The Structure and Function of an LSTM Neural Network

Evan Conley

MTH 490

6/6/18

1 Introduction

1.0 Neural Network Background

Neural networks are an incredibly popular problem-solving tool utilized today. Research into the field of machine learning has shown that they are effective over many different domains. For example, they are being utilized for problems as large as drug discovery[DR] to as trivial as creating and training an off-putting caption generator, like MIT's recently popularized, Norman Bates-themed AI.[NORM] In order to explore this field, it is the intent of this study to engage with the underlying mathematical structure of a neural network, while applying these methods to the selection and development of a specific model. By, first, understanding the background to machine learning and the types of problems, therein, we will develop a model capable of performing sentiment analysis on IMDB movie review data. Specifically, we aim to create a model that can correctly label a review as having 'positive' or 'negative' sentiment. Upon its construction, the model will also be put under different hyperparameter testing conditions, allowing us to review how capable the network is and ways to optimize its ability to perform its task. Further, in our decision to create a model that best fits our problem, there is an underlying question of efficacy: whether we chose the best-suited model architecture. To explore this, the neural network's performance will be examined alongside comparable models built to solve the same problem. To begin this process, though, we must develop an understanding of the requisite knowledge to address this topic.

1.1 Neural Network Background

An artificial neural network is a form of computational model, so called, because it seeks to mimic biological models of the same name. A graphical representation of neural network can

be seen in Figure 1, where the nodes of the graph are called neurons and the connections between the nodes are known as synapses, in line with the biological theme of their construction.[RR] Within Figure 1, note that each column of the graph is described as a different layer of the network, where the initial layer is known as the input layer, the layers comprising the central part of the network are known as hidden layers, and the final layer is known as output layer.

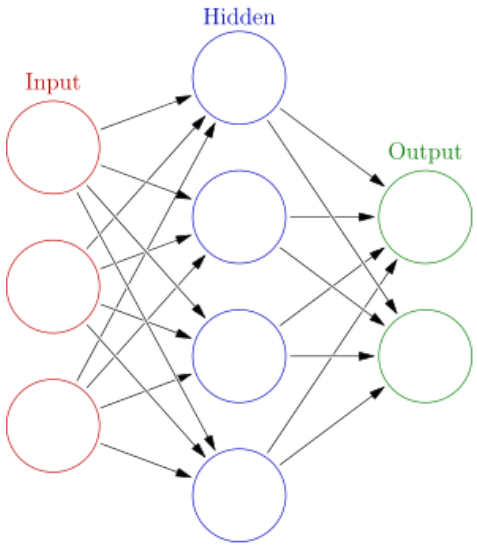

Figure 1: Single-Layer Perceptron

As the figure above only contains a single hidden layer, it is known as a single-layer perceptron. Further, since there exist no loops in the graph, the architecture shown is known as a feed-forward network architecture.

Each of the synaptic connections between layers is assigned a numerical weight, W, representative of the strength of the connection and applied to the input, X (also assigned a number), of each neuron. For the sake of variability of the architecture, when training a network, these weights are typically initialized with a random value between some defined range. Then, a

bias term, *b*, is applied to the output of the operation in the form of a constant term being added. Biases being added as a feature of a network (Figure 2), is done by creating an additional bias neuron at the input layer and any subsequent hidden layers, where the weight associated with a bias is always 1, and it takes no inputs from the layers previous to it.

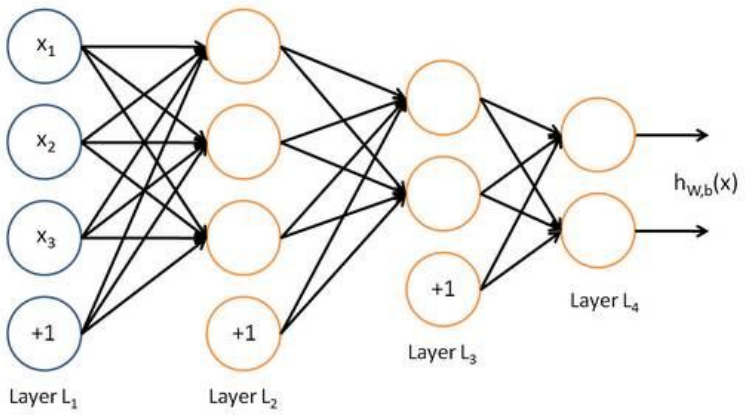

Figure 2: Network with biases

The functional purpose for a bias is that it skews the output, tending it away from a simple linear solution set.[BA] Finally, the output of the neuron has an activation function, σ, applied to it, giving us the general form of a neuron operation:

$$h = \sigma(WX + b). \qquad (1)$$

Here *h* is representative of the output of the neuron. The purpose of activation functions and how they are selected will be discussed later. Additionally, the structure and function of each of the layers will be described at length in later sections.

Essentially, neural networks are functional approximators, utilizing nested functions of the form of the general neuron output equation over the set of layers to reach a desired output. In fact, early work with neural network architectures by Cybenko led to the development of the

Universal Approximation Theorem, stating that, in theory, a single layer-perceptron is capable of approximating a continuous function of an arbitrary amount of variables on a compact space.[UAT] In practice, though, utilizing a basic, single-layer feedforward architecture is impractical due to issues of efficiency. Then, it follows that we should utilize more varied and, potentially, more complex models depending on the complexity of the function we seek to model. These considerations lead us to the process of designing an architecture that fits a specific problem. In order to do this, though, we must first describe the types of problems to which a neural network can be applied.

1.2 Types of Problems

To illustrate the mathematics underlying the structure of a neural network, we will apply a specific design architecture to a problem. Within the overall field of machine learning, there exists two broad categories to which neural networks are applied: supervised and unsupervised learning. Unsupervised learning is the set of problems such that there exists no known solution set; that is, the data is unlabeled and unquantified. The neural network architecture, in this case, is meant to discover similarities in the data and create clusters of similar data points by identifying key features of the dataset.[BAR] We will not be exploring an unsupervised architecture, and will instead focus on supervised learning. Supervised learning considers those problems where a dataset exists such that inputs have known outputs. This allows the model to use a metric to compare against, determining if the model is well-suited to a problem on a basis of its capability to correctly model expected behavior.[MUR] From here, the type of problem can be broken down further on a basis of the type of outputs expected by the approximated function. When the outputs described are discrete labels or categories, we know these problems to be

classification or pattern recognition problems.[MUR] When there are only two discrete categories in which the data can be sorted into, we know this as a binary classification problem. This gives us a means of classifying the issue of assigning our two sentiment labels (‘positive’ or ‘negative’) as being within the scope of binary classification problems.

1.3 Problem Statement

Now that we have an understanding as to the type of problem we will be discussing, we can specify our exact problem. To understand the specific requirements of our problem, though, we must discuss the field of Natural Language Processing (NLP). This is an area of study and research where the capabilities of computers to parse and find semantic meaning in natural language is tested and observed.[NLP] There are many subfields of NLP, including semantic analysis, terminology extraction, and question/answer search problems. Our problem will deal exclusively in the limits of sentiment analysis (popularly known as ‘opinion mining’). Sentiment Analysis is a demonstrative technique of being able to take in a form of input (text, vocal, or otherwise) and output one of a discrete number of classification labels that fit the data.[SA] With an understanding of the previous areas of study and problem types, we are able to fully explain our desired problem that we will be designing the network to solve. This problem is that we seek to take an IMDB movie review and output a binary classification label of either ‘positive’ or ‘negative’; a problem defined in the context of sentiment analysis, discussed above. With the defined problem in place, we are now ready to design and implement a neural network tailored for our problem.

2 Development

2.1 Input Layer

Sentiment analysis introduces a few obstacles into the process of designing our network. The primary difficulty encountered happens at the first layer of our network, the input layer. In particular, a neural network is comprised of composed matrix operations; for this reason, we require matrices that contain numerical input for our network to run. More than this, though, we need the mathematical representation associated with each input to effectively characterize the key features of the data, so that the network can act on and create meaningful interpretations of the data. Normally this would necessitate the use of a text corpus, or body of text, on which to train through a complex unsupervised learning method, ultimately resulting in a relation of the words in the corpus and the characteristics (called the feature map) about each word, represented as quantitative units.

Luckily, there exists a workaround to manually developing a secondary neural network in the form of pre trained word embeddings. Essentially, a word embedding is a model trained in the aforementioned way that is made freely available to help with input to neural networks involving textual data. One such model is the GloVe (Global Vector Word Representation model) designed and created at Stanford University, in 2014.[STAN] It is comparable to Google's word2vec vectorization model as far as efficiency and capability, and has varied feature sizes associated with each word. The feature map-lengths associated with each word are denoted as *n*-dimensional vectors. The options for the length of these feature maps are 50-dimensional, 200-dimensional and 300-dimensional vectors, of which, we are going to utilize the 50-dimensional embeddings. This allows for simple composition and development of the input

layer. From this point, then, the input layer of our network will be comprised of an embedding layer operation where each word in an a given review input will be looked up in the dictionary of word embeddings. If it exists in the dictionary, it will utilize the associated feature map vector as that level of input. In the case that the word does not exist in the dictionary, the matrix will be populated with a set of zeros that will act as placeholders for missing contextual information. In order to increase efficiency of output and create ease of use for the embedding layer, the network will only use the top 5,000 words (based upon word frequency) from the GloVe.

With this implementation, the network is capable of handling information at a low level. Another constraint is that the matrices taken as input to a network must be of a constant size, so as to match the given sizes of the weight matrices and bias matrices encountered throughout the rest of the network layers. As we have chosen a constant feature-map length for each word embedding, we are able to ensure that each input sample will have constant row lengths of 50. The need still exists to ensure that the column dimensions are of some constant size, though. As such, we will need to define a constant length that will be required for each of the input strings. Sources differ over what the correct sequence length should be for handling textual data where context matters. For the sake of being able to have a sequence length that was not excessive, but still capable of gathering enough information about the context of a review, the sequence length of 250 was chosen; this means that only 250 words of a review were considered for each input, after which the input was truncated. If a sequence was shorter than the 250 sequence length requirement, the sequence was padded. This means that a null character (character holding no embedding or meaningfulness in context) was used to extend a short sequence up to 250 words for input. So for each input, we now have a *250x50* matrix corresponding to it, and our input is

kept constant. Additionally, the embeddings ensure the data is able to be processed by further elements of the network.

2.2 Model Selection

Another pertinent issue to consider when developing a model involving textual information is that we are dealing with time series data. Time series data involves an ordered sequence of data where order of the sequence matters to the given output of a function or sequence.[GTS] This means that the output from a previous time step in a sequence can be just as important as the current input to our model. This allows us to develop a sense of context, where the order of language creates semantic importance. A simple example would be in the sentences "I hate that movie" versus "He is a complex character that you love to hate." In both sentences, the word 'hate' is utilized, but it is apparent that, through context, it is meant to imply a heavier negative connotation in the first sentence than in the second. In this manner, the order of the sequence can help to determine the overall intended sentiment inherent in that sequence.

Returning to Figure 1 and Figure 2, both of these architectures, while potentially robust models for other problems, are feedforward networks, as discussed previously. By definition, the network only forward propagates information through the model, meaning that previous inputs are not taken into consideration while evaluating current input. This is a drawback when working with time series data. As such, it is important to introduce another classification of network architectures.

So far, we have only reviewed the feedforward network architecture, but it is important to note that its key feature (a lack of cycles in its graph) is not a rule for the creation of networks. A recurrent neural network (RNN) is one, in which, there exists cycles in its graphical

representation, as seen in Figure 3. Each of the cycles is indicative of the output of a layer being utilized, in part, along with a new input to a previous layer.[RNN]

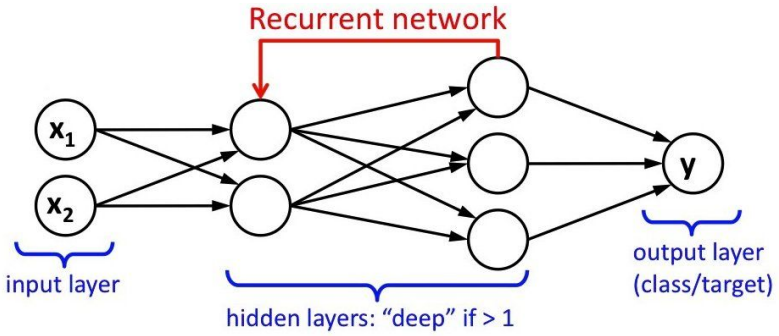

Figure 3: Recurrent Network

This modification to our network types vastly changes its functionality, and allows it the ability to develop a sense of context for time-series data analysis. Due to the inherent complexity of their architecture, RNN's have difficulty training due to the issue of vanishing or exploding gradient. During the process of backpropagation, the error changes too little or too drastically, and this can cause the RNN to miss an optimal solution set.[RNN] The process of backpropagation, and the issues of exploding or vanishing gradient will be described in later sections, but their existence is important to note for the selected model architecture. The Long Short-Term Memory (LSTM) network architecture was developed through the use of what are known as LSTM units. Each unit has several internal gates to keep track of importance of time series data through input and recurrent layers, as well a save state.[LSTM] The development of this architecture became very important to solving the issues with gradient mentioned above. The important thing to note is that the internal gates and ability to save data through its save state, make an LSTM network an optimal choice when handling time series data, because it is able to discern context. In fact,

LSTM's have been utilized in a variety of forms to solve issues of sentiment analysis, from text data to musical analysis.[MUS] For this reason, we will choose this model architecture in the development of our network.

2.3 Activation Functions

Before being able to explore the internal units that make up an LSTM network, we must return to Equation (1) to gain an understanding and appreciation for activation functions, and their capabilities. Considering the terms internal to (1), we see that they are of the form $WX + b$, yielding a linear function. This means that $WX + b$ satisfies the principles of linearity[CALC]:

Homogeneity: $f(ax) = af(x)$ *for some real valued constant* a

Superposition: $f(x+y) = f(x) + f(y)$.

As one might expect, continually nesting layers of this form will, ultimately, result in composition of linear functions, i.e., another linear function. This is limiting, since we would like our model to be able to approximate any continuous function on a compact space. There needs to be a means of introducing nonlinearity into the system. Returning to (1), we see that the linear function is ultimately the input to the final operator, σ, where this is indicative of the activation function.

Activation functions are meant to introduce nonlinearity into systems by breaking the above principles of linearity, allowing neural networks to approximate nonlinear functions, as well as linear ones. Commonly, the symbol for an activation function in a functional representation of a network layer is sigma, as sigmoidal curves are frequently used forms of activation functions. In fact, the internal activation function used on each of the gates in the LSTM units is the sigmoid function, visualized in Figure 4 to demonstrate its nonlinear form, as

well as the hyperbolic tangent function used for the save state and output of the unit. Further, sigmoid forces its output to range between 0 and 1, making each gate somewhat analogous to a neuron's firing rate (i.e. 1 is full fire of information while 0 means no information is let through). Activation functions must be continuously differentiable to allow for their use in the backpropagation algorithm.

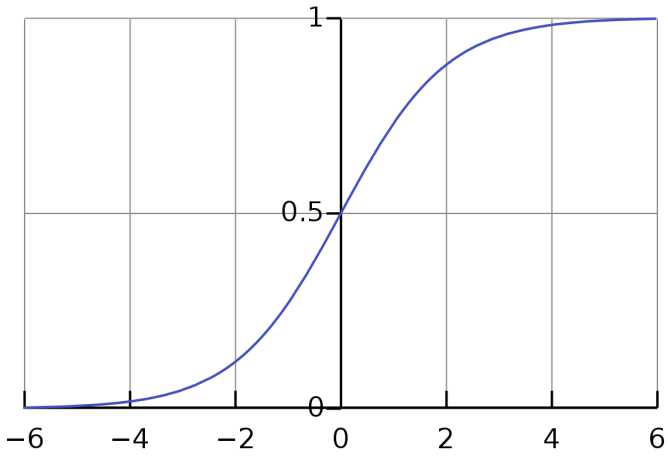

Figure 4: Sigmoid Curve

The functions of the activation functions are as follows:

$$sig(x) = \frac{1}{1+e^{-x}} \qquad (2)$$

$$tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} . \qquad (3)$$

2.4 LSTM Units

Now that we have some background on the activation functions that will be implemented in our network, we can discuss the internal structure of the network. Although the introduction of recurrent layers to the model is important to the development of the network, it is pertinent to note how data is handled internally to each LSTM unit so we can understand how the model is

capable of being so integral to the network's understanding of context. A representation of an LSTM network can be seen in Figure 5.

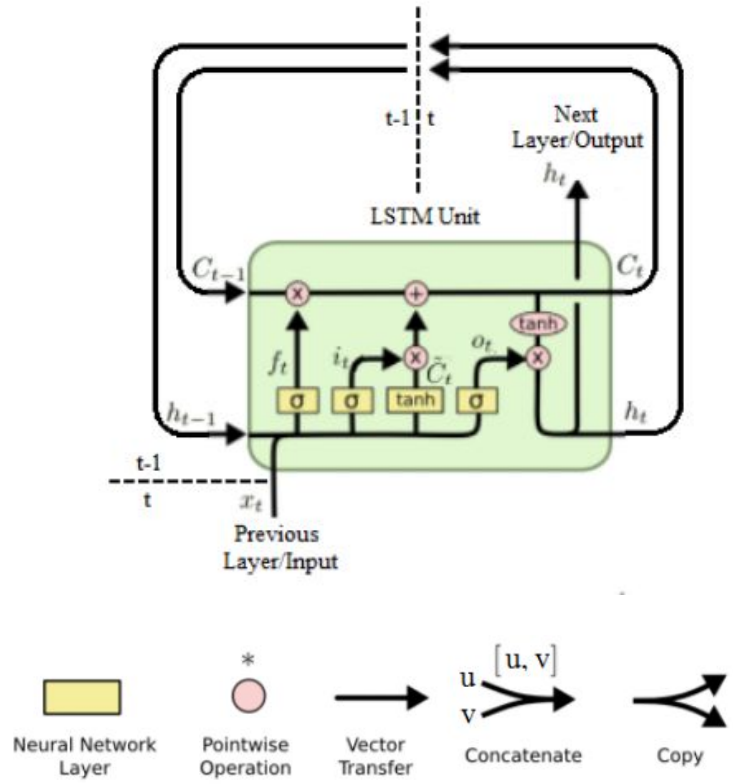

Figure 5: LSTM Unit

To fully explore the internal capability of these units, we can step through the process that information takes through an LSTM unit. Note, each of the inputs is subscripted with a timestep, *t,* that alerts us to their relative order in the sequence. Then, if *h* is representative of the output of a network layer, we have that h_{t-1} would be the output of the layer at the previous timestep (a recurrent input). The first operation performed is to take the current input sequence and the output of the previous network layer and concatenate them, feeding them into the 'forget

gate,' called as such because it determines the importance of the previous input and previous saved cell state. This can be represented as:

$$f_t = \sigma(W[X_t, h_{t-1}] + b), \quad (4)$$

From this point, the concatenated input and previous output are pushed through the 'input gate' which determines the importance of the current input. At this step through the LSTM unit, the cell state also calculates potential (or candidate) values for the save cell state:

$$i_t = \sigma(W[X_t, h_{t-1}] + b) \quad (5)$$

$$\overline{C_t} = tanh(W[X_t, h_{t-1}] + b). \quad (6)$$

It is at this point that the forget and input gate are used as scaling factors for the previous input and new candidate values respectively. Here C_{t-1} represents the previous cell save state, and the new cell state is computing using $C_t = f_t * C_{t-1} + i_t * \overline{C_t}$. Next, using the save state that will be utilized in the next iteration of the network, the output gate is calculated to act as a scaling factor similar to the input and forget gates; in this case, the output gate will determine how much of the current time step output, h_t, should be considered for the next iteration. Also, the output of the current time step is simply an activated version of the calculated current cell state, with respect to this scaling factor:

$$o_t = \sigma(W[X_t, h_{t-1}] + b) \quad (7)$$

$$h_t = o_t * tanh(C_t). \quad (8)$$

From here, the process continues, for each new input, the previous output of the layer is cycled back to act as a contextual argument for the rest of the matrix of the operations.[LSTM] Despite the seeming complexity of these operations, they are essentially a repeating composite of matrix

operations, resulting in a scaled output. This process happens for each of the LSTM units at each hidden layer of the network.

With the update to each of the layers for the duration of the input stream, each iteration causes a contextual update to the scaling of the given input. However, this is not enough to classify the overall output of the network. To achieve this we must pass the information from the hidden layer to the final layer of the network.

2.5 Output Layer

At this point, we have described a means to propagate information through the network and apply some contextual data to the overall output of the network. This is a big feat, but the total output of the network must also handle some constraints. In particular, our outputs are discrete labels, and we are aiming to express the most likely label that fits the current input stream. As such, we need to create a probability distribution such that some degree of probability is spread amongst our classification labels. A commonly chosen function for doing this is the softmax activation layer. To understand this is, it is necessary to look at some key aspects of the softmax function in terms of a functional model. For each potential output label from our final hidden layer, we have that

$$softmax(z_i) = \frac{e^{zi}}{\sum_{j=1}^{n} e^{zj}}, \tag{9}$$

where n is the number of potential classification labels and z_i is the current output label being considered. In addition to normalizing these input between 0 and 1, note the sum of the z_i's add to 1. This function also distributes some of the probability to every label, and this is is important

for differentiability of the terms.[HINT] And, with this, we have completed our description of the process of forward propagation in an LSTM network.

2.6 Selection of the Cost Function

With the softmax layer being the conclusive step of the forward propagation algorithm, the neural network will have completed a forward pass whose output is a probability distribution function. This can be used to determine the classification of the sentiment from a given input stream, where the label with the highest density is considered the output label. At this point, the process of learning, or backpropagation, will begin. The initial stage of backpropagation is done through determining the error associated with the outputs of the neural network by comparing the output probabilities with the expected values associated with the training data set. In order to accomplish this, a metric of accuracy, often called a loss function, must be chosen. A loss function for a given data $\mathscr{D} = \{(x^i, y^i) \mid i = 1, \ldots, n\}$ is concerned with the conditional probability $P(y^i \mid x^i)$.[BAR] That is, we wish to maximize the likelihood of conditional probability by minimizing the defined distance metric between the predicted output label and the expected value. Then, as the loss function is considered a metric, we have that $\mathscr{L}(y^i, y^p)$, where y^p is the predicted label of the neural network should be 0 if and only if $y^p = y^1$.[MEN]

From the softmax function classification layer, we have that the output vector will be in the form of the multinomial logit model, and will be such that the addition of the scores of all potential classification labels equals one; this ensures the output will be in the form of a probability distribution.[GRA] Additionally, the expected classification labels are one-hot encoded, meaning the output labels also form a discrete probability function. A difference between the two the output layer distribution and this one is that not all classes get some of the potential

probability. In fact, all output labels are associated with a value 0, unless it is the index of the output associated the correct expected output. That is, for the probability associated with an output classification label, y^j, $P(y^j) = 1$ for $j = i$ (where i is the index of the expected output label) and 0 otherwise. The distance metric employed by this model is a measure of statistical distance between distributions, and this is known as an *f*-divergence function.[SHEN] Of the common *f*-divergence functions available, one that has become ubiquitous in classification problems of machine learning is the cross-entropy loss function, often called the negative log-loss.

The cross-entropy loss function comes from the overarching study of information theory. Formally proposed by Claude Shannon, this study sought to quantify the data presented in a message regardless of the method of encoding.[CSH] The units utilized to create this quantified analysis became known as bits of information, where the values a bit can take on are 0 and 1. From this, we gain that the number of messages that can be created and the encoded bit stream are logarithmically related; that is, given n potential messages, where each is equally likely, each message is characterized by $log_2(n) = d$ bits. Keeping this in mind, the entropy (a measure of randomness of information in a system) of a system p with m different messages, where each message's probability of choice is denoted p_i, is calculated with the equation

$$H(p) = -\sum_{i=1}^{m} p_i log_2(p_i) .^{[CSH]} \tag{10}$$

Though this equation is helpful in characterizing the randomness of a single probability distribution, it is not enough to determine the differences of expected values between two separate probability distributions; for that, we need the relation of Kullback-Leibler (KL)

divergence. KL divergence is, alternatively, known as the relative entropy between two distributions. For probability distributions p and q, the KL divergence of the two is calculated as

$$KL(p,q) = \sum_{i=1}^{m} p_i log_2(\frac{p_i}{q_i}) \tag{11}$$

$$= \sum_{i=1}^{m} p_i log_2(p_i) - \sum_{i=1}^{m} p_i log_2(q_i) \tag{12}$$

$$= -H(p) + H(p,q). \tag{13}$$

where the $H(p,q)$ term is known as the cross-entropy of the distributions p and q. [MUR]

The cross-entropy of the distributions is the measure of the average bit encoding length of the distribution q to represent the true value of the distribution p; [MUR] a representation of this would be how efficiently (or with how little information loss) is the distribution q able to accurately represent the distribution p. Note, further, that the cross-entropy function would not be defined for a term that had no probability of being selected from the distribution q. Luckily, as defined above, the softmax output classification layer ensures that all potential classification labels receive some measure of potential probability greater than zero. Also, since the expected values of the training set are one-hot encoded in the probability distribution, p, for a specific input $(x, y) \in \mathbb{D}$, the expected value vector of $y = [y_1, y_2, \ldots, y_m]$, where there are m potential classification labels, implies that only one vector element holds all probability of selection. For example, assuming that the vector element of y that was one-hot encoded was denoted y_k, and q represents the output distribution, then the cross entropy of the output of the neural network and the predicted values of the dataset is given by the equation:

$$H(p,q) = -\sum_{i=1}^{m} y_i log_2(q_i) \tag{14}$$

$$= \ -[y_1 log_2(q_1) + ... \ + y_k log_2(q_k) + ... \ y_m log_2(q_m) \ ] \tag{15}$$

$$= -[0 * log_2(q_1)... + 1 * log_2(q_k) \ + ... + 0 * log_2(q_m)] = \ -log_2(q_k). \tag{16}$$

This simplification of the term yields that, in the case of classification, the cross-entropy of the predicted and expected values is determined by the log-loss only in the case of the true classification label. This is the final form of the loss function that we need to determine the error of the current state of the LSTM neural network. Now, the relationship between this loss function and the output layer of the neural network is clear; as stated previously above, the loss function is a metric, so it holds that,

$$\mathfrak{L}(y^i, y^p) = H(y^i, y^p) = 0 \ \textit{if and only if} \ y^i = y^p. \tag{17}$$

So, in holding with the goal of maximizing the likelihood of predicting the actual classification label, we must minimize the loss function. This is the aim of the backpropagation algorithm, of which, we know have the first required piece. The next step is to backpropagate the calculated error through the network in order to incrementally update the weights of the synapses. By doing this, the model will iteratively be improved.

2.7 The Backpropagation Algorithm

The backpropagation algorithm is the method by which neural networks are capable of learning. Backpropagation consists of a set of instructions for the neural network that allows for the update of the weights found in the nested matrix operations of each layer. The method by which these updates are made can be explained at a high level through the description of gradient-based descent optimization methods. Gradient-descent is an optimization method that seeks to find global minima of a function by taking a step in the negative gradient direction of the function.[KING] In the context of the neural network output, this would mean finding the

weight values in the hidden layer and input layer that minimized the loss function; in our case, this would be the cross-entropy loss function described in the previous section.

The overall steps of the algorithm include completing the forward propagation through the network, using this output to calculate the error via our loss function, calculating the gradient of each layer with respect to the current input through the network and summing this with the calculate error (essentially propagating the error back through time), and, finally, incrementally updating the weights of the neural network by adding the summation of the error scaled by a constant factor known as the learning rate.[KING] For now, consider the learning rate to be small constant value between zero and one that will scale the rate at which our updates change.

The preceding is a broad view of gradient-descent based optimization methods, and a choice for which specific method to utilize needs to be made in order to implement this successfully in the network. There are several issues with backpropagation, most of which are resultant from the chosen optimization method. The initial issue is that, oftentimes, gradient descent will step towards the direction of the nearest minimum. If the initial starting weights veer towards local minima, the network will settle on this solution, even though it is not optimal.[KING] Further, “vanishing gradient” (e.g. such as a step function) is an issue by which the propagated error has such a shallow gradient that it effectively stops the network from training. As such, stochastic gradient descent with a momentum term was developed. Stochastic gradient descent (SGD) utilizes batches of sample data, rather than training over the whole set at once. This allowed for updates to be made more frequently, and with a greater amount of variance, potentially veering away from local minima.[KING] This, alone, though is not enough to handle the issue of vanishing gradient. As such, a momentum term was introduced that dampened

oscillations of the gradient in directions that were not veering to a minimum, and increased the oscillations towards those that did. The result of this would be an optimizer that would be able to pull away from local minima, but could also cause the problem of "exploding gradient", wherein the optimizer would overshoot the ideal minimum, again settling on one of the local minima.[KING] These difficulties led to the development of the Adaptive Moment Estimation (ADAM) optimizer was created. Adam is a method by which the first and second moments of the gradient are calculated and utilized to scale our momentum factor before each iterative pass of updates. This ensures that our momentum 'slows' as we approach a optimal solution. As the newest and most stable methodology, we will utilize ADAM as the optimization method in our network. In both cases of vanishing or exploding gradient, it is noteworthy that the selection of model (LSTM) is helpful in avoiding the issues because the internal save state keeps track of error from previous time-steps, and this helps with finding the collected gradient at each hidden layer. Combined with ADAM, an LSTM is an effective means of dealing with error and optimization of weights and bias values. We are now ready to see how to implement this network to perform sentiment analysis on movie reviews.

3 Implementation

3.1 Python and Packages

The network was created using the Python programming language along with the Keras API built on top of the Tensorflow library. This allowed me to easily load in a pre-sorted dataset of 25,000 movie reviews where 12,500 were positive and 12,500 were negative. Using a randomly generated seed to determine distribution of the positive and negative reviews, the reviews were split into training and testing datasets. The data science library, scikit-learn, was

utilized for any further splitting of the data, as well as for visualization of data and confusion matrices; the latter will be explored later. The Keras layers were built with an initial embedding layer as the input to the model, utilizing the GloVe model for word lookup and value comparison, followed by 128 LSTM units in sequence. The output layer was the softmax layer, pushing the data into binary classification labels of 'positive' and 'negative'. The binary cross-entropy log loss function was set as the means of error checking with accuracy of prediction being the metric for comparisons. Further, the optimizer was set to ADAM and the model was compiled. At this point, the model was trained utilizing the *train* function, where a validation set could be set if one was created, the number of epochs could be set, and then the batch sizes could be set, testing mini samples of the training set at a time. Then, the model was called using the *predict* function being fed the remaining 12,500 testing samples. Then, scikit-learn was utilized to display the confusion matrices of the run, where a confusion matrix is a table of the form shown in Figure 6.

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Figure 6: Confusion Matrix

(Note that the actual values is this table are from a fictional model making a prediction of 'Yes' or 'No' on 165 data values). From this point, the number of True Positives (TP), True Negatives (TN), and False Positives (FP), and False Negatives (FN) were utilized to create data summaries of each attempt to train and test the network architecture. In particular, the following metrics of model predictions can be computed:

Accuracy: $\frac{TP + TN}{TN + TP + FP + FN}$ *(18)*

True Positive Rate(TPR): $\frac{TP}{TP+FN}$ *(19)*

True Negative Rate(TNR): $\frac{TN}{TN + FP}$ *(20)*

False Positive Rate(FPR): *1 - TPR* *(21)*

False Negative Rate(FNR): *1 - TNR.* *(22)*

We seek to reduce the FPR and FNR as much as possible, and aim to gain optimal levels of accuracy, TPR, and TNR. Though some measures are more important than others dependent on the details of a given study, in the case of our network, we wish to see a balance (or similarity) between accuracy, TPR, and TNR, as we hope that it is capable of detecting positive and negative sentiment with relatively similar rates, and with little bias for either classification label.

3.2 Experimentation

As described above, the output of the network allowed us to gain a few different metrics for comparative analysis of the model. For the sake of studying what different hyperparameters might skew the performance of our network, I ran the network at 2 different potential epochs and 2 different training/validation/testing data splits, yielding 4 permutations of hyperparameter

settings. Then, I compared the models created to find which yielded the highest and most balanced results. Before continuing on with the potential choices of our models, we should explain the training/validation/testing split in more detail. The training data set is the data on which the data undergoes the process of learning, and the testing data set is fed to the network to determine its performance. But validation is a bit different. A validation data set is tested at each iteration of training on a given epoch. If the validation set is tested and found to be yielding a high enough accuracy, training stops, ensuring we do not overfit or negatively train the model past an optimal point.

Now that we have the required information, we can discuss the experimental results. The models created were tested at 2 and 4 epochs. An epoch is considered to be one full pass over the training data set. That is, it uses a random selection method to break the training set into smaller batches and then trains on each batch, until it has seen every training sample one time. The process of passing through the total number of training samples is known as an epoch. Also, the network was tested at different tested at training/validation/testing splits of 50/0/50, and 50/25/25, where each number is representative of a given percent of the total existing data set. The results of the experiments can be found below in Figure 7.

LSTM Performance					
Model	Accuracy	Sensitivity (TPR)	Specificity (TNR)	Miss Rate (FNR)	Fallout (FPR)
50/25/25 Epochs:2	0.8487	0.8174	0.8024	0.1826	0.1976
50/25/25 Epochs:4	0.8243	0.8159	0.813	0.1841	0.187
50/0/50 Epochs:2	0.8146	0.8294	0.8302	0.1706	0.1698
50/0/50 Epochs:4	0.831	0.8451	0.8453	0.1549	0.1547

Figure 7: Test Outputs [Conley]

Note, the allusion to the probabilistic modeling of a coin can be made in the case of binary classification. That is, if the model was a truly random, untrained network, we would expect to see it classify the 'positive' and 'negative' samples correctly around fifty percent of the time. To be sure, it would appear that our model is capable of outperforming a randomized selection method, as the LSTM at all experimental hyperparameter settings was capable of an average accuracy score of 0.82965 (roughly 83% percent of the time). It is apparent from the above data that most of the models performed very similarly, but, overall, the most accurate and balanced model seems to have been the 50/25/25 2-epoch LSTM model. Still, the 50/0/50 4-epoch model, though lower in accuracy, did have a higher rate of true positives and true negatives. All models performed well at a relatively low amount of epochs and generalized well when comparing their test values to the training iterations. One could infer that the, at least within the scope of the changes made to the hyperparameters, that very little variance between scores was introduced as a result of these changes. As such, this is demonstrative of the strengths of the LSTM model, as stated previously. It is able to determine context, create a save-state for error, and defend against issues of vanishing and exploding gradient. Thus, the model performed quite well under varying issues, creating a balanced model architecture for our problem, overall.

Further experiments were done comparing different model architectures I constructed to solve the problem of classifying the IMDB movie reviews data to the performance our model. The model architectures included the convolutional neural networks (CNN), gated recurrent unit networks (GRU), and bi-directional LSTM networks (bi-LSTM), all tested at the above permutations for epochs and splits of data. There did seem to be a great deal of similarity

between LSTM and bi-LSTM and it is just a modded structure of a normal LSTM network, where inputs are fed from start and end, changing the order of input sequences. Again, LSTM performed similarly to the GRU networks as well, as the GRU is a proposed model that would simplify the internal gate structure of an LSTM, but provide similar results. A more shocking result was that the CNN (a feed-forward network architecture noted for its ability to handle visual data) seemed to outperform all other models greatly, even in testing. This led to me to believe that it generalized much better, even though it wasn't a recurrent architecture. By using sci-kit-learn to visualize the the training loss versus the validation loss over time, though, it became apparent that the CNN architecture, being more simple, was massively changing its weights to fit the problem, very quickly becoming overfit. It is my conjecture that the reason it held up against testing data is probably the small size of the dataset with a lack of diversity in the samples available. Fore more information on the performance of these nets, the calculated performance metrics for the neural networks can be found in Appendix A.

4 Conclusion

The process of development for a neural network begins far before touching any code. By determining the type of problem and assessing what kind of input the network will take, you are able to limit the potential architecture types and accompanying decisions for output and cost functions. Though there exist some choices in the hyperparameters of the network, the expected change caused by these differences in the parameters seems to vary dependent on architecture and problem. The generalized network model may seem somewhat simplistic in the utilization of linear algebra and matrix operations. In the process of designing a network to fit a specific problem, though, there exists the potential to push the boundaries of the process into fields like

information theory, natural language processing, statistical optimization, calculus and partial differential equations. Overall, this project, although designed for a relatively simple example, allowed us to effectively explore the process of design and implementation of neural network architectures in machine learning, yielding insights that can be generalized to far more complex examples.

Appendix A

Results of Multi-model Experimentation

Train/Valid/Test: 50/25/25 Epochs: 2					
Model	Accuracy	Sensitivity (TPR)	Specificity (TNR)	Miss Rate (FNR)	Fallout (FPR)
LSTM	0.8487	0.8174	0.8024	0.1826	0.1976
Bi-LSTM	0.8653	0.8423	0.892	0.1577	0.108
GRU	0.83648	0.8184	0.8053	0.1816	0.1947
CNN	0.871	0.9119	0.9195	0.0881	0.0805

Train/Valid/Test: 50/25/25 Epochs: 4					
Model	Accuracy	Sensitivity (TPR)	Specificity (TNR)	Miss Rate (FNR)	Fallout (FPR)
LSTM	0.8243	0.8159	0.813	0.1841	0.187
Bi-LSTM	0.837	0.91	0.925	0.0899	0.0749
GRU	0.8328	0.7896	0.7547	0.2104	0.2453
CNN	0.8712	0.9196	0.9279	0.0804	0.0721

Train/Valid/Test: 50/0/50 Epochs: 2					
Model	Accuracy	Sensitivity (TPR)	Specificity (TNR)	Miss Rate (FNR)	Fallout (FPR)
LSTM	0.8146	0.8294	0.8302	0.1706	0.1698
Bi-LSTM	0.8625	0.8455	0.8315	0.1545	0.1685
GRU	0.8426	0.8056	0.7733	0.1944	0.2267
CNN	0.8651	0.9271	0.9353	0.0729	0.0647

Train/Valid/Test: 50/0/50 Epochs: 4					
Model	Accuracy	Sensitivity (TPR)	Specificity (TNR)	Miss Rate (FNR)	Fallout (FPR)
LSTM	0.831	0.8451	0.8453	0.1549	0.1547
Bi-LSTM	0.8526	0.8496	0.8422	0.1504	0.1577
GRU	0.8365	0.8266	0.8141	0.1734	0.1859
CNN	0.87	0.9215	0.9284	0.0785	0.0716

Works Cited

[DR] Chen, H., Engkvist, O., Wang, Y., Olivecrona, M., & Blaschke, T. (2018). "The rise of deep learning in drug discovery". *Drug Discovery Today*. doi:10.1016/j.drudis.2018.01.039

[NORM] Norman by MIT Media Lab. *Norman*. *http://norman-ai.mit.edu/*. Accessed 5 June 2018

[RR] Rojas Raúl. (1996). *Neural networks: a systematic introduction* (pg. 1-10). Berlin: Springer.

[Figure1] Periasamy, A. R. P., & Mohan, S. (2017). A Review on Health Data Using Data Mining Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 7(3), 291–297. doi:10.23956/ijarcsse/v7i3/0136

[Figure2] Xianjun, H. (2015). "The Research on Decision Mechanism of Agricultural Products Logistics Transportation Based on Neural Network". *The Open Cybernetics & Systemics Journal*, 9(1), 1207–1211. doi:10.2174/1874110x01509011207

[BA] Trindade, L., Wang, H., Blackburn, W., & Rooney, N. (2013). "Effective sentiment classification based on words and word senses". *2013 International Conference on Machine Learning and Cybernetics*. doi:10.1109/icmlc.2013.6890481

[UAT] Cybenko, G. (1992). "Approximation by superpositions of a sigmoidal function". *Mathematics of Control, Signals, and Systems*, 5(4), 455–455. doi:10.1007/bf02134016

[BAR] Barber, D. (2015). *Bayesian reasoning and machine learning*. Cambridge: Cambridge University Press. (1-15,280-289,146)

[MUR] Murphy, Kevin P., *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge, 2012.

[NLP] Chowdhury, G. G. (2005). "Natural language processing". *Annual Review of Information Science and Technology*, 37(1), 51–89. doi:10.1002/aris.1440370103

[SA] M., V., Vala, J., & Balani, P. (2016). "A Survey on Sentiment Analysis Algorithms for Opinion Mining". *International Journal of Computer Applications*, 133(9), 7–11. doi:10.5120/ijca2016907977

[STAN] Pennington, J., Socher, R., & Manning, C. (2014). "Glove: Global Vectors for Word Representation". *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. doi:10.3115/v1/d14-1162

[GTS] 6.4.1. Definitions, Applications and Techniques. "1.3.3.14.6. Histogram Interpretation: Skewed (Non-Normal) Right". *https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc41.htm*. Accessed 25 May 2018

[RNN] “Neural Network Architectures for the Modeling of Dynamic Systems”. (2009). *A Field Guide to Dynamical Recurrent Networks*. doi:10.1109/9780470544037.ch18

[Figure3] Santos, L. A. dos. “Recurrent Neural Networks. Markov Decision process · Artificial Intelligence”. *https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/recurrent_neural_networks.html*. Accessed 25 May 2018

[LSTM] Hochreiter, S., & Schmidhuber, J. (1997). “Long Short-Term Memory. Neural Computation”, 9(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735

[MUS] Huang, C. (2017). “Combining convolutional neural networks for emotion recognition”. 2017 *IEEE MIT Undergraduate Research Technology Conference (URTC)*. doi:10.1109/urtc.2017.8284175

[CALC] Stewart, J. (2019). *Calculus: concepts and contexts*. Boston: Cengage.

[Figure4] Sigmoid function. *Wikipedia*. Wikimedia Foundation. *https://en.wikipedia.org/wiki/Sigmoid_function*. Accessed 6 June 2018

[Figure5] “In Keras, what exactly am I configuring when I create a stateful `LSTM` layer with N `units`?” *Stack Overflow*. *https://stackoverflow.com/questions/44273249/in-keras-what-exactly-am-i-configuring-when-i-create-a-stateful-lstm-layer-wi?rq=1*. Accessed 8 June 2018

[HINT] Hinton, G. (2016). “Lecture 4.3 — The softmax output function” [*Neural Networks for Machine Learning*].

[MEN] Mendelson, Bert. *Introduction to Topology 3rd Edition*. Dover Press. 1990

[GRA] Graves, A. (2012). “Supervised Sequence Labelling”. *Studies in Computational Intelligence Supervised Sequence Labelling with Recurrent Neural Networks*, 5–13. doi:10.1007/978-3-642-24797-2_2

[SHEN] Shen, Y. (2005). “Loss functions for binary classification and class probability estimation”.

[CSH] Shannon, C. E., & Weaver, W. (1999). “The mathematical theory of communication”. Urbana: University of Illinois Press.

[KING] Kingma, D.P. and Ba, Jimmy Lei. “Adam: a method for stochastic optimization”. arXiv:1412.6980v9

[Figure6] Confusion matrix. (2018, June 2). *Wikipedia*. Wikimedia Foundation. https://en.wikipedia.org/wiki/Confusion_matrix. Accessed 6 June 2018