batch_normalization_2[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 47, 64)	0	conv1d_3[0][0]
permute_1 (Permute)	(None, 1, 3197)	0	input_1[0][0]
batch_normalization_3 (BatchNor	(None, 47, 64)	256	max_pooling1d_3[0][0]
gru_1 (GRU)	(None, 1, 16)	154272	permute_1[0][0]
conv1d_4 (Conv1D)	(None, 37, 128)	90240	batch_normalization_3[0][0]
gru_2 (GRU)	(None, 1, 32)	4704	gru_1[0][0]
max_pooling1d_4 (MaxPooling1D)	(None, 9, 128)	0	conv1d_4[0][0]
gru_3 (GRU)	(None, 1, 64)	18624	gru_2[0][0]
flatten_1 (Flatten)	(None, 1152)	0	max_pooling1d_4[0][0]
gru_4 (GRU)	(None, 128)	74112	gru_3[0][0]
dropout_2 (Dropout)	(None, 1152)	0	flatten_1[0][0]
dropout_1 (Dropout)	(None, 128)	0	gru_4[0][0]
dense_1 (Dense)	(None, 64)	73792	dropout_2[0][0]
concatenate_1 (Concatenate)	(None, 192)	0	dropout_1[0][0] dense_1[0][0]
dense_2 (Dense)	(None, 32)	6176	concatenate_1[0][0]
dense_3 (Dense)	(None, 1)	33	dense_2[0][0]
Total params: 450,849			
Trainable params: 450,625			
Non-trainable params: 224			

In the original model, LSTM was used in the recurrent layers. I've tried both LSTM and GRU and finally used GRU. The key difference between a GRU and an LSTM is that a GRU has two gates (reset and update gates) whereas an LSTM has three gates (namely input, output and forget gates).

The GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. It just exposes the full hidden content without any control. The performance of GRU is on par with LSTM, but computationally more efficient. As can be seen from the equations LSTMs have a separate update gate and forget gate. This clearly makes LSTMs more sophisticated but at the same time more complex as well.

#### **FULL GRU Unit**

$$\tilde{c}_t = \tanh(W_c[G_r * c_{t-1}, x_t] + b_c)$$

$$G_u = \sigma(W_u[c_{t-1}, x_t] + b_u)$$

$$G_r = \sigma(W_r[c_{t-1}, x_t] + b_r)$$

$$c_t = G_u * \tilde{c}_t + (1 - G_u) * c_{t-1}$$

$$a_t = c_t$$

#### **LSTM Unit**

$$\tilde{c}_t = \tanh(W_c[a_{t-1}, x_t] + b_c)$$

$$G_u = \sigma(W_u[a_{t-1}, x_t] + b_u)$$

$$G_f = \sigma(W_f[a_{t-1}, x_t] + b_f)$$

$$G_o = \sigma(W_o[a_{t-1}, x_t] + b_o)$$

$$c_t = G_u * \tilde{c}_t + G_f * c_{t-1}$$

$$a_t = G_o * \tanh(c_t)$$

### **2.3.2. Experimental setup**

#### **2.3.2.1. Learning rate: 0.01**

#### **2.3.2.2. Optimizer: SGD**

The original model that I borrowed from was using Adam as the optimizer. When I plotted the loss for the original model, I found the val\_loss is very fluctuated. We guessed that Adam might have been too aggressive to make the validation loss decrease step by step. Since we already split

the training dataset into small batches, mini batch gradient descent can be used to calculate model error and update model coefficients. We then tried SGD and found the val loss gets smoother even using SGD makes the training process converge slower than using Adam.

Adam is spread used optimizer due to its power. But after a while people started noticing that despite superior training time, Adam in some areas does not converge to an optimal solution. Some research have revealed that adaptive methods (such as Adam or Adadelata) do not generalize as well as SGD with momentum when tested on a diverse set of deep learning tasks.

#### 2.3.2.3. Batch generator: batch size = 32

Before modelling, we created a function called `batch_generator`. The batch size was set to 32, and the input of the function is `x_train` and `y_train`. This batch generator gave an equal number of positive and negative samples, and then we used `np.roll()` to randomly rotate those samples. We transferred the original data into batches which was used as the input of the model.

#### 2.3.2.4. Loss: binary\_crossentropy

Since we were training a binary classifier, the target is either exoplanet or non-exoplanet, we were using binary cross-entropy as our loss function. Binary cross-entropy measures how far away from the true value (which is either 0 or 1) the prediction is for each of the classes and then averages these class-wise errors to obtain the final loss.

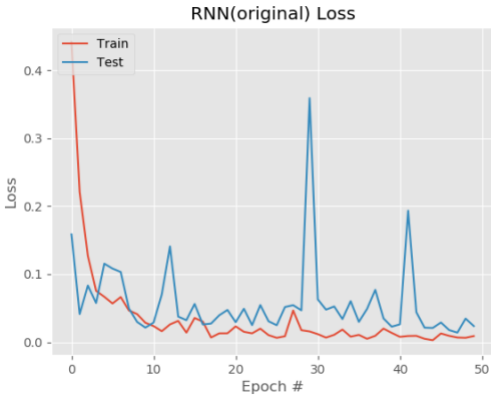
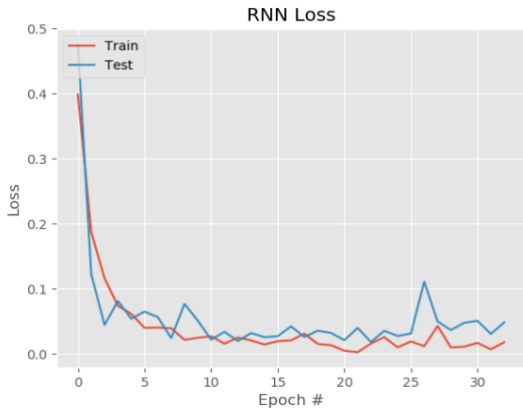
$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

#### 2.3.2.5. Callbacks: Early stopping

Here we set the model stop training once the validation loss is no longer decreased within 10 epochs (`monitor=val_loss, patience = 10`).

### 2.3.3. Results

Original model	My model
	
<pre> precision    recall  f1-score   support  0.0         1.00      1.00      1.00       565 1.0         1.00      0.80      0.89         5  accuracy macro avg       1.00      0.90      0.94       570 weighted avg    1.00      1.00      1.00       570 0 - exoplanet star 1 - non-exoplanet star </pre>	<pre> precision    recall  f1-score   support  0.0         1.00      1.00      1.00       565 1.0         0.67      0.80      0.73         5  accuracy macro avg       0.83      0.90      0.86       570 weighted avg    1.00      0.99      0.99       570 </pre>
LSTM; Adam	GRU; SGD; Early stopping

Compared to the results of the original model, the validation loss of our model is more smoothed, and both train loss and validation loss are showing a gradual downward trend. However, the precision value(0.67) and the F1 score(0.73) of our model are not as high as the original results.

## 3. Summary

### 3.1.Contribution

First, we tried different ways to balance our data. Unlike typical oversampling and undersampling methods in sklearn, what we did was first generate negative samples (non-exoplanet star) to the half number of positive samples by rotating them, and then we only used almost half of the numbers of positive samples rather than all of them to train the models. We found this way performs better than random oversampling or SMOTE.

Second, we tried to adopt a smooth method on kaggle, but we found the score getting lower. The smooth method was using `uniform_filter1d` to smooth features (flux) on each observation (star). The purpose was trying to move out noise from data. In general, the flux of exoplanets is more fluctuated than that of non-exoplanet. But when we plot some non-exoplanet samples before and after smoothing, we find that some flux shape has been changed. So we think it may not be appropriate to use the same smoothing method for all features.

Third, compared with other people's analysis on Kaggle, we use both loss value and F1 score as evaluation criteria. We all know that F1 score should be used as an evaluation when we deal with an imbalanced dataset. However, this dataset is so imbalanced that no matter what kinds of neural networks we use, the F1 score for positive samples can reach to almost 1. There are only 5 negative samples but 565 positive samples in the test set (hold out set). Therefore, we do not regard classification reports as the only criterion. Since we found that some of the results on the Kaggle have a very high F1, but they are actually overfitted. So on the other hand, we pay more attention to the loss value to prevent the model from overfitting

### **3.2.Future works**

First, in order to further improve the smooth method, we believe that stars can be classified according to the distribution shape of their flux. Even the stars all belong to exoplanet stars, but the shape of flux may be different, some are sin or cosine shape, some are line shape and others are curves. So we'd better to classify those stars and adopt different smoothing methods to different sub-categories.

Second, because this data set is too unbalanced, in addition to the need to balance the data during training, a more scientific method, such as k-fold validation should be used when assessing the performance of models.

## **4. Calculation**

Roughly:  $120-60/120+30 = 40\%$

## **5. References**

- [1]<https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>
- [2]<https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9>
- [3]<https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>