NORTH CAROLINA A&T STATE UNIVERSITY

TRADITIONAL MACHINE LEARNING ALGORITHMS VS. DEEP LEARNING ALGORITHMS FOR A BINARY CLASSIFICATION PROBLEM

Mehmet Arslan

Department of Mathematics

2022

NORTH CAROLINA A&T STATE UNIVERSITY

Submitted for approval

to the department of Mathematics for Bachelor of Science in Applied Math with Statistics Concentration

at North Carolina A&T State University

by

Mehmet Arslan

Department of Mathematics

2022

Abstract

Throughout my career at A&T I have learned and employed several methods of performing statistics and research through the use of statistical software such as RStudio, MATLAB, Mathematica, LaTeX, TensorFlow, and Python to list a few. I had some background knowledge in these languages, but nothing quite as extensive as what I've been exposed to here at A&T.

In this paper, I seek to determine the potential differences of performance between traditional Machine Learning algorithms and Deep Learning Algorithms. However, we can not use Mean Squared Error as a means of measuring of model performance as MSE is a metric reserved for Regression models. Because this project focuses on Classification, we will utilize Accuracy as our main metric.

When working with Classification problems, we wish to see how well the model that we've built performs in terms classifying the data that is found. This is the sole reason that we will be utilizing Accuracy as our metric for measuring model performance.

Furthermore, I seek to use programming languages such as RStudio and TensorFlow to generate the important results that I am seeking.

Acknowledgments

For this project, there have been several people who have been able to help me achieve my goals and achieve a finer understanding of the concepts of previous courses. At this point, I would like to acknowledge the following instructors for their incredible work in helping me along the way in completing my project. For their efforts, I do sincerely thank my advisor, Dr. Sayed Mostafa, Mr. Ryan P. Taylor, and my mentor on this project Dr. Seongtae "Ty" Kim.

I would also like to thank Dr. Liping Liu for instilling a sense of perfection in all that I have done academically and will continue to utilize their advice now and in my career in the future.

Furthermore, I would like to thank North Carolina A&T State University for allowing me the honor and the privilege of counting myself among truly great people known as Aggies! The university, along with the faculty in the department of Mathematics, has made my job as a student easier and significantly more rewarding in the process. I am proud to have known these people and proud to have made good friends of classmates and fellow Aggies along the way! Aggie Pride!

Contents

List of Figures

Chapter 1

Background Concepts

The motivation of this project was to collect and then compile all the information I've learned throughout my career here at A&T. From the outset, it is important to establish a few basic principles in how exactly programming languages work and their capabilities and/or their limitations. When working through this project, I learned how to utilize a process known as parallel-computing and how this differs from traditional methods such as sequential computing. That is, a given task is broken up into sequences and then passed to the CPU of the computer one sequence at a time. This is perfectly sufficient for myriad of tasks including but not limited to: conducting research, designing and building web-apps, along with designing and building physical objects such as buildings. Sequential computing is even sufficient for basic to moderately complex machine learning algorithms. However, sequential computing has a serious limitation in terms of how many operations can be performed at once. Every modern CPU that is in mainstream computers utilizes a 64-bit architecture. That is when we ask a computer a perform a task such as matrix multiplication, the task is broken up into binary operators, $1s$ & $0s$ and then operated on in pairs. Each pair is called a "bit". Hence, modern CPUs can perform the tasks in pairs of 64 at a time.

Once the instructions have been passed to the CPU from the operating system, the CPU distributes the workload over all available cores. The modern CPU, typically has 4 physical cores and then 8 "logical cores." This is known as hyper-threading. How

this differs in terms of what a GPU is capable of, however, is that a GPU operates on parallel computing. Parallel computing differs in one main aspect: the task we wish to perform, say matrix multiplication, is passed to the GPU all at once at which point the GPU will distribute the workload over all available cores simultaneously. Typically, most GPUs on the market have several hundred to a few thousand cores available to use. From this alone, one can see that GPUs are the preferred hardware options in both research and analytics.

Typically, most GPUs have a few-thousand cores that they can utilize. Note that the differences are highly exaggerated when we wish to perform a *large* in a timely manner. As with any product manufacturers on the market, there of course is a dominant figure enjoying a large market-share. In the case of GPUs, that manufacturer is nVidia. They have been on the cutting edge of technology in their field since their inception in the 1990s. In fact, this company holds the single largest market-share in the GPU marketplace because of their excellent high-quality products. I speculate that this is because their products are unparalleled in terms of performance alone. nVidia has also been a pioneer in the use of GPUs for artificial intelligence in the last few years. In fact, most new cars that feature an autonomous driving are utilizing nVidia solutions.

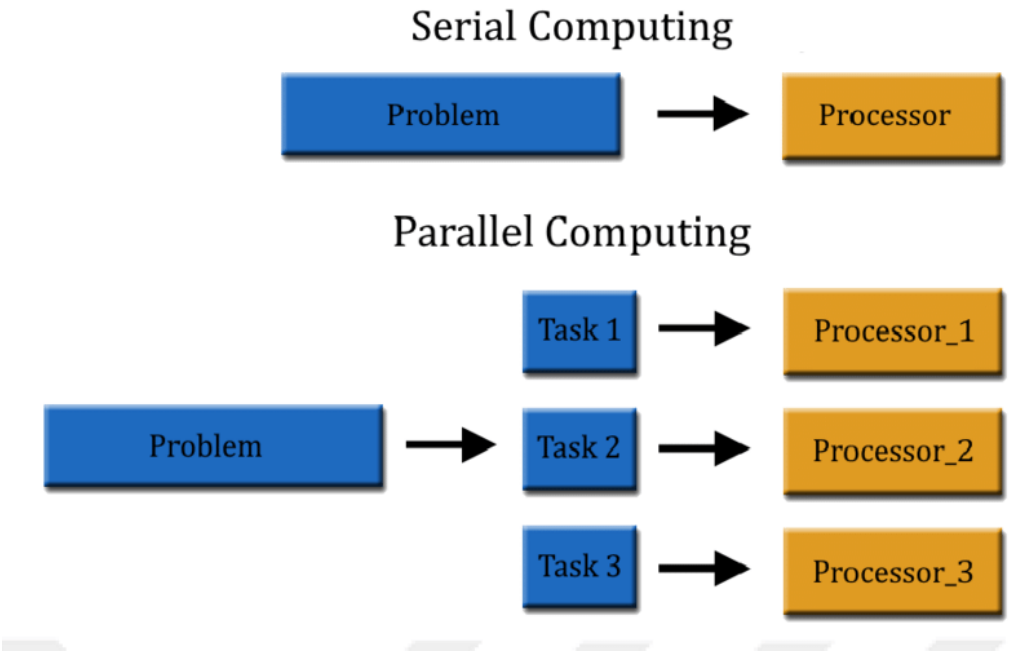

Figure 1.1: Sequential Computing (CPU) vs. Parallel Computing (GPU)

Note that figure 1.1 clearly illustrates how one method is likely to perform better than another one. Also, most research-grade high performance CPUs typically have 12 - 24 total cores whereas a modern GPU has more than 1,000 cores. This can become quite a serious bottleneck when performing long and arduous computations such as matrix multiplication. [1]

Both machine learning and deep learning algorithms simply perform the long-form tedious calculations on our behalf. In this project, I utilized the K-Nearest Neighbors classifier, a vanilla Artificial Neural Network, a Convolutional Neural Network with Dropouts, and then a Bidirectional LSTM (Long Short-Term Memory) type Recurrent Neural Network algorithm. Note that when performing something as simple as linear regression or binary classification, we are technically performing operations on matrices. This is where the sequential computing vs. parallel computing really shines. When performing these tasks on large datasets, it is important to note that the different methods begin to level out and deep learning algorithms tend to outperform them.

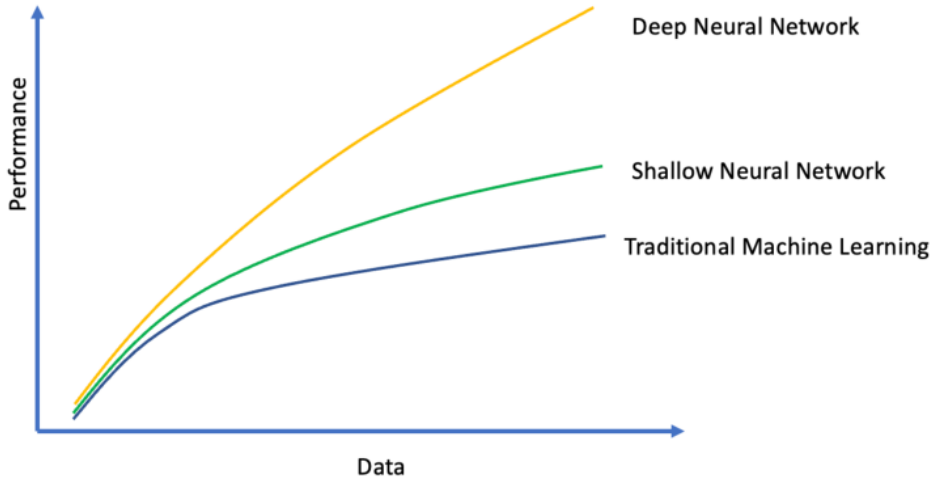

Figure 1.2: Model Performance vs. Data

Figure 1.2 illustrates the challenges of knowing which options to proceed with. But it also does an excellent job of illustrating the performance loss/gain over the size of data

and how much better Deep Neural Networks perform overall. Notice how the increased model performance only occurs when there is sufficiently large data.

Whenever we're performing operations to a dataset in a given language, we're utilizing the concepts of Linear Algebra & Matrices. Think of a large dataset as a matrix, then we can perform Linear Algebra related tasks on this dataset to obtain meaningful results. This of course is the intent of Statistical Inference.

This paper seeks to answer the following question "Is there a fundamental difference in performance between traditional Machine Learning algorithms and Deep Learning algorithms? If so, to what extent?"

We will be seeking the solutions to these questions throughout the remainder of this paper. Before we answer these questions, however, we must explore some background information first.

1.1 Fundamental Differences

The most salient difference Machine Learning and Deep Learning algorithms is how they are implemented. When performing our exploratory data analysis, EDA, we wish to employ methods that are best-suited to the task at hand. This challenge becomes significantly more difficult when dealing with unstructured data or even datasets or larger size.

Traditional Machine Learning algorithms are typically much simpler in their architecture, but the trade-off is how flexible these methods are. Deep Learning algorithms are significantly more complex in their architecture, but they are much more flexible. In this context, flexibility is defined as how simply we may manipulate the architecture of a predictive model, or even the data itself, to generate more meaningful insights. Moreover, when using a Deep Learning algorithm we have significantly more options in terms of data preprocessing, model complexity, and model architecture. Furthermore, we can fine tune nearly every aspect of a Deep Learning algorithm through hyperparameter fine-tuning.

It is through the use of data preprocessing and other clever techniques that we can achieve much higher accuracy, and therefore more meaningful results. These techniques will be covered later in this report.

1.2 Essential Research Questions

In this project, I sought to answer the following question:

- Can a Deep Learning Algorithm Predict the Sentiment of a Movie Review?
- If so, How Well Can They Perform as Compared to Traditional Machine Learning Algorithms?

The following sections in this report will flow in the order of:

- Background Information
- Implementation
- Results

Each section will include a detailed summary of the findings.

Chapter 2

Methodology

2.1 Data Preprocessing

For the problem of Binary Classification, I chose the infamous IMDB Reviews dataset found in the Keras package in TensorFlow. TensorFlow is a library that works with the python programming language and is optimized to utilize NVIDIA's CUDA core technology in their GPU architecture. [2]

Note that the IMDB reviews themselves are in fact just the reviews in their text form. The dataset has labels attached to each of 100,000 reviews as either positive or negative. This is an example of unstructured data. I utilized a data preprocessing technique called one-hot encoding. The overall method is fairly straightforward in its approach and implementation in python. The purpose of doing this is to process the data in such a way that we can actually use them in our predictive models.

2.1.1 One - Hot Encoding

The IMDB Reviews dataset comes somewhat preprocessed for us in that all the reviews are already one-hot encoded when loading them. That is, all the reviews have been transformed into an index value based on the 10,000 most commonly used words in the English Dictionary. Suppose that the review uses words such as: "the", "and", "movie", then the word itself is transformed into a numerical value between $1 - 10,000$.

This was done to ensure that the model, or algorithm, couldn't detect a hierarchy within the sequences and just treated the reviews as numbers. This alone would help to train the model efficiently and accurately.

After the one-hot encoding, I built a training dataset, a testing dataset, and validation datasets for the training data. The training and testing sets were 25,000 observations large and then I randomly sampled 10,000 from the training set to build the validation dataset.

2.1.2 Sequencing/Tensorizing and Padding

After the various datasets were built, I then sequenced the reviews themselves into a sequence of 500 words. That is, each review was now only a sequence of 500 words long regardless of however long they may have been beforehand. I arrived at this value because the average length of the reviews was about 500 words. There were clear outliers with one nearly using 10,000 words but 500 seemed more than sufficient to capture most of the data.

When we perform a one-hot encoding, we are essentially transforming the reviews into a vector of numbers. This presented a unique challenge because we need to convert this vector into a tensor. I achieved this through the use of zero-padding and then indexing. The indexing was indexing the 10,000 most commonly used words in English and the zero-padding just transformed the review into a vector of 0s or 1s that were 10,000 observations large for each word in the review. That is, there were now 25,000 sequences of 10,000 observations in size! The purpose of doing so was to standardize the data in such a way that we can operate on them mathematically.

2.1.3 K-Nearest Neighbors Classifier Implementation

For the purposes of the choosing a traditional machine learning algorithm, I decided that a K-Nearest algorithm was the best option as this algorithm performs binary classifications rather well. The reasoning behind this was rather simple: I assumed that the nature of the data itself would be near impossible to utilize logistic regression on. Recall that when working with logistic regression, or regression of any type for that matter, we wish to build a model out of independent variables. This was a problem because the dataset itself was unstructured in that it did not contain variables to speak of. As such, any regression model would simply be futile.

When I took STAT 328 - Intro to Machine Learning with Dr. Kim, we worked on several classification problems in homework assignments and the projects and I found that the K-Nearest Neighbors algorithm was the most flexible to work with.

Having prepared the data into now padded sequences of equal length, I build a K-Nearest Neighbors classifier in TensorFlow.

Note that the K-Nearest seeks to find the distance between two points in space. Through my coursework, I found that the method of Euclidean to be the most efficient:

$$D(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Where y_i is our true value that we wish to predict and $\hat{y}_i$ is our predicted value. Though it might be difficult to conceptualize in terms of sequenced data, but the following image will make it more clear:

In 2.1, we see how the K-Nearest Neighbors seeks to divide between the classes available by using a straight line. This line is determined by the Euclidean distance, in effect the line of best fit for this case. [3]

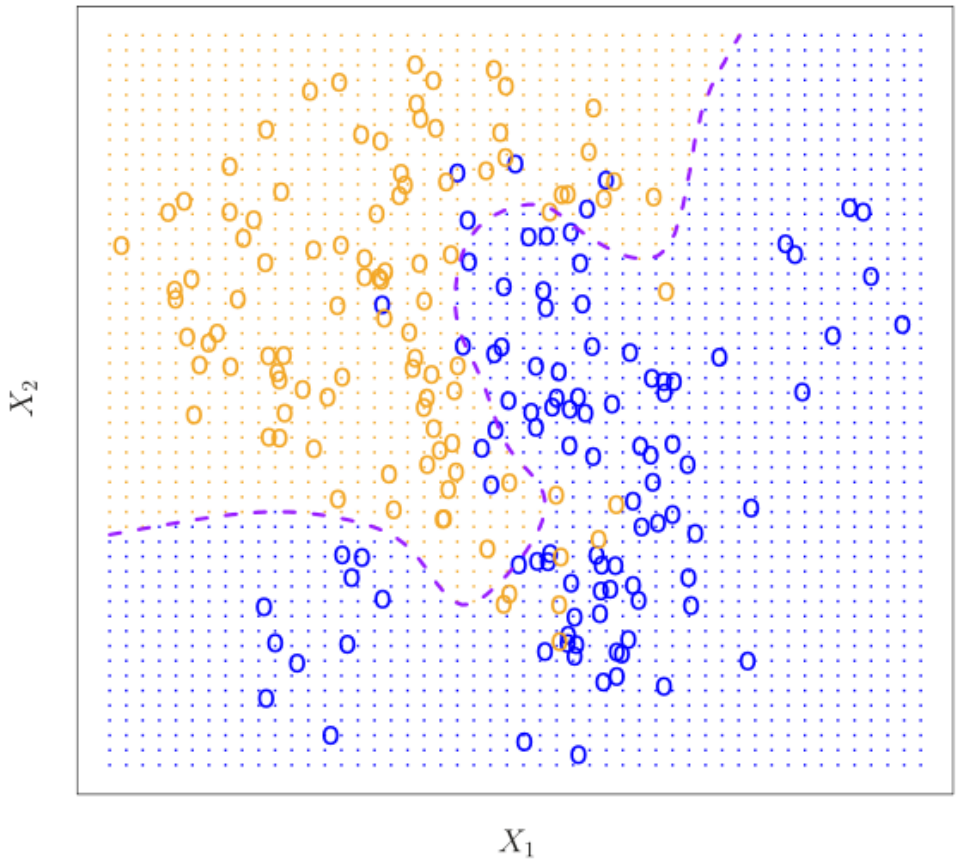

Figure 2.1: K-Nearest Neighbors Classifier

2.1.4 K-Nearest Neighbors Algorithm Results

When building the algorithm, I was astonished to see the trend that this algorithm only managed a peak accuracy of 50.2% This was truly an astonishing result. I expected that this algorithm could perform this task and have much more meaningful results. I speculate that this algorithm was simply overwhelmed when it came to trying to predict the sentiment.

Although this was an interesting result, it wasn't to be expected. I speculate that the model so overwhelmed, in terms how much data there was, that it simply began preferring the training dataset.

2.1.5 Artificial Neural Networks

When working through this paper, and indeed taking STAT 428 - Intro to Deep Learning, I learned how Artificial Neural Networks function. They function on a process known as feed-forward and back-propagation. The feed forward aspect is self explanatory, the net moves forward from the first input layer and then through the hidden layers, and then finally to the output layer. The input layer is our input data, the hidden layers are the layers of neurons that either activate or not and then their combinations determines the outcome or output layer. There is an interesting phenomenon to note when working with Neural Networks, and it stems from the following quote from the Hebbian Theory: "Neurons that fire together, wire together"

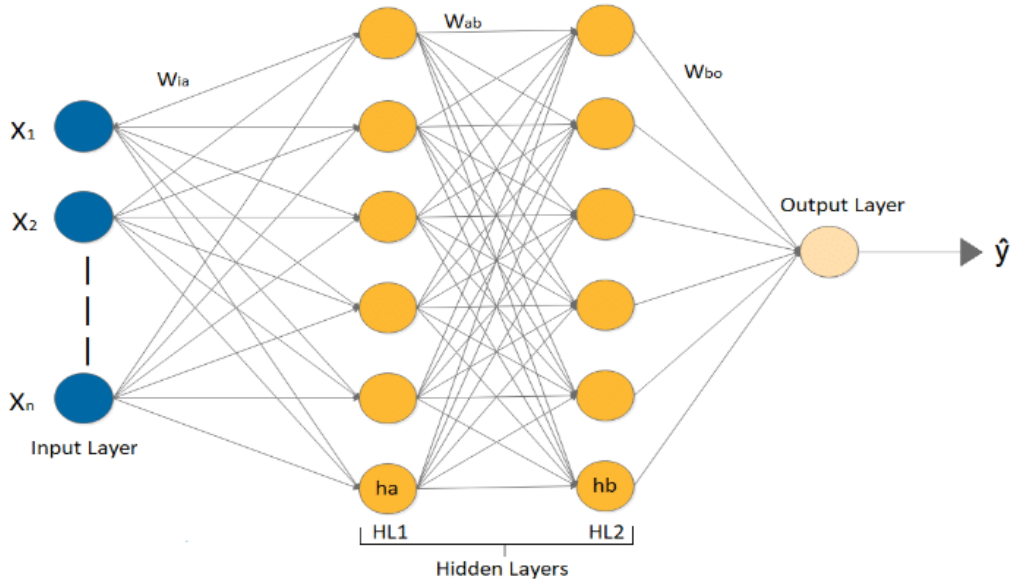

Figure 2.2: Architecture of Artificial Neural Networks

In 2.1 we see how the model progresses forward, from the first layer of input all the way to the final output or prediction. Recall the quote from the Hebbian Theory here. Interestingly, we can systematically wire fire our nodes through the use of weights and biases.

Note that each node requires quite a bit of computational resources:

$$Z = \sum_{i=1}^{n} (W_i \, x_i \; + \; b)$$

The activation matrix can be given as:

$$Z = \begin{bmatrix} w_{11} & w_{12} & \dots & w_1 k_1 \\ w_{21} & w_{22} & \dots & w_1 k_1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_m k_1 \end{bmatrix}^T \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(2)} & \dots & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(2)} & \dots & x_2^{(n)} \\ \vdots & \vdots & \ddots & \vdots & \\ x_m^{(1)} & x_m^{(2)} & x_m^{(2)} & \dots & x_m^{(n)} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_k 1^{[1]} \end{bmatrix}$$

Where W_i is the weighted matrix of an input, x_i is the input itself, Z is our threshold for activating the node or not, and b is the bias at that step. Afterwards, the activation function is activated and then it generates a value. From there, the loss is calculated through the use of back-propagation and an attempt to minimize the loss function(which is at the core of an ANN).

The minimization of the loss function and the cost functions are the core concept of Gradient Decent, which seeks to use the information of a gradient of a differentiable function, and then minimize it. This is one such example of a single node firing in a single instance, and a single iteration of the loss function. This process is repeated by the algorithm all the way until it converges to zero for each node and then, it will propagate to the next layer. These calculations can become extremely burdensome for even the best CPU that money can buy, but this is where the advantage of a GPU really comes into play. All of these calculations can be performed at once, and the entire Neural Network can converge in a matter of minutes!

The Gradient Decent Algorithm first requires a differentiable function. Often we may select this function as the MSE function or the mean-squared error function. The MSE cost given function is given as:

$$MSE = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y} - y_i)^2$$

The derivatives of the cost function are automatically calculated for us in the ANN architecture and thus makes it easier for us to make meaningful insights. Note that these two options together are present in every single node at every single layer of the Dense(fully connected) Neural Network. This is the reason why Deep Learning algorithms outperform traditional machine learning so heavily, because their cost is automatically minimized and so is the loss. This of course yields to a better fitting model more of the time.

2.1.6 Artificial Neural Network Implementation & Results

When implemented, the model performed rather well, but as the plots show, their results were sub-par. When training a Deep Learning model, we wish to train the model in such a way that the model performs well overall and indeed has a robust fit with the validation data. Otherwise, if the model prefers the training data, then it suffers from over fitting. Note that this model does in fact suffer from over fitting.

Through the use of the vanilla ANN, I achieved a peak accuracy of 90.8% Which is excellent, but it could definitely be better. In later parts of this report, I explored more state-of-the-art techniques.

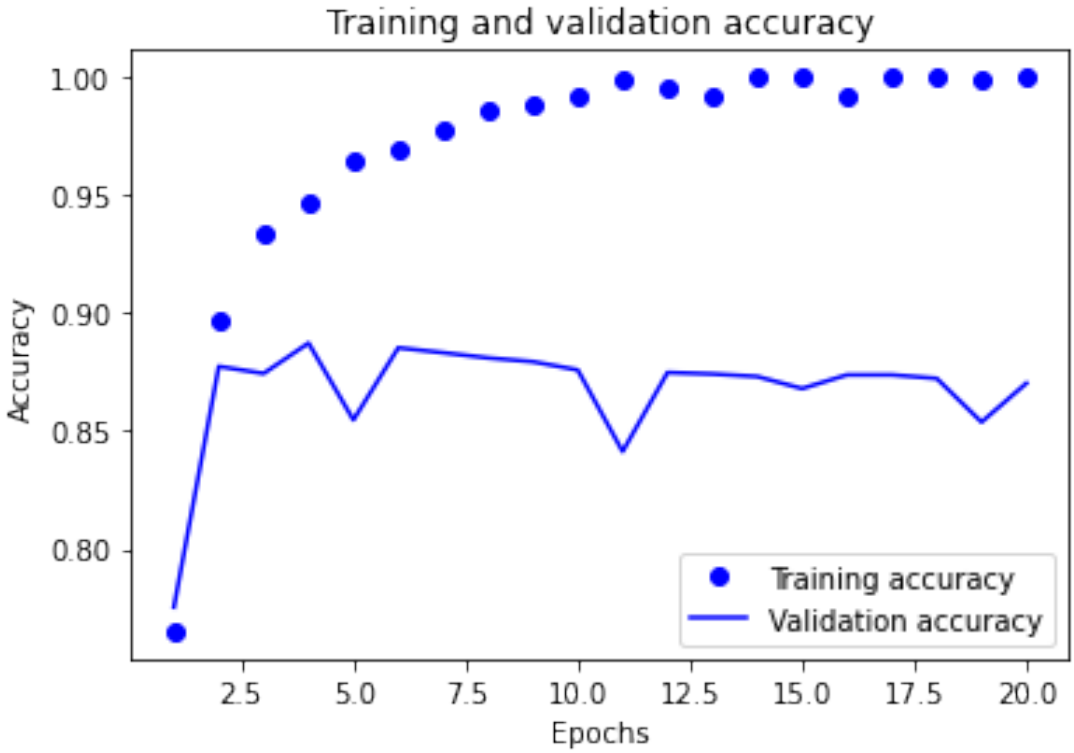

Figure 2.3: Vanilla Artificial Neural Network Performance

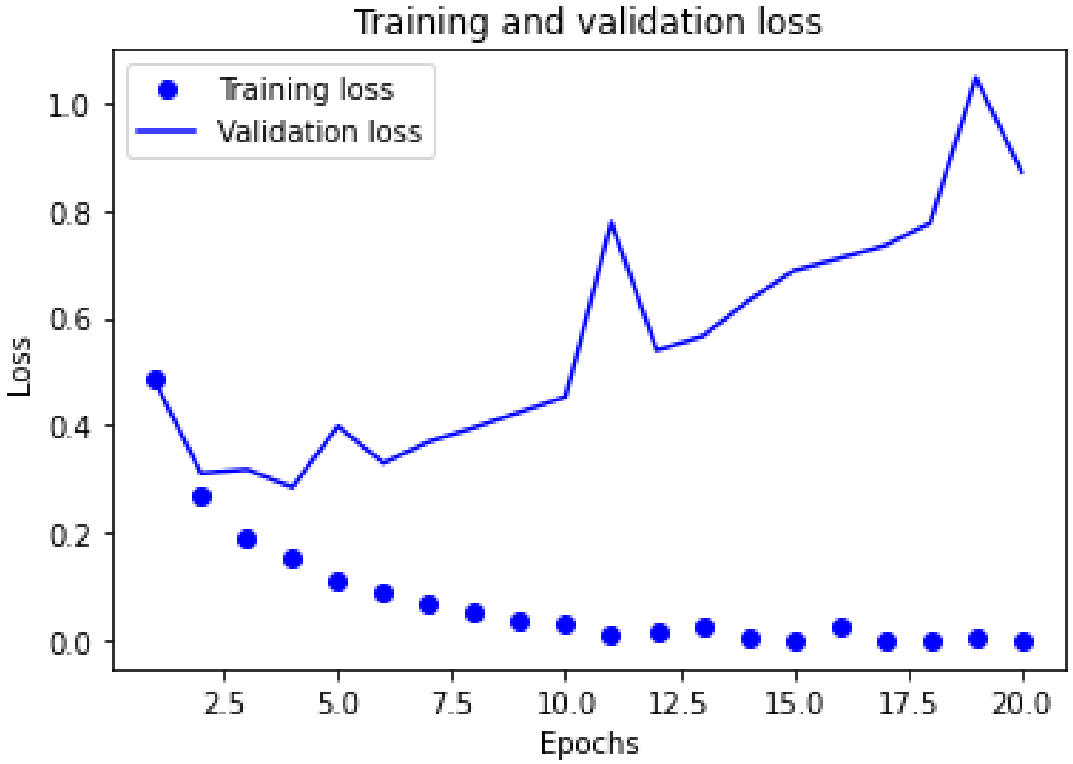

Figure 2.4: Vanilla Artificial Neural Network Performance

2.1.7 Convolutional Neural Networks

Convolutional Neural Networks are similar to vanilla ANNs but they differ in one main aspect: The implementation of convolutions. A convolution is a mathematical operation that produces a third expressing how the shape of one is modified by the other.

Let X and Y be two random discrete variables. Then the distribution of the sum $Z = X + Y$ is given as:

$$P(Z = z) = \sum_{k=-\infty}^{\infty} P(X = k)P(Y = z - k)$$

For the case of a binary classification, out input variables, X are continuous and our Y variables are discrete. The convolution of a continuous variable is given as:

$$h(z) = (f * g)(z) \triangleq \int_{-\infty}^{\infty} f(\tau)g(z - \tau)\, d\tau = \int_{-\infty}^{\infty} f(z-\tau)g(\tau)\, d\tau$$

This of course allows us to extract a feature map from things like an image and then train the model to detect edges, colors, in effect *all* characteristics of an image.

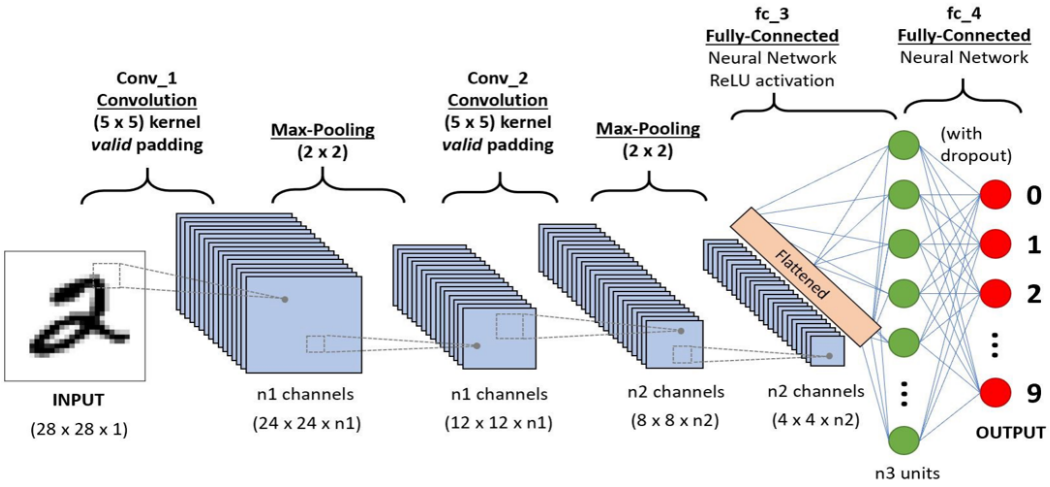

Figure 2.5: Vanilla Artificial Neural Network Performance

In figure 2.10 we see the operation required to perform a convolution on an image. The same operations can be performed on text data also, in fact more simply, instead of a 2×2 filter, or 2-dimensional filter, I used a 1-dimensional filter as this was not an image. [4]

2.1.8 Convolutional Neural Network Results

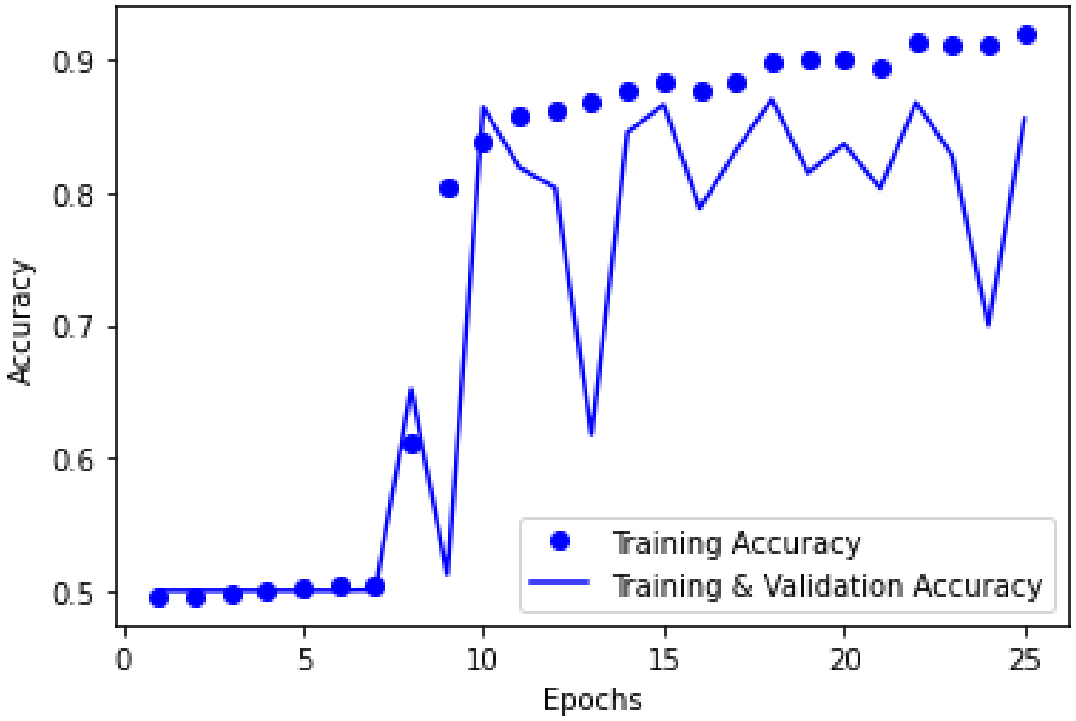

Figure 2.6: Accuracy of CNN w/ Dropout

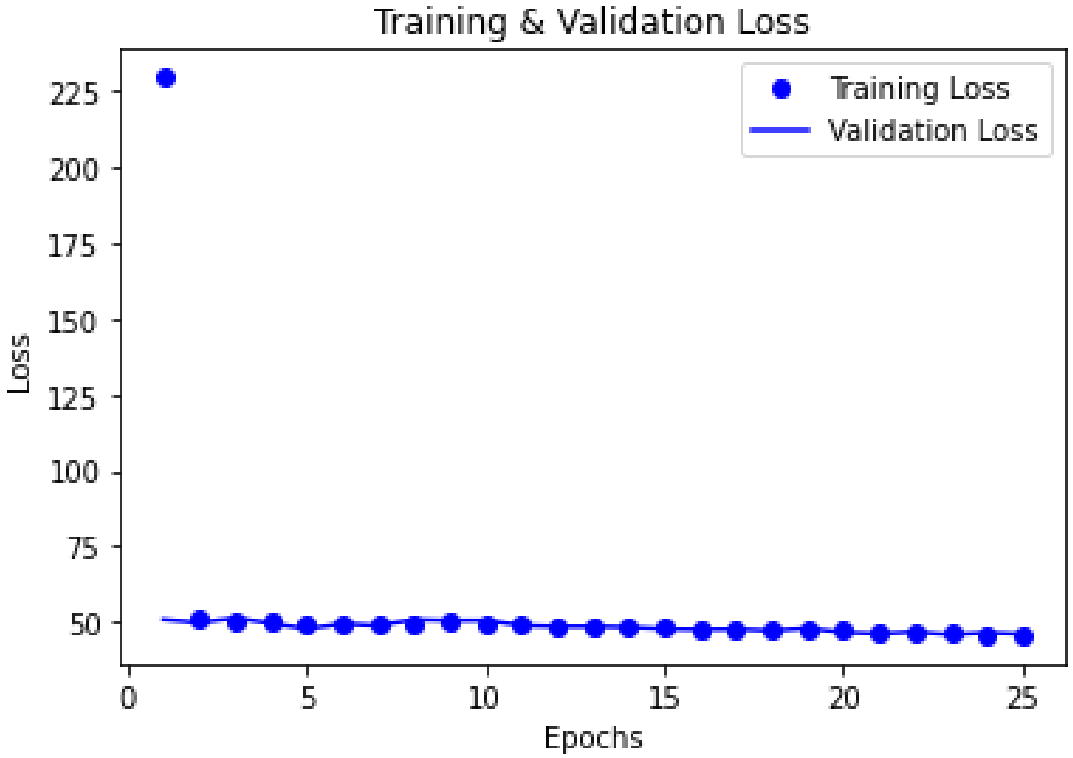

Figure 2.7: Loss of CNN w/ Dropout

With the CNN with a 50% dropout layer, I achieved a peak accuracy of 87.1% with a loss value of 50.14. This was nothing short of sub optimal, but the model did eventually achieve a robust fit. For me, while learning how these operations are performed, achieving the robust fit was the most important.

2.1.9 Bidirectional LSTM with 128 units

Finally, the implementation of a Bidirectional LSTM was the best performing model overall. The Bidirectional LSTM achieved a peak accuracy of 92.4% with a loss value of 0.07 which was outstanding to say the least. Note the robust fit and how few epochs it took to train this model as compared to other methods.

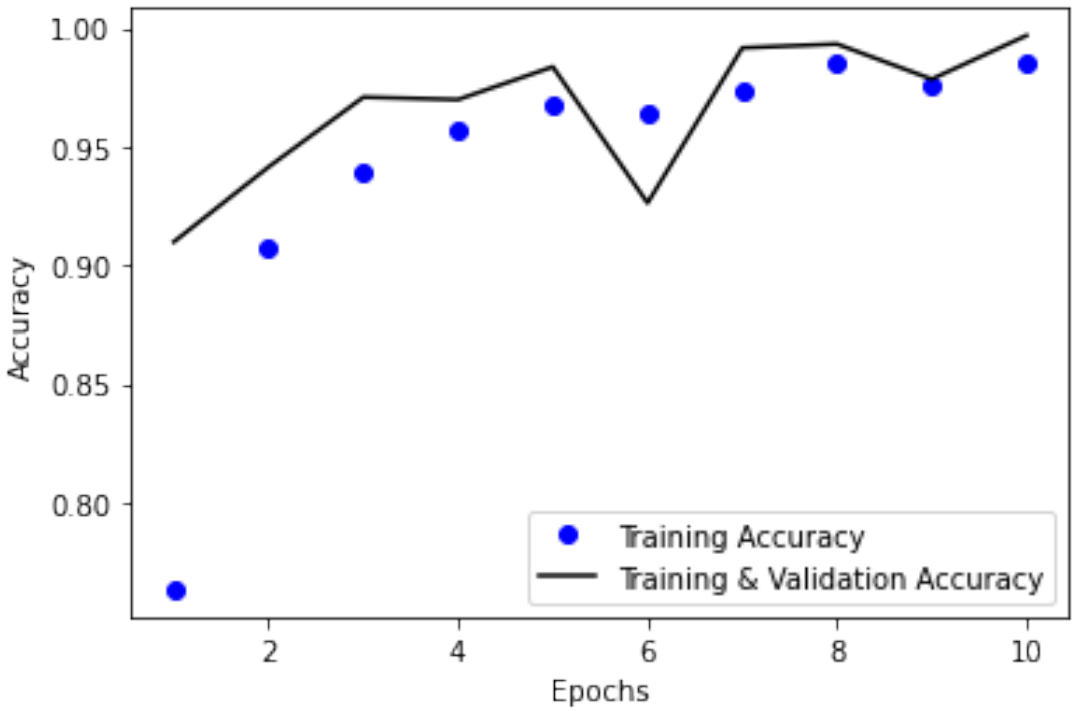

Figure 2.8: Bidirectional LSTM w/ 128 units Accuracy

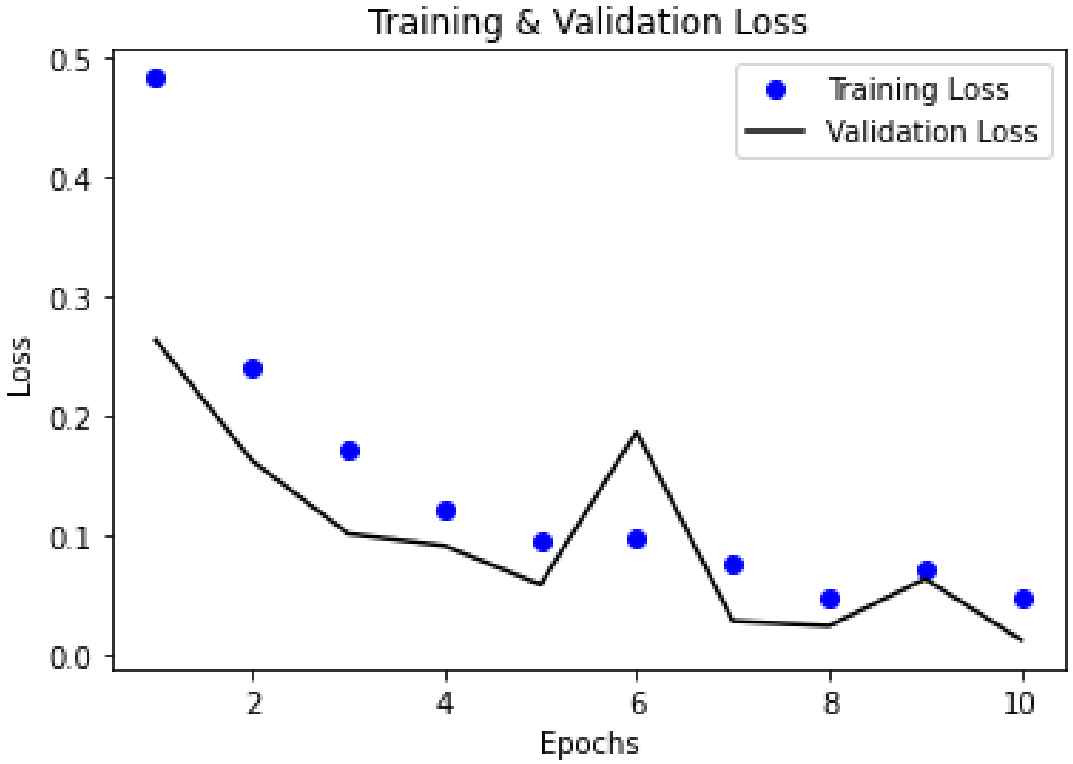

Figure 2.9: Bidirectional LSTM w/ 128 units Loss

2.2 Conclusions & Future Future Study

Throughout this semester, I learned many new and exciting skills and I plan to utilize these skills over my career in terms of implementing these high accuracy learning models. In conclusion:

Model Type	Accuracy
K-Nearest Neighbors	50.2%
Vanilla ANN	90.8%
CNN w/ Dropout	87.1%
RNN/Bidirectional LSTM	92.4%

Figure 2.10: Overall Accuracy Table

I also learned that the Bidirectional LSTM with 128 units was more context aware in that it could correctly guess the sentiment more correctly more of the time.

Bibliography

[1] Batuhan Hangün. "Performance evaluation of a parallel image enhancement technique for dark images on multithreaded CPU and GPU architectures". PhD thesis. Jan. 2022. DOI: 10.13140/RG.2.2.32436.17280.

[2] TensorFlow. *Introduction To Tensors*. https://www.tensorflow.org/guide/tensor. Accessed on 2022-05-07. Apr. 2022.

[3] Gareth James. *Introduction to Statistical Learning with Applications in R*. Springer Publishing, Aug. 2021.

[4] Phani Rhatan. *Convolutional Neural Network Architecture*. Oct. 2020. URL: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/.