Predicting Exoplanet Temperatures using Deep Learning

Konstantinos Varakliotis
Electrical and Computer Engineering, University of Thessaly

22 March 2025

Abstract
The characterization of exoplanets is fundamental for understanding planetary formation, atmospheric composition, and potential habitability. One key property, planetary temperature, is often missing from exoplanetary databases due to observational limitations. In this study, we employ deep learning to estimate exoplanet temperatures based on available parameters such as stellar characteristics and orbital properties. Using data from NASA's Exoplanet Archive, we develop a feedforward neural network optimized with hyperparameter tuning. Our model achieves a Mean Absolute Error (MAE) of 25 K for a basic feature set and 17 K for an extended set. While the model performs well for exoplanets closer to their stars, its accuracy declines for more distant planets, highlighting a potential avenue for improvement. This approach provides a valuable tool for supplementing missing exoplanet data and enhancing planetary classification efforts

Key words: Deep Learning – Exoplanet Temperatures – Machine Learning

1 Introduction

The study of exoplanets has grown significantly in recent years, driven by the search for potentially habitable worlds beyond our solar system. However, characterizing exoplanets remains a major challenge due to their vast distances and the limited light they emit. These obstacles make it difficult to obtain crucial planetary properties, leading to incomplete datasets that hinder our understanding of exoplanetary atmospheres and habitability.

One of the most fundamental properties of an exoplanet is its temperature, as it plays a key role in atmospheric composition, climate, and the possibility of liquid water. Yet, for many exoplanets, temperature cannot be directly measured or confidently calculated. This gap in data presents a significant challenge in planetary classification and habitability studies.

In this work, we explore the use of deep learning to bridge this gap. By leveraging confirmed exoplanet data, we develop predictive models that estimate exoplanet temperatures based on readily available parameters, such as host star characteristics. This approach provides a powerful tool for inferring missing planetary data and enhancing our ability to analyze exoplanets, even in the absence of direct temperature measurements.

2 Literature Review

There have been many studies indicating the importance of the temperature in exoplanets and the process that it is calculated. For instance, [Jones 2008] indicated that the Earth is a potential habitable planet since we can calculate its surface temperature to be 300 K, which is warm enough for water to liquefy. The technique that they proposed was to study the black body radiation emitted from the planet. Furthermore, in the same study Hot Jupiters were examined. They are gas giants with masses in the same order of magnitude as Jupiter and Saturn that orbit close to their host star, causing extreme surface temperature. In another study (Unterborn 2018) the upper temperature limits for potential rocky super-Earth planets was discussed and calculateed. Hence, the surface temperature of the planet contributes significantly in the study of exoplanets.

Additionally, the field of exoplanet study has been aided by machine learning numerous times. One such time is being examine in (Armstrong 2021) where 50 new exoplanets were discovered through machine learning by a method called statistical validation. Moreover, (Lalande 2024) used machine learning techniques, such as kNN and Imputation to estimate the mass on exoplanets in an incomplete database. In this study, we aim to do something similar by providing values for the missing exoplanet temperatures.

3 Data Collection and Preprocessing

3.1 Dataset Description

For this project we used NASA's Exoplanet Archive Dataset, reference in [1], that has a complete record of all confirmed exoplanets with a variety of properties for all purposes, from the telescoped used to discover it, to the stellar metallicity of the host star.

For our purposes we deemed that the initial properties that could potential prove useful for temperature estimation are many and it would be beneficial to include many different properties and then judge their relevance on the temperature of the planet. All the datasets of the projects can be found in its github referenced at [2].

As we found on the dataset, out of the 6000 individual planets listed on the dataset, almost 14% of the planets is missing its temperature value. However, due to the high amount of confirmed data we can be confident that the prediction our algorithm makes are accurate and credible.

3.2 Feature Engineering

As mentioned before, we used a variety of properties from the database. This means that, in order to achieve the best possible model, we have to estimate their impact on the training and remove those with low impact. To accomplish this, we constructed the correlation matrix of the features. Firstly, we consider a correlation threshold of 0.3 and, as a result, we remove the features with absolute correlation value of less than 0.3. The correlation matrix with the reduced number of features can be seen in figure 1.

The features chosen by the correlation matrix method are the following:

- **Number of Stars** : The number of stars in the planetary system in question, it makes sense to include this because

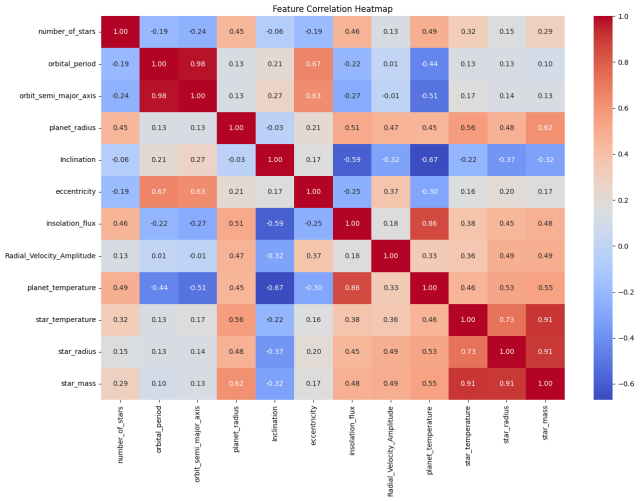

Figure 1. Correlation Matrix of Important Features

the more amount of stars the most heat that is being radiated to the planet

- **Orbital Period** : The time, in days, that takes the planet to orbit its parent star. Even though this does not make intuitive sense, planets with longer orbital periods tend to be further away from their stat, which results in lower temperatures
- **Orbit Semi Major Axis** : The distance between between the long axis of the elliptical orbit of the planet. For Earth, this value is 1 AU.
- **Planet Radius** : The radius of the planet in Earth Radius values. Larger planets, such as Jupiter, retain more of their heat due to their size and compositions, while they have more surface for the star to radiate on. It is measured on Earth Radii values.
- **Inclination** : Inclination, measured in degrees, is the tilt of the planet's orbit. It should not directly affect the temperature and it is probably related with the discovery method of the planets in the dataset.
- **Eccentricity** : It measures how much a planet's orbit deviates from a perfect circle. Planet's with large eccentricity experience huge temperature deviations to the fluctuations in the distance to the their star.
- **Insolation Flux** : It directly affects the temperature since it is the amount of stellar radiation a planets receives.
- **Star Temperature** : The temperature of the host star in the system, counted in Sun Temperature values.
- **Star Radius** : The radius of the host star, measured in solar radii.
- **Star Mass** : The mass of the host star, measured in solar masses.

There is another issue raised regarding the features. Even though these are the features that affect the temperature the most, some of them are very scarce inside the database. For example, out of the close to 6000 planets, if we include all those features we are left with just 1600 planets, a number that would hinder the effectiveness of our model. However, if we just exclude the insolation flux, the eccentricity and the inclination, we are left with 2996 planets, which is an acceptable amount. This serves the aim of the project as well, since the goal is to create a machine learning model that can predict the temperature of a planet with as little data as possible. This is why we created multiple models so that the user can choose the amount of accuracy they want depending on the amount of data that they have available. All models are included in the github and new models, depending on the available feature, can be easily created using the python scripts.

3.3 Dataset Cleaning

In the original dataset, for each planet there were multiple rows of data based on the properties that were calculated by different scientific teams. Even though, they were in accord in many of the properties, they were in disagreement in others, something that would prove detrimental to the machine learning model. To resolve this, we chose to include the row of data with the least amount of NaNs. This turns out to be a happy accident, since we can use the trained model on the complete dataset so that we can conclude if the different numbers calculated by the scientists affect our model.

Also, the names for the columns used by NASA where relatively incomprehensible, so we changed the columns names with more descriptive names. Additionally, the largest temperature in the dataset was an outlier and it negatively affected the training, so it was removed. Lastly, we removed the planets with NaN values in properties that interested us so that we can be as accurate as possible. We decided against using traditional imputation techniques because we wanted to use as much confirmed data as possible.

4 Methodology

4.1 Model Architecture

The model that was implemented is a fully-connected feed forward neural network. It was created using TensorFlow and Keras using Python,

The architecture consists of multiple layers. The first layer is the input layer, where the data from the dataset are inserted. The number of neurons in the input layer equals the number of features in the dataset. After the input layer, the data are passed onto two hidden layers. The first hidden layer consists of 192 neurons with a ReLU (Rectified Linear Unit) activation function and a dropout rate of 0.2. The high number of neurons is vital in order to capture the complex patterns inside the dataset. The ReLU activation function is the most popular function for neural network as it assists with the vanishing gradient problem, where the gradients become extremely small. The Dropout Layer aids in preventing overfitting by randomly deactivating 20% of the neurons. Similarly, the data are next connected to the second hidden layer that has 128 neurons instead of 192 and has the same architecture as the previous hidden layer. We opted for only two hidden layers because after adding more layers the evaluation metrics did not deviate at all. The code for the this model can be seen in Appendix 1.

4.2 Hyperparameter Tuning

The number of neurons has been decided by the techinique of Hyparameter Tuning. Specifically, the HyperBand by the Keras Tuner was utilized to calculate the most effective hyperparameters for the model. We ran the tuning algorithm for 10 epochs and the objective of the function was to minimize

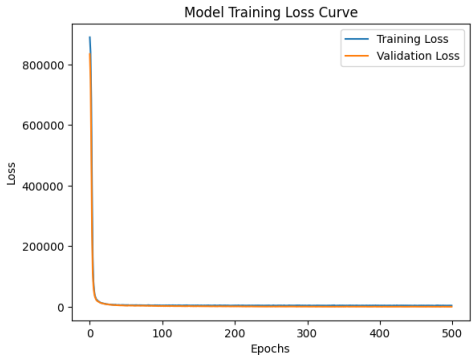

Figure 2. The Loss Curve based on the number of epochs

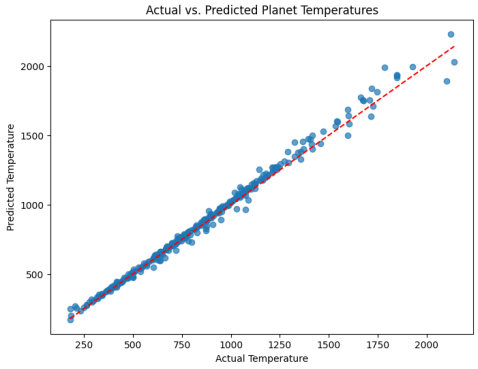

Figure 3. Actual Values versus Predictions

the MAE (Mean Absolute Error) metric. The metrics that the hyperparameter tuning script outputted assisted the model heavily as it greatly reduced the MAE. The script for the Hyperparameter tuning can be seen in Appendix 2.

Moreover, one more important parameter that was considered was the number of epochs that the model would ran. In the first model, we chose arbitrarily the number of 500 epochs and worked experimentally from there. Finally, the optimal number was 2000 epochs as there was no further improvement from there based on figure 2. We can clearly distinguish that no matter the number of epochs the model would not improve.

Lastly, as far as the batch size is concerned, we tried experimenting with 32, 64 and 128 batch sizes, however, it did not affect the model at all. For this reason, we chose in the final model the number 32 for the batch size as it would assist in reducing the training time.

4.3 Scaling

After the construction of the first, unsuccessful model, that resulted in a MAE of about 150, the first thing that was considered for its improvement was the inclusion of scaling the X variables, this would result in the input variables being on a similar scale, and thus, perform better in training. For our purposes, we utilized the StandardScaler() function from sklearn and before the training we saved the scaler in the scaler.pkl file so that it can be reused in a different script. The inclusion of the scaler greatly affected the results of the model and led to a greatly reduced MAE.

4.4 Loss Function

The loss function that was used for the training of the machine learning model was the standard Mean Absolute Error (MAE) function, because our goal was for the model to work correctly for as many planets as possible and the MAE metric provided just that, given that it gives equal weight to all errors. One more option that was considered was a physics-based approach, like the equation for calculating the equilibrium temperature of a planet:

$$T_{eq} = T_{star}\sqrt{\frac{R}{2a}}(1 - A_B)^{1/4} \quad (1)$$

However, such an approach was discouraged, at least for now, so that the results would not be limited by our existing understanding of the parameters that influence a planet's temperature.

Metric	Basic Features	All Features
Mean Absolute Error (MAE)	25.9	17.16
Root Mean Squared Error (RMSE)	37.69	35.78
R Squared Score	0.99	0.99

Table 1. Comparison of actual and predicted temperatures of selected exoplanets using the basic model

5 Results and Discussion

5.1 Evaluation Metrics

As mentioned before, we implemented multiple models depending on the scarcity of the available data. For the evaluation, we will consider both the model with all of the features including, meaning the best performing model, and the one with the least amount of features included, the basic model.

The evaluation metrics that were considered are the Mean Absoluter Error (MAE) , the Root Mean Squared Error (RMSE) and the R square score. The MAE and RMSE are both relatively accurate considering that the target variable is the temperature of a planet. In particular, the fact that the MAE on the basic model is 25 informs us that, on average, the model is 25 degrees off in the temperature of an exoplanet. This difference is insignificant when considering that different places here on Earth can vary that much. Also, the error of the model is not important if it is used for planet classification. For a planet like a hot Jupiter 25 degrees off is unimportant and it can be easily classified based on its temperature regardless of the error.

In addition, R Squared, also known as the coefficient of determinations is extremely high at 0.99. This means that the total variability of the target variable, the temperature, can be easily determined by the independent variables used in the model.

Lastly, in figure 3 we can see the graph of the actual temperatures in the dataset compared with the predictions from the model. It is easily determined that the vast majority of predictions are highly accurate with little deviations, as most planets fit within the ideal fit line.

In Table 2, we can see an example of the dataset in question along with the model's prediction for their

temperature. It can be easily distinguished that there are little differences between the actual and the proposed temperature. The full table for the predictions for the entire NASA Exoplanet Archive is located in the github of the project.

5.2 Solar System Verification

In order to further analyze the model and compare its strengths and weaknesses, we decided to run the trained model against the planets of our Solar System. The results can be seen in Table 3. There are two major conclusions we can draw based on the table. The first is that it performs very well in determining the temperature of the planets close to the Sun and it performs horrifically when it comes to planets far in the Solar System. The reason for this dramatic steep in accuracy is simple. Due to the current methods for detecting exoplanets, like the transit method, the majority of detected, well-documented exoplanets are close to their host star and our model, being trained using the exoplanet database, is not used to dealing with planets like Saturn and Neptune and it does not perform well at all. However, the model is trained to estimate the temperature of exoplanets, hence, for its purpose it is acceptable.

6 Conclusion

In conclusion, the temperature of a planet is one of its most important parameter when considering its habitability and its general scientific interest. The most accurate exoplanet database, the NASA Archive, is missing the temperature for almost 14$ of its planets. For this reason, we have developed a machine learning model, trained on this database, to predict the surface temperature of planets requiring very little data. This model can be used to fill the temperature value on the existing databases as well as estimate the temperature of the exoplanet at the moment of their detection. The model performs extremely well with exoplanets, achive a Mean Absolute Error of 25 degrees on the basic model and 17 degrees on the advanced one.

In the future, it is vital to enhance the model, by probably using synthetic databases in order to increase its accuracy for planets with great distances from their parent star, like Neptune. Additionally, it is critical to experiment with different machince learning architectures, like Random Forest or Gradient Boost in order to exhaust the possibilty of improving the model based on the architecture. Finally, as the science of exoplanets is further developed, it is vital to update the model with new discoveries to increase its accuracy to real world data.

References

[1] NASA, *https://exoplanetarchive.ipac.caltech.edu/docs/data.html*
[2] *https://github.com/KostasVarak/ExoplanetTemperaturePredictor*
[3] B. W. Jones, *Exoplanets – search methods, discoveries, and prospects for astrobiology*, International Journal of Astrobiology, 2008
[4] Florian Lalande et al, *Estimating Exoplanet Mass using Machine Learning on Incomplete Datasets*, astro-ph.EP, 2024
[5] Abhishek Malik, Benjamin P. Moster and Christian Obermeier, *Exoplanet detection using machine learning*, Royal Astronomical Society, 2022
[6] R.L. Akeson et al, *The NASA Exoplanet Archive: Data and Tools for Exoplanet Research*, The Astronomical Society of the Pacific, 2013
[7] David J. Armstrong, Jevgenij Gamper and Theodoros Damoulas, *Exoplanet validation with machine learning: 50 new validated Kepler planets*, Royal Astronomical Society, 2020
[8] C. T. Unterborn and W. R. Panero, *The Pressure and Temperature Limits of Likely Rocky Exoplanets*, Journal of Geophysical Research: Planets, 2018

Planet Name	Orbital Period (days)	Planet Radius ($R_\oplus$)	Star Temp (K)	Star Radius ($R_\odot$)	Actual Temp (K)	Predicted Temp (K)
55 Cnc e	0.7365	2.08	5234	0.94	1958	1923.54
AU Mic b	8.4630	4.07	3700	0.75	593	548.63
AU Mic c	18.8597	2.522	3678	0.74	459	456.67
BD+20 594 b	41.6855	2.23	5766	0.93	546	506.45
BD-14 3065 b	4.2890	21.59	6935	2.35	2001	2184.97

Table 2. Comparison of actual and predicted temperatures of selected exoplanets

Planet Name	Stars	Orbital Period	Semi-Major Axis	Radius	Temp (Actual)	Star Temp	Star Radius	Predicted Temp
Mercury	1	87.9	0.387	0.3	440	5780	1	414.94
Venus	1	224.7	0.723	0.9	415	5780	1	309.30
Earth	1	365.2	1.0	1.0	288	5780	1	267.03
Mars	1	687.0	1.52	0.5	208	5780	1	194.35
Jupiter	1	4331.0	5.2	11.0	163	5780	1	-559.64
Saturn	1	10747.0	9.58	9.0	133	5780	1	-1899.77
Uranus	1	30589.0	19.21	4.0	78	5780	1	-5297.92
Neptune	1	90560.0	30.11	3.9	72	5780	1	4047.69

Table 3. Solar System Planets: Actual vs Predicted Temperatures

Appendix A: THE MACHINE LEARNING MODEL

Listing 1: Sample Python Code

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
import pandas as pd
import shap
import matplotlib.pyplot as plt

# Load the cleaned dataset
file_path = "/content/sample_data/exoplanet_database_new.csv"
df = pd.read_csv(file_path)

# Remove the outlier (if it's the highest value)
df = df[df['planet_temperature'] < df['planet_temperature'].max()]

# Define features (drop 'planet_temperature' which is the target)
X = df.drop(columns=['planet_temperature', 'planet_name', 'insolation_flux', 'eccentricity', 'Incl

# Define target variable
y = df['planet_temperature'].values

# Normalize features (important for deep learning)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Define the model with the best hyperparameters from your search
model = Sequential([
    Dense(192, activation='relu', input_shape=(X_train.shape[1],)),  # units=192
    Dropout(0.2),  # dropout_rate=0.2
    Dense(128, activation='relu'),  # units_2=128
    Dropout(0.2),  # dropout_rate_2=0.2
    Dense(1)  # Output layer
])

# Compile the model with the best learning rate
model.compile(optimizer=Adam(learning_rate=0.0009), loss='mean_squared_error', metrics=['mae'])

# Display the model summary
model.summary()

# Train the model with the specified hyperparameters
history = model.fit(X_train, y_train, epochs=2000, batch_size=32, validation_data=(X_test, y_test)

# Predict on the test set
y_pred = model.predict(X_test)

# Compute evaluation metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
```

```
59
60 print(f"Mean Absolute Error (MAE): {mae:.2f}")
61 print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
62 print(f" R  Score: {r2:.2f}")
63
64 # Plot the training history
65 plt.plot(history.history['loss'], label='Training Loss')
66 plt.plot(history.history['val_loss'], label='Validation Loss')
67 plt.xlabel("Epochs")
68 plt.ylabel("Loss")
69 plt.legend()
70 plt.title("Model Training Loss Curve")
71 plt.show()
72
73 # Actual vs Predicted Plot
74 plt.figure(figsize=(8, 6))
75 plt.scatter(y_test, y_pred, alpha=0.7)
76 plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='dashed')
   # Ideal line
77 plt.xlabel("Actual Temperature")
78 plt.ylabel("Predicted Temperature")
79 plt.title("Actual vs. Predicted Planet Temperatures")
80 plt.show()
81
82 import joblib
83
84 # Save feature names before scaling
85 feature_columns = df.drop(columns=['planet_temperature', 'planet_name', 'insolation_flux' , 'eccen
86 joblib.dump(feature_columns, "feature_columns_basic.pkl")
87
88 # Save the trained model
89 model.save("exoplanet_model_basic.h5")
90
91 # Save the scaler (used for feature normalization)
92 joblib.dump(scaler, "scalerbasic.pkl")
```

Appendix B: THE HYPERPARAMETER TUNER

```
1     # Function to create the model for hyperparameter tuning
2 def build_model(hp):
3     model = Sequential()
4     model.add(Dense(units=hp.Int('units', min_value=64, max_value=256, step=64), activation='relu'
5     model.add(Dropout(rate=hp.Float('dropout_rate', min_value=0.2, max_value=0.5, step=0.1)))
6     model.add(Dense(units=hp.Int('units_2', min_value=32, max_value=128, step=32), activation='rel
7     model.add(Dropout(rate=hp.Float('dropout_rate_2', min_value=0.2, max_value=0.5, step=0.1)))
8     model.add(Dense(1))  # Output layer
9
10    # Compile the model with a tunable learning rate
11    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', min_value=0.0001, max_val
12                  loss='mean_squared_error',
13                  metrics=['mae'])
14
15    return model
16
17 # Initialize the tuner
18 tuner = kt.Hyperband(
19     build_model,
20     objective='val_mae',  # Objective to minimize
21     max_epochs=10,  # Limit to a number of epochs
22     hyperband_iterations=2,  # Number of iterations to run
23     directory='my_dir',  # Where to save the logs and search results
```

```
    project_name='exoplanet_tuning'  # Name of the project
)

# Start the search for the best hyperparameters
tuner.search(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# Retrieve the best hyperparameters and model
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"Best Hyperparameters: {best_hps.values}")

# Build the best model with the found hyperparameters
best_model = tuner.hypermodel.build(best_hps)

# Train the best model
best_model.fit(X_train, y_train, epochs=500, batch_size=32, validation_data=(X_test, y_test))

# Predict on the test set
y_pred = best_model.predict(X_test)
```