CSCI 490 Write-Up

By: Nick Chandler, Mwangi P., Ian P.

June 30, 2024

Abstract

The problem our research targets is providing fast, reliable predictions of stellar parameters with a quantification of the predictions' uncertainty at a level of performance higher than that of the traditional methods. A large part of the research is devoted to the quantification of uncertainty in model predictions due to the inexact nature of machine learning. Additionally, designing the model to be robust to missing data and features constitutes another significant portion of the research. To accomplish these goals, we draw inspiration from ideas in Probabilistic Machine Learning[3] as well as methodologies in astro-informatics. We detail the results of these analyses and address the effective and ineffective aspects, examining their effects on the pipeline and therefore on predictions.

1 Introduction

Most of the stars in our galaxy are a part of eclipsing binary systems. These systems are composed of two stars revolving around a shared center. The light emitted from these systems follows a periodic pattern, decreasing when one star eclipses the other, reaching a minimum when the larger star eclipses the smaller star. These light curves, equivalently time series of flux values, are captured by multiple satellites. Along with the radial velocity data, we use deep learning to predict various stellar parameters about the systems and the constituent stars.

In order to accomplish this, we use a deep neural network created with the Pytorch Library [1]. Specifically, we use a convolutional neural network architecture which allows us to learn from the spatially related information in the light curves and radial velocities. Additionally, we experiment with a custom loss function derived using maximum likelihood estimation as well as a reparameterization of the model weights. Then we examine data dropping to see how our model behaves under the real-world conditions of missing data. We also examine phase folding to address data security, then we detail the results of applying the Discrete Fourier Transform (DFT) on our input data before passing it into the neural network. Finally, we examine the information richness of our data and the usefulness it has for our model by omitting specific pieces and gauging the model performance after omission. We conclude with a summary of our processes, an analysis of the results we obtained, an evaluation of which techniques performed the best, and directions for future work.

2 Literature Review

A few papers related to the project are detailed in the following sections. Specifically, we examined two papers on astronomy which detail the problem we are trying to solve and then one on statistics which further explores the problem and provides background for some of the methods.

2.1 PHOEBE[2]

Here we detail the PHOEBE Paper which concerns the previous methods for the estimation of stellar parameters.

2.1.1 Introduction and Scientific Goals

The introduction highlights the focus of the paper on the release of PHOEBE 2.3, specifically designed to tackle the challenging inverse problem inherent in the analysis of eclipsing binary systems. Eclipsing

binaries are acknowledged for their crucial role in enhancing our understanding of stellar parameters and refining stellar evolution models. The complexity of the inverse problem is attributed to the intricate parameter space and the diverse morphologies exhibited by these binary systems. PHOEBE is introduced as a modeling code, and a noteworthy aspect is the provision of a common interface for engaging with a variety of algorithms. The historical context is provided, underscoring that previous versions of PHOEBE primarily concentrated on improving the forward model, often leaving the inverse problem unaddressed.

2.1.2 Algorithmic Approach

The paper categorizes algorithms into three distinct types: "estimators," "optimizers," and "samplers." This categorization sets the stage for a more detailed exploration of the functionalities that these algorithms provide in the context of solving the inverse problem.

2.1.3 Estimators

Estimators are defined as algorithms that play a crucial role in determining the parameter space of a system based on the available observational data. These algorithms, as detailed in the paper, offer a means to obtain initial solutions for the system parameters directly from the observational data, alleviating the need for a comprehensive forward model in the initial stages of analysis.

2.1.4 Optimizers and Samplers

The paper introduces optimizers as algorithms designed to minimize the merit function by maximizing the log probability. This minimization process contributes significantly to finding optimal solutions for the system parameters. Simultaneously, samplers are described as algorithms adept at exploring the local parameter space. Their primary role is to uncover any degeneracy between parameters, thereby providing a more nuanced understanding of the characteristics of the system under consideration.

2.1.5 PHOEBE 2.3 Features and Future Plans

The latest release, PHOEBE 2.3, is presented as a significant advancement aiming to address the challenges associated with estimating model parameters without necessitating the construction of the entire model. This innovation is particularly noteworthy as it reduces the learning curve for users, allowing them to estimate parameters efficiently. The introduction of a common interface for multiple algorithms further enhances user accessibility. The paper closes by emphasizing the ongoing developmental trajectory of PHOEBE, with a commitment to incorporating additional algorithms and forward models within the same framework, ensuring its continued relevance and adaptability in the field of eclipsing binary system analysis.

2.2 The EBAI Project

"Artificial Intelligence Approach to the Determination of Physical Properties of Eclipsing Binaries. I." highlights the importance of automatic data analysis due to an anticipated surge in astronomical data from space-based surveys. These surveys were anticipated to generate millions of new Eclipsing Binary (EB) light curves. This fills the scarcity in data analysis in an area where data has typically analyzed through intensive manual methods.

2.2.1 Methodology

The authors propose the use of an artificial neural network (ANN) trained on a sample of 33,235 model light curves. The ANN is designed to output approximate model parameters, such as temperature ratio (T_2/T_1), the sum of relative radii ($(R_1+R_2)/a$), and eccentricity components ($e\sin(\omega), e\cos(\omega)$), as well as the sine of inclination ($\sin i$) for each input light curve. The ANN's training uses a computationally intensive back-propagation algorithm, so the authors use a parallelized version of the algorithm on a Beowulf cluster to speed up the training phase. This bypasses the manual step of parameter estimation, automating the analysis of EB light curves.

2.2.2 Data Pre-processing

Observational light curves pose a challenge to ANNs as they suffer from irregularities, such as incomplete phase coverage and noise. For this problem, authors developed a novel solution called polyfit which fits a smooth curve to the observed data points before feeding them into the ANN. This creates a polynomial chain that fits the data smoothly without requiring the function to be differentiable at the knots. This process improves the data input by making parameter estimation more accurate.

2.2.3 Results

The ANN is tested on both synthetic and real data sets. The synthetic data set includes 10,000 light curves generated similarly to the training set, ensuring a robust evaluation of the network's performance. The real data tests involve 50 binaries from the Catalog and Atlas of Eclipsing Binaries (CALEB) database and 2580 light curves from the OGLE survey.The ANN demonstrates high accuracy, with about 90% of the OGLE sample and nearly 100% of the CALEB sample showing less than a 10% error in output parameter values.

2.3 Bayesian Neural Networks

Given that neural networks have a problem of making overconfident predictions, we seek to mitigate and account for the uncertainty of these predictions using bayesian neural networks.

2.3.1 Bayes Theorem

Recall Baye's Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(A \cap B)}{P(B)},\ P(B) > 0$$

This essentially says that the conditional probability of an event A given an event B can be computed using the (easier to compute) probabilities: $P(B|A)$, $P(B)$, and $P(A)$. While this is the basis for the ideas in Bayesian deep learning, we utilize a more application specific formulation of this theorem. In our case, we define the parameters of our neural network to be Θ, the data that we are training on to be D and Θ' to be a component of the partition of Θ. So, the formula we utilize is:

$$P(\Theta|D) = \frac{P(\Theta)P(D|\Theta)}{P(D)} = \frac{P(\Theta)P(D|\Theta)}{\int P(\Theta')P(D|\Theta')d\Theta'} \propto P(\Theta)P(D|\Theta)$$

Our task is to compute the posterior which represents the uncertainty about the parameters after seeing the data. This leads us to the idea of epistemic uncertainty.

2.3.2 Epistemic Uncertainty

Epistemology concerns itself with the study of knowledge. Epistemic uncertainty follows this definition and can be expressed as, "a lack of knowledge of a system of interest which can be reduced by obtaining additional information." [4] In our approach we use the Bayesian framework as represented by the above equations to account for this uncertainty and take action to mitigate it. In theory, the reparameterization of the model weights can accomplish this mitigation.

2.3.3 Aleatory Uncertainty

Aleatory means random or involving elements of random choice. "Aleatory uncertainty accounts for the natural variation of inputs and parameters; it is irreducible and cannot be decreased with additional knowledge." [4] While we cannot mitigate this, we can still understand how it is affecting our model.

3 Methods

Here we outline several techniques that we tried in pursuit of acceptable results. They consist of the the following: The derivation and implementation of a custom loss function via maximum likelihood estimation, the reparameterization of the model weights, the dropping of portions of the input data, phase folding, and the DFT on our input data.

3.1 Custom Loss Function

To obtain the variance on the model mean estimates for each target, we use a custom loss function. With the assumption that we are trying to predict: $p(y|x)$ for targets y and input data x. Recall that before, the objective was to minimize mean squared error loss:

$$\arg\min_{\theta} \frac{1}{N} \sum_{(x,y)} \|h_\theta(x) - y\|^2 \tag{1}$$

We are now doing maximum likelihood estimation (Derivation 1 in Appendix A.1):

$$\arg\max_{\theta} \prod_{(x,y)} p_\theta(y|x)^{1/N} \tag{2}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{(x,y)} \left(\log|\Sigma_\theta(x)| + (y - \mu_\theta(x))^T \Sigma_\theta(x)^{-1} (y - \mu_\theta(x))\right) \tag{3}$$

We make the assumption that the distribution defined by $p(y|x)$ does not use unit variances. That is the multivariate normal distribution has a learnable diagonal covariance matrix. This gives the following equality (Derivation 2 in Appendix A.1):

$$\arg\min_{\theta} \frac{1}{N} \sum_{(x,y)} \left(\log|\Sigma_\theta(x)| + (y - \mu_\theta(x))^T \Sigma_\theta(x)^{-1} (y - \mu_\theta(x))\right) \tag{4}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{(x,y)} \left(\sum_{i=1}^{k} \log \sigma_\theta(x)_i + \sum_{i=1}^{k} (y_i - \mu_\theta(x)_i)^2 / \sigma_\theta(x)_i\right) \tag{5}$$

We take the output of our model to be: $z = h(x) \in R^{14\times 2}$. We can get μ and σ with:

$$\mu_\theta(x) = z[:, :, 0] \tag{6}$$

$$\sigma_\theta(x) = \exp\left(z[:, :, 1]\right) \tag{7}$$

We exponentiate the second column of the output to ensure no values are negative. The network should learn this but the standard deviations should be logged when they want to be interpreted. This work results in the definition of a python class in Appendix A.2.

3.2 Model Weight Reparameterization

Another method we experimented with was the reparameterization of the model weights. This changes the trained neural network from a deterministic model to a non-deterministic model through the learning of two parameters in place of the traditional one. Specifically, the model learns the mean and standard deviation of a distribution rather than a fixed parameter. Upon inference, the model will sample from each of these learned distributions of parameters before conducting a forward pass through the model as done with a traditional neural network. This was done using the IntelLabs© Bayesian Torch library[5] [7]. This all yields a Bayesian Neural Network (BNN)

3.2.1 Experimental Setup

As seen in the following plot, we tried a few different models trained for varying amounts of time. A short description of each is as follows:

- Baseline_1_25 is the baseline model which uses BossNetPlus architecture with MSE Loss
- bnn_reparam_initial_trial is the BNN without MOPED (pre-training before further optimizing with the reparameterized model) and with MSE loss
- BNN_MOPED_MSE is the bnn with MOPED and MSE loss
- ManyEpsBNNMOPED_MSE is the same as BNN_MOPED_MSE but with more epochs

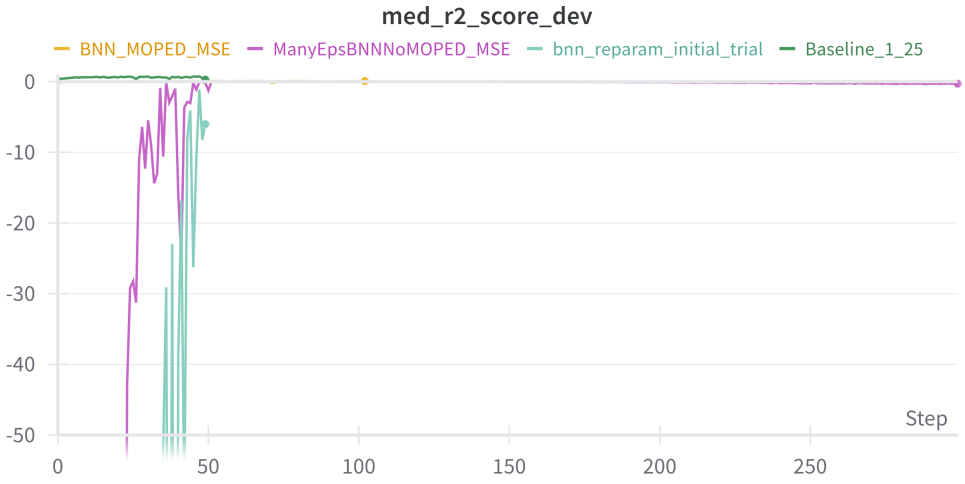

Figure 1: Median Dev R^2 vs. Epoch

3.2.2 Results

From the figure, we can see the baseline outperforms the variations of the bnn we tried. Additionally, the one to one plots had exceptionally poor performance when compared to the baseline. This led to an abandonment of the idea under advisement from Professor Hutchinson.

3.3 Data Dropping

The initial versions of our model were trained using synthetic data, which provides us with complete light curves, radial velocities, and metadata. Real data, however, is very sparse in comparison. Only a small fraction of the light curves and other data is readily available to utilize. In order to prevent our model from overfitting from our large and complete set of synthetic data, we mask our training data to resemble that of the real data the model will handle in production. Our previous implementation of data masking simply dropped entire rows of light curves. For example, for each batch, the model is trained with a random twenty five of the 50 originally available light curves. Our final implementation drops most entire light curves and radial velocities, and for the ones not completely dropped, a portion of each of them will be masked out as well.

3.4 Phase Unfolding

The original observations of data we work with exhibit different cadences and the light curves have different periods and duration of eclipses. This can lead to moments where there are very few data points per eclipse. However, as you continue to observe these systems over long periods of time, you build up phase coverage, which allows us to apply a modulo operation with the orbital period of a system to produce a meaningful light curve. This is referred to as phase folding, and all the data we use to train our models goes through this process. We believe that by reversing this process and using the "unfolded" data will provide us with a better performing model due to the model's new opportunity to differentiate between data at different observed periods. Along with the possible performance improvements that can be achieved, utilizing this phase unfolding technique allows astronomers to leverage our model on a much larger array of data. This stems from the fact that some data does not have a precise enough corresponding period value recorded to employ the phase folding technique with. If our proposed method is correctly integrated, the prevalence of a precise period value will be unnecessary to pass observations through our model.

3.4.1 Implementation

Our proposed method of implementation consists of several steps. For each light channel that is not fully zeroed out by our data dropping methods we significantly up sample our original 512 data points by interpolating between them. We then draw samples at random points in a period and time within the range of time that all our samples span and assign corresponding time encodings to those samples. We also assign a channel encoding to each sample to provide a method to differentiate samples from separate channels. The goal of our process is to efficiently mimic samples that occur at different cadences and points in time within countless slightly varying instances of periods over long lengths of observation. The caveat with this method, however, is the requirement of shifting the model architecture from a convolutional neural network into a transformer. This of course is due to the usage of the time and channel encodings of which transformers are best fit for.

3.5 DFT Transformation

In this section, we detail the results of our use of the Discrete Fourier Transform on the features before they were input to the model.

3.5.1 Preliminaries on the DFT

The DFT is a signal processing technique used to take a discrete (and in this case digital) signal and provide a representation of the signal in the frequency domain. This can be used to provide a different representation of the data input to the model, allowing it to learn different information.

Mathematically it is an operation on the elements of a sequence, defined:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi kn}{N}} \tag{8}$$

for a sequence of length n, $\{x_i\}_{i=0}^{n-1}$.

3.5.2 Experimental Setup

We compared three different instances of the DFT on the model inputs:

- Direct DFT, the DFT was directly applied to the input data. No original input was passed to the model.
- Forward Concatenation, the transformed data was concatenated to the original before being passed into the model. The norm $\frac{1}{n}$ was used.
- Orthonormal Concatenation, the transformed data was concatenated to the original input before being passed into the model. The norm $\frac{1}{\sqrt{n}}$ was used which makes the fourier transform orthonormal.
- Baseline, the input data was unaltered

We trained all methods for 10 epochs as we saw convergence occur within this training length. The loss function in all trials was MSE.

3.5.3 Results

We now examine the figure showing median dev R^2 scores, an indicator of performance on unseen observations.

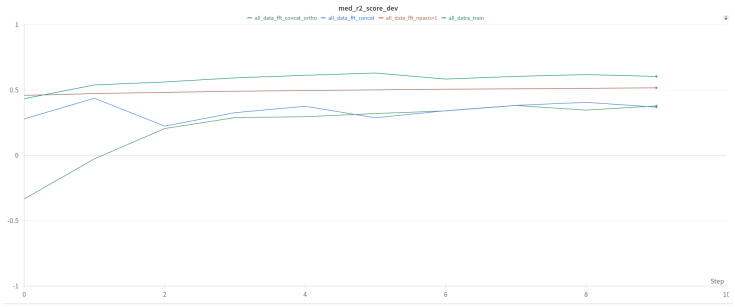

Figure 2: Median Dev R^2 vs. Epoch

Each curve in the figure corresponds to one of the treatments outlined above as follows:

- Direct DFT - all_data_fft_npaxis=1 (rust)
- Forward Concatenation - all_data_fft_concat (blue)
- Orthonormal Concatenation - all_data_fft_concat_ortho (green)
- Baseline - all_data_train (teal)

From the figure we can see that the Baseline performs the best overall (R^2 of 0.6), with the direct DFT performing the second best overall (R^2 of 0.52). We examine selected one to one plots on the dev set after the model is fully trained to analyze the performance of the Direct DFT.

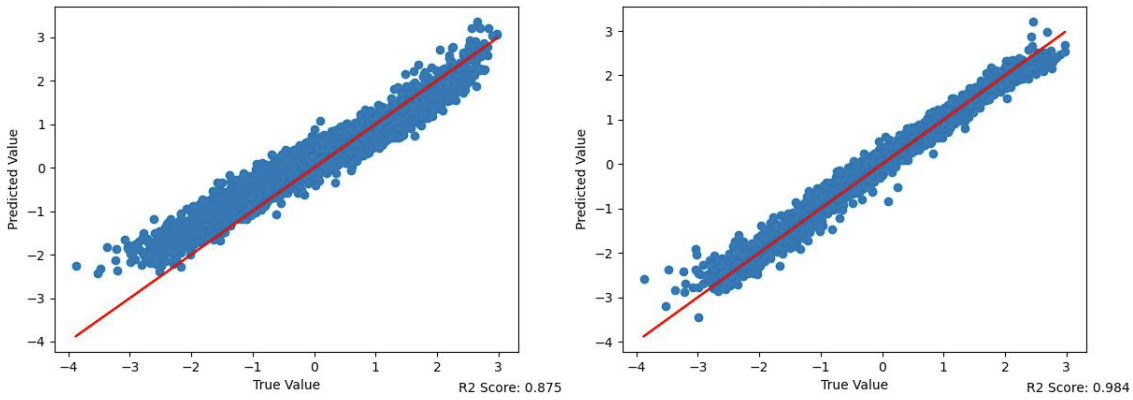

Figure 3: SMA - Baseline (left) & DFT (right)

Here we see the Direct DFT had better performance in predicting the SMA target than the untransformed version.

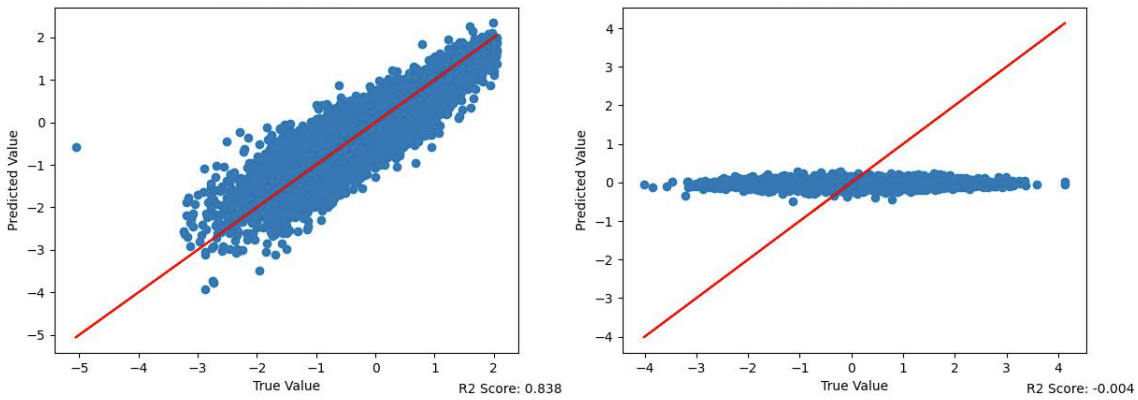

Figure 4: VGamma - Baseline (left) & DFT (right)

As we can see in the VGamma one to one plot the Direct DFT has performance near that of predicting the mean whereas the untransformed version is able to perform moderately well. This is potentially due to the fourier transform removing temporal information. This experiment was fruitful as we found that the Direct DFT can yield some performance gains over the untransformed data on certain parameters. Still, the Direct DFT is unusable on some targets like VGamma. We also experimented with concatenating the fourier transformed data to the untransformed data and then passing this into the model. The results were mostly mediocre and are described well in the plot of R^2 curves since no single target one to one plots stood out.

3.6 Gauging Information Richness of Input Data

In this section, we examine the performance of the model when leaving out entire components of data. It builds upon the techniques used in data dropping but is focused on what components of the input data provide useful information in predicting the targets.

3.6.1 Experimental Setup

We examined several scenarios:

- No Light Curves, No Radial Velocities, Full Metadata - lcnone_rvnone_metafull (blue solid)
- No Light Curves, Full Radial Velocities, No Metadata - lcnone_rvfull_metanone (solid yellow)
- Full Light Curves, No Radial Velocities, No Metadata - lcfull_rvnone_metanone (magenta)
- No Light Curves, Sparse Radial Velocities, Sparse Metadata - lc0_rvsparse_metasparse (grey)
- One Light Curve, Sparse Radial Velocities, Sparse Metadata - lc1_rvsparse_metasparse (blue dotted)
- No Light Curves Full Radial Velocities, Full Metadata - lc0_rvfull_metafull (dotted yellow)

The sparse data is simply the data with some random dropping enabled to simulate inference in production.

3.6.2 Results

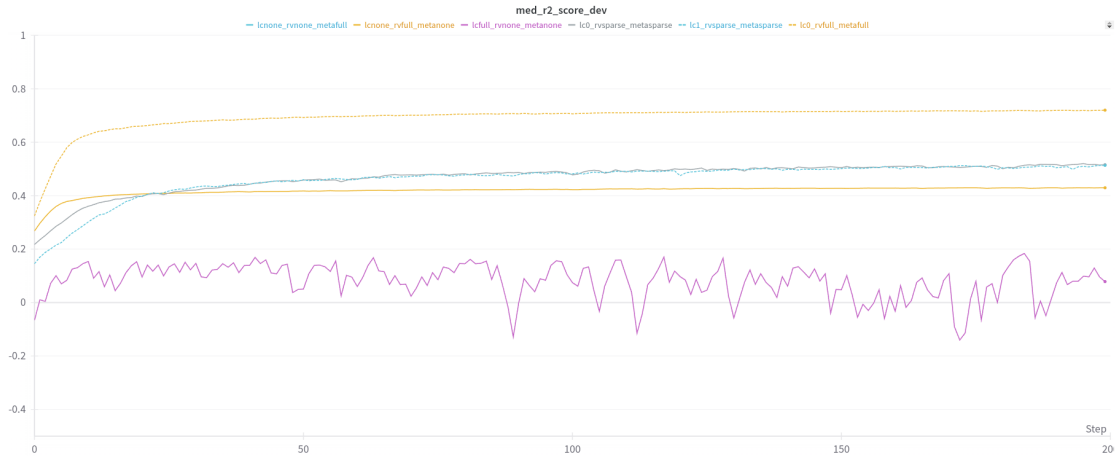

Figure 5: Median Dev R^2 vs. Epochs

From the figure, we can see that the No Light Curves, Full Radial Velocities, Full Metadata scenario performed best with the highest R^2 score at the end of the training period. We can also see that the No Light Curves, No Radial Velocities, Full Metadata performed worst with the R^2 curve not present on the plot due to exceptionally low values. Additionally, the Full Light Curves, No Radial Velocities, No Metadata scenario has the second lowest final R^2 score with very erratic behavior during training. These findings suggest that the light curves do not provide very much useful information in predicting the targets. Selected one to one plots are displayed to further evaluate the information found in each of the data sources.

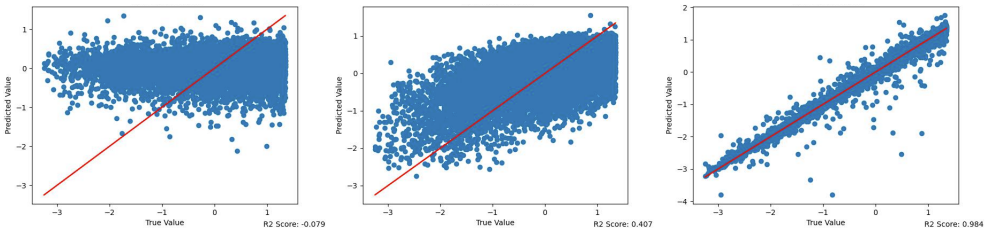

Figure 6: Mass - Full Light Curves (left), Full Radial Velocities (center), Full Radial Velocities & Full Metadata (right)

These figures display the performance of the model with different input data sources on the Mass target. The model trained on all light curves and no other data has the worst performance (shown in the leftmost plot). This can be discerned by the trend in the one to one plot which does not follow the line $y = x$ The model trained on only full radial velocity data (the center plot) has decent performance but not the best as shown by the trend which follows the $y = x$ line somewhat. Finally, the model trained on the full radial velocities and full metadata has very good performance relative to the other two as shown by the approximately linear trend in the one to one plot.
We now examine the difference between the models trained with some metadata, some radial velocities, and either 0 or 1 randomly selected light curves. This was done under advisement by Professor Hutchinson with the idea that one curve, if it is useful, should make a difference in the performance.

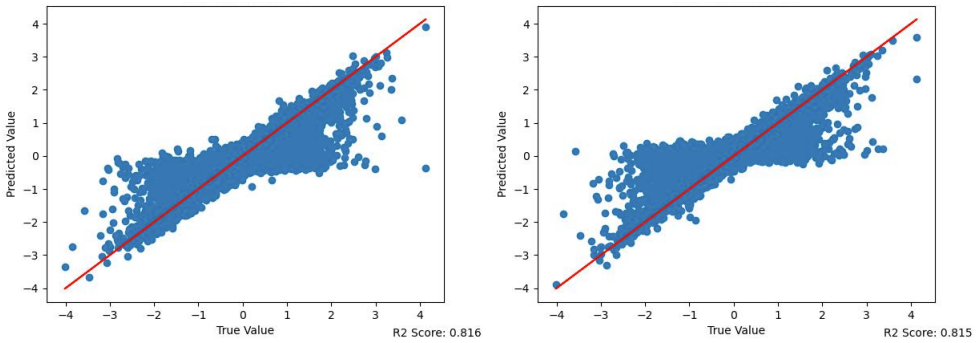

Figure 7: VGamma - 0 Light Curves (left), 1 Light Curve (right)

The above figures display sparse metadata and sparse radial velocities with either 0 or 1 light curves. The conclusion that can be drawn from these plots is that the 0 light curves performs slightly better if not the same due to the slightly higher R^2 value displayed in the plot. Consequently, the light curves likely do not provide very much (if any) useful information to the model in predicting the targets.

4 Conclusion

4.1 Analysis of Results

Through our analyses we have been successful in experimenting with a novel model to predict stellar parameters, and quantify the uncertainty of those predictions given light curve data. While there are still many improvements to be made, the model itself functions relatively well on some parameters given the small amount of time it requires to inference and train. Specifically, we found that the Bayesian reparameterization of the model weights is not very applicable to this problem given the poor performance. While, theoretically, it provided benefits to the construction of the model, in practice it was ineffective. We find that the custom loss function is effective in practice. While it does make the training procedure more volatile, it does yield benefits in that it regularizes the model during training and provides more information on inference. Namely, the uncertainty of the predictions. The data dropping slightly reduces performance but allows the model to be applied within the problem domain it was designed for. The DFT allows the model to learn from new information that may have been hard for it to obtain without, this helps performance on some parameters but hurts it on others, as expected. Finally, the analysis of the information-richness of our input data leads to the conclusion that the light curves provide little information relative to the radial velocities.

4.2 Future Work

Given the conclusions drawn in the preceding section, there are a few directions of exploration that could prove fruitful. Specifically, the lack of usefulness of the light curves could inspire the design of a model which places more weight on radial velocities. The lack of and sparsity of input data in the problem space could prompt the use of interpolation methods to reconstruct the data before passing it into the model as seen in the EBAI paper. Additional research on the application of the DFT to the input data and how the model can use this data for some parameters and the original data for others could be another direction. Finally, exploring the transformer architecture to allow for the analysis of the effect of phase unfolding could be another fruitful direction of further inquiry.

References

[1] https://pytorch.org/docs/stable/index.html

[2] Conroy, K. E., Kochoska, A., Hey, D., Pablo, H., Hambleton, K. M., Jones, D., Giammarco, J., Abdul-Masih, M., & Prša, A. (2020) *Physics of Eclipsing Binaries. V. General Framework for Solving the Inverse Problem* The Astrophysical Journal Supplement Series, 250:34. American Astronomical Society.

[3] Kevin P. Murphy (2023) *Probabilistic Machine Learning*, Kevin P. Murphy.

[4] N.W. Porter & V.A. Mousseau (2020) *Understanding Aleatory and Epistemic Parameter Uncertainty in Statistical Models*, Sandia National Laboratories,

[5] J. -L. Lin et al., "Quantization for Bayesian Deep Learning: Low-Precision Characterization and Robustness," 2023 IEEE International Symposium on Workload Characterization (IISWC), Ghent, Belgium, 2023, pp. 180-192, doi: 10.1109/IISWC59245.2023.00020. keywords: Deep learning;Uniform resource locators;Quantization (signal);Uncertainty;Codes;Computational modeling;Neural networks,

[6] Krishnan, R., Subedar, M., & Tickoo, O. (2020). Specifying Weight Priors in Bayesian Deep Neural Networks with Empirical Bayes. Proceedings of the AAAI Conference on Artificial Intelligence, 34(04), 4477-4484. https://doi.org/10.1609/aaai.v34i04.5875

[7] Prša, A., Guinan, E. F., Devinney, E. J., DeGeorge, M., Bradstreet, D. H., Giammarco, J. M., Alcock, C. R., & Engle, S. G. (2008). Artificial Intelligence Approach to the Determination of Physical Properties of Eclipsing Binaries. I. The EBAI Project. arXiv:0807.1724.

A Appendix

A.1 Custom Loss Derivation

The following derivations come from the mixture density network document, created internally by Professor Hutchinson.

Derivation 1:

We are now doing maximum likelihood estimation:

$$\arg\max_\theta \prod_{(x,y)} p_\theta(y|x)^{1/N} \tag{9}$$

$$= \arg\max_\theta \frac{1}{N} \sum_{(x,y)} \log p_\theta(y|x) \tag{10}$$

$$= \arg\max_\theta \frac{1}{N} \sum_{(x,y)} \log \left((2\pi)^{k/2} \left|\Sigma_\theta(x)\right|^{-1/2} \exp\left(-\frac{1}{2}(y-\mu_\theta(x))^T \Sigma_\theta(x)^{-1}(y-\mu_\theta(x)) \right) \right) \tag{11}$$

$$= \arg\max_\theta \frac{1}{N} \sum_{(x,y)} \log \left(\left|\Sigma_\theta(x)\right|^{-1/2} \exp\left(-\frac{1}{2}(y-\mu_\theta(x))^T \Sigma_\theta(x)^{-1}(y-\mu_\theta(x)) \right) \right) \tag{12}$$

$$= \arg\max_\theta \frac{1}{N} \sum_{(x,y)} \left(\log \left|\Sigma_\theta(x)\right|^{-1/2} + \log \exp\left(-\frac{1}{2}(y-\mu_\theta(x))^T \Sigma_\theta(x)^{-1}(y-\mu_\theta(x)) \right) \right) \tag{13}$$

$$= \arg\max_\theta \frac{1}{N} \sum_{(x,y)} \left(-\frac{1}{2}\log \left|\Sigma_\theta(x)\right| - \frac{1}{2}(y-\mu_\theta(x))^T \Sigma_\theta(x)^{-1}(y-\mu_\theta(x)) \right) \tag{14}$$

$$= \arg\min_\theta \frac{1}{2N} \sum_{(x,y)} \left(\log \left|\Sigma_\theta(x)\right| + (y-\mu_\theta(x))^T \Sigma_\theta(x)^{-1}(y-\mu_\theta(x)) \right) \tag{15}$$

$$= \arg\min_\theta \frac{1}{N} \sum_{(x,y)} \left(\log \left|\Sigma_\theta(x)\right| + (y-\mu_\theta(x))^T \Sigma_\theta(x)^{-1}(y-\mu_\theta(x)) \right) \tag{16}$$

Derivation 2:

$$\arg\min_\theta \frac{1}{N} \sum_{(x,y)} \left(\log \left|\Sigma_\theta(x)\right| + (y-\mu_\theta(x))^T \Sigma_\theta(x)^{-1}(y-\mu_\theta(x)) \right) \tag{17}$$

$$= \arg\min_\theta \frac{1}{N} \sum_{(x,y)} \left(\sum_i \log \sigma_\theta(x)_i + (y-\mu_\theta(x))^T \Sigma_\theta(x)^{-1}(y-\mu_\theta(x)) \right) \tag{18}$$

$$= \arg\min_\theta \frac{1}{N} \sum_{(x,y)} \left(\sum_i \log \sigma_{\theta(x)_i} + \sum_i (y_i - \mu_{\theta(x)_i})^2 \Sigma^{-1}_{\theta(x)^{ii}} \right) \tag{19}$$

$$= \arg\min_\theta \frac{1}{N} \sum_{(x,y)} \left(\sum_i \log \sigma_\theta(x)_i + \sum_i (y_i - \mu_\theta(x)_i)^2 \sigma_\theta(x)_i^{-1} \right) \tag{20}$$

$$= \arg\min_\theta \frac{1}{N} \sum_{(x,y)} \left(\sum_{i=1}^{k} \log \sigma_\theta(x)_i + \sum_{i=1}^{k} (y_i - \mu_\theta(x)_i)^2 / \sigma_\theta(x)_i \right) \tag{21}$$

A.2 Implementation of Custom Loss

```
# New loss function
class DiagCovLoss(nn.Module):
"""
Implements the Diagonal covariance loss as shown in the overleaf doc
"""
def __init__(self, reset_std, reset_condition):
      super(DiagCovLoss, self).__init__()

      self.reset_std = reset_std # for reseting std threshold
      self.reset_condition = reset_condition # for reseting std condition

def forward(self, model_output, targets):
      """
      Inputs:
        model_output -- the output vector of the model of shape (batch_size, num_targets, 2)
        targets -- a vector of shape (batch_size, num_targets)
                    which correspond to the respective inputs
      Outputs:
         adjusted MSE for diagonal Cov for the MVG Dist
      """
      mu_vector = model_output[:,:,0]
      assert mu_vector.shape == targets.shape
      sigma_vector = torch.exp(model_output[:,:,1])
      condition = sigma_vector < self.reset_condition
      sigma_vector[condition].data = self.reset_std*torch.ones_like(sigma_vector[condition])
      loss = torch.sum(torch.log(sigma_vector), dim=1)
            + torch.sum(torch.div(torch.pow((targets - mu_vector), 2), sigma_vector), dim=1)
      return loss.mean(dim=0)
```

Note that in the code, we check that the values of the standard deviations are not too small. That is, we replace any values below some threshold with a larger value. This was done to remedy the problem of vanishing gradients and numerical errors associated with taking the logarithm of 0. These are also hyper parameters that we can tune.