2018

Developing a Recurrent Neural Network with High Accuracy for Binary Sentiment Analysis

Kevin Cunanan

Pomona College
Department of Computer Science

Developing a Recurrent Neural Network with High Accuracy for Binary Sentiment Analysis

Kevin Andrew Cunanan

April 29', 2018

Submitted as part of the senior exercise for the degree of
Bachelor of Arts in Computer Science

Professor Mariam Salloum

Abstract

Sentiment analysis has taken on various machine learning approaches in order to optimize accuracy, precision, and recall. However, Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs) account for the context of a sentence by using previous predictions as additional input for future sentence predictions. Our approach focused on developing an LSTM RNN that could perform binary sentiment analysis for positively and negatively labeled sentences. In collaboration with Mariam Salloum, I developed a collection of programs to classify individual sentences as either positive or negative. This paper additionally looks into machine learning, neural networks, data preprocessing, implementation, and resulting comparisons.

Acknowledgments

I would like to thank my advisor, Professor Mariam Salloum, for providing guidance throughout the year to help me develop this topic and research. She was instrumental in providing me with resources, guidance, and feedback throughout the entire process.

I would also like to thank Professor Kim Bruce, Professor Alexandra Papoutsaki, Professor Everett Bull, and Professor Michael Greenberg for their help these past two semesters. Professor Bruce gave incredibly insightful feedback and suggestions for my paper and its numerous revisions. Professor Alexandra Papoutsaki gave my presentation and slides feedback in order to make it more engaging and palatable. Professor Bull and Professor Greenberg helped me cultivate and refine my knowledge on many interesting computer science topics which kickstarted this project.

Lastly, I'd like to thank my brother, Kenny Cunanan, for reviewing my paper and improving its overall readability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human beings are naturally endowed with the ability to understand language. Language allows for innumerable manners of verbal expression which we can almost always understand. However, that skill is not inherent in machines which require strict parameters in order to function. Fortunately, considerable progress has been made in Natural Language Processing (NLP) which allows computers to understand language. NLP has become culturally relevant due to consumer inventions such as Apple's Siri, Amazon Alexa, and Google Home which provide powerful voice recognition and command-based software to everyday consumers at a relatively low price point [Ohl]. NLP employs machines to function in a manner that's user-friendly and natural. However, NLP is by no means perfect; Amazon recently dealt with a NLP bug that triggered the Amazon Alexa to eerily laugh for no apparent reason. This occurred because of a somewhat frequent sound that could easily create a false positive hit for the words "Alexa, laugh." [Day18]. To further improve NLP, engineers are constantly developing new programs and implementing revolutionary techniques.

One very important field to NLP is sentiment analysis. Although sentiment, a feeling or emotion, may be something nascent to humans and living creatures, there is a lot of value in building programs that can detect basic emotions such as happiness and sadness, and even more complex emotions like sarcasm. The continued development of analytical patterns that provide both singular and multiscale sentiment analysis confirms that importance of accurately analyzing emotion from text-based inputs. Singular classifications attempt to determine whether the text has only one of two emotions while more complex sentiment analysis can make an educated guess from a variety of emotions.

This paper will first cover general machine learning basics by reviewing important steps to train a basic AI while covering key algorithms and techniques. I then examine neural networks specifically including how they can tackle problems such as NLP. Next, I review noteworthy related work in general sentiment analysis as well as sentiment analysis in regards to neural networks. I then walk through my implementation for training a sentiment analyzing neural network and compare my results to other experiments.

Chapter 2

Machine Learning Overview

When most people think about machine learning, it's commonly perceived as some super intelligent machine that can think for itself. However, that's far from today's reality; most machine learning techniques train a program to be very good at finding patterns and making inferences from those patterns.

The beauty of machine learning, for many, is the ability to develop extremely nuanced and personal answers to various questions. One prominent use of machine learning stems from recommendation systems like those used in prominent companies such as Netflix and Yelp. Recommender systems are especially important in ensuring the platform can find the user meaningful content. Many of these examples from industry are constructed using fundamental algorithms and steps which are described below.

2.1 Preprocessing and Feature Extraction

It's important to note that before beginning any sort of machine learning on a set of documents, one must clean each document to remove any unwanted data. This could be anything from improperly formatted data, to removing stop words (the, is, it, etc.) that we don't want to feed into our program. For example, if we were to analyze a collection of books from *Project Gutenberg*, we'd notice there is metadata stored at the beginning of every book; it would make sense to systematically remove this information because it offers no insights into the text that we do not already know about. Note, we do this before doing any sort of machine learning because it would only take more time to read through that information in the ML process. What's next is a more in depth explanation of various types of machine learning techniques and implementations.

Next, the user needs to make important decisions regarding feature extraction; in other words, what characteristics about the data will likely lead to an accurate prediction? For sentiment analysis, we need to ask, what qualities of a sentence or tweet can potentially provide insight into the sentiment? Basic features can include but are not limited to: adjectives, word frequency, punctuation, and more. More complex features include TF-IDF, WORD2VECTOR, and metadata.

We can extract features from the metadata of an individual data record and use that to make predictions. For example, if it were the case that people made more negative remarks at night, then we could potentially use the time of day as a feature. Any other information associated with the actual text can be used in addition to the text.

Term Frequency-Inverse Document Frequency (TF-IDF) is a specific feature extraction approach based on word frequencies. The Term Frequency is the calculation of terms or phrases that appear most frequently within a document 'whereas Inverse Document Frequency represents the importance of a term relative to the entire corpus' [MB12]. The algorithm ultimately helps find words associated with specific topics by looking for words unique to one document (or a group), but not present in all of the corpus. Consequently, if we have documents with different expected categories, then there should be keywords that do not appear in other documents. One example of TF-IDF in practice would be analyzing various store reviews. The assumption is that reviewers likely drop important keywords that denote the type of store. Many reviews for a coffee shop might mention *frappuccino* and *barista*; using TF-IDF we can see how a high frequency of these words in the review likely indicate the review is for a coffee shop rather than, say, a hot dog stand.

Term Frequency Inverse Document Frequency (TF-IDF) calculates the frequency of words in a given document or data record in relation to the frequency of those words in the entire corpus or dataset [Ram]. These calculations then should give an estimate for how important those words are to a given document.

WORD2VECTOR relies on the premise that some words have similarities and connections to one another which can help us perform natural language processing [MCCD13]. Since words can have different word-endings but retain the meaning, WORD2VECTOR attempts to encode that in a vector that can have varying degrees of similarity and dissimilarity determined through vector distance.

WORD2VECTOR models need to be trained on a corpus of text before it can be used for another ML pre-processing task.

[0 0 0 0 1 0 0 0 0 0] = cat

[0 0 0 0 0 1 0 0 0 0] = dog

Figure 2.1: A given WORD2VECTOR model may encode CAT and DOG as these vectors. Since the two words are different in meaning but similar in category (i.e. mammal), then their vector distances may be further apart compared to CAT and FELINE or DOG and POODLE.

2.2 Machine Learning Algorithms

After properly identifying characteristics of a sentence using feature extraction, its possible to train a machine learning algorithm to properly categorize a sentences with some confidence. Naive Bayes classifiers use words independently as their primary feature and calculate probabilities based on the appearance of certain words. Support vector machines (SVMs) attempt to find the most suitable features that accurately generates a consistent classification scheme that correctly splits the training data [ZS14]. K-Nearest Neighbor (KNN) classifies sentiments based on the greatest number of similar documents to the one in question [GC12].

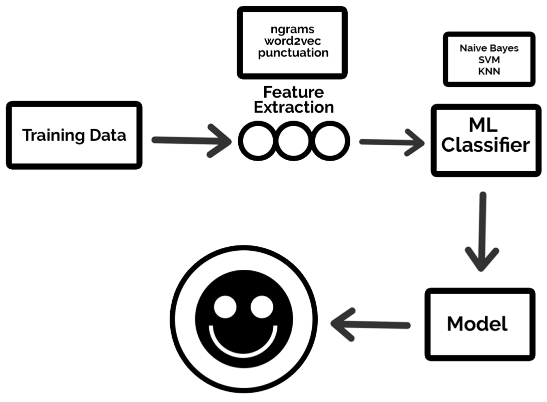


Figure 2.2: In general machine learning, learning goes through a series of steps until it can make a prediction.

Thus, the entirety of machine learning can be broken down into a distinct workflow as seen in Figure 2.2. After understanding what our machine learning task is, we can find a dataset. From there we attempt to identify significant features to provide valuable information for the ML classifier we will pass that data into. After some number of training iterations, we can save the current state of the learning process as a model and use it to make predictions or further examine the testing accuracy.

Unfortunately, rudimentary machine learning approaches are unable to grasp the context of a sentence. Because of the complexity of human language, we can have several different meanings for the same sentence and also several different sentences for the same meaning. In this respect, the challenge for natural language processing is 'the translation of potentially ambiguous natural language queries and texts into unambiguous internal representations on which matching and retrieval can take place' [Cho03].

The ML approaches mentioned above would be unable to accurately classify in this manner since they cannot account for words or phrases except in isolation; the features are used independently and the order of the sentence or document is ultimately unaccounted for. The next section of this paper explain how neural networks can better approach sentiment analysis.

Chapter 3

Neural Networks

3.1 History

At its fundamental level, a neural network attempts to replicate the basic structure of a human brain [Gra07]. Humans learn through a repetitive learning process that strengthens our neural pathways. The growth of neural networks originated from extensive nervous system research conducted by Donald Hebb in the 1940s. He concluded that humans and other animals learn by strengthening the neural pathways "by averaging the results from a number of animals, trial by trial, that one can manage to get a simple curve showing steady improvement with practice" [Heb49]. These insights looked more into how humans and living creatures learn and motivated others to replicate that procedure using computers.

Dr. Bernard Widrow came up with a mathematical and technical approach to create "trainable neural elements" by using the steepest descent algorithm to "direct the weight vector" which effectively helped generate better predictive models [Wid05]. Since Widrow could determine how accurate the output was, this could be fed back into the algorithm to adjust the weights to maximally increase accuracy. These initial findings, however, were not developed using software but circuits due to technological limitations of the time. Not until much later could software advancements for neural networks be implemented.

3.2 How Neural Networks Work

Today, neural networks have become exceedingly important because of their ability to mimic human learning. They are used for solving problems that are “complex, ill-defined, highly nonlinear, of many and different variables, and / or stochastic” [Gra07]. Neural networks are capable of this because they process the input through different layers which allow them to analyze an input from several different aspects and then come up with a prediction.

Neural networks are composed of cells. Each cell contains an activation function which is a mathematical function that accepts one or more inputs, but only has one output. These activation functions can be associated with the firing of neurons. In order for a cell to activate, it needs to reach a certain number typically determined by a sigmoid function given by the function $f(x) = \frac{1}{1+e^{-x}}$. However, the network also has a global bias term that can

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

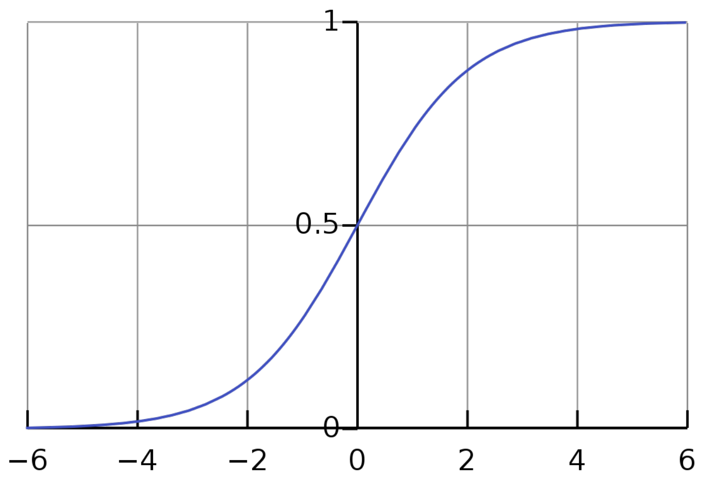


Figure 3.1: A cell in a neural network activates after passing through a sigmoid function. Determined by $f(x) = \frac{1}{1+e^{-x}}$. Values quickly approach either 1 or -1 which equate to positive or negative activation.

be configured manually. The bias subtracts a fixed number from the cells activation function before being plugged into the sigmoid function. This effectively allows it to shift the cells output in another direction which can help correct activation functions that may be too eager to activate based on a certain dataset.

The complexity of neural networks lies in complex connections cells have between one another; a given cell’s outputs serve as another cell’s inputs.

However, every neural network separates cells into layers. These layers consist of input layers, output layers, and hidden layers. The input layer starts as the initial point where data is fed into the network while the output layer is the final layer that makes a prediction. The hidden layers serve as intermediary layers that make inferences on different parts of the input.

One additional layer that many choose to add to their neural networks is a dropout layer. These were created to prevent a common phenomenon known as overfitting. This occurs when a neural network has essentially memorized the training dataset. Memorization is bad because although the network has optimized for the training data, it will perform horribly on anything outside of that. In order to combat this, we can randomly remove cells during training and testing by blocking their incoming and outgoing signals. Doing so helps prevent the network from receiving familiar input consistently and will be forced to make an inference from fewer inputs [SHK+14]. In order to train the neural network, the inputs of each cell are given individual weights. While these weights are relatively arbitrary at the beginning of training, theyre constantly being adjusted through backpropagation; the neural network changes these weights by checking how well its predictions performed against the labels and calculates the loss.

Neural networks can choose from a variety of optimizers that may be more effective for different types of training tasks. These optimizers serve as different calculations during backpropagation. Bernard Widrow developed the signature gradient descent algorithm that changes the weights in such a way that it will find the change in weights that will result in the steepest decrease in error.

Common to other types of machine learning processes, the neural network typically also undergoes validation and testing in addition to training. First, the validation phase occurs intermittently between training iterations. The validation stage tests the current model against data the model has never seen. The accuracy from the validation can be used to determine how well the training is progressing. At this point, the programmer can choose to end training. If training ends, then the model moves into a testing phase that gauges how well the network performs against data not in the validation or training set. The accuracy from testing is then the official accuracy generated for the entire network model.

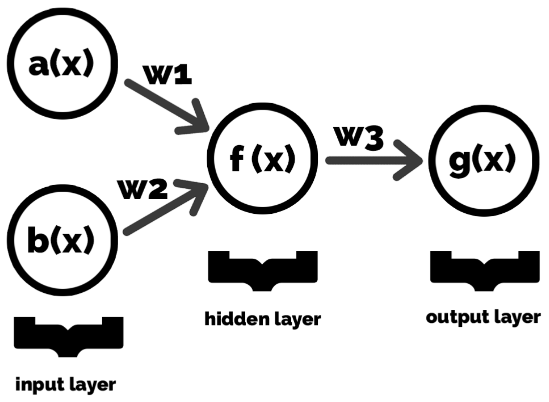


Figure 3.2: A basic neural network with weighted inputs and outputs that feed into the next layers. This network has one hidden layer.

3.3 Recurrent Neural Networks

General artificial neural networks perform quite well on a number of problems, however, these neural networks still consider words independent of the entire sentence. In order to resolve this, recurrent neural networks (RNNs) were developed. RNNs have feedback connections which provide previous neuron states as input [JM99]. This allows RNN to effectively handle NLP because they use in predictions about previous parts of the text as input back into the network.

By the end of the process, the network will have effectively used previous prediction scores to make a general prediction about the entire sentence. In order to understand this, let's investigate a basic sentence example.

In Figure 3.4, the sentence is broken up into 5 different parts. Assuming that I has no sentiment value, then by $t = 2$, we'd have processed I DON'T which would likely be a -1 prediction for negative. A RNN would then use that prediction from $t = 2$ to make a new prediction for $t = 3$. A well trained RNN would likely recognize that a negative prediction at $t - 1$, followed by another negative word such as NOT should be positive instead. Therefore, the following prediction score for everything up to $t = 3$ would be $+1$. This process would continually happen until the end of the input has been reached.

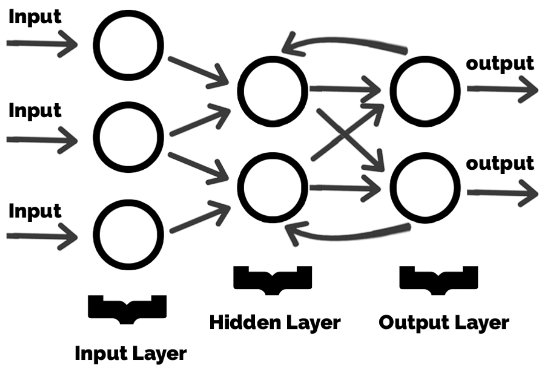


Figure 3.3: The figure above highlights that information passing through the RNN is re-directed back into the hidden layers as input for later predictions. This feature allows RNNs to hold onto information in a sequential fashion.

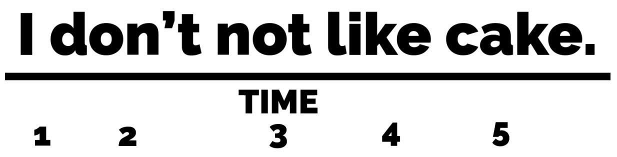


Figure 3.4: An example sentence for better understanding how an RNN would work.

3.4 Long Short-Term Memory

Recurrent Neural Networks with Long Short-Term Memory (LSTM) are a specific RNN that replicate memory using a memory cell. Each memory cell contains an input, forget, and output gate. Gates provide finer control over information and decide what passes through and what does not. These gates all have the same inputs passed into them, but interact with them differently. LSTMs have a forget gate to enable "the LSTM to reset its own state" [GSK$^+$15]. This allows the LSTM to look back many timesteps in order to make a more accurate prediction.

Let's imagine we wanted to create a LSTM to produce a 6 digit number without repeating previous numbers. If our current number was 123, then our LSTM would have these inputs passed into the neural network. The forget gate would be in charge of forgetting the current input, namely 3, but relay information that 1 and 2 cannot be used. The rest of the neural network would still have the current input of 3, and factor in output from the forget gate to choose the next number. In terms of sentiment analysis, the LSTM is especially useful for analyzing words farther back in the sentence since not all significant words occur right after each other.

Chapter 4

Related Work

4.1 Early Work

Sentiment Analysis has grown significantly since the early 2000s. However, one of the first significant attempts to classify words began with Hatzivassilogou and Mckneown in 1998 with their work on *Predicting the Semantic Orientation of Adjectives* [MGK16]. Because dictionaries do not contain sentiment, they made a basic assumption that the sentence itself would provide that information. Their algorithm looks for conjunctions between adjectives and utilized that to generate similarity scores between words which could then be used to find the best clusters of adjectives [HM97].

For example, using the sentence I THOUGHT DISNEYLAND WAS FUN, BUT EXPENSIVE reveals that the words FUN and EXPENSIVE are dissimilar words evident from the conjunction BUT. After analyzing numerous documents, they were able to train an algorithm that could infer sentiment with accuracy scores of around 90%. However, their work involved people manually verifying their results which appeared to focus specifically on identifying the sentiment orientation of adjectives and not sentences as a whole.

4.2 Recent Work

A study by Sharma and Dey utilized artificial neural networks (ANNs) for sentiment analysis by training on movie reviews [SD12]. Notably, they used a feature extraction technique called INFORMATION GAIN (IG). IG measures the entropy which is the anticipated information needed to classify something which can then be used to build decision trees for likely classifications given appearances of certain features. Information gain correlates roughly to information that strongly connects to a given document; by finding a collection of attributes that strongly correlate to certain documents, we can reduce the amount of information needed to make classifications.

Let's look at a restaurant categorization problem that can train on a set of reviews. IG can possibly check unigrams and find ones that maximize entropy in the given review, but is small in the entire dataset. Thus, information about the name of a food in a given review such as CHICKEN KATSU should have a high entropy in the review, but is likely uncommon amongst all restaurant reviews for many different restaurants. IG differs from TF-IDF since it usually breaks down features into subsets. In other words, a restaurant may likely be Hawaiian if they not only serve chicken katsu, but also serve macaroni salad.

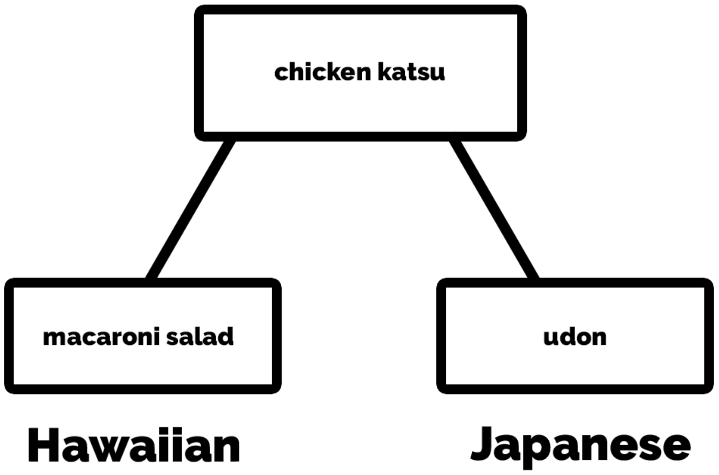


Figure 4.1: A simple information gain decision tree based on the appearance of the words chicken katsu, udon, and macaroni salad. Information gain can make more specific classifications by associating features together.

They then used a standard ANN to calculate the weights using back-propagation. Their ANN was able to achieve around 95% accuracy on movie review classifications. Their results primarily highlighted the advantage of using the INFORMATION GAIN feature selection technique over others.

In 2014, Zainuddin and Selamat investigated binary sentiment analysis using SVM (Support Vector Machine) [ZS14]. Their study involved implementing various feature extraction techniques such as TF-IDF in conjunction using unigrams, bigrams, and trigrams. Using TF-IDF with different n-grams will yield various results because the frequency of two words (bigrams) would theoretically be different than one word (unigram). For example, the frequency of the unigram PIZZA may be different than the bigram COOKING PIZZA. The latter more definitively highlights the process of the pizza being made which could potentially be a cooking class on pizza. On the other hand, the former could easily be referring to either a pizza cooking class or a pizza restaurant. Using TF-IDF with unigrams, however, yielded the highest accuracy of 77.50%.

There has been recent work by Severyn and Moschitti in 2015 that specifically focused on convolutional neural network (CNN) sentiment analysis using Twitter tweets [SM15]. CNNs are special because of their additional convolutional layers. Convolutional layers are more adept at detecting patterns especially in images and speech; these layers have filters that help identify certain patterns. Their work analyzed how the initialization of weights at the beginning of training affects overall accuracy.

In order to test different weight initializations, Severyn and Moschitti had three different evaluations which were random, derived from word embeddings, and one where word embeddings are further tuned using a distant supervised dataset. After training, each was tested on various labeled Twitter datasets. Their official results revealed that the distant supervision improved test results overall with message-level accuracies of around 73%.

In 2018, Wang, Sun, Han, Liu, and Zhu published a paper in 2018 on *Recurrent Neural Network Capsules* for positive and negative sentiment classification [WSH$^+$18]. Their experiment trained a capsule network which is an augmentation of CNNs. After processing data through a neural network, the information is represented as a vector entity passed to some number of capsule determined by the number of potential predictions or categories an algorithm can make; these capsules then compute the prediction and also reconstruct the representation of the input. For this paper, since each capsule is associated with a different sentiment, it provides two important computations; how certain is the capsule that the input matches the capsule's sentiment, and how accurately can the capsule rebuild the original input from the computed input.

For example, a capsule network for sentiment analysis would attempt to reconstruct the actual sentence and check how well that sentence matches up against the original sentence. From there, it computes a reconstruction loss coupled with the prediction score. If these values combined reach a certain number, the capsule can activate (like a cell), which then feeds its output into a final prediction score. Capsule networks are usually utilized for image recognition in order to identify images that are rotated or skewed a certain way; other neural networks are unable to handle this since they look for features in the same general location and cannot look for features in different regions of the image. However, this method highlights that capsule networking can also provide high accuracy for NLP problems.

Using three different datasets (movie reviews, hospital feedback, and a proprietary dataset), their capsule network outperformed LSTM, Naive Bayes, and SVM by yielding an accuracy of 91.6% on a hospital feedback dataset and 83.8% on the movie review dataset.

Chapter 5

Environment

The coding environment we worked under immensely improved our workflow by increasing code reusability, reducing run-time, and providing valuable data representations. All programming was done using Python 3.5.4.

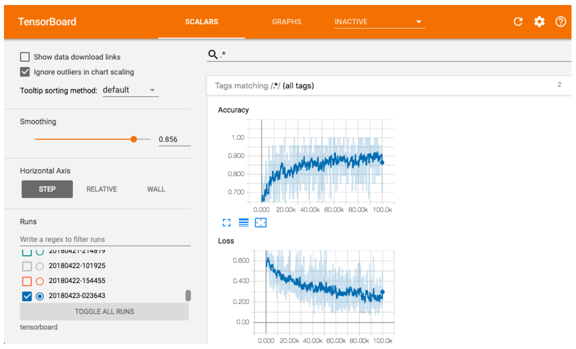
5.1 Jupyter Notebook

Jupyter Notebook provides users the ability to use literate programming. This allows programmers a user-friendly and readable way to add paragraphs of text to describe and highlight their code. Jupyter supports various markdown and languages including Python, Markdown, and LaTeX. Jupyter supports a cell structure where code can be written, executed, and re-executed on demand. As a result, Jupyter Notebook is ideal for machine learning since it helps you effectively organize and configure your code to minimize unnecessary code reruns. We utilized Jupyter by splitting our code into various areas for preprocessing, configuration, training, and testing. That way, if we needed to rerun one of those code blocks, we would not have to rerun them all at once. For example, changing code in the configuration cell allowed us to immediately run the training without needing to rerun preprocessing.

5.2 Tensorflow and Tensorboard

Tensorflow by Google is a machine learning framework with great documentation for configuring and training neural networks. We decided to use TensorFlow because its library already has many key neural network features already built into the API. The API allows users to easily plug in different layers into the network without much overhead. In my project, I utilized Tensorflow 1.3.

Tensorboard is used in conjunction with Tensorflow and provides users with a ready-made interactive dashboard to view all training iterations. These are essential for viewing accuracy and loss plots while the network is training.



[ ] Show data download links
[✓] Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing: 0.856
Horizontal Axis: [✓ STEP] RELATIVE WALL
Runs
Write a regex to filter runs
[ ] 20180421-214819
[ ] 20180422-101925
[ ] 20180422-154455
[✓] 20180423-023643
.*
Tags matching /.*/ (all tags) 2

Figure 5.1: The Tensorboard dashboard with options to manipulate views of various training runs.

5.3 Matplotlib

Matplotlib allows users to easily and quickly plot large amounts of data. It offers standard line and bar plots as well as histograms, color coordination, labeling, and more. We utilized Matplotlib to perform necessary data analysis on the individual data sets and to visualize testing and training iterations. These visualizations provided us with valuable insights into how to better optimize our preprocessing and training configurations for improved performance.

Chapter 6

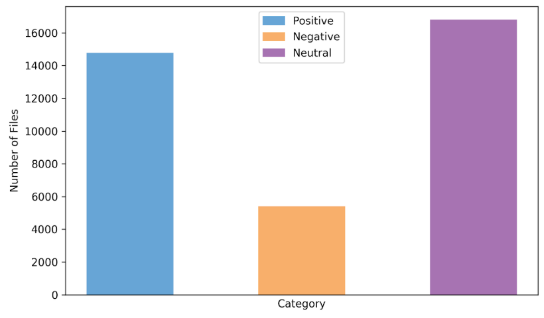
Approach

While many have attempted to implement sentiment analysis through various algorithms, we wanted to gauge the performance of a recurrent LSTM neural network using different Twitter datasets. Some previous work utilized significantly more complex neural networks and algorithms and achieved high accuracy doing so. While they compared their results to other ML approaches, most results are not compared against other neural networks. Thus, we wanted to measure their results against an LSTM recurrent neural network.

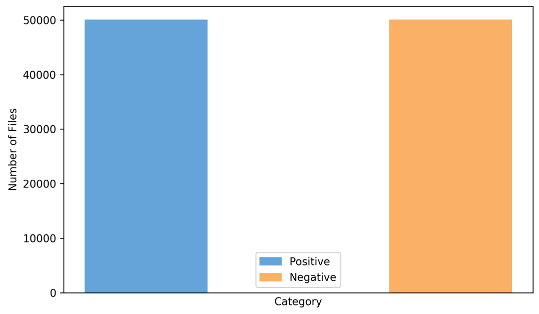
6.1 Dataset

This study focused on binary classification of sentiments using a LSTM recurrent neural network from two labeled Twitter datasets [RFN16, GBH09]. The first dataset (SemEval '16) contained manually labeled tweets categorized into various years. The second dataset (Sentiment140) was automatically generated under the assumption that tweets with happy emoticons were positive and tweets with sad emoticons were negative. Both datasets labeled each tweet as either positive, negative, or neutral.

After basic data processing, we found the SemEval '16 dataset to be heavily unbalanced for positive tweets. The dataset had approximately $16,000$ positive tweets and $5,000$ negative tweets. On the other hand, the Sentiment140 dataset had $800,000$ tweets for positive and negative. Additionally, due to the processing power limitations of my machine, we could only use approximately $100,000$ tweets in total. It's important to note that a balanced dataset is not necessarily best. If Twitter typically has more positive tweets than negative, then it may be appropriate. Whether or not this is the case is unfortunately outside the scope of this paper, however, it is important to consider trends such as these when applying the neural network to texts and inputs outside of Twitter.



(a) SEMEVAL'16 manually labeled dataset.



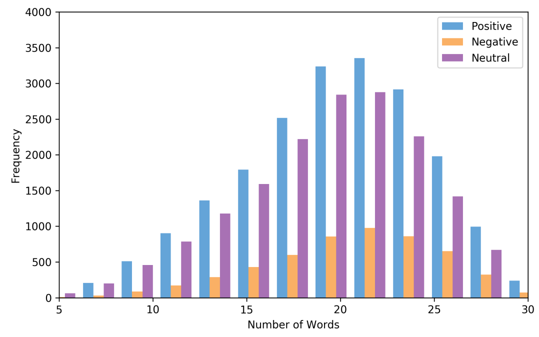
(b) SENTIMENT140 automatically generated dataset.

Figure 6.1: Tweet count by sentiment.

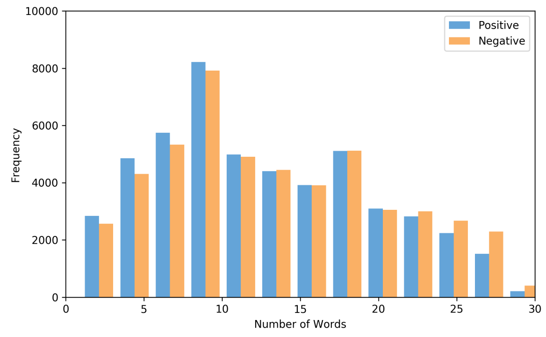
We were concerned with the unbalanced SEMEVAL dataset, because a trained model may be over-eager to choose positive sentiments. However, we decided to use it as a test against the balanced SENTIMENT140 dataset in future tests.

6.2 Preprocessing and Configuration

After loading the tweets into separate lists for positive and negative tweets, we analyzed the average words per tweet in order to determine the necessary sequence length for the neural network. Sequence length is the size of input we feed into the neural network determined for us by the number of words in a given tweet. After deciding on these basic configurations, we processed



(a) SEMEVAL'16 average word histogram.



(b) SENTIMENT140 number of words histogram.

Figure 6.2: Histogram plots depicting number of words per tweet.

each tweet through a WORD2VECTOR model trained from a news dataset with over $400,000$ words [PSM14]. This model then converted each word to a vector in the model. If a word could not be identified, the word was assigned a generic unknown word vector (the same vector was used for all unknown words).

After all tweets were successfully preprocessed, the dataset was shuffled, then split into a training, validation, and testing set. The training set was given approximately 80% of the dataset, while the validation and testing set were each given 10%.

6.3 Training

With the data preprocessed, properly shuffled, and split, we began training our neural network. We additionally configured an LSTM RNN with a dropout layer in order to prevent the neural network from memorizing the training set.

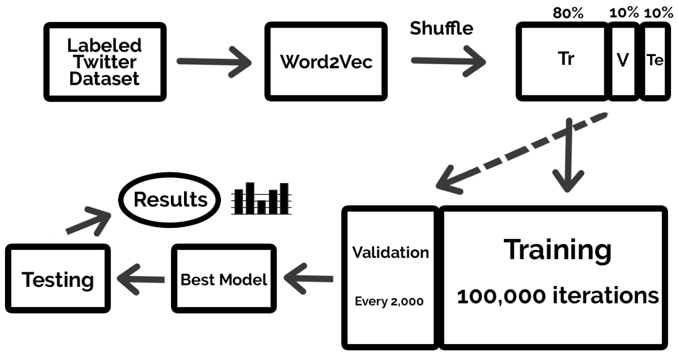
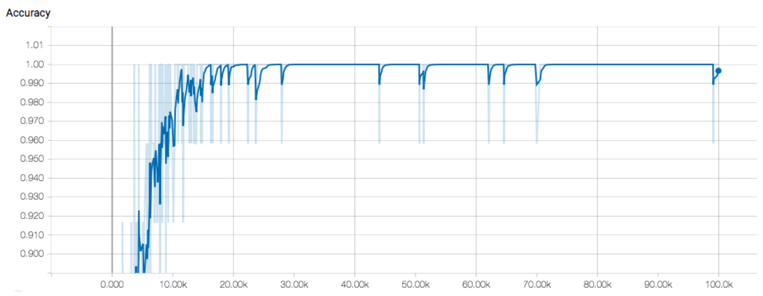


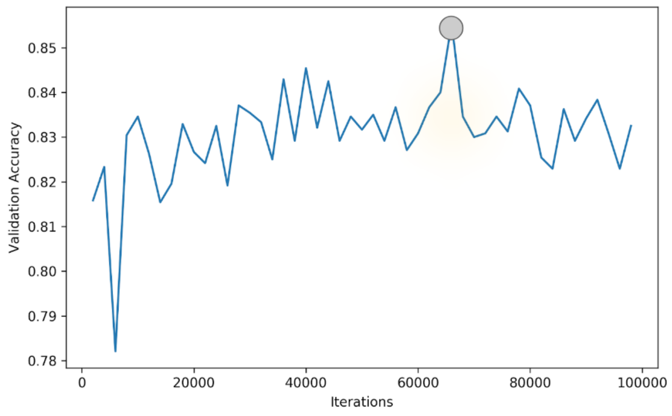
Figure 6.3: Workflow diagram.

To train, we randomly select 24 tweets from the training set to have the neural network learn from. We could then add controls to check progress every 24 tweets. For most training experiments, we chose to train on 100,000 iterations or 2,400,000 tweets with validation intervals every 5,000 iterations or 120,000 tweets. Every validation test checked the model's accuracy against 2,400 tweets. This setup allowed us to intermittently check how well our model performed against data not in the training set. Throughout this process, Tensorflow logged training accuracy and loss to Tensorboard every 50 iterations or 1,200 tweets.

During training, we logged the validation accuracy and kept track of the model with the best performance. After training completed, we took the best model and computed its accuracy on the testing dataset. While we logged general testing accuracy, we also calculated the total number of



(a) Tensorboard training accuracy with data logged every 50 iterations.



(b) SemEval validation accuracy over iterations. At around $66K$ iterations, we logged a validation of over 85% which we eventually used for testing.

correct and incorrect positive and negative predictions.

Chapter 7

Results

In order to test our models against data and measure it, we needed to capture information on how many correct predictions for positive and negative occurred as well as incorrect predictions. Correct predictions (true predictions) were those that matched with their label, whereas incorrect predictions (false predictions) did not.

$$\text{accuracy} = \frac{\text{truePositive} + \text{trueNegative}}{\text{totalPredictions}}$$

Figure 7.1: Our basic formula for calculating accuracy involves summing the total number of correct predictions by total number of predictions.

Using the formula in Figure 7.1, we made basic accuracy calculations and plots. General testing against our own datasets yielded validation accuracies up to 85.6% after training for 66K iterations on the SemEval '16 dataset. That model translated into a testing accuracy of 84.62%. However, after testing the same model on the Sentiment 140 testing set, it scored an accuracy of 64.64%.

We made further calculations of recall and precision using formulas presented in Table 7.1. The recall calculates the correct predictions for one sentiment divided by the incorrect and correct predictions for the sentiment. This helps us understand how good the model is for one sentiment; a higher recall means the overall sentiment category is more complete or accurate. Precision calculates the true positives divided by the true positives and false positives. In other words, how correct are the results? If the precision is poor, that implies the results for that sentiment are not useful or reliable.

Variable	Definition
tp	true positive
tn	true negative
fp	false positive
fn	false negative

Name	Formula
accuracy	$\frac{tp+tn}{tp+tn+fp+fn}$
recall	$\frac{tp}{tp+fn}$
precision	$\frac{tp}{tp+fp}$

Table 7.1: Formulas for recall, precision, and accuracy. These formulas provide us with helpful information to that lets us know if the model is biased.

Ideally, both the precision and recall should be high for a good model since that shows that the category itself is reliable with few incorrect guesses.

Our initial assumptions were correct when we looked into the prediction and recall graphs which split the predictions into true and false counts. A look into the prediction counts revealed that the SemEval model was heavily biased toward guessing positive sentiments. While this worked within its own dataset, it needed to be able to accurately predict sentiment from any potential tweet. When we tested the SemEval model using the Sentiment 140 dataset, our accuracy dropped to about 65% because it still made significantly more positive guesses.

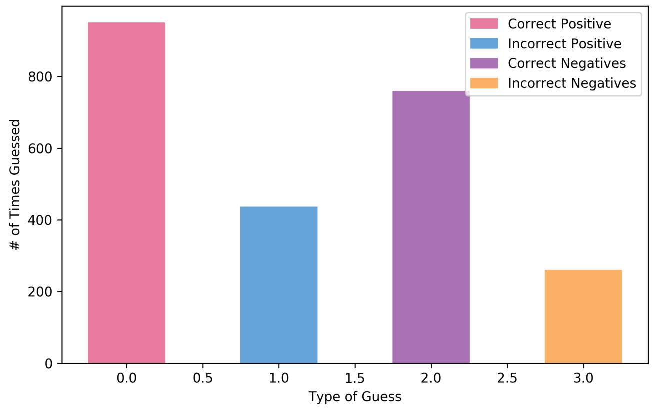


Figure 7.2: SemEval model specificity and sensitivity statistics from testing on Sentiment 140.

On the other hand, the LSTM recurrent neural network had trouble

Approach	**Testing Dataset**	**Accuracy**
SVM + TF-IDF Unigrams	Taboada Corpus	77.50%
RNN-Capsule	Movie Reviews	83.8%
Weight-Initilizated ANN	SemEval '15	64.59%
LSTM RNN SemEval	SemEval '16	84.62%
LSTM RNN SemEval	Sentiment 140	64.64%
LSTM RNN Sentiment 140	Sentiment 140	74.55%
LSTM RNN Sentiment 140	SemEval '16	70.38%

Table 7.2: Comparison for our LSTM RNN against various other networks and ML algorithms mentioned in our previous work section. Comparing accuracies between different datasets is difficult, but we can most closely compare our score to the Weight-Initialized ANN since the datasets are both from SemEval.

training on the Sentiment 140 dataset. As a result, testing and validation accuracy hovered around 75%. However, we noticed this model performed much more accurately against the skewed SemEval '16 testing set with a accuracy of 70.38%. Thus, although the network trained on a balanced dataset, it was not biased by the dataset it trained on. Because we do not have access to some of the datasets that other neural networks in previous works were trained on, it's ambiguous in some cases when comparing accuracies and recall. However, we can still use our general accuracy scores to compare to some degree. Our models rank well against the capsule network, but the hospital feedback and movie review dataset could potentially be working on larger inputs and have more sentences. On the other hand, both our networks slightly outperformed the ANN neural network. This can potentially indicate that recurrent neural networks are preferable to ANNs since they can capture context. The only other study also utilized recurrent neural networks was the capsule network experiment. Given that recurrent neural networks on average performed higher than other machine learning algorithms, we can reaffirm the importance of context within the sentiment analysis field.

Unfortunately, due to the limitations of a semester long study, we were unable to optimally train our LSTM RNN. We noticed that over 10% of words were labeled unknown during preprocessing runs which dilutes the inputs to the RNN. The primarily occurred because words used in Twitter appear as slang and typos which a WORD2VECTOR model trained from news articles would not be able to recognize. Creating a WORD2VECTOR model from our dataset could easily increase testing accuracy on both datasets. We also could experiment with different optimizing functions for training; although the ADAM'S OPTIMIZER is popular, there could be an optimizer better fitted for binary sentiment classification. More testing with these variables would allow us to make better judgments about how to improve recurrent neural networks for sentiment analysis.

Chapter 8

Conclusion

Achieving accuracy scores of 75% - 80% in sentiment analysis are surprisingly possible with the right dataset and configurations. Despite machine and time limitations, we were able to train a LSTM RNN to recognize positively and negatively labeled tweets with high accuracy. Although we cannot make sweeping generalizations with comparisons to related sentiment analysis work, our recurrent neural networks did outperform artificial neural networks performing similar tasks. However, the results also highlight possible guessing habits that our neural network developed from a severely unbalanced dataset. The higher accuracies of recurrent neural networks with long short-term memory over other networks and machine learning algorithms highlights the value of retaining contextual information for sentiment analysis. All in all, while these results do not provide any groundbreaking revelations, our work highlighted the ability to effectively capture sentence context in order to provide more accurate sentiment classifications. Investing more work into developing a Word2Vector model from our Twitter datasets and experimenting with various optimizers can potentially increase performance even more.

Bibliography

[Cho03] Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, 2003.

[Day18] The Day. Amazon promises to fix Alexas creepy laugh, Mar 2018.

[GBH09] Alec Go, Richa Bhayani, and Lei Huang. Twitter Sentiment classification using distant supervision. In *Twitter Sentiment Classification using Distant Supervision*, Sentiment140 - A Twitter Sentiment Analysis Tool. Sentiment 140, 2009.

[GC12] Vinodhini G and Dr Chandrasekaran. Sentiment analysis and opinion mining: A survey. 2, 06 2012.

[Gra07] D. Graupe. *Principles of Artificial Neural Networks*. Advanced series on circuits and systems. World Scientific, 2007.

[GSK$^+$15] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

[Heb49] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949.

[HM97] Vasileios Hatzivassiloglou and Kathleen R. McKeown. Predicting the semantic orientation of adjectives. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, ACL '98, pages 174–181, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics.

[JM99] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications.* CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1999.

[MB12] Teng-Sheng Moh and Surya Bhagvat. Clustering of technology tweets and the impact of stop words on clusters. In *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE '12, pages 226–231, New York, NY, USA, 2012. ACM.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[MGK16] Mika Viking Mäntylä, Daniel Graziotin, and Miikka Kuutila. The evolution of sentiment analysis - A review of research topics, venues, and top cited papers. *CoRR*, abs/1612.01556, 2016.

[Ohl] Frank J. Ohlhorst. Amazon Alexa poised to bring natural language processing to businesses.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[Ram] Juan Ramos. Using TF-IDF to determine word relevance in document queries.

[RFN16] Sara Rosenthal, Noura Farra, and Preslav Nakov. SemEval-2016 task 4: Sentiment analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation*, SemEval '16, Vancouver, Canada, August 2016. Association for Computational Linguistics.

[SD12] Anuj Sharma and Shubhamoy Dey. An artificial neural network based approach for sentiment analysis of opinionated text. In *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, RACS '12, pages 37–42, New York, NY, USA, 2012. ACM.

[SHK$^+$14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SM15] Aliaksei Severyn and Alessandro Moschitti. Twitter sentiment analysis with deep convolutional neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 959–962, New York, NY, USA, 2015. ACM.

[Wid05] B. Widrow. Thinking about thinking: the discovery of the LMS algorithm. *IEEE Signal Processing Magazine*, 22(1):100–106, Jan 2005.

[WSH$^+$18] Yequan Wang, Aixin Sun, Jialong Han, Ying Liu, and Xiaoyan Zhu. Sentiment analysis by capsules. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1165–1174, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee.

[ZS14] N. Zainuddin and A. Selamat. Sentiment analysis using support vector machine. In *2014 International Conference on Computer, Communications, and Control Technology (I4CT)*, pages 333–337, Sept 2014.