Neural Networks in MATLAB
Numerical Methods Project Report

VICTORIA CATLETT,[1,2] AMANDA EHNIS,[1] AND EVAN MEADE[1,2]

[1]*Department of Physics, The University of Texas at Dallas, Richardson, TX, 75080, USA*
[2]*Department of Mathematics, The University of Texas at Dallas, Richardson, TX, 75080, USA*

1. ABSTRACT

We explored machine learning in the Matrix Laboratory (MATLAB), specifically with regard to neural networks. We successfully built our own shallow classification neural network, which achieved up to 92.59% accuracy, comparable to the performance of MATLAB's built-in networks applied to the same data. We also explored modifying MATLAB's convolutional neural networks in the Deep Learning Toolbox™, which achieved 86.57% accuracy when trained on only a few thousand images.

2. INTRODUCTION

Machine learning is a term which encompasses a wide variety of computational techniques to find patterns and trends in data. While there are many types of machine learning, we chose to focus on a specific subset, neural networks.

2.1. *Machine Learning*

Machine learning allows one implement select statistical algorithms to automate data analysis. The computer "learns" what to do over time by finding patterns and trends in data. There are countless possible applications of machine learning, which typically fall into one of the following categories: supervised learning, unsupervised learning, and reinforcement learning (4).

Supervised learning begins with labeled data, that is, data for which desired output values are known. The computer searches for patterns in the labeled data in a process called "training", and once it it "trained," it can apply its findings to decide likely categories for unlabeled data (8). An simple type of supervised learning is a linear regression on x and $y = f(x)$ data points. A computer can easily find some m and b which minimizs the error between the points and the line $y = mx + b$. Then, it can make predictions on y values for new x values. For a more complicated example, consider a set of images which contain cats or dogs. A human can likely identify each image as either containing a cat or a dog. One can present these images with their labels of "cat" or "dog" to the computer so that it can establish a way to tell the two species apart. Then, it will make an educated guess on the species of animal in new images.

Unsupervised learning begins with unlabeled data, that is, desired output values are not known (4). Consider again an example with images of animals, only this time, a human does not go through them beforehand provide the computer with a list of which animal is in which image. Instead, the computer must find common features in the images, then group the images by those features. With sophisticated models, these image groups may be recognizable as different animal species (8).

Reinforcement learning is essentially a combination of supervised and unsupervised learning. In reinforcement learning, the computer trains on labeled data, but the labels are hidden from the program. Thus, the process begins like unsupervised learning, which lets the algorithms freely explore. However, the process is guided by the labels, in which the algorithm is positively or negatively reinforced depending on how well its findings match the known outputs.

2.2. *Neural Networks*

Neural networks are a subset of machine learning which break down large problems into smaller, manageable components and organizes them in a manner which mimics the processes in the human brain.

A neuron in the human brain takes in many inputs, evaluates the inputs, and ultimately decides whether or not to release an electrical pulse. The outputs of neurons act as the inputs to others, creating a vast web which ultimately results in our human experience of thinking and living (3). Similarly, a neural network consists of layers which take in inputs and become the inputs of another layer until there is a final output.

The human brain is adaptive, otherwise one could never learn or remember anything. While the exact processes are not known, neurons can at least adjust their behavior after repeated, similar inputs. Similarly, when neural networks train on labeled data, they can adjust how they weight certain inputs until the final output most closely matches what it should.

To visualize how a neural network works, consider the comparison between a neuron and a simple neural network in Figure 1. Both have inputs, evaluate the inputs with internal mechanisms, then ultimately produce a discrete output. The given neural network has 3 layers: a layer of inputs, a layer of calculations (called a "hidden" layer), and an output layer. The second layer is called "hidden" because a user of a neural network typically only interacts with the inputs and the outputs.

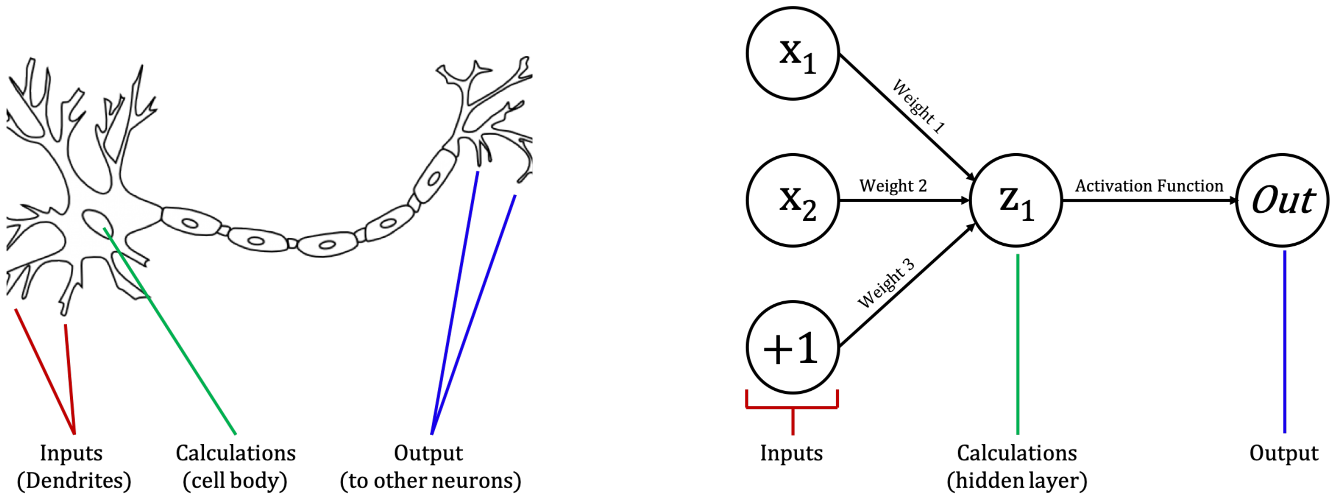

Figure 1. A neuron compared with a simple neural network. Both have inputs, calculations on those inputs, and outputs based off of those calculations. Neuron image from (3).

An activation function transforms continuous data into discrete outputs. Much like how a neuron decides to fire or not, an activation function evaluates inputs and returns a value in a predictable range. The discrete outputs are assigned to numbers in this range, so the closest number to the value of the activation function determines the output. Three common types of nonlinear activation functions are the sigmoid function, the hyperbolic tangent, and the rectified linear unit (10).

The sigmoid function, which we use in our example in Section 4.1.1, is given by

$$S(x) = \frac{1}{1+e^x}$$

The hyperbolic tangent function is simply $tanh(x)$. A simple rectified linear unit function can be defined for some $c \in \mathbb{R}$ as

$$RLU(x) = \begin{cases} 0, & \text{if } x < c \\ x - c, & \text{if } x \geq c \end{cases}$$

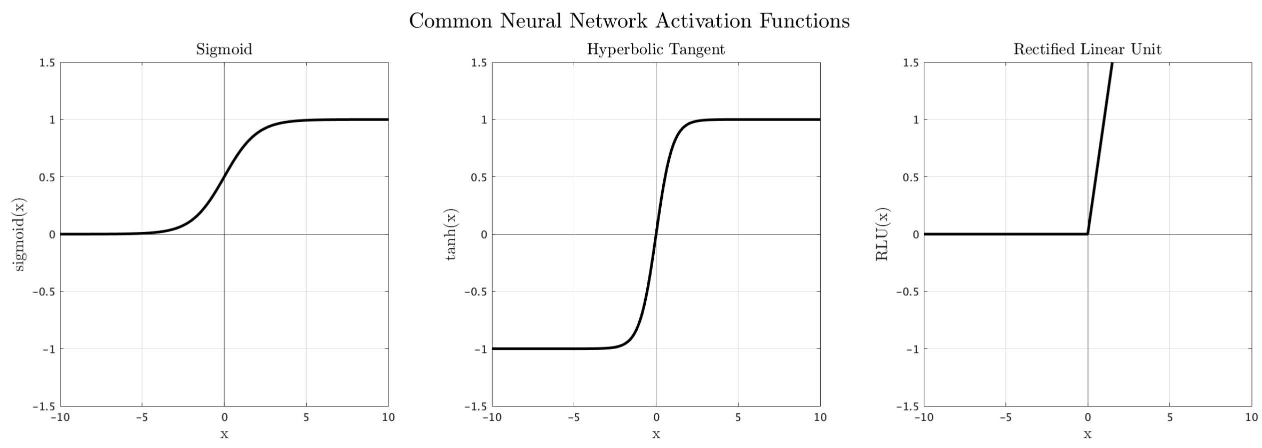

Figure 2. Common activation functions for a neural network. The RLU in this example used $c = 0$. For the sigmoid function, $\lim_{x\to\infty} S(x) = 1$ and $\lim_{x\to-\infty} S(x) = 0$. For the hyperbolic tangent function, $\lim_{x\to\infty} tanh(x) = 1$ and $\lim_{x\to-\infty} tanh(x) = -1$. For the rectified linear unit function, $\lim_{x\to\infty} RLU(x) = \infty$ and $\lim_{x\to-\infty} S(x) = 0$.

2.3. *Deep Learning*

Deep learning involves using neural networks with many layers so that the network trains itself with little to no human intervention. Deep learning is typically unstructured, meaning that the network doesn't have any preset features it is trying to identify in the image.

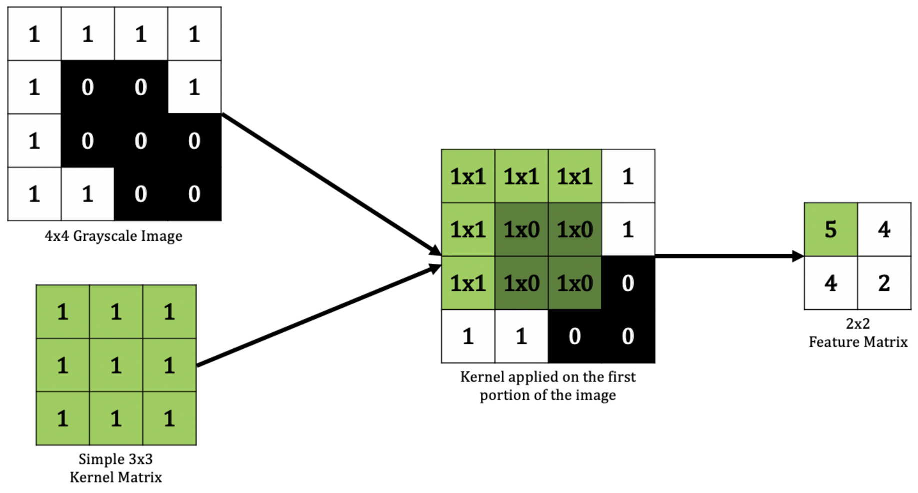

Figure 3. A kernel matrix operating on a 4x4 image.

A specific type of network used in deep learning is a convolutional neural network (CNN), which is typically applied to image classification. An image is simply a large matrix of pixels, either $m \times n$ (greyscale images) or $m \times n \times 3$ (color images) in size. A CNN extracts key features from an image by running a smaller "kernel" matrix over each two-dimensional matrix to produce a smaller "feature" matrix. As shown in Figure 3, the kernel matrix multiplies element-by-element with a region of the same size, then produces the sum of those elements as an entry for the feature matrix. It continues until it has evaluated over every possible region in the image (7). The relative strengths of each feature in an image determines how the network classifies an image. In a CNN, every output of a layer is connected to every neuron in the following layer in a situation called "full connectedness."

3. MOTIVATIONS

The amount of existing data is growing exponentially. According to the International Data Corporation (IDC), the amount of data in the world in 2025 will be approximately 5 times the amount that was present in 2018, growing from 33 zettabytes ($3.3 * 10^{13}$ gigabytes) to 175 zettabytes ($1.75 * 10^{14}$ gigabytes) (11). However, without sufficient analysis to place this data in context, it is merely information, not knowledge. Nate Silver, a professional statistician and editor-in-chief of the political forecasting blog *FiveThirtyEight* describes this dilemma perfectly:

> *"Meanwhile, if the quantity of information is increasing by 2.5 quintillion bytes per day, the amount of useful information almost certainly isn't. Most of it is just noise, and the noise is increasing faster than the signal. There are so many hypotheses to test, so many data sets to mine–but a relatively constant amount of objective truth." (13).*

Nowhere is this need for data interpretation in addition to mere measurement more apparent than in the basic sciences. Without the application of human judgement based on physical intuition, blind application of computational techniques can lead researchers to misleading or false conclusions. It is therefore crucial for scientists at any point in their career to understand the principles behind their tools, and in particular, their limitations.

Due to the expansion of both digital information and digital processing power, these analytical tools increasingly resemble complex and incomprehensible black boxes. Some of the most exciting modern tools are machine learning methods, which can help expose subtle relationships in large datasets. However, such methods are also prone to overfitting, a logical flaw which can suggest relationships that do not actually exist (5). That said, responsible use of such techniques is already producing real scientific results.

For instance, in the field of particle physics, some scientists use convolutional neural networks in the aftermath of particle collisions to reconstruct what happened. These reconstructions determine the types of interactions and particle energies at each point in time (14). The results of such calculations aid in the understanding of neutrino flavors and their energies, breaking down analytical barriers and furthering fundamental work in particle physics.

Neural networks present great promise and possibility, but they must be used responsibly. This project aims to increase understanding of their methods and limitations by demonstrating them on relatively basic, yet large, data sets.

4. DEMONSTRATION

We explored two types of neural networks in MATLAB. First, we built our own *classification* neural network to find boundaries in a set of data, and we compared its accuracy to that of built-in networks in MATLAB. Then, we modified a *convolutional* neural network from MATLAB's Deep Learning Toolbox™ to classify a set of images.

4.1. *Classification Neural Networks*

A data set is classifiable if its data points can be split into a finite number of groups with similar characteristics. A classification neural network is a form of supervised learning which uses mathematical processes on data with established classes to find boundaries for each group. It then uses those boundaries to determine the likeliest groups new data points fall into.

4.1.1. *Our From-Scratch Classification Network*

The Hipparcos, Yale, and Gliese (HYG) catalog from (9) contains 115,015 stars from the Hipparcos Catalog, the Yale Bright Star Catalog, and the Gliese Catalog of Nearby Stars. We reduced the data set to only contain the stars' ID numbers, their absolute magnitudes (relative to Vega), B-V color index, and spectral type. A star's absolute magnitude relative to Vega is a measure of its intrinsic brightness, where the magnitude of the star Vega is assigned to be 0, and dimmer stars have *more positive* absolute magnitudes. The B-V color index is the magnitude of the star through the V ("visible," near yellow and green wavelengths) filter subtracted from the magnitude in the B ("blue") filter. The spectral type is analogous to the temperature (9). The HYG data is shown in Figure 4.

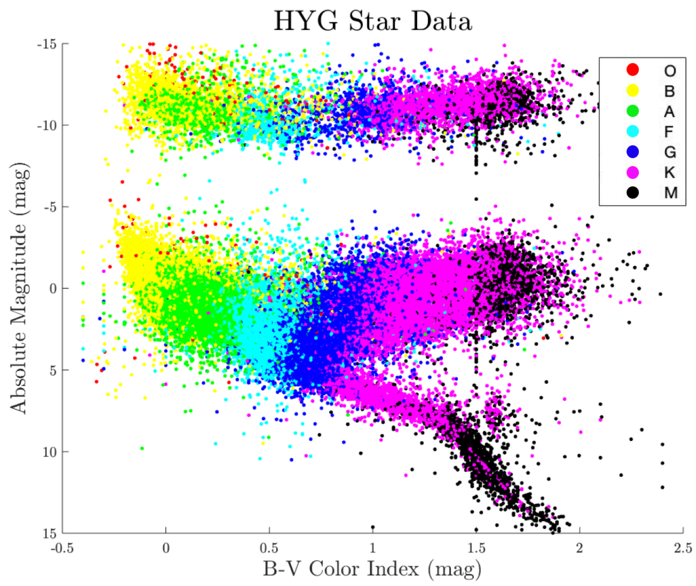

Figure 4. In each plot, the B-V color indices of stars are plotted against their absolute magnitudes. The colors indicate their spectral types (O, B, A, F, G, K, or M), which match the colors of the boundaries found during training. Note that the y-axis is inverted so that brighter stars are higher on the plot.

We built a classification neural network which has the capability to fit boundaries of different degrees. For data in the form of x_1 and x_2 and a boundary degree of n, we create the vector x which contains all combinations of x_1 and x_2 such that their powers sum to *n*:

$$\begin{bmatrix} 1 & x_1 & x_2 & ... & x_1^n & ... & x_1 x_2^{n-1} & x_2^n \end{bmatrix}$$

The length of this vector is $\binom{n+2}{2}$; if $n = 2$, x will have 6 elements (1, x_1, x_2, x_1^2, $x_1 x_2$, and x_2^2). The goal is to find a vector α such that the cost function is minimized for $S(x * \alpha)$. This creates a boundary curve where, ideally, points inside the boundary belong to the class and points outside do not. The cost function for a classification problem with m training examples, as described in (10), is:

$$cost(\alpha) = \frac{1}{m} \sum_{i=1}^{m} -y_i \log(S(x_i * \alpha)) - (1 - y_i) \log(1 - S(x_i * \alpha))$$

In this function, y_i is a Boolean (0 or 1) representing if the data point x_i belongs in the class (1) or not (0). To understand this function better, consider a training example x_t that belongs in the class in question (that is, $y_t = 1$). If α is a good fit, then $S(x_t * \alpha) \approx 1$. The cost for this example would be:

$$cost(\alpha) = -1 \log(1) - (1 - 1) \log(1 - 1) = 0$$

However, if α is a very poor fit, then $S(x_t * \alpha) \approx 0$. The cost for this α becomes:

$$cost(\alpha) = -1 \log(0) - (1 - 1) \log(1 - 0) = \infty$$

Minimizing the cost function will find an α such that $cost(\alpha)$ is relatively close to zero, so $S(x_i * \alpha) \approx y_i$ for as many i^{th} training examples as possible. That is, most values inside the boundary will belong to the class, and most values outside of the boundary will not belong to the class.

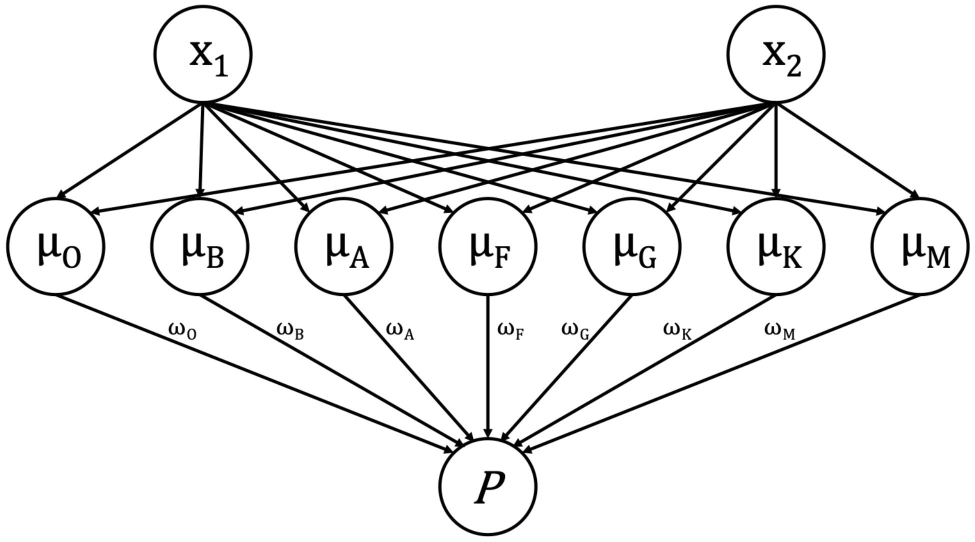

Figure 5. A diagram of our shallow neural network, where μ_n is the probability that a point with values x_1 (B-V color index) and x_2 (absolute magnitude) is of class n. ω_n is the weight for each class, calculated from the relative error of the fit for that class. *P* is the final prediction based off of the found weighted probabilities.

Notice that the cost function only works if there are two possibilities: x_i is either in the class or it is not. However, our example has 7 classes! To deal with this, we divide the problem into 7 sections: we consider each class separately and find an α for each class. Once such boundaries are found with the training data, we can begin to classify new data points based on their position relative to each boundary. For each input point, our network finds the probability that the point is in each class. Then, it multiplies each probability by a weight for that class, which was determined by the error of that class's α. Finally, the network makes a prediction of which class the point belongs to by choosing the class with the highest weighted probability. The basic structure of this network is shown in Figure 5.

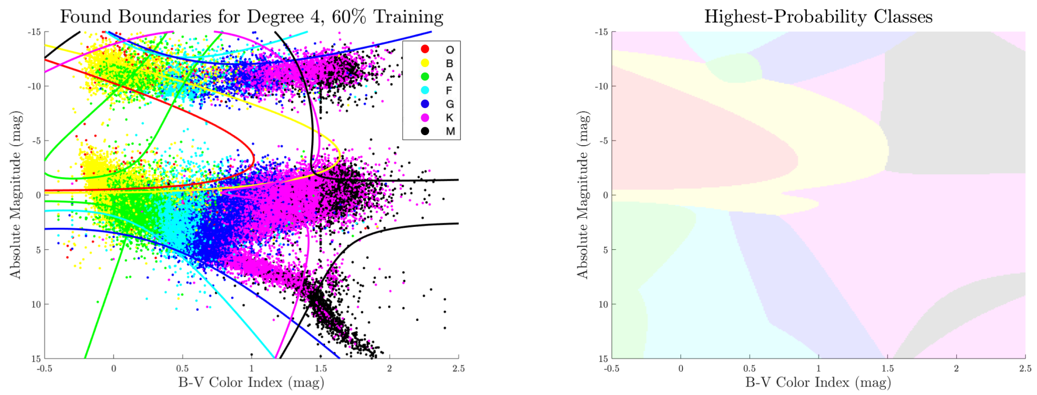

Figure 6. Found boundaries for fourth-degree fits. The plot on the left shows the boundaries themselves, which are clearly very messy. The plot on the right shows how a point anywhere on the graph would be classified. To calculate these regions, we created a 301x301 grid of rectangles on the domain $[-0.5, 2.5] \times [-15, 15]$, used the network to classify the lower-left point of each rectangle, then colored each rectangle accordingly. The colors correspond to the spectral type for the boundaries, shaded regions, and data points, as indicated in the legend.

Figure 9 in Section 4.1.3 includes plots of the best results, which came from a second-degree fit trained on 60% of the data. Figure 6, however, shows one of the *worst* fits we found, which was a fourth-degree fit trained on 60% of the data. The plot shows the individual boundaries found for each class and which regions of the plot would correspond to each class as a result of those boundaries. This is a clear example of "overfitting": the found boundaries fit the training data as well as possible, but the resulting network classified fewer than half of new points correctly. Overfitting arises when α has too many degrees of freedom (in this case, 15), so the boundary fits the training data *too well*. The resulting boundary may not correspond to any real trends, so it may not classify new inputs well.

4.1.2. *NNStart: Built-In Neural Networks*

MATLAB's Deep Learning Toolbox™ comes with many tools for rapidly creating and training custom neural networks. One of the simplest ways to do this is to use the *NNStart* function, which helps automate creation of the neural network using a graphical user interface (GUI). To start out, there are options for what kind of neural network is needed (curve fitting, classification, clustering, and time series networks). Because the output of this network should describe a discrete categorical variable (spectral type), classification was the appropriate option for the problem. MathWorks includes robust default parameters for these networks, including an input layer, a number of hidden layers, and an output layer.

The GUI for *NNStart* is limited in advanced functionality; instead, it generates a modifiable script. We adjusted the script to find average the performance of the *NNStart* networks with different parameters on classifying the HYG data. These parameters included the number of nodes in the hidden layer and the percentage of the data set used for training.

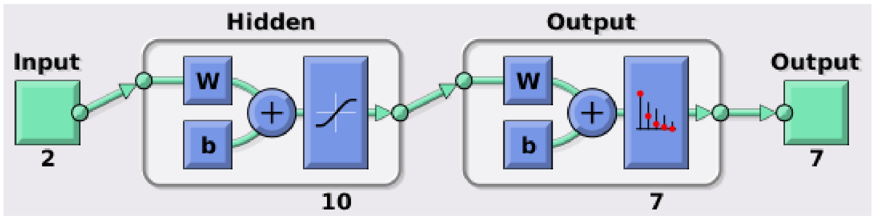

Figure 7. Output image from *NNStart* in MATLAB R2019b showing the structure of a neural network which meets the user's specifications. This particular example has an input layer of 2 nodes, a single hidden layer of 10 nodes, and an output layer of 7 nodes.

The resulting graphs in Figure 8 demonstrate three key observations. First, larger hidden layers decreased the error and misclassification rate on average, but also demonstrated declining returns to an asymptotic limit. Second, lower training percentages actually resulted in better performance on average. This counter-intuitive effect is likely because the data set is very large, yet noisy, so higher training percentages overfit to the noise instead of true physical patterns, thus decreasing real-world accuracy. Third, the fact that these curves are not monotonically decreasing even when averaged over 10 runs demonstrates neural networks' sensitivity to initial conditions. Logically, a larger hidden layer should allow for the capture of smaller patterns, and so should exceed or at least match the performance of a smaller layer. However, either due to the random initial values of the network weights and biases, or the division of the subsets, performance clearly differs significantly from case to case, resulting in rougher, yet still clearly decreasing curves over this particular parameter space.

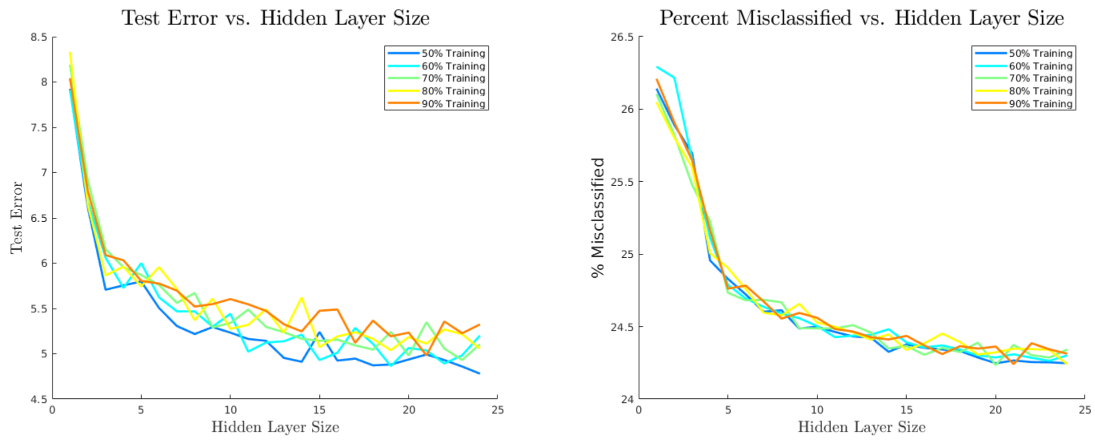

Figure 8. Performance of *NNStart* networks with differing percentages of data used for training and hidden layer sizes, averaged over 10 runs for each parameter combination. Error is measured by taking the norm of the 7-dimensional error vector, found by subtracting the prediction vector from the binary target vector. Each dimension represents the probability that the star belongs to a specific spectral class. The misclassification percentage is simply the average percentage of test cases which received the wrong classification based on the highest-probability prediction.

4.1.3. *Comparison*

Although our network and the *NNStart* networks operate along similar principles of self-training to minimize a cost function, their relative performances vary significantly with input parameters.

Table 1 compares the accuracy of our neural network (up to a fourth-degree boundary) against those of a *NNStart* network with a single hidden layer for various percentages of the total data used for training. All networks were tested with 5% of the data which was not in the training set.

Percentage for Training	NNStart	Degree 1	Degree 2	Degree 3	Degree 4
50%	73.86	48.63	91.30	87.29	53.05
60%	73.71	49.47	92.59	80.27	49.01
70%	73.90	49.69	92.00	81.81	55.19
80%	73.95	49.39	91.40	80.00	56.26
90%	73.79	49.32	91.42	80.29	56.21

Table 1. The percentages of data which were accurately classified by the neural networks. Each network used 50%-90% of the original data to train and tested with 5% of the total data (excluding points used for training) to keep results consistent. The first data column contains the accuracy of the *NNStart* classification network. The next four data columns contain the accuracies of our network for boundary fit degrees of 1, 2, 3, and 4.

Table 1 provides solid insight into how two different network parameters affect overall network accuracy. First, observe that the different percentages of data used for training made little impact on overall accuracy at these values. Each column contains very similar values, indicating that the variable may not have a significant impact. Conversely, it is clear that the default network parameters from *NNStart* perform better than our runs with degree 1 and 4 boundaries, but worse than the network runs with degree 2 and degree 3 boundaries. While the true astrophysical boundary is linear, as shown in Figure 9, the data is too noisy for an extremely constrained fit to capture. Degree 4 is likely overfitting to noise due to having too many degrees of freedom, reducing overall accuracy, as discussed in Section 4.1.1 with Figure 6. However, the degree 2 and degree 3 runs may have performed well because they are similar in degree to the given data set, which has two parameters.

Figure 9 compares the true boundaries (as defined in astronomy) to the best boundaries found by each network. In the first plot of Figure 9, the boundaries are much straighter than they appear to be in the data. A huge source of such error comes from intervening dust between a star and the observer, which preferentially absorbs certain wavelengths in the visible spectrum, changing the color index.

The found boundaries seem to fall roughly where they should be except for Type O. The Type O points are not nicely clustered like the other types; rather, they are dispersed among the other points. Additionally, Type O points only constitute 0.23% of all points in the data set and are relatively dispersed, making it extremely difficult for the training algorithm to determine an accurate boundary. The resulting error of the found boundary is much larger than for the other boundaries, making its probability weight quite low compared to the other classes. In fact, its error was so high that *none* of the grid points for either network had Type O as the highest-probability class, so there is no red on the second or third plots of Figure 9.

It can clearly be seen from Figure 9 that our network has smoother boundaries which appear to more-closely match physical reality than the selected *NNStart* network. This is likely due to the fact that these networks effectively have different degrees of freedom. Our network was of degree 2, while the *NNStart* network had 24 nodes in its hidden layer, which are roughly comparable to the networks' degrees of freedom. However, because the physical reality is so low in dimensionality–it really only depends on the B-V index–having too many degrees of freedom results in overfitting to noise, resulting in the lower accuracy

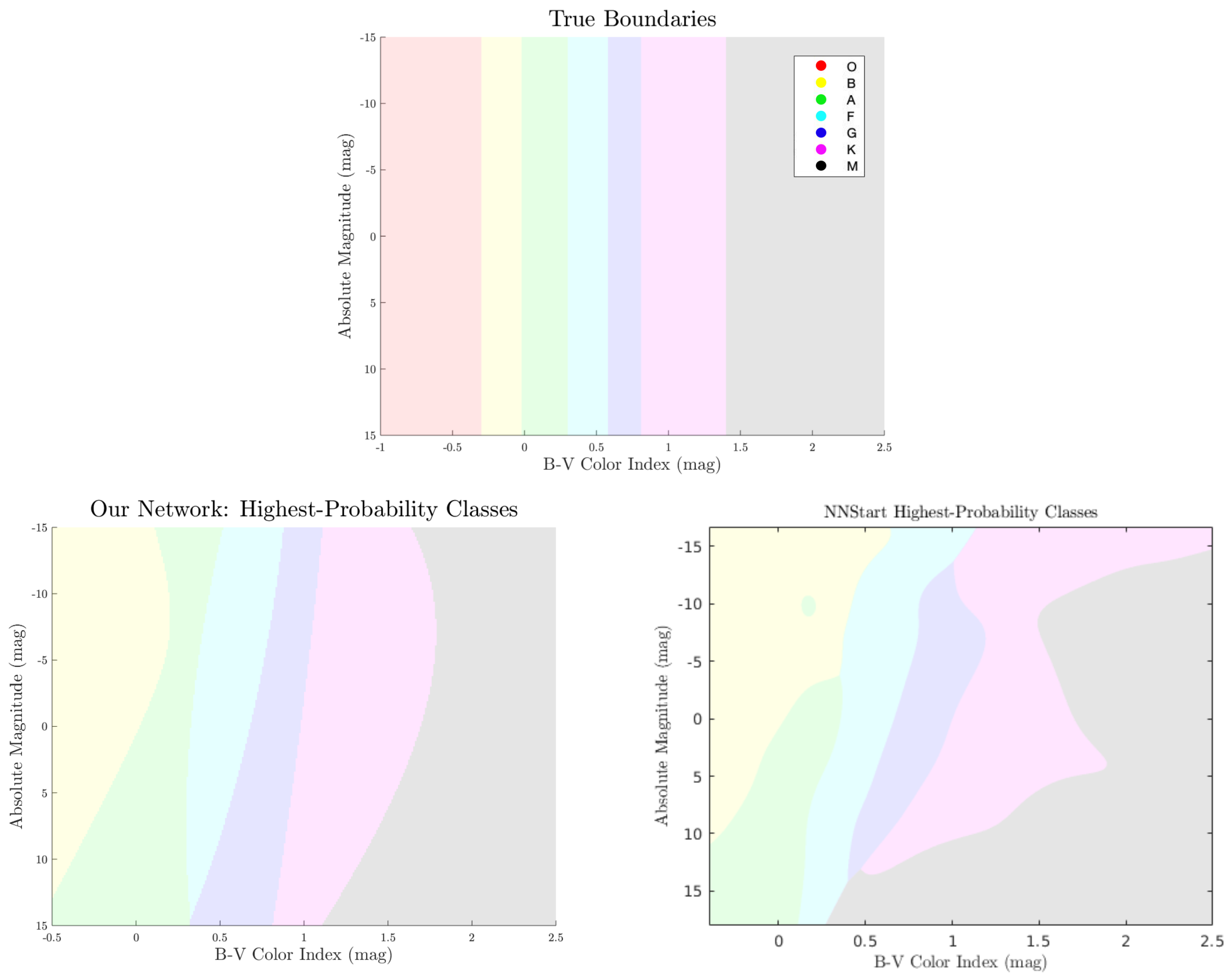

Figure 9. The boundaries found from our neural network (degree 2) and NNStart (24 node hidden layer) compared with the true boundaries (as defined in astronomy). The colors indicate the spectral type, and the colors are slightly transparent so that the plots are not visually overwhelming.

seen in Table 1. This is also reflected in the irregular boundaries of the thirs plot in Figure 9, which deviate widely from physical reality.

Although our network ultimately had the highest accuracy, it took a very long time to construct and is only optimized for this data set. *NNStart* networks, on the other hand, expedite the construction of an initial network, allowing the user to spend more time experimenting with different parameters and data configurations. This startup speed allows for thorough analysis of multiple parameters' impact on overall accuracy.

The advantages of *NNStart* come at a price: a user sacrifices true comprehension of the methods in use. Without intense research, it acts like one of the dangerous "black boxes" discussed in Section 3. Using an *NNStart* network required frequent reference to documentation and outside sources. While documentation is extensive for well-established programs such as MATLAB, it can still be difficult to understand. For instance, even after much research into the "Scaled Conjugate Gradient Backpropogation" algorithm, which is the defualt training algorithm in *NNStart*, we still felt like we were trusting it to do magic. All *NNStart* had to offer on the matter was this snippet inside of the auto-generated code (1):

```
% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.
```

However, our network was far from a “black box.” Even though it took time to build, once it was done, we could tweak at our leisure because we understood it completely. All of the pros and cons we have discussed for these networks are useful considerations when deciding whether or not to build a custom network for a scientific application.

4.2. *Convolutional Neural Networks*

MathWorks, the corporation which produces MATLAB, has many pre-trained convolutional neural networks available on their website to be used with their MATLAB Deep Learning Toolbox™. One of the most state-of-the-art networks is *AlexNet*, named after one of its head creators, Alex Krizhevsky. Krizhevsky and other researchers at the University of Toronto trained the network on 1.2 million images, resulting in it being able to label images as belonging to one of 1,000 different classes (6).

We used data from the Kaggle 2013 Galaxy Classification Challenge (2). This data set consists of 61,578 color images from the Sloan Digital Sky Survey which were identified as containing a galaxy. The data set also contains a table with the names of all of the galaxies with columns for 37 different features, as described in (12). We simplified the table to identify galaxies that are relatively featureless (”smooth”), contain noticeable features (”featured”), or are not galaxies at all (”artifacts”). Sorting the images by those classes resulted in 43.35% (26,693 images) containing smooth galaxies, 56.55% (34,826 images) containing galaxies with prominent features, and 0.10% (59 images) containing artifacts which were initially misidentified as galaxies.

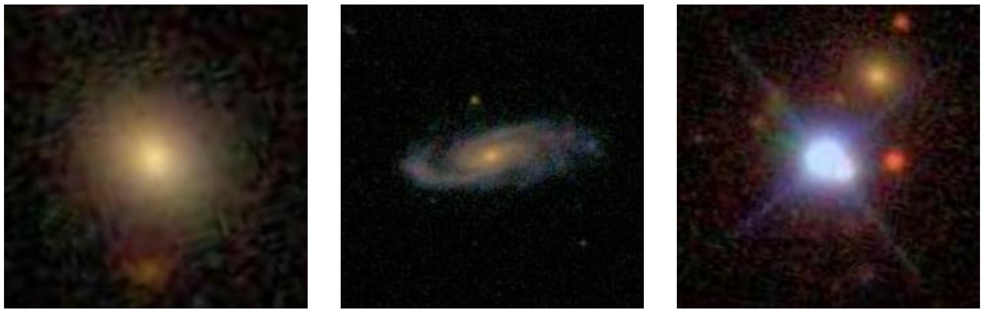

Figure 10. Examples of the three image classes from (2). The image on the left is a smooth galaxy, the center image is a galaxy with features, and the right image is an artifact (the four outgoing streaks indicate it is a nearby point source, such as a star).

We used a modified version of *AlexNet* and the MATLAB Deep Learning Toolbox™ to create a convolutional neural network which can classify galaxy images as smooth, featured, or artifacts. First, we created and ran a script to sort the given images by their classifications into separate folders and resize them to be 227x227x3 pixels. Then, we randomly chose 1000 smooth, 1000 featured, and 50 artifacts to set aside in separate folders for training.

AlexNet has a complicated structure of 25 layers, but we only needed to modify two of them per the guidance of (7). First, we changed the 23^{rd} layer to contain three classes, not 1000, for our three types of images. Then, we changed the final (25^{th}) layer to classify an input into one of the three classes. Now, *AlexNet* could use its build-in feature detection to figure out common features which distinguish the three classes. We trained the modified *AlexNet*, which we lovingly named *GalaxyNet*, on the 2,050 images that we set aside. Training took 43 minutes on a personal laptop even with the assistance of MATLAB's Parallel Computing Toolbox™, nearly causing the computer to overheat.

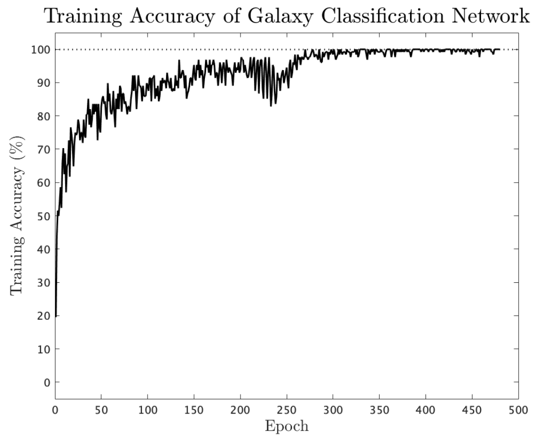

Figure 11. The accuracy of the model as a function of epoch. An epoch is a single iteration of training on a subset of all of the training images.

This network had 86.57% accuracy on 209 test images after being trained on the 2,050 images. Training on a larger set of images would almost certainly increase the accuracy; however, we anticipate that it would be too computationally intense for our computer.

5. CONCLUSIONS

Neural networks are a powerful machine learning tool, which are becoming especially important as the amount of data in the world increases. The neural networks which we explored rely on matrix mathematics, so it makes sense that the Matrix Laboratory (MATLAB) has good tools for them. Although it is possible to build neural networks on your own, MATLAB's powerful, ready-to-go tools can expedite the process to allow more time for analyzing results rather than building something which is already available and effective.

Additionally, critical thought must be applied to the results of any machine learning process, and outputs must not be taken on blind faith. Because neural networks and other machine learning algorithms can become so complex, their particular inner workings are often beyond human comprehension. Rather than reading through them, they can only be analyzed approximately through critical analysis of input and output. It is here that the professional judgement of the scientist is most needed to tap the full potential of these powerful techniques.

All code is available in our technical appendix: https://github.com/Evan-Meade/Stellar-Type-Neural-Networks

REFERENCES

[1]Mark Beale, Martin Hagan, and Howard Demuth. Neural network toolbox: User's guide, 2010.

[2]Galaxy Zoo. The galaxy challenge, 2013.

[3]Virtual Amrita Laboratories. Understanding the passive properties of a simple neuron (remote trigger). 2019.

[4]Karen Hao. What is machine learning?, 2018.

[5]Douglas M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004. PMID: 14741005.

[6]Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[7]MathWorks. Matlab deep learning onramp, 2019.

[8]Eileen McNulty. What's the difference between supervised and unsupervised learning?, 2015.

[9]David Nash. The hyg database, 2006.

[10]Andrew Ng. Stanford university machine learning coursera, 2011.

[11]David Reinsel, John Gantz, and John Rydning. The digitization of the world: From edge to core, 2018.

[12]Honghui Shi. *Galaxy Classification with Deep Convolutional Neural Networks*. PhD thesis, University of Illinois at Urbana-Champaign, 2016.

[13]Nate Silver. *The Signal and the Noise: Why So Many Predictions Fail–but Some Don't*. Penguin Books, 2 edition, 2015.

[14]Neutrino Group SLAC. Algorithms & machine learning.