Draft version May 21, 2025
Typeset using LaTeX **default** style in AASTeX7

CTA200 Mini Project: JWST Stellar Spectroscopy

Nava Wolfish[1]

[1] *University of Toronto, DADDAA*

1. INTRODUCTION

JWST's Near Infrared Spectroscopic Instrument (NIRSpec) will enable astronomers to decode the spectra of stars at higher resolution and fainter luminosities than previous surveys. However, the implementation of NIRSpec has been stalled due to difficulties in the data pipeline. In this report, I recreate a portion of the progress made by Julia Kim on deciphering JWST spectroscopy. She utilizes past APOGEE spectroscopic surveys and synthetic JWST data, mapping them to a shared latent space using a convolutional neural network (CNN).

In my script `cta200_project.ipynb`, I perform preliminary data analysis on APOGEE and synthetic JWST data. Namely, I will examine APOGEE stellar data and the distributions of the stellar labels T_{eff}, $\log(g)$, and [Fe/H]. I will also briefly discuss the distributions of other elemental fractions. I will inspect the JWST and APOGEE databases, ensuring that the indices correspond to matching stars. I will also drop duplicates and stars only present in one dataset. Next, using Julia's `StarNet.ipynb` notebook and (S. Fabbro et al. 2018), I implement a 1D CNN that, given an input of a star's spectra, can predict the stellar labels T_{eff}, $\log(g)$, and [Fe/H]. This process is a first step to solving the overall JWST data pipeline and closely resembles StarNet from S. Fabbro et al. (2018).

2. QUESTION 1: INSPECTING THE DATASET

2.1. *Read in the catalogue of selected APOGEE measurements. How many stars are included in this catalogue? What stellar parameters and elemental abundances are provided?*

I used `pandas` to read in the catalogue of the APOGEE-DR17 measurements. Before removing duplicates, there are 19800 stars in the sample. In the catalogue, T_{eff}, $\log(g)$, and [Fe/H] are recorded, along with elemental abundances of M, C, N, O, Na, Mg, Al, Si, S, K, Ca, Ti, V, Cr, Mn, Co, Ni, and Ce as they compare to Fe. Also included is the error in each abundance, as well as the star's location (RA, Dec) and average radial velocity.

2.2. *Visualize the dataset by creating the following plots:*

2.2.1. *A 2-dimensional histogram* $\log(g)$ *(surface gravity) vs.* T_{eff} *(effective temperature)*

First, I created a 2-D histogram of $\log(g)$ vs. T_{eff} for the stars in the APOGEE catalogue (Figure 1). I notice a broad-lined linear relationship, where higher temperatures correspond to higher surface gravity.

2.2.2. *A 1-dimensional histogram of the distribution of stars as a function of [Fe/H]*

Next, I plotted a histogram of the iron abundances for the APOGEE star catalogue (Figure 2. The result is a roughly normal distribution around [Fe/H] $= -0.4$, with a small but significant second peak around [Fe/H] $= -1.2$. This indicates that there may be two different stellar populations within the dataset.

2.2.3. *A 2-dimensional histogram of [Mg/Fe] vs. [Fe/H]*

Figure 3 displays the abundance of Mg as a function of [Fe/H] for stars in the APOGEE catalogue. Most points are congregated in a lobe spanning from [Fe/H] $\in [-0.6, 0.25]$, and from [Mg/Fe] $\in (0, 0.2)$. There is a second, fainter lobe spanning from [Fe/H] $\in [-0.8, -0.25]$, where [Mg/Fe] $\in (0.2, 0.4)$.

2.2.4. *A corner plot (a.k.a. pair plot) of all of the stellar labels in the sample*

While all the stellar labels could not fit into one corner plot, I was able to plot a subset of the labels in Figure 4.

Email: nava.wolfish@mail.utoronto.ca

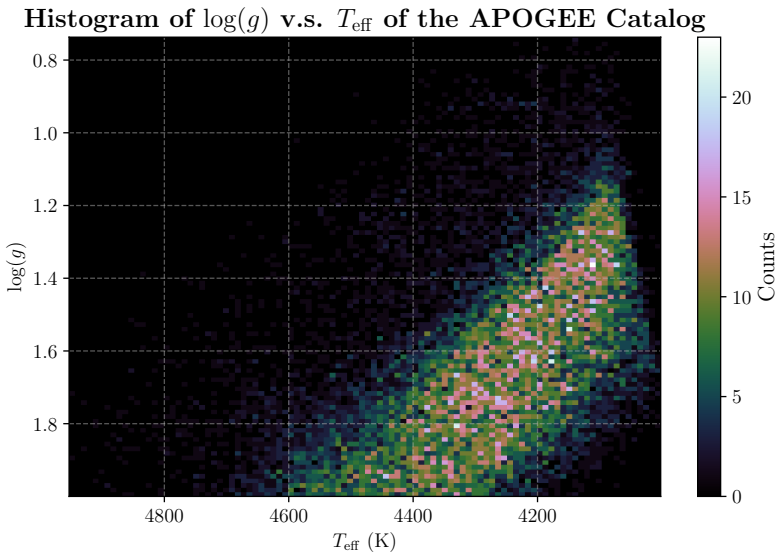

Figure 1. Histogram of $\log(g)$ and $T_{\rm eff}$ in the APOGEE catalogue.

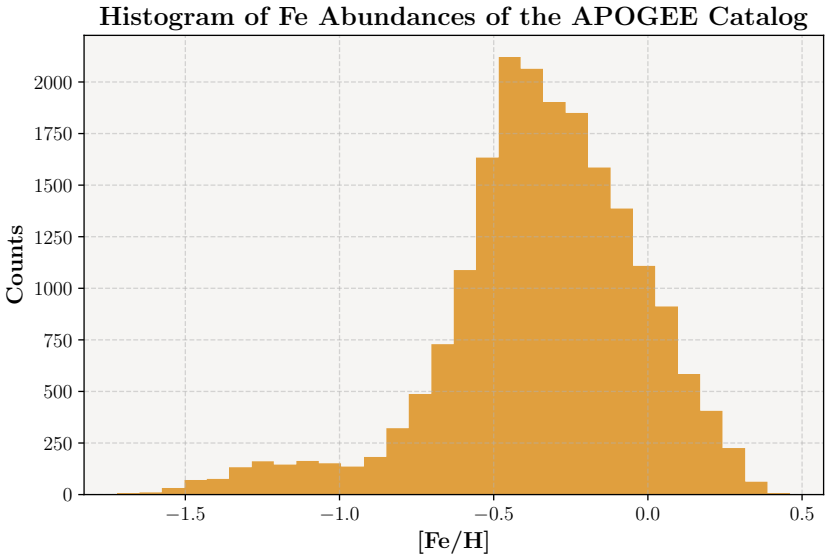

Figure 2. Histogram of iron (Fe) abundances for stars in the APOGEE catalogue.

2.3. *From this APOGEE dataset, choose 10 stars that roughly span the range of T_{eff}, $\log(g)$, and [Fe/H] represented in the dataset. Read in the pseudo-continuum normalized spectra for these stars and separately plot each vs. the APOGEE wavelength grid. What is the wavelength coverage of APOGEE and how many pixels are in each APOGEE spectrum? Why are there portions missing from each spectrum?*

I began by picking 10 random stars using the built-in `sample()` method for `pandas.DataFrame` objects. Next, I reconstructed the APOGEE wavelength grid with guidance from `load_apogee_spec.ipynb`. APOGEE captures wavelengths from $\lambda_{\rm min} = 15100$ Å to $\lambda_{\rm max} = 17000$ Å, with a spacing $\Delta\lambda \approx 0.2$ and a total of 8575 pixels.

Figure 5 displays plots of the selected stellar spectra. I observed that there are wavelengths that consistently have no spectral data. APOGEE's detectors have a gap in these wavelengths, approximately between $\lambda = 15799.5$ Å and $\lambda = 15867.3$ Å, and $\lambda = 16424.0$ Å and $\lambda = 16483.8$ Å.

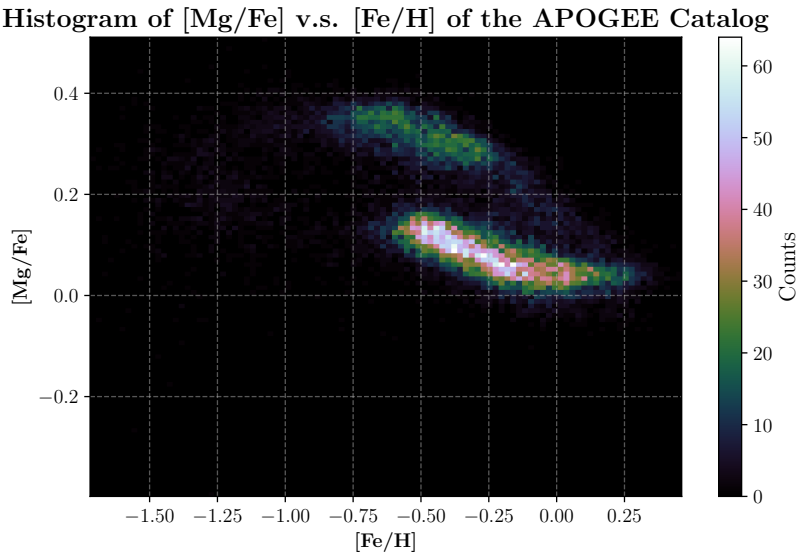

Figure 3. Histogram of Mg v.s. Fe abundances for stars in the APOGEE catalogue.

2.4. *Read in the synthetic JWST dataset, including both stellar labels and spectra. There is one star in the APOGEE dataset missing from this dataset due to convergence issues in the stellar models. Which is it and what are its T_{eff}, log g, and [Fe/H]?*

I loaded in the JWST stellar labels and spectra. JWST data is indexed as a string in the form 'abcded_00'. I borrowed the functions `name_to_idx` and `idx_to_name` from `load_apogee_spec.ipynb` to convert between the JWST index strings to the numeric indices in the APOGEE labels. Next, I compared both indices to find the APOGEE star without a match. This star has index 11215 in the APOGEE dataset, with $T_{\rm eff} = 4256.675781$ K, $\log(g) = 1.841577$, and $[\mathrm{Fe/H}] = -0.023263$.

2.5. *For the 10 stars chosen in part (c), find the corresponding JWST spectra and verify that the stellar labels from both sources match. Recreate the figures from part (c), overplotting the JWST spectra on top of the corresponding APOGEE spectra. What is the wavelength coverage of the JWST/NIRSpec spectra and how many pixels are in each spectrum?*

I mapped each JWST star to the corresponding APOGEE spectrum. I selected the same 10 stars, found their corresponding JWST spectra, and overplotted them in Figure 6. The stellar labels matched to an impressive precision of 10^{-5}. I noticed that the spectral range for the JWST data was much wider than for APOGEE, as expected, as it captures wavelengths from $\lambda_{\rm min} = 9000$ Å to $\lambda_{\rm max} = 18000$ Å with 8192 pixels per spectrum.

2.6. *For one of the selected stars, repeat the above plot but zoom in on the APOGEE wavelength range. Briefly describe the similarities and differences between the real APOGEE and synthetic JWST spectra. Does it seem reasonable that the two spectra correspond to the same star?*

Figure 7 displays the spectrum of one star, but with both APOGEE and JWST mappings. While the data range and SNR clearly differs between both stars, the locations of the spectral lines in each line up succinctly. From this, it is reasonable to conclude both spectra originate from the same star.

4.

5. IMPLEMENTING A NEURAL NETWORK PREDICTOR

In this section, I use Julia Kim's `StarNet.ipynb` notebook to implement a 1D CNN that inputs a star's spectrum and outputs the stellar labels $T_{\rm eff}$, $\log(g)$, and [Fe/H]. The CNN is inspired by S. Fabbro et al. (2018)'s StarNet. First, I dropped stars from the dataset that may have appeared twice, and ended up with 19001 unique stars.

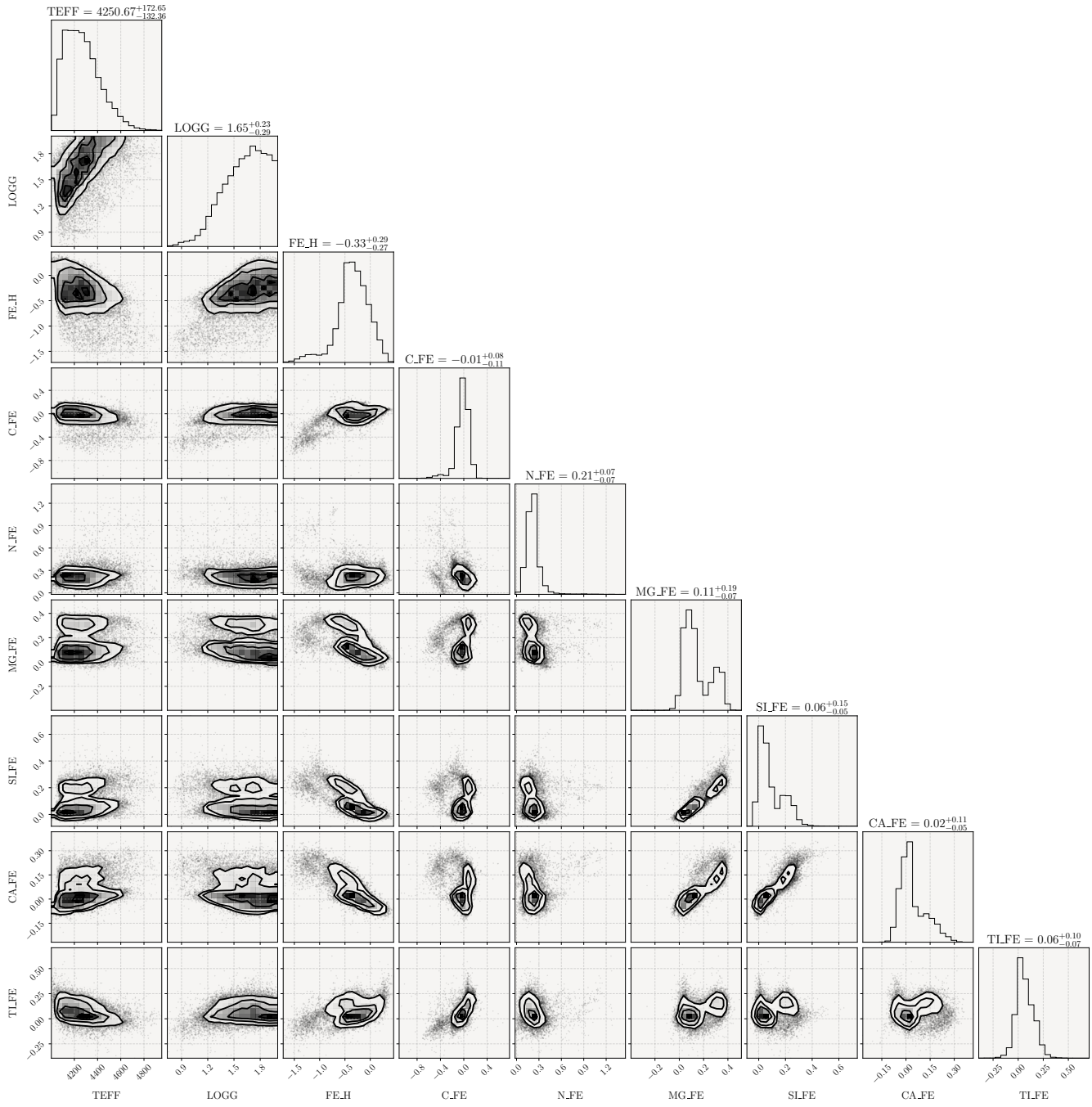

Figure 4. Corner plot of a subset of stellar parameters in the APOGEE catalogue

5.1. *When working with neural networks (and other machine learning techniques), it is common practice to either normalize or standardize inputs and outputs to improve model training and stability. The former scales values to between 0 and 1, while the latter scales the values to have a mean of 0 and a standard deviation of 1. Use either method to rescale all of the stellar labels. Feel free to use the built scikit-learn transforms: sklearn.preprocessing.MinMaxScaler and sklearn.preprocessing.StandardScaler.*

Using `StandardScaler`, I rescaled all stellar labels to have a mean of 0 and standard deviation of 1. `StandardScaler` transforms a dataset according to Equation 1, where μ is the data's mean and σ is the standard deviation. I ensured

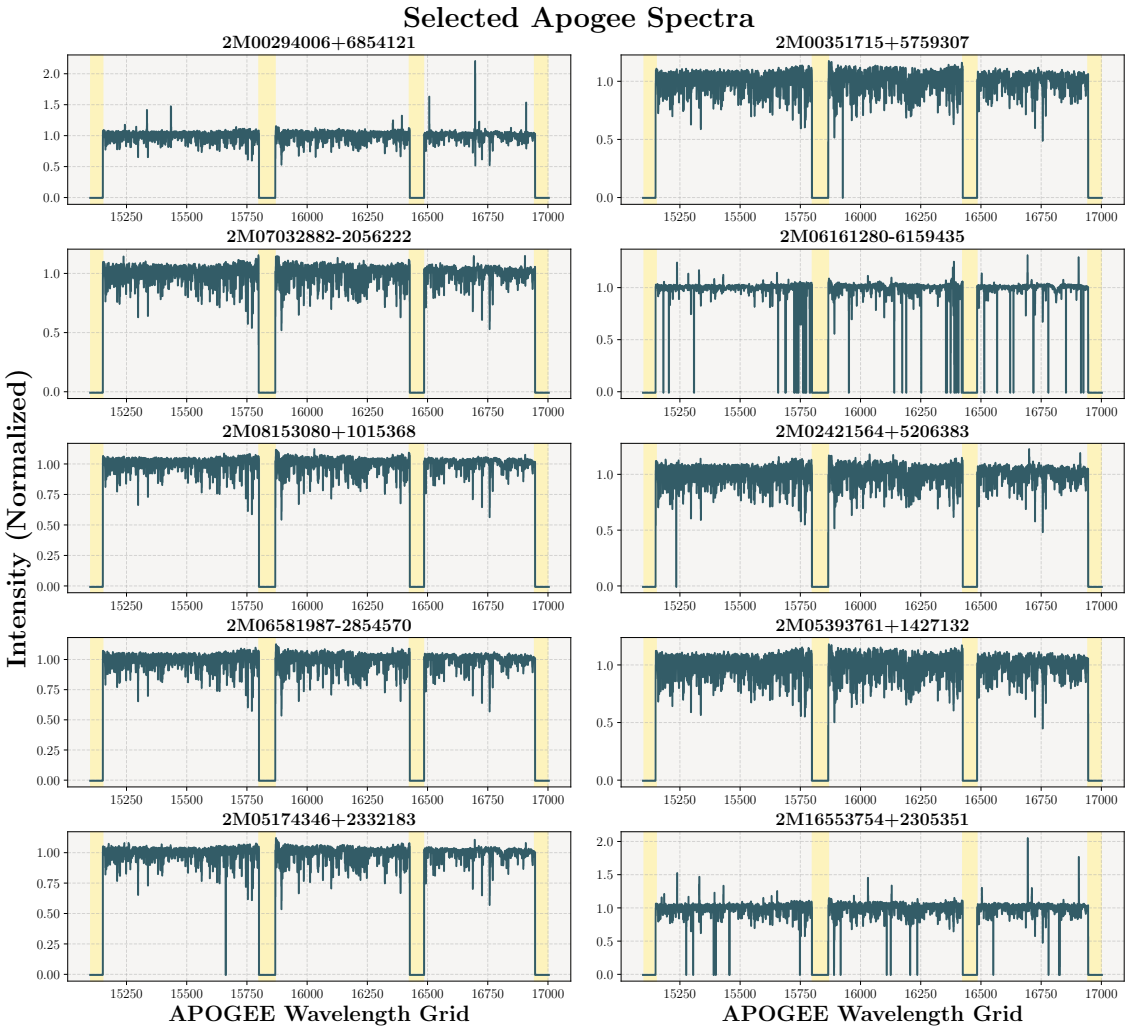

Figure 5. Selected plots of APOGEE spectral data

that I labeled the scaler so I could recover my original values after implementing the NN.

$$x \to \frac{x - \mu}{\sigma} \tag{1}$$

5.2. *Pseudo-randomly assign each star in the sample to one of three sub-samples: a training set, a validation set, and a testing set with the ratio of stars in each set roughly 5:1:2. This is not a magical ratio by any means, but rather a reasonable starting place. You may return to this choice later in the project and realize that the sample should be broken up in a different ratio.*

Prior to splitting the dataset, I set my random seed to 42 so my results would be reproducible using Julia's `set_seed` function. Next, I created a function `split_data` that, given a dataset, would output train, validation, and test sets in the default ratio 5:1:2. The function also returns the sequence used to assign each star to a set, in case it is required later in the code. The function distinguishes between X, my NN's input data, and y. Given that both sets are mutually indexed, my function will separate them accordingly.

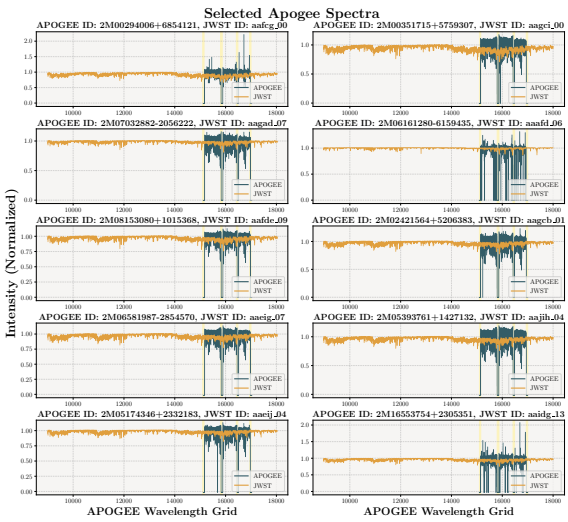

Figure 6. Same selected spectra as in Figure 5, but with JWST synthetic spectra overplotted.

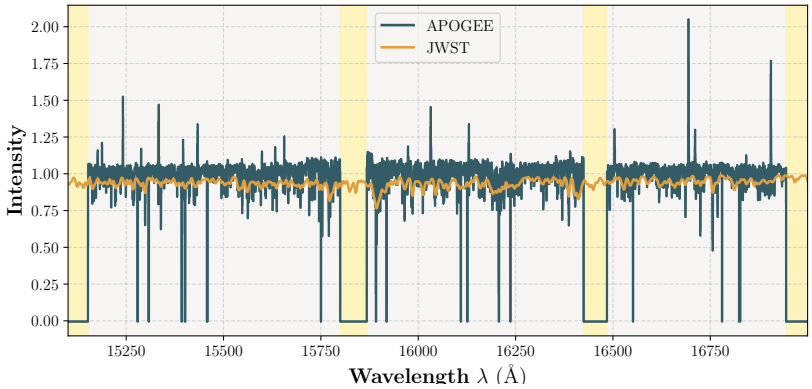

Figure 7. Zoomed in Sample of matching APOGEE and JWST Spectra

5.3. *Using StarNet.ipynb as inspiration, create a convolutional neural network (CNN) that takes a JWST spectrum as input and outputs three stellar labels: T_{eff}, log g, and [Fe/H]. Play around with the number of filters, the filter and pool lengths, etc. to see how they change the input/output shapes and model size. You will likely find the torchsummary Python package helpful. Of course, you may choose to ultimately adopt the same architecture as in StarNet.ipynb, but just know that these are also not necessarily optimized values—just a good place to start.*

I created a class `StarNet`, as guided by Julia's notebook, that stores the network according to S. Fabbro et al. (2018). The class utilizes the `torch` package's `nn.Module` to inherit all the capabilities required for our network. Next, I initiated the model with 4 filters in the first convolutional layer and 16 in the next. Each convolutional filter had a length of 8 pixels. The max pooling window had a length of 4 pixels. The fully connected layers had 256 and 128 nodes respectively, before outputting 3 stellar labels.

The architecture I chose for my NN we ultimately the same as Julia's, largely to enable me to compare my results against hers in the sequential steps of my code. In total, my network includes 8406327 trainable parameters.

5.4. *Following StarNet.ipynb, train your model on the synthetic JWST data. The training process takes two additional hyperparameters, the batch size and the learning rate. What is the importance of these hyperparameters and what values were adopted? Plot the training and validation loss as a function of training step (e.g., "epoch"). What is their trend over time? Is that what you expect? Would you say that the training was completed successfully? Why?*

Next, I trained my model on the synthetic JWST data with help from Julia's `train_model`. To train the model, I defined the batch size N to be 64 and the learning rate η to be 10^{-3}. In a neural network, weights and biases (trainable parameters θ) are updated according to Equation 2. In each pass through the neural network, the loss J_i is determined as a function of the weights and biases θ. The gradient $\nabla J_i(\theta)$ points in the direction of steepest descent, indicating the direction of change to minimize the loss as fast as possible. To update the weights and biases, each epoch dictates a small step in the direction of steepest descent. The size of the step is dictated by the learning rate η, as a step too large may overshoot the optimal θ and a step too small will require more epochs for training. I allow η to decrease and take smaller and smaller steps as I approach the minimum of the loss function. The batch size N determines how many training samples to use for each epoch, or how many samples to "consider" before updating the weights and biases. A batch size that is too large will require many computations of $\nabla J_i(\theta)$ before any improvement in θ, while N too small (Stochastic Gradient Descent) may not converge as quickly or directly as batch gradient descent.

$$\theta_{k+1} = \theta_k + \eta \sum_{i}^{N} \nabla J_i(\theta) \tag{2}$$

At each epoch, I note the validation and training loss. A model is said to converge if both the losses are sufficiently small, and their change over multiple epochs is sufficiently small. I define $\epsilon = 10^{-3}$ as the required improvement in validation loss. If the validation loss has not improved by epsilon in 5 consecutive epochs, the model has converged early and the training ceases. While Julia allowed for a maximum of 15 epochs, I allowed for a maximum of 20 to give my model more time to converge. My model converged after 16 epochs, with no relevant increase in performance after 5 updates of parameters.

As dictated by the process of gradient descent, I'd expect my losses to decrease exponentially with each epoch. Figure 8 confirms my prediction, with an initial sharp decrease in both losses and an eventual plateau as the model converges. Thus, I conclude that the training of my model is successful, as my parameters indicate a minimum in the loss.

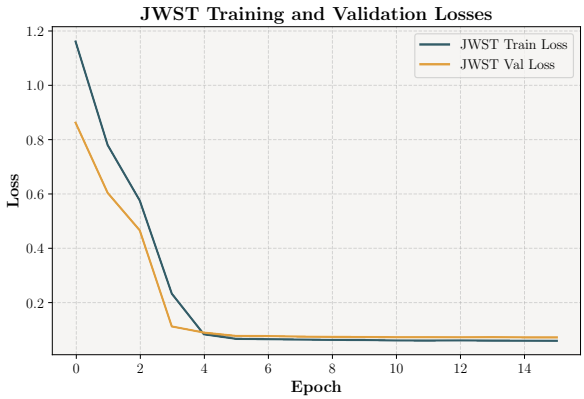

Figure 8. Training and validation losses over 16 epochs for my CNN on JWST spectra

5.5. *Check how well the trained model can predict the T_{eff}, $\log(g)$, and $[Fe/H]$ of the test dataset. For each of these stellar labels, plot the predicted vs. true value as well as the residuals of the predictions (i.e., predicted - true values vs. true values). Visually, does it look like the CNN does a good/bad job at predicting the stellar labels? Are there regions of parameter space where the model performs better or worse? Numerically quantify the performance of the CNN for each stellar label using one or more of the following: 1) the root-mean square error of the predictions, 2) the mean bias of the predictions, and 3) the R^2 score.*

I inputted all my test data into my CNN to see how it would perform on unseen spectra. I obtain the predicted stellar labels and compare to their actual values. I see the residuals between the two are stellar (sorry for my overused pun). All three labels have residuals centered about 0. Furthermore, the predicted and actual labels are related 1-1 with r^2 scores consistently above 0.9. From these results, visualized in Figure 9, I conclude that my CNN did a good job predicting the stellar labels.

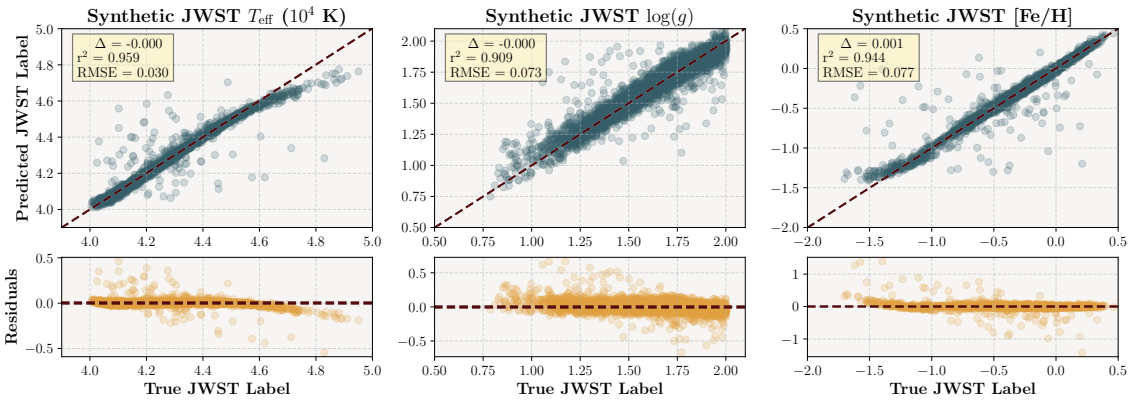

Figure 9. Predictions of stellar labels indicated by my CNN, and their true counterparts. Also included are the residuals for each label, which further indicate the impressive performance of my model.

There are, however, regions where my model did not perform as well. Namely, these are regions with higher temperatures and regions with lower metallicity. We see in these regions the data points tip away from the 1-1 line, resulting in higher absolute residuals.

REFERENCES

Fabbro, S., Venn, K. A., O'Briain, T., et al. 2018, Monthly Notices of the Royal Astronomical Society, 475, 2978, doi: 10.1093/mnras/stx3298