Predicting Gas Temperature in Protoplanetary Discs

Jason Zheng (jkzheng@umich.edu)
Mentored by Maria Han Veiga (mhanveig@umich.edu),
Arthur Bosman (arbos@umich.edu)

August 27, 2022

Abstract

Predicting gas temperatures in protoplanetary discs is an important area of research in astrophysics. A model that is accurate, efficient, and generalizable to new cases provides an understanding of how properties of these discs are related to the eventual planet. Existing models to predict gas temperature are inefficient. We use feed-forward neural networks to efficiently achieve this goal.

Contents

1 Introduction

A protoplanetary disc is a large circumstellar rotating disc of gas and dust in space surrounding a newly formed star [wik]. Protoplanetary discs are young and gas-rich, and possess the physical properties necessary to form a planetary system, as opposed to debris disks, which are older and less gaseous, and typically do not form planetary systems [Bos]. Protoplanetary discs are integral to the modern theory of stellar and planetary formation; thus, understanding how they form and change is crucial. An understanding of the physical and chemical properties of these discs allows us to predict the properties of the eventual formed planets[BvDDH12]. One of the very important properties of these discs is gas temperature and its distribution across the disc. Prediction of gas temperature based on other variables is an important part of our understanding.

Currently, computational models exist to model the relationship between gas temperature, dust temperature, and chemical properties of discs such as the behavior of agents like $[C]$, $[C^+]$, $[CO]$, and $[O]$[BvDDH12] [Bru13]. They investigate the dependence of planetary properties on carbon prevalence, gas/dust ratio, and other properties. These models seek to understand the "chemistry and excitation of simple species (C, C^+, CO, and O) in the disk around a pre-main-sequence Herbig Ae/Be star."[BvDDH12] Unfortunately, current models do not model the crucial gas temperature effectively, as gas temperature is sensitive to the assumptions made about chemical behaviors, as well as certain design implementations. [Bru13]

1.1 Existing Models

An existing model used to study protoplanetary discs comes from Simon Bruderer et al. in 2013 [BvDDH12][Bru13], which uses a series of successive calculations to compute many variables. The model divides a disc into geometric cells, and calculates each cell individually. The resulting representation provides information about geometric trends across a disc.

The model inputs a gas density distribution, and calculates dust temperature. Then, using an initial guess of gas temperature, it computes UV radiation, several chemical properties of agents acting as coolants, and finally an updated guess of gas temperature from the balance between heating and cooling rates. However, due to the codependency between UV radiation and gas temperature, the gas temperature processes must be recalculated many times until convergence.

Formally, the mean UV intensity across a cell is calculated as

$$J_\lambda = \frac{1}{4\pi V} \sum_i I_{\lambda,i} \triangle_{s_i} \frac{1 - e^{-\triangle \tau_{\lambda,i}}}{\triangle \tau_{\lambda,i}}, \tag{1}$$

where V represents the volume of the cell, λ represents the input flux, and $\triangle_{s_i}, \triangle\tau_{\lambda,i}$ represent the distance and optical depth for a photon package i, respectively[BvDDH12].

The prevalences of certain molecular species are obtained from the solutions of the equation

$$\frac{dn(i,t)}{dt} = \sum_i k_{ij} n(j,t) + \sum_{jl} k_{ijl} n(j,t) n(l,t) \tag{2}$$

for a species i, time t, where k_{ij}, k_{ijl} represent the rates of destruction and formation of a species, respectively. This equation is solved for

$$\frac{dn(i,t)}{dt} = 0. \tag{3}$$

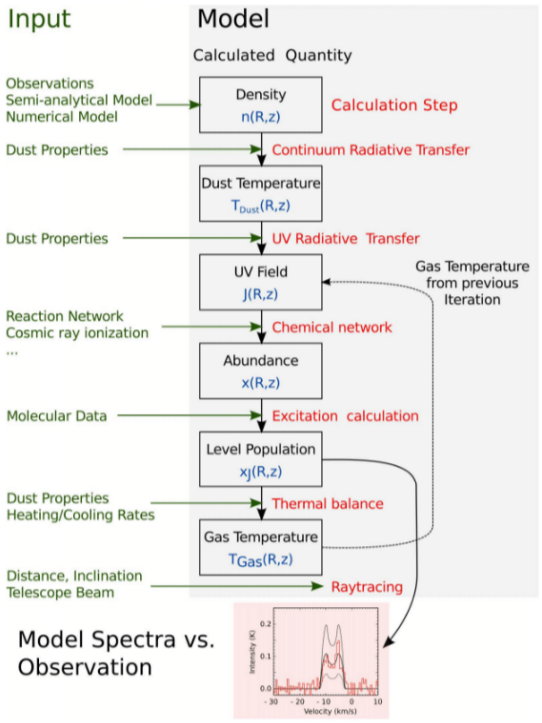

Figure 1: Dust and Lines (DaLi) gas temperature calculation[BvDDH12]. The calculations of UV Field through Gas Temperature must be performed many times.

Finally, from the previous results, the gas temperature is obtained from balancing the heating and cooling rates:[BvDDH12]

$$\frac{d\epsilon}{dt} = \sum_i \Gamma_i - \sum_i \Lambda_i = 0 \tag{4}$$

where ϵ represents the internal gas energy, and Γ_i, Λ_i represent the heating and cooling rates of the cell, respectively. The entire process is visualized in Figure 1.

Due to the strict conditions for convergence, the calculation of gas temperature represents a significant bottleneck in the efficiency of the entire model. At times, a single protoplanetary disc can take up to a full day to model[HV22].

Furthermore, the current model is often inaccurate in predicting gas temperature. The final predicted temperature is very sensitive to assumptions made about chemical properties, as well as the implementation of gas models. As a result, the "calculated temperature is prone to large uncertainty" and contains a "scatter of a factor of a few"[Bru13] across observations.

1.2 Improvements From Existing Models

Instead, we have applied machine learning techniques to learn a direct model to approximate these gas temperatures given the existing dataset of known temperatures. Our model, which we present in later sections, approximates gas temperature significantly faster with reasonable accuracy.

The rest of this report is organized as follows: In section 2, we establish the theory of supervised learning and neural networks that underlies our project. In section 3, we describe the data used and its treatments. In section 4, we analyze processes used in our training and their implementations. In section

5, we present the results of our training. In section 6, we conclude and present possible future works.

2 Preliminary

2.1 Supervised Learning

To learn our model, we employed supervised learning, a subcategory of machine learning in which we learn a function that maps an input to an output based on example input-output pairs[RN09]. In supervised learning, we adopt a function mapping points to labels based on a set of given labeled points. Formally, given a set $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, where each $x^{(i)} \in \mathbb{R}^d$ represents a point and each $y^{(i)} \in \mathbb{R}$ represents a label, we seek to learn a model $f_\theta : \mathbb{R}^d \to \mathbb{R}$ that accurately simulates the relationship between points and labels. For a particular point $x^{(i)}$, each $x_k^{(i)}$ is called a $feature$.

In our model, the number of features in each point $d = 190$. We used a neural network to learn the model.

2.2 Neural Networks

A neural network is a model that approximates a continuous function by applying successive composed linear transformations and nonlinear "activation" functions to an input. Formally, for some $L \geq 2, n_0, \ldots, n_L \geq 1$, $\sigma_i : \mathbb{R} \to \mathbb{R}$ for each $1 \leq i \leq L$, we define the network function $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ as

$$f_\theta(x) = \sigma_L(W^{(L)}\sigma_{L-1}(\ldots(W^{(2)}\sigma_1(W^{(1)}x + b^{(1)}) + b^{(2)})\cdots + b^{(L)}), \qquad (5)$$

where for $1 \leq i \leq L-1$ each $\sigma_i : \mathbb{R} \to \mathbb{R}$ is often nonlinear[Vei22]. The activation function of the last layer, σ_L, is sometimes linear. This network function is highly adaptable, and can fit the needs of our problem by selecting the hyperparameters

- $\sigma_1, \ldots, \sigma_L$, the activation functions
- $n_0, \ldots, n_L$, the sizes of each layer
- L, the total number of layers in the network.

In a training process, a training algorithm iteratively learns the parameters

- $W^{(l)} \in \mathbb{R}^{n_l \cdot (n_{l-1})}$, the linear weights
- $b^{(l)} \in \mathbb{R}^{n_l}$, the layer biases,

for each $1 \leq l \leq L$, in an attempt to approximate a true function $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$.

2.2.1 Activation

Because activation functions give neural networks their nonlinearity and subsequent flexibility, it is important that they are chosen carefully. There are a few desirable features when selecting these functions. Activation functions are more effective when they are:

- Differentiable: The training algorithm will need to compute the derivative in the gradient descent step
- Monotonically increasing[Jai19].

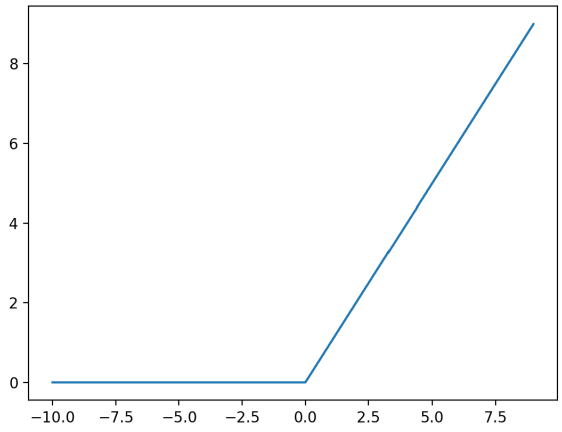

Figure 2: Nonlinear ReLU (Rectified Linear Unit) activation function.

In our model, we used the ReLU (Rectified Linear Unit)[Bro20] activation function (Figure 2), defined as

$$f(x) = max(0, x), \tag{6}$$

in all layers $1 \leq i \leq L-1$ because it is very quickly differentiable. This property of the activation function is crucial- derivatives need to be calculated many times throughout training, so they must be computationally inexpensive to compute[Jai19].

2.3 Loss

Loss functions are used to evaluate a network after the training cycle has completed. A loss function inputs a set of given labels, provided by the dataset, and a set of predicted labels, computed by the network, and outputs a real number representing the model's performance[Sei22]. A lower loss indicates a more effective model. Like activation functions, the particular loss function used is an engineering decision. We used MSE (Mean Squared Error), defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i^{'} - y_i)^2, \tag{7}$$

where n represents the number of points, and for each $1 \leq i \leq n$, y_i represents the true label of point i, and $y_i^{'}$ represents its computed label[Sei22]. We use MSE (Figure 3) because it disproportionately penalizes large errors while forgiving small ones.

2.4 Universal Approximation

We now present an important theoretical property of neural networks. The Universal Approximation Theorem (UAT), proved in 1989, is an extremely important result in machine learning that establishes neural networks as universal models of approximation. [Cyb89]

Theorem 2.1 *Any continuous function* $f : \mathbb{R}^{\aleph} \to \mathbb{R}^{\gtrdot}$ *can be approximated within* ϵ *by a feed-forward neural network for any* $\epsilon > 0$ *iff its activation function* σ *is non-polynomial.*

The UAT is impactful because it guarantees that a solution exists for any applied problem fitting its bounds. However, the theorem comes with important

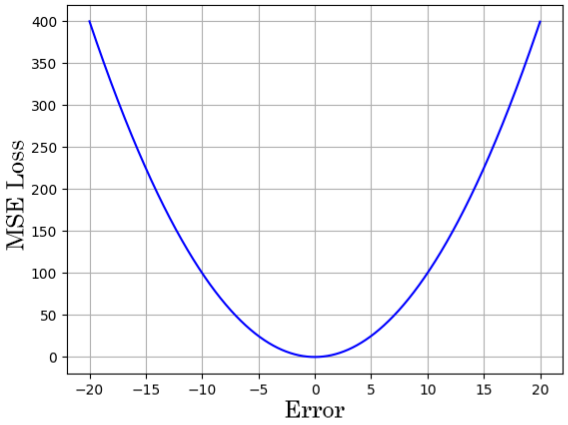

Figure 3: Mean Squared Error (MSE) loss. $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i^{'} - y_i)^2$.

caveats: First, for an arbitrary function, it provides no guarantee for the efficiency of a corresponding network. In practice, for a complicated f and small ϵ, the resulting network may be too large and computation-intensive to be useful. Furthermore, the theorem is nonconstructive. Although it guarantees the existence of a sufficient network for an arbitrary f, ϵ, it does not provide the hyperparameters necessary to achieve this approximation. In practice, we are almost always left to find them ourselves.

3 Data

3.1 Data Sets

To learn our model, we used a dataset of approximately $1,080,000$ vectors in $\mathbb{R}^{191}$, provided by Arthur Bosman at the University of Michigan Department of Astronomy[Bos22], to perform supervised learning. The dataset was generated using existing models of predicting gas temperature[BvDDH12]. Each vector describes the properties of a cell of gas, or a small geometric subdivision, of a protoplanetary disc, as well as includes its gas temperature. We used the Pandas library to randomly split the set into three smaller sets as follows:

- A training dataset, consisting of approximately 74% of the total set
- A validation dataset, consisting of approximately 19% of the total set
- A testing dataset, consisting of the remaining approximately 7% of the total set.

We used the training set to learn the model, the validation set to evaluate the model between epochs, and the testing set to provide a final unbiased evaluation after the training cycle to compare different models. The testing set was only used once the model would no longer be changed. Furthermore, each dataset was separated into a set of features, $x^{(i)} \in \mathbb{R}^{190}$, and a set of labels, $y^{(i)} \in \mathbb{R}$.

3.2 Data Normalization

Before starting the training cycle, it is important to first normalize the data sets to have 0 mean and 1 variance in order to provide easier interpretation of values and maintain consistency across features[Ala20].

The training set was normalized as follows: For each of the 190 feature dimensions and the label dimension, we added and multiplied by a constant factor to achieve 0 mean and 1 variance. For each point $x^{(i)} \in \mathbb{R}^{191}$, each feature $x_d^{(i)}$ was computed as

$$x_d^{(i)} := (x_d^{(i)} - \mu_d)/\sigma_d, \tag{8}$$

where μ_d and σ_d represent the mean and standard deviation of feature d across training points, respectively.

The validation set was normalized analogously using the mean and standard deviation of the training set, in order to preserve consistency. Thus, each feature $x_d^{(i)}$ was normalized as

$$x_d^{(i)} := (x_d^{(i)} - \mu_{d_{train}})/\sigma_{d_{train}}, \tag{9}$$

where $\mu_{d_{train}}$ and $\sigma_{d_{train}}$ represent the mean and standard deviation of feature d in the training set, respectively.

The test set labels were not initially normalized, in order to preserve the unbiased model. However, because our neural network was trained on normalized data and computes a nonlinear function, it was necessary to normalize test set features to have a consistent scale. Thus, each $x_d^{(i)}$ was computed as

$$x_d^{(i)} := (x_d^{(i)} - \mu_{d_{train}})/\sigma_{d_{train}}, \tag{10}$$

where $\mu_{d_{train}}$ and $\sigma_{d_{train}}$ represent the mean and standard deviation of feature d in the training set, respectively. However, when reporting results and final metrics, these normalizations were inverted and labels were reported in the original scale.

4 Training Processes

4.1 Regularization

We performed regularization to limit overfitting the model to training data. Specifically, we implemented early stopping, a process in which the training cycle terminates when it detects that the model's performance on validation data is decreasing[Bro19]. We used MSE score, defined in a previous section, to evaluate the effectiveness of each model.

Because MSE score as a function of epoch is not monotone and can be locally erratic, it would not be appropriate for our cycle to terminate at the first decrease in performance. Instead, the early stopping algorithm evaluates the model on validation data every 5 epochs and stops training when the score has increased from its last computation[i2t19]. Furthermore, since validation metrics tend to be volatile in the first epochs of the training cycle, we added an arbitrarily chosen "grace period" of 8 epochs at the beginning of the cycle in which early stopping was not considered. Formally, the final model was selected through Algorithm 1.

4.2 Crossvalidation and Reporting

Because our training cycle includes significant randomness in the assignment of points to different datasets, we performed crossvalidation to ensure the model generalizes well to new data.

In each program run, the training cycle is run in 5 total folds, with the validation and testing sets making up a distinct portion of the total set in each fold. The final testing metrics are reported as the best of the 5 iterations, with the corresponding model representing the final recorded parameters. This

Algorithm 1 EarlyStopping($numEpochs$)

Require: $numEpochs \geq 1$

$epoch \leftarrow 0$
$bestLoss \leftarrow 0$
$bestModel \leftarrow currentModel$
while $epoch < numEpochs$ **do**
    learnModel() ▷ Updates current model
    if $epoch == 8$ **then**
        $bestLoss \leftarrow currentValidationLoss()$ ▷ Calculates MSE Loss
        $bestModel \leftarrow currentModel$
    else if $epoch > 8 \wedge (epoch - 8)\%5 == 0$ **then**
        if $currentValidationLoss() > bestLoss$ **then**
            return $currentNet$
        else
            $bestLoss = currentValidationLoss()$
            $bestModel = currentModel$
        end if
        $epoch \leftarrow epoch + 1$
    end if
end while
return $bestNet$

practice accounts for statistical differences across portions of the data set and is likely to produce more consistent results. A visualization is provided in Table 1.

5 Results

All program runs were conducted with a 5000 batch size and 0.002 learning rate to maintain consistency. The final learned model was relatively consistent across runs.

5.1 Presented Metrics

In our training cycle, the algorithm decides when to terminate based on validation MSE loss. However, we instead present the final metrics as an R^2 regression score, a number between 0 and 1 representing the quality of the regression model.

Formally, the R^2 score of a model is defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - y_i')^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{11}$$

where n represents the number of points in the testing set and for each $1 \leq i \leq n$, y_i represents the actual gas temperature of point i, y_i' represents its temperature predicted by our model, and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ represents the mean gas temperature across all testing points. We report R^2 score instead of MSE loss because R^2 is not sensitive to constant scaling factors in data and is more intuitive to understand model performance[Hal21].

5.2 Performance and R^2 Score

Figure 4 shows the R^2 score of our model over time on both training and validation data.

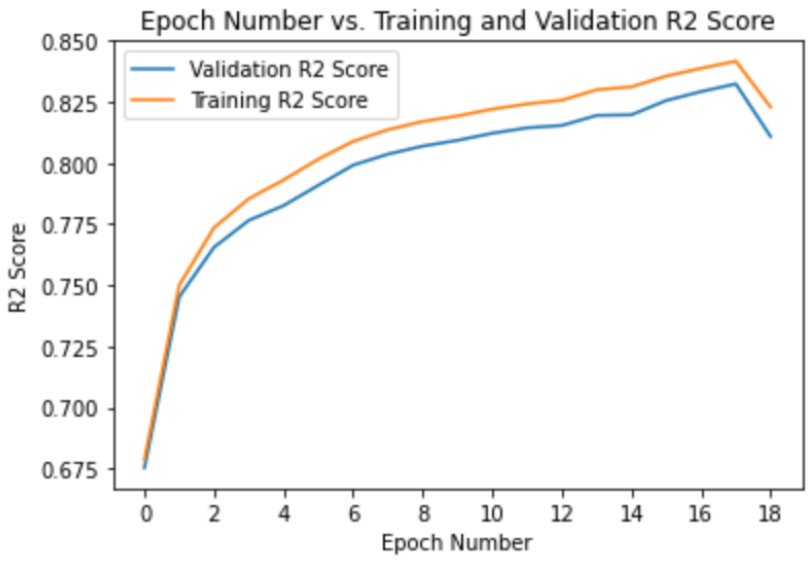

Figure 4: Training and testing R^2 coefficient as a function of epoch number.

Due to the nature of our early stopping algorithm, described in a previous section, the training cycle computes the R^2 score for more epochs than necessary to produce an optimal model. The algorithm computes R^2 score every 5 epochs after epoch 8 and terminates when it detects a performance decrease. Thus, in pictured graph, the algorithm selected the model trained at epoch 13.

5.3 Crossvalidation Folds

The preceding graphs present the metrics of a single training cycle. In practice, 5 total crossvalidation folds were run, each with varying final metrics, in order to find the best model. Table 1 shows the final testing R^2 for each fold of 6 program runs, illustrating how performances varied depending on which training/validation set was selected.

Run	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1	0.805	0.870	0.843	0.908	0.845
2	0.744	0.902	0.872	0.868	0.819
3	0.801	0.833	0.905	0.839	0.827
4	0.803	0.825	0.865	0.861	0.815
5	0.781	0.801	0.871	0.901	0.845
6	0.810	0.866	0.780	0.834	0.837

Table 1: Final testing R^2 for each fold of 6 program runs. Note the nontrivial difference between folds of the same run.

5.4 Model Visualization

In addition to R^2 score, we also present an intuitive visual representation of each testing point's predicted gas temperature by our model as a function of its actual gas temperature. Figure 5 shows this relationship.

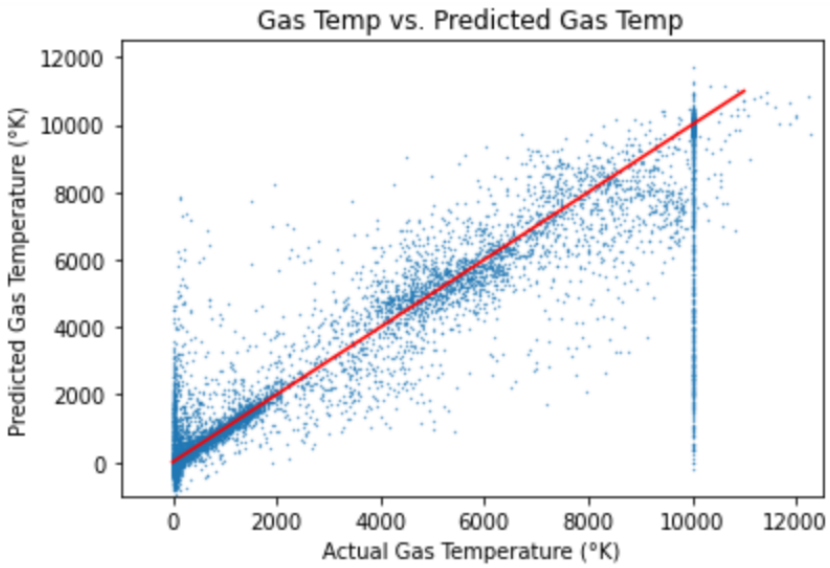

Figure 5: Actual vs. Predicted gas temperature for testing points. $10,000$ randomly selected points of $83,906$ total included for best visual representation.

6 Conclusion and Future Explorations

6.1 Conclusion

In conclusion, protoplanetary disks are subjects of active study in astrophysics, and predicting gas temperature is crucial to understanding the relationship between disks and their eventual planetary systems. An existing model developed by Simon Bruderer et al. in 2013 [Bru13] uses an iterative method of solving several equations to predict gas temperature, but this method is inefficient and often inaccurate. We have developed a new model using supervised learning and neural networks to predict gas temperature quickly.

6.2 Future Explorations

We include a few areas of possible further explorations and improvements to our model.

Firstly, as is obvious in Figure 5, our model appears to be ineffective at predicting points with exactly $10,000°K$ true gas temperature. The network's predicted temperatures for this subset are not drastically better than a simple random guess between $0°K$ and $10,000°K$. An additional observation is that there are more points with $10,000°K$ true temperature than any other value, suggesting a pattern in the underlying model used to generate this dataset. A possible future exploration is to analyze the points with $10,000°K$ true temperature to understand why our model predicts them poorly and how it can be improved.

Furthermore, our model also seems to be not very effective at predicting points near $0°K$. One possible solution to this problem is to modify the loss function to add additional penalty for poorly predicting points with low true temperature.

References

[Ala20] Mahbubul Alam. Data normalization in machine learning, Dec 2020.

[Bos] Arthur Bosman. Protoplanetary disks.

[Bos22] Arthur Bosman. Private correspondence, Jun 2022.

[Bro19] Jason Brownlee. A gentle introduction to early stopping to avoid overtraining neural networks, Aug 2019.

[Bro20] Jason Brownlee. A gentle introduction to the rectified linear unit (relu), Aug 2020.

[Bru13] Simon Bruderer. Survival of molecular gas in cavities of transition disks-i. co. *Astronomy & Astrophysics*, 559:A46, 2013.

[BvDDH12] Simon Bruderer, Ewine F van Dishoeck, Steven D Doty, and Gregory J Herczeg. The warm gas atmosphere of the hd 100546 disk seen by herschel-evidence of a gas-rich, carbon-poor atmosphere? *Astronomy & Astrophysics*, 541:A91, 2012.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[Hal21] Jeff Hale. Which evaluation metric should you use in machine learning regression problems?, Sep 2021.

[HV22] Maria Han Veiga. Private correspondence, Jun 2022.

[i2t19] When do we apply early stopping and how it is helpful?, Oct 2019.

[Jai19] Vandit Jain. Everything you need to know about "activation functions" in deep learning models, Dec 2019.

[RN09] Stuart J. Russell and Peter Norvig. *Artificial Intelligence*. Pearson Education, 2009.

[Sei22] George Seif. Understanding the 3 most common loss functions for machine learning regression, Feb 2022.

[Vei22] Maria Han Veiga. Fully connected neural networks. *Mathematical Foundations of Machine Learning*, pages 133–134, 2022.

[wik] Protoplanetary disk.