Publications of the Astronomical Society of the Pacific, 132:044502 (9pp), 2020 April https://doi.org/10.1088/1538-3873/ab740d

A Multilayer Perceptron for Obtaining Quick Parameter Estimations of Cool Exoplanets from Geometric Albedo Spectra

Timothy K Johnsen[1,2], Mark S Marley[2], and Virginia C. Gulick[1,2]
[1] SETI Institute, 189 N Bernardo Ave suite 200, Mountain View, CA 94043, USA; Tim.K.Johnsen@gmail.com
[2] NASA Ames Research Center, Moffett Blvd, Mountain View, CA 94035, USA; Mark.S.Marley@NASA.gov, VGulick@seti.org

Received 2019 September 12; accepted 2020 February 7; published 2020 March 10

Abstract

Future space telescopes now in the concept and design stage aim to observe reflected light spectra of extrasolar planets. Assessing whether given notional mission and instrument design parameters will provide data suitable for constraining quantities of interest typically requires time consuming retrieval studies in which tens to hundreds of thousands of models are compared to data with a given assumed signal to noise ratio, thereby limiting the rapidity of design iterations. Here we present a machine learning approach employing a Multilayer Perceptron (MLP) trained on model albedo spectra of extrasolar giant planets to estimate a planet's atmospheric metallicity, gravity, effective temperature, and cloud properties given simulated observed spectra. The stand-alone C++ code we have developed can train new MLP's on new training sets within minutes to hours, depending upon the dimensions of input spectra, size of the training set, desired output, and desired accuracy. After the MLP is trained, it can classify new input spectra within a second, potentially helping speed observation and mission design planning. Our MLP's were trained using a grid of model spectra that varied in metallicity, gravity, temperature, and cloud properties. The results show that a trained MLP is an elegant means for reliable in situ estimations when applied to model spectra. We analyzed the effect of using models in a grid range known to have degeneracies.

Key words: methods: data analysis – methods: miscellaneous – planets and satellites: fundamental parameters – planets and satellites: atmospheres

Online material: color figures

1. Introduction

Exoplanet spectroscopic science to date has focused primarily on transiting planets (Kreidberg 2018) and a handful of young giant planets directly imaged in thermal emission (e.g., Nielsen et al. 2019). Next generation space telescopes, including the coronagraph instrument on *WFIRST* (Mennesson et al. 2018) and mission concepts such as the Large Ultraviolet Optical InfraRed Surveyor (LUVOIR) and the Habitable Planets Explorer space telescopes, will have the capability of obtaining photometry and spectroscopy of directly imaged planets in reflected light. Design of mission concepts and notional observing plans for such telescopes requires establishing the desired signal-to-noise ratio and spectral resolution required to meet specified science goals, such as determining the atmospheric metallicity of an extrasolar giant planet. To date such requirements have been set by iterative retrieval studies of simulated spectra (e.g., Brandt & David 2014; Lupu et al. 2016; Nayak et al. 2017; Feng et al. 2018) or information content studies (Batalha et al. 2018).

Retrieval studies are very time consuming as they involve tens to hundreds of thousands of iterative model spectra to be evaluated in order to ascertain which ranges of models best fit the simulated data sets. Retrieval approaches are thus ill-suited for applications involving iterative design studies which aim to specify instrument capabilities or observation parameters. For example, a user may desire to understand if a given observation of a target with a given integration time and other instrument settings, simulated with an online observation simulation tool,[3] would be sufficient to answer a particular science question. Running a full retrieval of the simulated data in such a situation is generally not practical.

Here we report the development of a machine learning tool to facilitate rapid data interpretation given a simulated observation. Our goal is to understand if such a tool could be productively used for quick estimations of planetary properties, not to replace retrievals.

Machine learning algorithms have previously been applied to exoplanetary spectra such as these. Waldmann (2016) used stacked Restricted Boltzmann Machines topped with a perceptron, called RobERt, to identify gasses in simulated highly irradiated exoplanet emission spectra. Zingales & Waldmann (2018) also aimed to address what they termed the "computational bottleneck" associated with retrieval methods by developing a Generative

[3] e.g., http://luvoir.stsci.edu/coron_model.

Adversarial Network, called ExoGAN, to estimate mass, radius, temperature and chemical abundances from hot Jupiter transit data. Márquez-Neila et al. (2018) used a random forest of regression trees to estimate temperature and chemical abundances from simulated transit data sets. Soboczenski et al. (2018) used a convolutional network to estimate chemical abundances of simulated spectra of terrestrial exoplanets in reflected light comparable to those expected from LUVOIR. Cobb et al. (2019) used an ensemble of Bayesian neural networks, called plan-net, to estimate the isothermal temperature and water abundance using a transmission spectrum of WASP-12b. Notably absent from these previous studies are applications to a set of extrasolar giant planets in reflected light, such as we consider here.

Specifically, we begin with a training set drawn from approximately 50,000 model geometric albedo spectra of cool extrasolar giant planets computed by MacDonald et al. (2018) and available online.[4] The models are down sampled in resolution to be comparable to that expected to be obtained by future large space observatories which aim to measure reflected light spectra of terrestrial and giant planets.

We present a Multilayer Perceptron (MLP) to add to the array of machine learning algorithms used to classify exoplanets. As in previous studies the models are distinguished by effective temperature, gravity and metallicity. Unlike the previous studies which considered cloudless spectra we also train on the cloud parameter f_{sed} (Ackerman & Marley 2001), a measure of cloud vertical thickness. Small values are associated with decreased sedimentation efficiency and vertically thicker and more optically thick clouds. Conversely larger values are associated with thinner clouds and spectra which approach the cloudless limit. The MLP we present has 22,308 trainable parameters, which keeps training time on the order of minutes.

The purpose of this study was to provide a quick and easily accessible tool for scientists in the preliminary stage of investigating exoplanet reflectance spectra. Thus, the machine learning algorithm we investigated in this study was an MLP selected to have a relatively low complexity, high portability, and quick train/test times while maintaining an accuracy appropriate for the job of order-of-magnitude on-the-fly estimations. This allows for quick optimizations and estimations based on input dimensions of the spectra and desired output parameters, which both may vary by the scientists' preliminary investigations and goals.

An MLP outputs a continuous response for each output parameter and thus struggles when input degenerate models. Future studies will need to consider utility of other approaches, such as those which address model degeneracy. However, we found that at a given range of parameters, specifically at effective temperatures below 240 K, model degeneracy is not an issue. Even when using models with temperatures above 230 K, the only degeneracies that were present in our data set were due to the clouds being too thin and thus not measurably impacting the spectra.

[4] https://zenodo.org/record/3470301#.XhZ2WchKheU

2. Database

As a database we chose the geometric albedo spectra (A_{g}) of cool gas giant extrasolar planet models computed by MacDonald et al. (2018). Such planets will make bright, appealing targets for future direct imaging observations and their distinctive optical spectra are well suited to test machine learning algorithms. These reflectivity spectra for planets observed at full phase were computed employing the modeling approach of Marley & McKay (1999) as extended to exoplanet spectroscopy (Marley et al. 1999; Cahoy et al. 2010). In reality exoplanet reflected light spectra will neither be available at full phase (see Nayak et al. 2018 for retrievals against such data) nor be cast as albedo spectra since the planetary radius will not be known. Nevertheless, we make both approximations to make the immediate problem more tractable and the capabilities of the MLP to be more straightforwardly evaluated. Each model is for a value of atmospheric metallicity [M/H], gravity, $\log g(\mathrm{cm\ s^{-2}})$, effective temperature, T_{eff}, and cloud sedimentation efficiency f_{sed} (Ackerman & Marley 2001).

The original model set by MacDonald et al. consists of about 50,000 individual geometric albedo spectra. Here we limited [M/H] from 0.0 to 2.0 with intervals of 0.5, $\log g(\mathrm{cm\ s^{-2}})$ from 2 to 4 with intervals of 0.1, T_{eff} from 150 to 400 K with intervals of 10 K, and f_{sed} from 1 to 10 with intervals of 1. We choose this range to better capture the planets most easily detected by direct imaging missions as the warmer planets will both likely be too close to their stars to be detectable and are generally dark in reflected light as they lack clouds (Marley et al. 1999; Sudarsky et al. 2000; Heng & Demory 2013; Angerhausen et al. 2015). Figure 1 illustrates some of these models at a wide range of parameters.

When cloud opacity is negligible, f_{sed} does not appreciably affect the spectra and there are model degeneracies for a given [M/H], $\log g(\mathrm{cm\ s^{-2}})$, and T_{eff}. Those models that were perfectly degenerate with f_{sed} were given an f_{sed} value of 11, because higher f_{sed} corresponds to thinner cloud profiles. There were no duplicates of these degenerate models used in our training, validation, or testing sets. This resulted in 12,783 models in our entire database used to train, validate, and test the MLP. Figure 2 illustrates the distribution of degenerate models for each parameter. There were no other degeneracies outside of those models degenerate with f_{sed}.

2.1. Adding Gaussian Noise to Model Spectra

We simulated uncorrelated noise in the model spectra at various levels. Random, Gaussian noise was added by individually calculating the average albedo of a spectrum and using a percentage of that average as the standard deviation, σ, for the random distribution. We synthesized noise for every

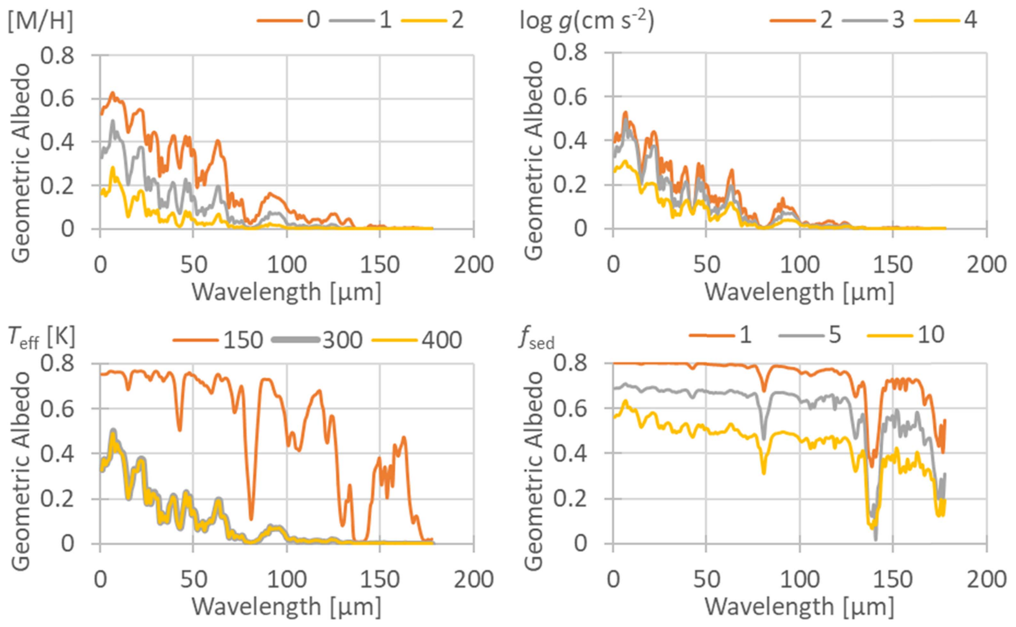

Figure 1. Model albedo spectra of generated exoplanets. Each panel shows a range of model spectra over the domain considered here for that given parameter. The gray spectrum is the same for panels (a) through (c): ([M/H] = 1.0, log(g) = 3.0 cm s^{-2}, $T_{\rm eff}$ = 300 K, $f_{\rm sed}$ = 5). The orange and yellow spectra in each figure are respectively the smallest and largest values for the given output parameter, assuming the other parameters remain the same as the gray spectrum. However, panel (d) uses (1.0, 3.5, 190, 5) for better contrast.

(A color version of this figure is available in the online journal.)

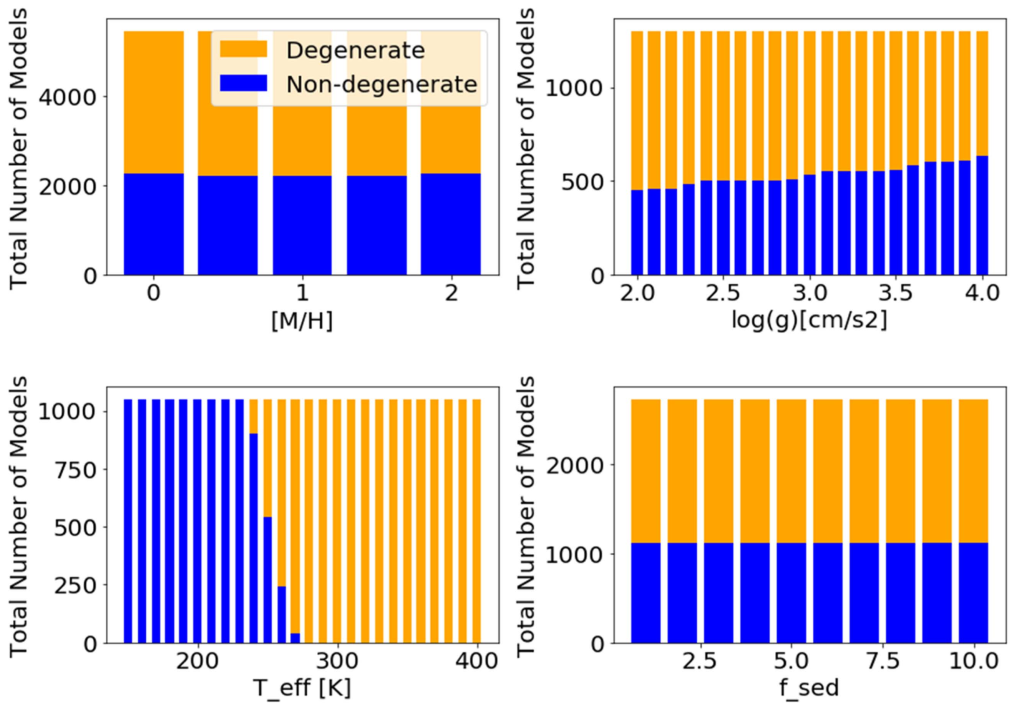

Figure 2. Bar charts comparing the number of models which are either degenerate or non-degenerate at different parameter values. At temperatures of 230 K and below, there are no degeneracies. At temperatures between 240 and 270 K, the number of models degenerate with $f_{\rm sed}$ is a function of log g(cm s^{-2}). At temperatures of 280 K and above, all models are perfectly degenerate with $f_{\rm sed}$.

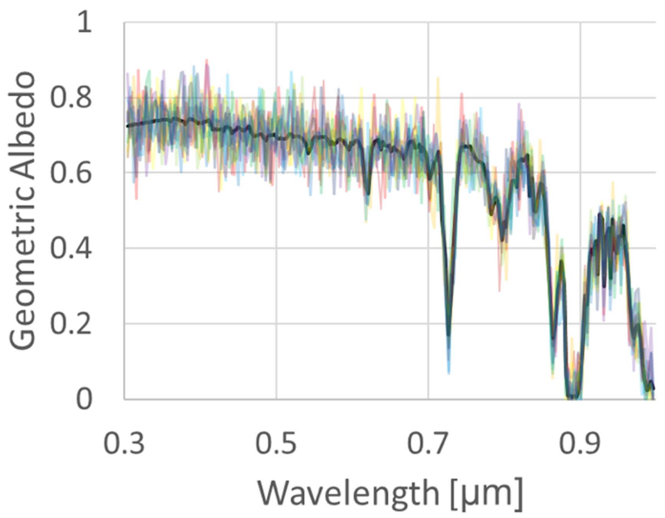

Figure 3. Model spectrum (black) with added noise (color), each color corresponds to 1 of 10 different runs of adding 10% Gaussian noise, uncorrelated in wavelength. Model: ([M/H] = 2.0, $\log(g) = 2.7$ cm s^{-2}, $T_{\mathrm{eff}} = 210$ K, $f_{\mathrm{sed}} = 4$).
(A color version of this figure is available in the online journal.)

data point by adding a normally distributed number with the corresponding σ for a given spectrum. Albedo values were bound between 0 and 1. If random noise would otherwise exceed this threshold, the values were set to either a 0 or 1, whichever was closer. We used three levels of noise: setting σ equal to either 5%, 10%, or 20% of the average, corresponding to signal to noise ratios of 20, 10, and 5, respectively. Figure 3 shows an example of Gaussian noise added to a model spectrum.

Noise draws were randomly computed ten times for each model spectrum at each noise level. An original model spectrum was thus associated with 30 additional noisy models (10 for each noise level). Note that we did not attempt to emulate any noise model for a particular notional instrument as these frequently change. However, such a modification could easily be included in our approach. Noisy models were used in all three data sets: training, validation, and testing.

2.2. Spectral Wavelength Range and Resolution

As the number of data points input into the MLP increases, the training time exponentially increases as each new input dimension requires a new set of weights that then need to be iteratively trained. This phenomenon can cause the training time to increase to an infeasibly high value. We require training time to be minimal, so that scientists may iteratively retrain new MLP's to cater toward their goals. At full resolution, training was not able to converge to a reasonable accuracy (yielding R^2 values between .7 and .8 with no noise added) nor within a feasible training time (12 hr for 8000 epochs). To bring training time down to the order of minutes, a feasible on-

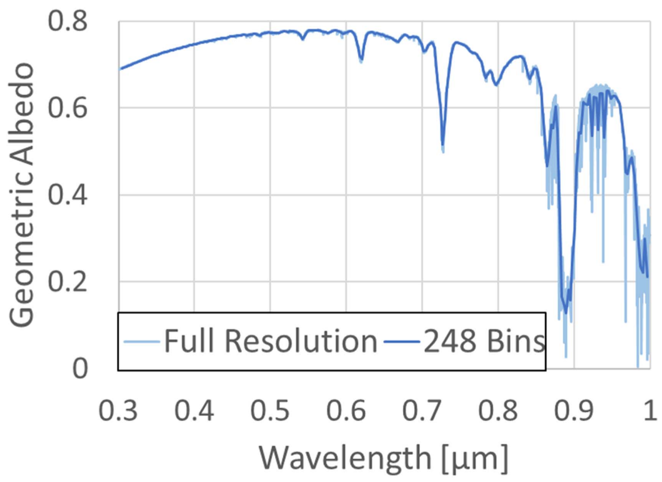

Figure 4. Comparing a model spectrum ([M/H] = 0.0, $\log(g) = 2.0$ cm s^{-2}, $T_{\mathrm{eff}} = 150$ K, $f_{\mathrm{sed}} = 1$) at full resolution to one after being binned to a lower resolution.

Table 1
Example Spectral Matrix

A_g (0.3042 μm)	⋯	A_g (0.9958 μm)
$s_{1,1} = 0.772$	⋯	$s_{1,I} = 0.322$
⋯	⋯	⋯
$s_{N,1} = 0.107$	⋯	$s_{N,I} = 0.811$

the-fly training time, we binned the models to a lower number of dimensions.

The wavelength, λ, range used was $0.3028 \leqslant \lambda < 0.9972$ μm with bin intervals of 0.0028 μm, corresponding to a spectral resolution $R = \lambda/\Delta\lambda \sim 270$ at 0.75 μm. Other decomposition and machine learning techniques can be used to reduce dimensionality, however we found binning to be sufficient. Figure 4 compares a model at full resolution to one binned down.

2.3. Matrix Notation

Let $\boldsymbol{S}$ be an $N \times I$ matrix of spectra for a given data set. Each row in $\boldsymbol{S}$ ($s_{n,*}$) is a different spectrum where each element ($s_{n,i}$) is a binned albedo value indexed as the ith component. Bin names are the average wavelength for that bin's range. Each $\boldsymbol{S}$ has a corresponding $N \times K$ matrix, $\boldsymbol{L}$, containing labels for the spectra. Each row in $\boldsymbol{L}$ ($l_{n,*}$) is the respective label for the appropriate spectrum ($s_{n,*}$), where each element ($l_{n,k}$) is a parameter value for that given spectrum: [M/H], log g(cm s^{-2}), T_{eff}, and f_{sed}. An albedo matrix corresponds to a matrix of labels, L, and an example is in Tables 1 and 2.

3. Multi-layer Perceptron

A neural network is a machine learning algorithm that uses a network of connected, weighted nodes to fit input to output.

Table 2
Example Labels Matrix

[M/H]	Log g(cm s^{-2})	$T_{\rm eff}$ (K)	$f_{\rm sed}$
$l_{1,1} = 0.0$	$l_{1,2} = 2.0$	$l_{1,3} = 150$	$l_{1,4} = 1$
…	…	…	…
$l_{N,1} = 2.5$	$l_{N,2} = 4.0$	$l_{N,3} = 400$	$l_{N,4} = 10$

The network is supervised, in the sense that it trains on rows in $\boldsymbol{S}$ to adjust the weights to better fit to $\boldsymbol{L}$. With trained weights, the network can estimate parameters of novel spectra.

Figure 5(a) illustrates the type of neural network we used, a MLP (Rosenblatt 1958). Each color indicates a different layer or nodes. The red, input, layer is used to input a row of $\boldsymbol{S}$. Each input node reads a different element in the row. The set of orange, hidden, layers is the heart of an MLP. Each hidden node is an obfuscated feature that the MLP must learn by summing weighted nodes in the previous layer and inputting the sum into an activation function. There can be multiple hidden layers with a different number of nodes in each layer, referenced by the set $\boldsymbol{H}$. The output, green, layer consists of output parameters we are trying to map from input. Output nodes work like hidden nodes, by summing weighted hidden nodes from the previous layer and inputting the sum into an activation function. Different activation functions can be used depending on the purpose and limitations of data flowing into and out of that node.

Figure 5(b) shows the activation function that we used for hidden nodes, Exponential Linear Units (ELU) (Clevert et al. 2015). ELU functions are simple linear functions for positive weighted sums, but output is bound between -1 and positive infinity. ELU is a modern version of Linear Units (LU) which is a linear function bound between 0 and positive infinity. The training process, as detailed later, relies on derivatives to update weights. Giving ELU a simple diminishing output for negative sums improves the training process over LU, because training can better escape negative values.

Figure 5(c) shows the activation function that we used for output nodes, Sigmoid, otherwise known as the logistic function. Sigmoid activation functions were used for the output nodes to bind output values to the range used for each parameter. For example, we trained the network on spectra modeled from planets with varying temperature between 150 and 400 K. We can stretch the sigmoid function so that it asymptotically reaches 140 K as a lower bound (giving a buffer of 10 K as determined by the intervals between 150 and 400 K) and similarly an upper bound of 410 K.

Let M be the total number of layers in an MLP including input, hidden, and output. Quantity m varies from 1 to M, representing the indexed layer in the network where the input layer corresponds to $m = 1$ and output layer corresponds to $m = M$. The hidden layers have the indices $m = 2$ to $m = (M-1)$.

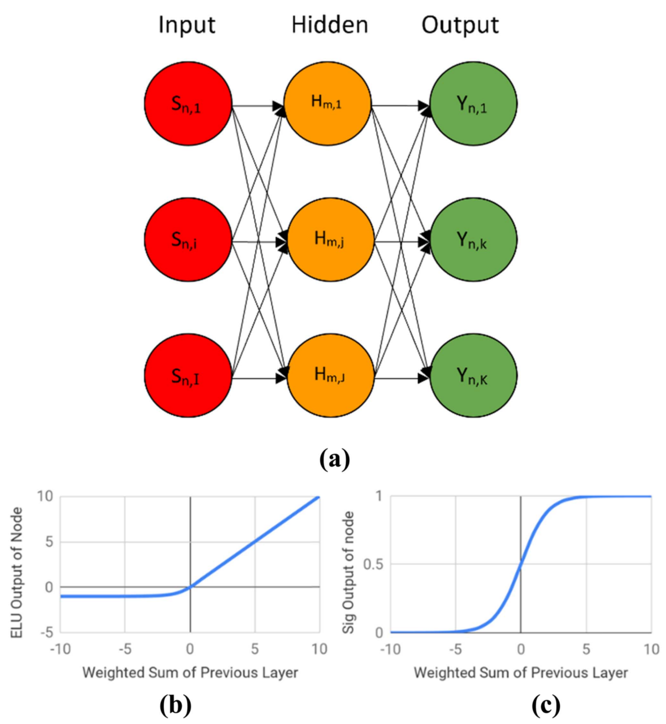

Figure 5. (a): MLP. An input row of $\boldsymbol{S}$ ($s_{n,*}$) feeds information into global hidden nodes $H = \{h_{m,1} \ldots h_{m,J\_m}\}$ which are used to estimate an output row of Y ($y_{n,*}$). (b): ELU activation function, $y = x$ for positive input values and $y = e^{x}-1$ for negative input values. (c): Illustration of a Sigmoid activation function, $y = 1/(1+e^{-x})$.

The number of nodes in each layer can vary. The input layer has I number of nodes, as determined by the dimension I of $\boldsymbol{S}$. The number of nodes in each hidden layer must be determined by the user, as presented in Section 4. We used 4 output nodes, one for each label: [M/H], log g(cm s^{-2}), $T_{\rm eff}$, and $f_{\rm sed}$.

There are trainable parameters that the MLP must learn to map input to output. These parameters consist of a set of weights, $\boldsymbol{W}$, and bias terms, $\boldsymbol{B}$, with elements in the sets indexed to each node by the layer number, m, and node number in that layer, j (such as $w_{m,j}$ and $b_{m,j}$). Quantity J_m refers to the number of nodes in that layer. Weights are used to take a weighted sum of nodes in the previous layer. Bias terms are used as a shift in the activation function. With weights and bias terms, data propagates forward from input layer to hidden layer using Equation (1), hidden to another hidden using Equation (2), and hidden to output with Equation (3). Where f is the activation function. The flow of data, as propagated forward with Equations (1)–(3), is illustrated in Figure 5(a).

$$h_{2,j} = \left(b_{2,j} + \sum_{i=1}^{J_1=I} (w_{1,i} \ * \ s_{n,i})\right) \qquad (1)$$

$$h_{m,j} = \left(b_{m,j} + \sum_{i=1}^{J_{m-1}} (w_{m-1,i} \ * \ h_{m-1,i})\right) \qquad (2)$$

$$y_{n,k} = \left(b_{M,k} + \sum_{i=1}^{J_{M-1}} (w_{M-1,i} \ * \ h_{M-1,i})\right). \tag{3}$$

The grid of trainable MLP parameters, $\boldsymbol{\Theta} = \{\boldsymbol{W}, \boldsymbol{B}\}$, must be explored to find the optimal values that give lowest output error. Supervised machine learning refers to iteratively introduce training vectors, with known labels, to the MLP. The MLP trains on the feature vectors by exploring the grid space to find values that minimize the error between output values and labels. Stochastic Gradient Descent (Rumelhart et al. 1986) is the process used to explore the grid space.

Trainable MLP parameters in Equations (1) and (2), $\boldsymbol{\Theta}$, were at first randomized to small values normally distributed around 0 with a standard deviation, STD, found from Equation (3). This gives a near zero deviation which scales to the size of the MLP. Near zero values were adequate starting points for the grid search, as zero is a critical point in both ELU and Sigmoid activation functions.

$$\mathrm{STD} = 1/\sqrt{I + K}. \tag{4}$$

We then iteratively introduced a subset of training spectra to the MLP. After each iteration, a loss function was used to measure the error between labels and output values. We used mean squared error as our loss function, Equation (5). The 0.5 coefficient was used so that it vanishes after taking the derivative. Squared error was used because: (1) it measured error in continuous values as opposed to other loss functions which focus on binary classification error; and (2) it punishes larger residuals, as opposed to other loss functions which balance out small and large residuals more.

$$\varepsilon_k = 0.5(l_{n,k} - y_{n,k})^2. \tag{5}$$

After each iteration the average error, ε, for each output node was calculated for a "mini batch" of 128 training spectra. After measuring error each iteration, $\boldsymbol{\Theta}$ was updated to lower error. The gradient of ε with respect to each parameter, $\frac{\partial\varepsilon}{\partial\theta}$, was averaged over the 128 spectra. By averaging the gradient with mini batches each iteration, we were able to parallelize the training process which speeds up training (Li et al. 2014).

The quantities $\partial\varepsilon/\partial\theta$ were used to update $\boldsymbol{\Theta}$ after each iteration. Each update adjusted $\boldsymbol{\Theta}$ in the gradient's direction scaled by the learning rate, η, and adjusted by the learning momentum, α. If η is too large, training can overshoot a global minimum in error. If η is too small, training can get stuck at a local minimum in error. Quantity α was used to help get out of local minima in error by adding a percent of the previous update.

Regularization techniques were also employed to keep the values of $\boldsymbol{\Theta}$ close to zero. Otherwise, the values could blow up, causing the MLP to overfit on training data. This is analogous to how a polynomial fit with high coefficients gives erroneous results on novel data. L_1 and L_2 regularization constants were used for this purpose.

The update size for each parameter, $\Delta\theta$, was determined by Equation (6). The $\pm$ in front of L_1 and L_2 indicates which ever pulls the MLP trainable parameter closer to zero.

$$\Delta\theta = \alpha * \Delta\theta_{\mathrm{last}} - \eta * \left\langle \frac{\partial\varepsilon}{\partial\theta} \right\rangle \pm L_1 \pm L_2 * \theta. \tag{6}$$

We utilized a few popular techniques that further improved our grid search. The following terminology refers to an epoch which means iterating through the entire training set one time.

Smith (2017) presents a technique to cycle between different values of η, called cyclic learning. First, one finds a small range of optimal learning rates. The grid search then cycles between the minimum and maximum optimal learning rates. Cyclic learning alleviates the need to over-optimize the learning rate, and helps training get over saddle points in the loss function. To cycle through learning rates, we used a linear triangle function with a cycle step size of 2 times the number of iterations in each epoch, as shown by Smith to be robust.

Srivastava et al. (2014) used a pseudo ensemble of neural networks called dropout. During training, input and hidden nodes are initially turned off and only turned on with a given probability each epoch. After training, when using the MLP to make estimations, each node is weighted by the probability that it was turned on. This is a quick and synthetic way of averaging the response of multiple trained networks. It can also be used as a regularization technique, as parameters are not trained as often and are scaled down by their respective probability of being turned on during training.

Srivastava et al. (2014) also showed that using another regularization technique, Max-norm, helps when used in combination with dropout. Max-norm works by normalizing weights to fit into an n-dimensional sphere with a given radius.

MLP's and data-processing algorithms were invoked from a lightweight, stand-alone C++ program that we developed.

4. Training the MLP on Model Spectra

Model spectra from our database were randomly split into 3 sets: 80% into a training set, $\boldsymbol{S}^{\mathrm{train}}$; 10% into the validation set, $\boldsymbol{S}^{\mathrm{validate}}$; and 10% into the testing set, $\boldsymbol{S}^{\mathrm{test}}$.

When training a neural network, it is important to include many diverse data in the training set. This way the network can estimate a wide range of novel data. A learning curve that plots error versus the size of the training set can be used to determine if enough training data was used. At first the network was completely trained on by only one spectrum from $\boldsymbol{S}^{\mathrm{train}}$, then another spectrum was iteratively added to the training data and the network was completely trained again from scratch. Figure 6 is such a learning curve that shows we used enough training spectra in $\boldsymbol{S}^{\mathrm{train}}$, because the error reaches an asymptote when using the entire training set.

We used greedy stopping to determine when to stop training. Greedy stopping simply lets the MLP train for a number of

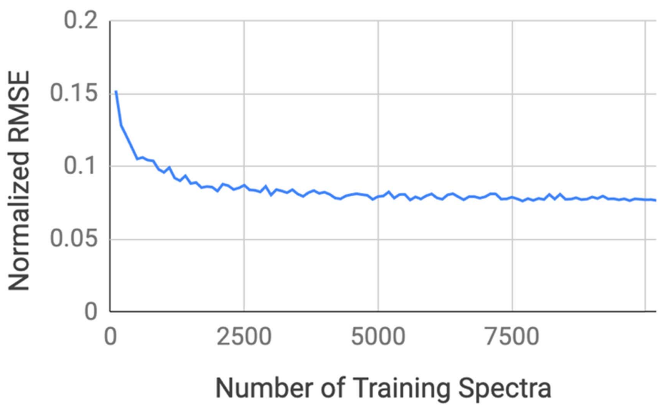

Figure 6. Normalized error as a function of the number of models used in the training set. These results indicate that for this model set about four thousand models are required. We used about 10,000 models in our training set.
(A color version of this figure is available in the online journal.)

epochs and uses the MLP parameters which resulted in the lowest validation error. We trained for up to 1000 epochs, as this gave enough training for the validation error from all MLP's to converge.

We investigated multiple MLP hyperparameters, changing: the number of hidden layers and nodes in each layer, learning rate, learning momentum, and if we used dropout, cyclic learning, $L1$, $L2$, and/or Max-Norm. After investigating different MLP hyperparameters, we found that using three hidden layers with 64 nodes in the first hidden layer, 64 in the second, and 32 in the third yielded the lowest validation error. We used a learning momentum of 0.9, as changing it did not significantly affect results. Learning rate cycled between 0.1 and 0.9, as this was the optimal range. For this study we trained the four MLP's listed in Table 3. Dropout did not improve validation error so it was not used.

Validation results of thousands of different MLP's showed a general trend that noisier models benefit from stronger regularization techniques which oppositely increases error when using less noisy models. This trend is apparent in Table 3 by the large drop in Max-Norm radius when using noisy models. We assume regularization is more effective with noisy models because it more generalizes the MLP hyperparameters, which allows for higher uncertainty in albedo.

5. Results

The training set was used to train the MLP, and the validation set was used to determine both optimal hyperparameters and when to stop training. The results shown in Figure 7 are from testing the trained models with models in the testing set, which have been left out of the process until now. The coefficient of determination, R^2, is shown along with plots illustrating the spread of actual estimations. Quantity R^2 is the proportion of variance between residuals to variance of

Table 3
List of Trained MLP's

Noise	# Epochs	log($L1$)	log($L2$)	Max-Norm radius
None	873	−6	−4	6.1
5%	951	−6	−4	1.4
10%	839	−6	−4	1.4
20%	859	−6	−4	1.4

expected values for a given output parameter. It is defined so that the closer R^2 is to 1, the more accurate the estimations are.

$$R^2 = 1 - \frac{\sum(l_{n,k} - y_{n,k})^2}{\sum(l_{n,k} - \overline{l_k})^2}. \qquad (7)$$

6. Discussion

As might be expected, the best performance was for the noise-free cases. As shown in these sub-panels of Figure 7, the R^2 values for all four parameters varied between 0.933 and 0.995. Metallicity and gravity showed the best behavior, followed by $f_{\rm sed}$ and $T_{\rm eff}$. The addition of noise had a significant effect on accuracy. Each increase in the percent of noise added to the models corresponded to a noticeable drop in the R^2 values, dropping as low as 0.722 for $f_{\rm sed}$ when adding 20% noise.

The importance of clouds and the effect of model degeneracy among the cloudless cases is also apparent in Figure 7. The plots of expected verses estimated $T_{\rm eff}$ show that the estimated values deviate further from expected at temperatures above 230 K, where degeneracies begin to appear where the clouds dissipate. Although the error is larger, the estimated temperatures are all higher than 230 K, demonstrating that the MLP can accurately recognize the cloudless region (above about 230 K). Likewise, the $f_{\rm sed}$ estimations for models with expected values of 11 (cloudless in our nomenclature) are well clustered, and there are very few cases with expected values between 1 and 10 that are estimated to have a $f_{\rm sed}$ value of 11. This further shows that the MLP robustly recognizes if clouds are present.

When clouds are present the MLP generally does very well estimating other model parameters. This is likely because cloudy models show consistent variations with model parameters. Above this cloud threshold the model spectra change more slowly with temperature and the MLP struggles to assign parameters. The R^2 values displayed in Figure 7 corroborate this showing better overall accuracy in the cloudy, non-degenerate, region. However, metallicity and gravity estimations do somewhat better in the warmer cases when clouds are not present. This is likely because for our model set the cloud opacity dominates over the spectral changes arising from the other parameters once clouds form.

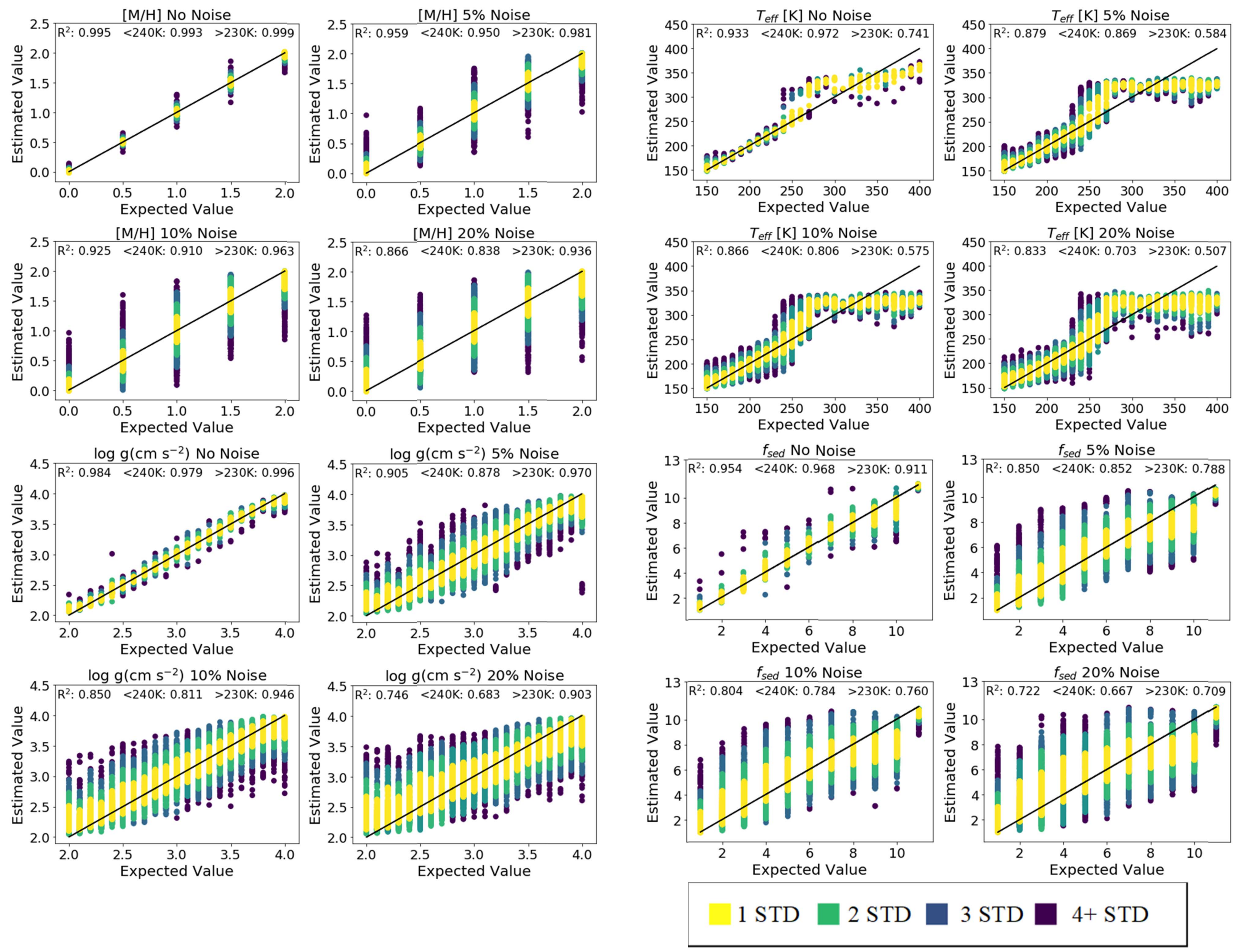

Figure 7. Test results from using the binned model spectra, and varying the level of Gaussian noise added to models. Each dot in the scatter plot represents one test model. The plots are heat mapped by relative model density for the given estimated value. Hotter color means more models at that point, within fewer Standard Deviations (STD) from the average estimated value for the given expected value. The straight line is a slope of 1, showing theoretical perfect estimations. Three R^2 values are shown in each plot when considering: full range of models, models with $T_{\rm eff}$ less than 240 K (non-degenerate with $f_{\rm sed}$), and models with $T_{\rm eff}$ greater than 230 K (degenerate with $f_{\rm sed}$).

(A color version of this figure is available in the online journal.)

These sensitivities can be further seen in Figure 8. The top panel illustrates a case where the MLP estimation is close to the input model parameters of a given test model. Since the MLP can find solutions between individual model grid points we show several of the closest neighboring training models. The similarity of the spectra demonstrate that the estimation is comfortably close to the actual input spectrum. However the bottom panel illustrates a particularly bad outlier case where the MLP finds plausible values for all the parameters except the cloud thickness. In this case the input model had $f_{\rm sed} = 1$ but the MLP settled on 3.3, producing much thinner clouds and a lower albedo continuum level. In this case the solution seems to have been driven by the depth of the strongest CH_4 molecular band at 0.89 μm. Perhaps this $T_{\rm eff}$, lying close to the boundary edge, made the selection more difficult. This dichotomy demonstrates that all of the parameter domain must be carefully tested before any MLP is deployed.

7. Conclusion

We found that an MLP is an elegant supervised machine learning approach for rapidly interpreting albedo spectra arising from simulated observations, and noted the effect of using models in areas known to be degenerate (in this study the models began to be degenerate with $f_{\rm sed}$ at temperatures above 230 K). Particularly, the MLP is robust in estimating [M/H] and log g(cm s^{-2}) and recognizing cloudless cases, especially when no noise is added to the models. Noise mostly affects estimations in log g(cm s^{-2}) and $f_{\rm sed}$. The MLP struggled the most when estimating $T_{\rm eff}$ of models degenerate in $f_{\rm sed}$, starting

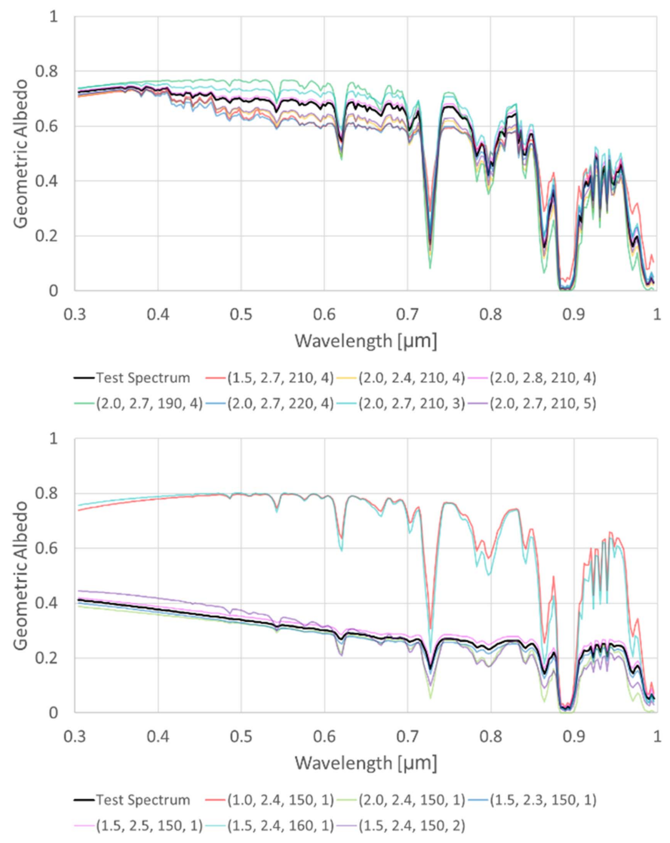

Figure 8. Plots comparing a test model to the training models with parameters closest to it. There was no noise added to the models. The top panel shows a model ([M/H] = 2.0, log g(cm s^{-2}) = 2.7, $T_{\rm eff}$ = 210, $f_{\rm sed}$ = 4) which was accurately estimated (1.99, 2.69, 209, 4.1). The bottom panel shows a model (1.5, 2.4, 150, 1) which was poorly estimated (1.9, 3.0, 151, 3.3).
(A color version of this figure is available in the online journal.)

at temperatures above 230 K and struggling further at temperatures above 260 K where clouds are negligible.

A tool such as that explored here could be deployed online in concert with observation simulation tools to provide an interpretation of a given simulated spectrum. The MLP thus can provide the user with insight, within less than a second for the cases explored here, into whether a given signal-to-noise ratio is "good enough" to meet the scientific goals desired, for example identifying the correct atmospheric metallicity.

New MLP's can be trained to satisfy different spectra dimensions and desired output parameters within minutes to hours depending on the size of the MLP, training set, and desired accuracy. We infer that future work will combine the machine learning methods circulating in the community, to create an artificial intelligence robust in estimating many different output parameters of planetary spectra at various resolutions, wavelength ranges, and levels of noise; along with combining algorithms to output both a continuous value with uncertainty and posterior distribution for each parameter.

Support for this research was provided by the Sellers Exoplanet Environment Collaboration. This research was ran in conjunction with a project which used similar techniques to classify rocks and minerals with Raman spectroscopy (Ishikawa & Gulick 2013; Johnsen & Gulick 2019; Johnsen & Gulick 2020).

References

Ackerman, A. S., & Marley, M. S. 2001, ApJ, 556, 872
Angerhausen, D., DeLarme, E., & Morse, J. A. 2015, PASP, 127, 1113
Batalha, N. E., Smith, A. J. R. W., Lewis, N. K., et al. 2018, AJ, 156, 158
Brandt, T. D., & David, S. S. 2014, PNAS, 111, 13278
Cahoy, K. L., Marley, M. S., & Fortney, J. J. 2010, ApJ, 724, 189
Clevert, D., Unterthiner, T., Hochreiter, S., et al. 2015, Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) CoRR, arXiv:1511.07289
Cobb, A. D., Himes, M. D., Soboczenski, F., et al. 2019, AJ, 158, 33
Feng, Y. K., Robinson, T. D., & Fortney, J. J. 2018, AJ, 155, 200
Heng, K., & Demory, B.-O. 2013, ApJ, 777, 100
Ishikawa, S. T., & Gulick, V. C. 2013, Comput. Graph., 54, 259
Johnsen, T., & Gulick, V. C. 2019, in 2019 Fall Meeting, AGU (San Francisco, 9–13 December 2019) (Washington, DC: AGU), https://agu.confex.com/agu/fm19/meetingapp.cgi/Paper/624340
Johnsen, T. K., & Gulick, V. C. 2020, CG, submitted
Kreidberg, L. 2018, in Handbook of Exoplanets, ed. H. Deeg & J. Belmonte (Berlin: Springer)
Li, Mu, Zhang, T., Chen, Y., Smola, A. J., et al. 2014, in Proc. 20th ACM. SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, ACM (New York: Association for Computing Machinery), 661
Lupu, R. E., Marley, M. S., Lewis, N., et al. 2016, AJ, 152, 217
MacDonald, R. J., Marley, M. S., Fortney, J. J., & Lewis, N. K. 2018, ApJ, 858, 69
Marley, M. S., & McKay, C. P. 1999, Icarus, 138, 268
Marley, M. S., & Tyler, D. R. 2015, ARA&A, 53, 279
Marley, M. S., Gelino, C., Stephens, D., Lunine, J. I., & Freedman, R. 1999, ApJ, 513, 879
Márquez-Neila, P., Fisher, C., Sznitman, R., & Heng, K. 2018, NatAs, 2, 719
Mennesson, B., Debes, J., Douglas, E., et al. 2018, Proc. SPIE, 10698, 106982I
Nayak, M., Lupu, R., Marley, M. S., et al. 2017, PASP, 129, 034401
Nielsen, E., De Rosa, R. J., Macintosh, B., et al. 2019, AJ, 158, 13
Rosenblatt, F. 1958, Psychological Review, 65, 386
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986, Natur, 323, 533
Smith, L. N. 2017, Cyclical Learning Rates for Training Neural Networks, in IEEE Winter Conf. Applications of Computer Vision (WACV) 2017, 2015 (IEEE: Santa Ros, CA), 464
Soboczenski, F., Himes, M. D., O'Beirne, M. D., et al. 2018, arXiv:1811.03390
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., et al. 2014, The Journal of Machine Learning Research, 15, 1929
Sudarsky, D., Burrows, A., & Pinto, P. 2000, ApJ, 538, 885
Waldmann, I. P. 2016, ApJ, 820, 107
Zingales, T., & Waldmann, I. P. 2018, AJ, 156, 268