**MTE546 Multisensor Data Fusion**

Analyzing Humor in Yelp Reviews using Neural Networks



Professor Melek & Eugene Li

Vincent McLoughlin 20575319

Sebastian Fiorini 20558540

March 29, 2019

## 1.0 Introduction

The goal of this project is to determine the ability of neural networks to understand abstract concepts such as humour. Natural language processing (NLP) is a common application of neural networks, so it is worthwhile to examine the underlying tools used to allow machines to understand human sentiment. Yelp reviews are freely available and can be downloaded in a json format for data science purposes [1]. This dataset contains over 6 million reviews and their associated data. One segment of this data includes the review text and how many times a review is upvoted as being cool, funny, or helpful. In this study, a neural network is trained and generated to determine how effectively it can estimate if a review is going to be upvoted ‘funny’. An upvote occurs when another user reads the review and marks it as ‘funny’.

Two different types of neural networks will be used to analyze humour and will be compared based on their effectiveness. First a Convolution Neural Network (CNN) is used, followed by an LSTM (Long Short Term Memory) Network. The development is done in Python using Keras with a TensorFlow backend. The results are compared to a paper from Stanford where a similar experiment is conducted to ensure the results are consistent with what others have found [2].

## 

## 2.0 Background Knowledge and Information

In the following section background knowledge is presented prior to explaining the implementation of the neural network. This is in order to understand the underlying reasons for the choices and difficulties faced during the implementation process. Concepts explained include text tokenization methods, natural language processing and sensor data fusion, recurrent neural networks, long short term memory networks, and convolutional neural networks.

### 2.1 Tokenization

In order to prepare the data to be fed into a digital neural network, the text input is converted into a representation the digital network understands and can work with. To do this a unique representation for a word must be established.

There are two popular ways this tends to be done. The first is achieved by building a set from the words seen in the training data by parsing through it before training. Once parsed, a unique identifier is assigned for each word in the set which becomes the initial layer of the neural network. The second approach, which is more sophisticated, vectorizes the word tokens to try and determine similarities between them. Word vectorization attempts to group words with similar meanings, for example, the words “frog” and “toad”, “king” and “queen”, “strong” and “stronger” share similar meaning. When using this approach, the vector is passed in to the network as opposed to the token value itself.

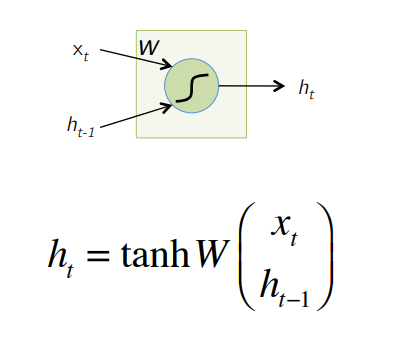
### 2.2 Neural Networks, Natural Language Processing, and Sensor Fusion

Neural networks have been applied to NLP with great success in recent years. Google’s speech recognition uses neural networks in their speech recognition software [3]. One of the most popular models recently has been the LSTM model, a type of Recurrent Neural Network (RNN), which is popular due to its ability to remember previous information. This enables LSTM networks to track information across several sentences, something other types of networks struggle with.

In both the LSTM Networks and CNNs, the neural networks are taking the data from the words and establishing previously unknown information based on the data extracted from the review. Once the words in the review have been converted into numbers using the tokenization method described in section 2.1, the neural network evaluates what combinations of words have a high correlation with funny reviews. From there it can build a model to predict if a given review will be funny or not. Using the data extracted from the words to determine this new information qualifies as high level data fusion.

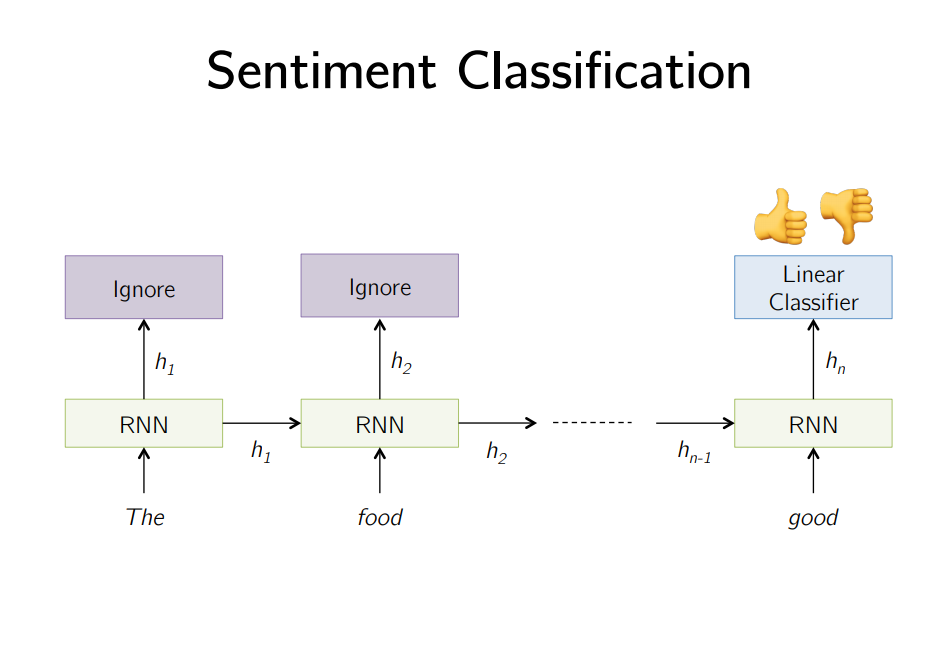
### 2.3 RNNs

Before looking specifically at LSTM Networks, it is useful to understand how RNNs work. Fundamentally, RNNs have memory and pass along information from previous states. This means RNNs are able to utilize information from both their direct inputs, as well as information from the recent past. A typical RNN node is shown in figure 2.3.1.



**Figure 2.3.1:** Typical RNN Node [4]

An example of this can be seen in figure 2.3.2. Note how input ht is passed along to different nodes in the RNN. In figure 2.3.2 the sentence “The food was good” is being analyzed to determine if the sentence is a positive opinion or a negative opinion.

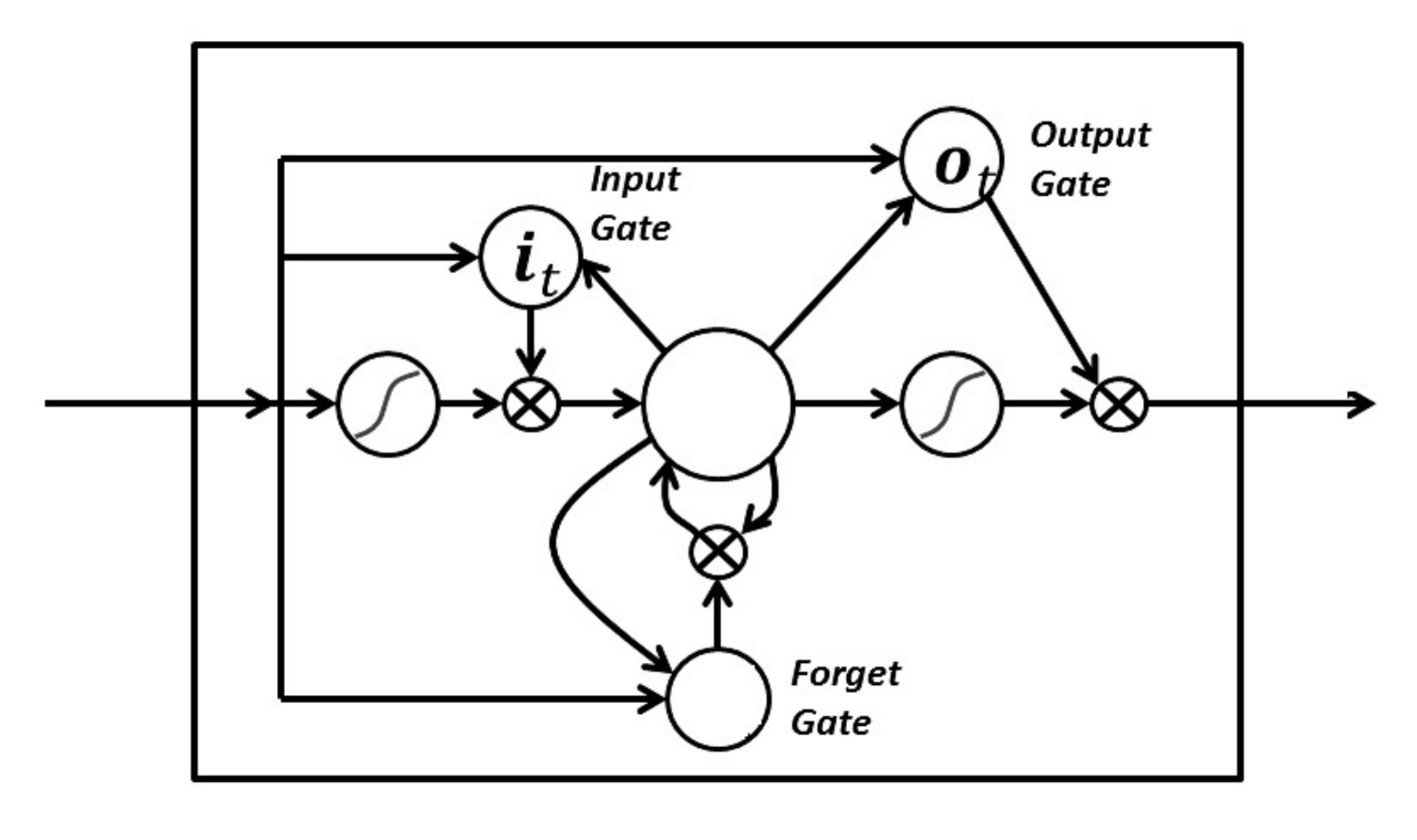


**Figure 2.3.2:** RNN Illustration [4]

Due to the fact ht is affected as it passes through each individual RNN node, it can propagate information through the network and affect the end result. This is particularly useful in sentences where the meaning changes throughout the sentence, so recording data word-by-word is useful.

### 2.4 LSTM Networks

LSTM Networks are a special kind of RNN that have a memory gate that can control if the node retains or forgets previous information. The node structure is slightly different from the typical RNN node. This can be seen in figure 2.4.1.



**Figure 2.4.1:** LSTM Node [5]

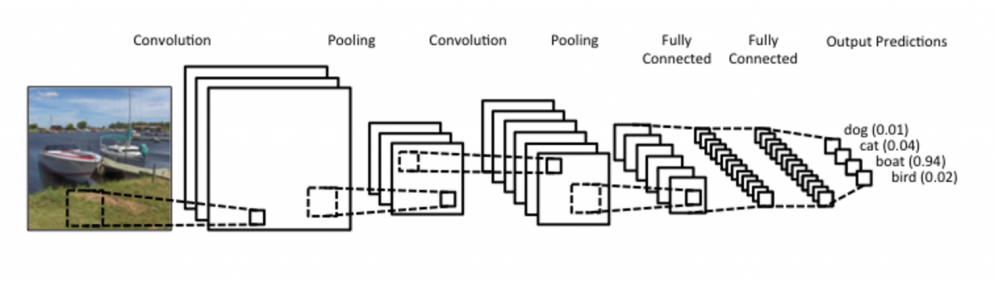
The LSTM node has an input gate, output gate, and a forget gate. While the input and output gates function similarly to how they do in a typical RNN, the forget gate is a new addition. The forget gate controls access to the nodes previously held information. The node saves previous information and uses it in its calculations. The results of these calculations can then be passed on to the rest of the network. In this way, information from some time ago can be retained and used if it is still relevant. The forget gate allows the node to discard information if it becomes unimportant, and allows newer, more relevant information to take its place. This may occur in scenarios such as when a new sentence is encountered. Old information could be useful but the beginning of a new sentence is more likely to assist the LSTM network with understanding the rest of the sentence. Access to the forget gate is controlled by the node so only more useful information will overwrite the old information.

### 2.5 Strengths and Weaknesses of LSTM Networks

LSTM networks are very good at remembering data from a previous datapoint, and can even distinguish between what information is worth remembering and what information is not worth remembering. This is particularly useful in NLP as the LSTM network can remember context from the start of the sentence as it moves toward the end of the sentence. This ability to combine words from different parts of the sentence assists it in understanding if the combination of words should be considered funny or not. Where the network is less successful is in overall classification. LSTM networks are not designed to excel at picking out notable features from a set of data points which may may hinder its ability to classify humour.

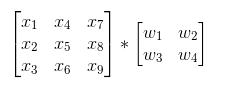
### 2.6 Convolutional Neural Networks

Typical Artificial Neural Networks work by summing the weights determined by the nodes as inputs, and passing them to an activation function. CNNs operate by taking the matrix representing the data and convolving this matrix with a filter. The resulting matrices are then pooled for their max value to highlight notable features. This process is repeated several times. Once the dataset has been downsized to its most relevant features, a fully connected neural network runs on the remaining data. A diagram of this process for an image can be seen figure 2.6.1.



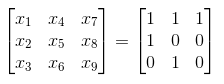
**Figure 2.6.1:** Convolution Process [6]

Let’s look at these steps individually to see how the apply to NLP, beginning with the convolution step. To actually convolve the inputs, a filter must be created so it can be convolved with the data matrix. This looks something like the case in figure 2.6.2. In a CNN the weights of the network form the coefficients of the filter. This allows one filter to be used for more than one set of data, so long as features being extracted are the same. This results in the following operation being carried out across all data points where xn represents the inputs and wn represents the weights of the nodes.



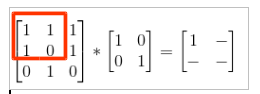
**Figure 2.6.2:** Convolution of Inputs and Weights

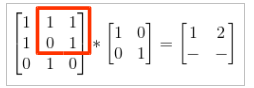
The convolution step is carried out across all sets of four inputs in the input data matrix. An example of this process is shown in figures 2.6.3 and 2.6.4.





**Figure 2.6.3:** Example of convolution of inputs and weights

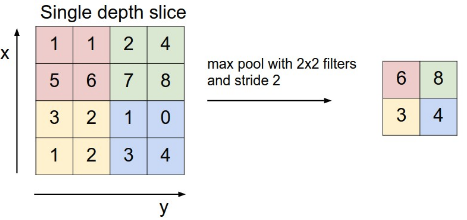




**Figure 2.6.4:** Example of Convolution Steps

One filter does not contain all the necessary weights to fully model and accurately predict the required output. To address this issue many filters must be learned and applied to extract the relevant features from the input. While this is very computationally expensive, it is still far less work than a fully connected network, as many inputs can share a few weights. This results in a sparse network where a feature is identified only by the inputs that are relevant to it.

Once data has been convoluted, it can be pooled (also known as downsampled) to decrease its size. Usually this is done by taking the largest number in a subset of a matrix as it best preserves relevant features. An example of this is shown in figure 2.6.5, stride here refers to the number of cells the filter will move across before being reapplied. This data reduction preserves the most important information while reducing the dimensionality.



**Figure 2.6.5:** Pooling Step [7]

The convolution and pooling steps are repeated until the dataset has been reduced to a manageable size that preserves its key features. From there a fully connected net can be run on the smaller dataset to determine a model.

The primary advantage of CNNs is that they are much less dense than fully connected networks, due to weights being shared among inputs. This makes them much faster than other forms of fully connected neural nets where connections would be required for all data points. They are still able to preserve relevant data across the convolution and pooling steps so they are very well suited for classification tasks such as the one attempted in this case, where classifying an overarching trend in a text is desired.

### 2.7 Strengths and Weaknesses of CNNs

CNNs are excellent at classification tasks. Their use of convolution to extract key features means that they can pick out the important ones, while max pooling highlights them and reduces the dimensionality of the data. The convolution operation also results in inputs only forming connections with nodes where they will be particularly impactful. These sparse connections make CNNs very fast as a result. Where CNNs struggle is in identifying the relationship between words. While CNNs are good at picking an overall impression out of a string of text, they have no method for remembering specific words, or remembering the value that specific word added to the humour of the review.

## 

## 3.0 Implementation of Experiments and Simulations

The following section explains different engineering choices and tasks that are completed in order to create a neural net to understand humour from Yelp reviews. Tasks and difficulties faced include assumptions made, data setup, tokenization, normalization, selecting loss and optimization functions, and an overview of the final selected models.

### 3.1 Assumptions

Three assumptions were made for this experiment:

1. All reviews upvoted ‘funny’ are considered equally funny, even if some ‘funny’ reviews have more upvotes than others.
2. A vocabulary of 10,000 unique tokens is used and assumed to be sufficient for understanding humour. This assumption is made due to a memory constraint explained in section 3.5.
3. It is assumed that a funny review has text that is considered funny in the first 100 words. This assumption is made due to a memory constraint explained in section 3.5.

### 3.2 Data Setup

To extract the necessary information from the training, the Yelp dataset is converted into a more efficient and convenient method for the neural network trainer to parse. A script is written in Python to parse the number of funny upvotes and review’s text into separate files from the review.json document provided by Yelp. Regular expressions are applied to each review’s text to remove punctuation and replace all words by spaces so that they can be correctly identified during the tokenization phase where words are matched to a unique token (number). At this point, there were two files, one with reviews with removed punctuation, and the other with the number of funny ‘upvotes’ received. To manage the file of reviews more easily, both files are split up into smaller chunked files of 5000 text reviews and 5000 funny ‘upvotes’. This approach is chosen over iterating the large files in chunks as it allows for easier neural network training data changes and running the software on smaller sample tests.

### 3.3 Generator Function Coroutine

Since the Yelp review data does not fit all at once in memory on the computer, the data has to be chunked and processed in parts when training the Yelp dataset. A generator function is made by continuously looping over all of the provided training data files and returning a set of training examples and their ground truths. This functionality is achieved using the ‘yield’ keyword to promote a routine in to a coroutine such that it remembers where it left off. Subsequent calls to the coroutine return the following batch of training data when called again as shown in the figure 3.3.1.

|  |
| --- |
| def **trainingGenerator**():  while True:  for i in range(len(filesText)):  # data is loaded from file from index i  # ground truth normalized (Explained in Section 3.3)  # review data is tokenized (Explained in Section 3.4)  yield(tokenizedReviews, normGroundTruth) |

**Figure 3.3.1**: Training Generator Pseudocode

### 3.4 Normalizing Data

To ensure that the data was bounded and fair, the number of funny reviews has to be normalized. A review that gets many upvotes may not be a lot funnier than one that only receives a few, so reviews should not be treated or weighed differently because of the number of times that it is upvoted funny. In the implementation here, the ground truth for either test case has their inputs normalized from 0 to 1 by iterating over the list of funny upvotes the review receives and checking if the number of upvotes is greater than the set threshold as seen in the figure 3.3.1.

|  |
| --- |
| normGroundTruth = np.array([1 if int(i) > 0 else 0 for i in dfSentiment]) |

**Figure 3.4.1**: Normalizing the Data for Number of Funny Upvotes Greater Than 0

### 3.5 Tokenization

For our experiment, the vectorization of words uses the GloVe library [8] which is an open source set of word vectors from Stanford’s NLP group. The GloVe library data consists of 400,000 words ranked based on that word’s popularity. For the study, the smallest GloVe set is used and has 50 vector dimensions. Due to limitations on the computers available to the team, the larger models with 100 dimensions or more were not used. Since the training and test data is so large, to simplify the design, instead of iterating through the training data and selecting the word tokens seen in reviews to be added to a bank of used words, the first 10,000 words from GloVe are used. This means that the model is limited to understanding 10,000 words, which to put in to perspective is the average vocabulary of an 8 year old [9].

Loading this tokenizer and building the word bank is done once at the start of the program before loading in any training data or building the network. Inside of the training generator, as shown in figure 3.5.1, a two dimensional array of tokens is made. This array is populated when parsing the text of each review and inserts the token at the index of the word in the review if the word is recognized. If the word is not recognized, the token 0 is used to show that the word is not known. Due to memory constraints on the group’s computer, a review’s length is limited to a length of 100 words. If a review is longer than 100 words, the data after that 100th index is ignored.

|  |
| --- |
| def **trainingGenerator**():  …  # Tokenizing each word for each review  tokenizedReviews = np.zeros((len(dfText), max\_words\_review))  for i, review in enumerate(dfText):  for j, word in enumerate(review.split(' ')):  tokenizedReviews[i][j] = word\_bank[word] if word in word\_bank else 0  ... |

**Figure 3.5.1**: Tokenizing the Words Used For Each Review

### 3.6 Filtering Reviews to Better Comprehend Humour

There are roughly 6.68 million reviews that are supplied by Yelp and 1.37 million of those are upvoted funny at least once. It is clear that most reviews do not end up being upvoted funny, but in analyzing some of the data that is being used for training, it is clear there is a large discrepancy in what can be considered funny and what is not considered funny. A funny review may not have been seen by a user which finds it funny and may not have any funny vote. Likewise, many comments on Yelp are upvoted funny when they are not actually funny.

A second dataset is produced where 50% of the reviews have 2 funny upvotes or more and the other 50% of the dataset consist of reviews with 0 or 1 funny vote. This second dataset, where the results are evenly split between reviews with 2 or more funny upvotes ends up being around 1.1 million reviews. The goal of this second split is to compare how effective the neural network is at determining humour when there is a even split of humour and non-humour. This selection of 2 upvotes or more is chosen because a review with multiple upvotes is more likely to actually be funny as opposed to a review with only 1.

### 3.7 Splitting Ratio of Training and Test Data

A important consideration when training a neural network is to go through a testing phase to validate that the neural network is properly ‘learning’ and not overfitting training data into its model. A split ratio, known as the ratio between training and testing data determines what proportion is used for training and what is used for test. A split ratio of 80/20 was selected for all test cases based on the suggestions found in the Amazon Web Services Machine Learning documents [10].

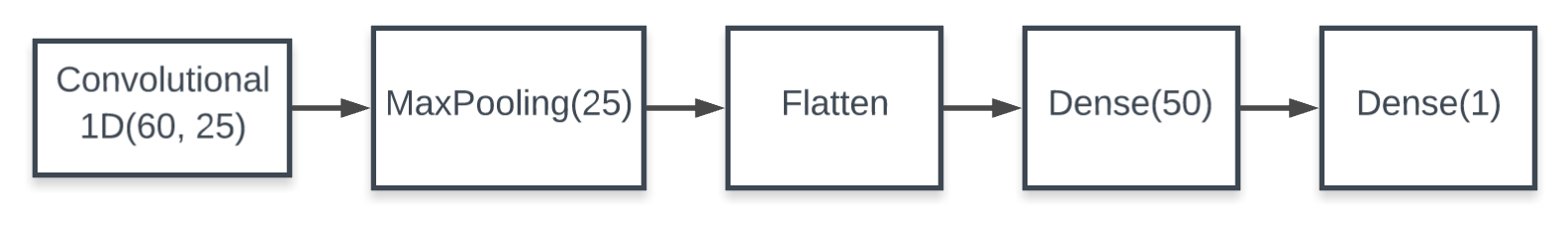
### 3.8 Selecting Loss Function and Optimizer For Training

The binary cross entropy loss function is selected as humour should be classified in a binary manner as reviews should be classified as ‘funny’ or ‘not funny’ and multiple states should not exist for the output neuron. This requirement for a binary classification eliminates many of the multi-class loss functions available in Keras. Another alternative loss function that can be used for classification is the mean square error (MSE) loss function but was not selected due to diminishing popularity. “When modeling a distribution, SE is bounded and the optimization is therefore more robust to outliers than minimization of cross entropy. In practice, however, cross entropy mostly leads to faster convergence and better results in terms of classification error rates. Hence, square error became less popular over the last years” [11].

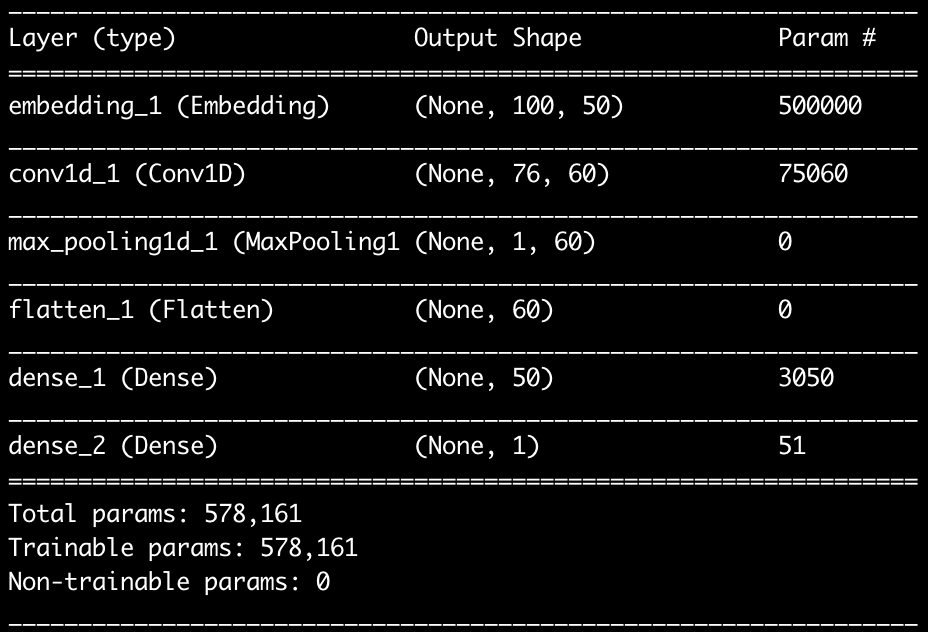
The adaptive moment estimator (ADAM) optimizer is selected due to its lower memory requirements over other popular optimizers. An alternative optimizer that is considered is stochastic gradient descent (SGD). The ADAM optimizer is favoured over SGD due to the lower memory cost and its relatively high performance without the need to tune the parameters of the optimizer. In the case of SGD, specifying the learning rate is critical to how the optimizer performs.

### 3.9 Convolutional Neural Network Setup

Figure 3.9.1 shows the final selected network model based off of convolutional network layers. Note that the 60, and 25 represent the filters and kernel size on the convolutional layer. The max pooling selects the 25 popular filters from the convolutional layer as shown in figure 2.5.5. To feed the max pool to the dense layer of 50 neurons the type of the network has to be casted to work with this dense layer and is flattened. The final dense layer of size 1 classifies whether or not the input text is funny or not by outputting the probability of the text being recognized as funny. These selections were made based on empirically testing the performance of the network as well as avoiding allocation errors due to the size of the model becoming too large to run on the team’s laptops due to memory constraints. A summary of the model used is shown in figure 3.9.2.



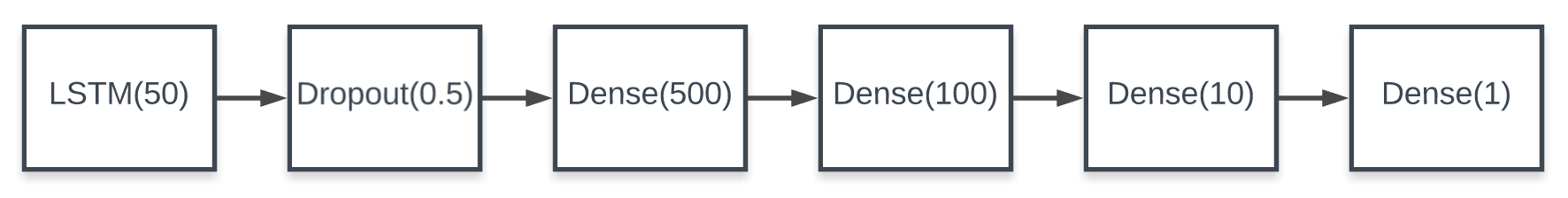
**Figure 3.9.1**: Neural Network Model for Convolutional Neural Network



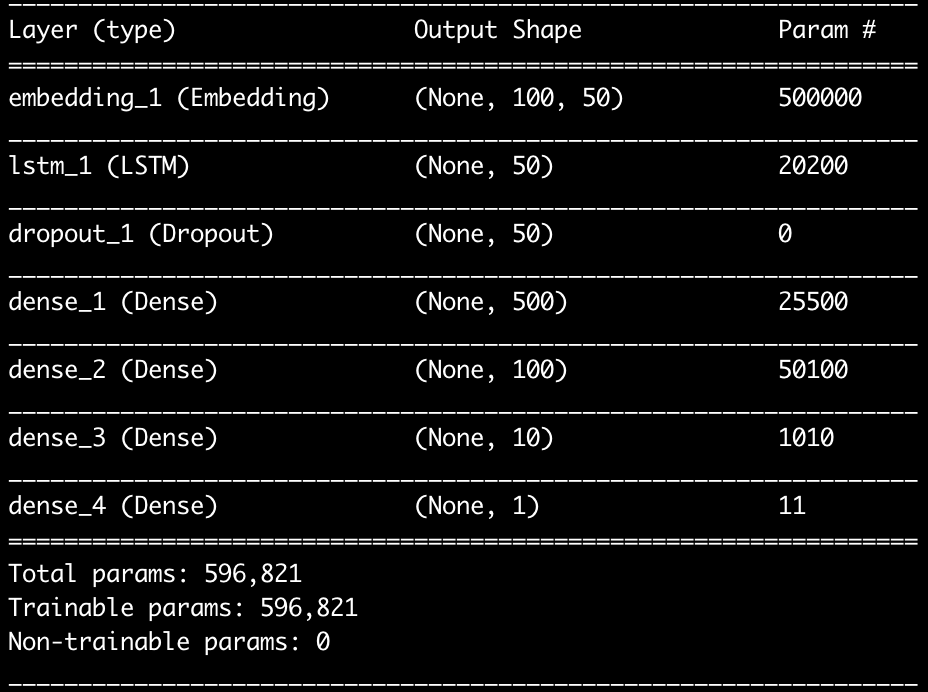
**Figure 3.9.2**: Summary of Convolutional Network Model

### 3.10 LSTM Network Setup

Figure 3.10.1 shows the final selected network based off of long short term memory layers. Note that the 50 parameter in the LSTM layer represents the dimensionality of the embedding vector. Explained in the tokenization section 3.5, a dimensionality of 50 is chosen due to memory constraints on the group’s computers. The drop out rate of 0.5 sets 50% of inputs during training to 0 to help avoid overfitting. This drop out prevents overfitting because adding neurons restricts the size of the solution space the model considers valid. This leads to the model not being able to generalize to data points not in the training dataset. By dropping half of the neurons, the risk of overfitting is greatly reduced. A drop out rate of 50% is picked as an initial starting point. The ensuing results are good enough that the value is kept during further testing. From the LSTM layer, three additional hidden dense layers of size 500, 100, and 10 are added. The final output layer is a single neuron, like in the convolutional example, where the output shows a percentage of how funny the input text is perceived to be. A summary of the LSTM based model is shown in figure 3.10.2.



**Figure 3.10.1**: Neural Network Model for Convolutional Neural Network



**Figure 3.10.2**: Summary of LSTM Network Model

## 

## 4.0 Results with Discussion of Key findings

After running the LSTM network and CNN on the different datasets, the results in table 4.0.1 are obtained.

**Table 4.0.1:** Results of Neural Networks

|  |  |  |
| --- | --- | --- |
|  | Accuracy (%) | Loss |
| CNN (Raw) | 80.12 | 0.4678 |
| CNN (Even Funny Split) | 71.4 | 0.5593 |
| LSTM (Raw) | 79.6 | 0.5017 |
| LSTM (Even Funny Split) | 64.91 | 0.6489 |

From the results, the CNN outperforms the LSTM network, and both models are more accurate when using the unfiltered data as opposed to the evenly split data. The output of the model returns the likelihood of something being upvoted funny on Yelp. The raw CNN only returns results in the range of 10-35%. The CNN model with an even funny split returns a much broader range of likelihoods, covering a range from about 10% to 90%.

Examples can be seen in tables 4.0.1 and 4.0.2. The results are generated from a web app created to evaluate the detected humour in Yelp reviews. Table 4.0.1 evaluates a review which is not funny and Table 4.0.2 evaluates a review which is funny.

**Table 4.0.1:** Example of a Review Which is Not Funny Using CNN

|  |  |
| --- | --- |
| Input Text | This was good. I enjoyed the restaurant and I would recommend it to anyone who likes good food. |
| Percent Likelihood of Being Upvoted Funny (Unfiltered) | 10.5427 % |
| Percent Likelihood of Being Upvoted Funny (Funny Split) | 18.9112 % |

**Table 4.0.2:** Example of a Funny Review Using CNN

|  |  |
| --- | --- |
| Input Text | The waiter and waitress forgot that I hadn't left and I was in the washroom and they locked the place up. I had to call 911 to open the door for me. They gave me a grilled chicken breast as a apology, quite disappointed with their choice of apology, would've appreciated steak more. After the 1 hour of torture inside that restaurant, I deserve that piece of meat. |
| Percent Likelihood of Being Upvoted Funny (Unfiltered) | 21.9937 % |
| Percent Likelihood of Being Upvoted Funny (Funny Split) | 88.8706 % |

### 4.1 Filtered vs Unfiltered Data

The filtering of data evidently has a major impact on the network and its ability to classify reviews. Both networks are trained to maximize accuracy. Using a dataset with 79% of reviews not being upvoted funny, the easiest way for a network to model humour is to guess that the review is not funny. This is what the network does when trained on the raw dataset. It moves toward classifying all reviews as not funny because it has been instructed to optimize accuracy. From the experiments with evenly split data (half reviews voted funny, the other half not funny), it is difficult for the network to achieve a comparable degree of accuracy to the unfiltered data (~80%). Once the training classes are balanced, a more useful model is obtained that makes intelligent choices on the data rather than by the disproportion of classifications in the data.

### 4.2 Comparing Model Accuracy

The accuracy achieved by the neural network is somewhat lower than that generated by the Stanford paper. The Stanford paper manages to achieve 83% CNN accuracy, about 11% higher than what was achieved here, while they achieved LSTM results of about 79% with a balanced dataset, 14% higher than what was achieved here [2]. There are several potential reasons for this. One large factor for this difference between the two models is due to the word bank selection. The word bank that is generated does not select words from the training data, instead it only considers the first 10,000 words in the GloVe dataset. The GloVe contains a list of words ranked by their popularity, many of which are likely are not in the domain of food reviews. A more suitable method is to build a word bank based on the words in the training set, which would effectively expand on the vocabulary the model can understand as was done in the Stanford paper. A vocabulary larger than 10,000 words would also improve the results here, as the network would be better able to recognize less common humorous words. Another reason for this discrepancy in accuracy could be that the Stanford paper considers only reviews with greater than 2 funny upvotes when normalizing data but the unfiltered model that is generated considers reviews with greater than 0 funny upvotes to be funny. This likely introduces noise in to what is considered funny as all reviews that are marked funny are marked equally funny (as per the assumptions described in section 3.1).

Lastly, in general, humour is subjective to the reader and many reviews exist on Yelp that are very funny which haven’t been seen by others and similarly there are reviews which aren’t funny and received upvotes for being funny. The network is working off imperfectly labelled data, but the network does get the general idea of humour from the large dataset it has available.

### 4.3 CNN vs LSTM Networks

The CNN achieves better results than the LSTM Network. The CNN may outperform the LSTM due to the nature of the task. Humour detection is a classification problem. The goal of any method in this task is to take the dataset as a whole and determine if it is humorous or not. The specific context of each word might be useful but it is not essential so long as the network can determine the overall impression of the review. Determining these impressions is a task CNNs are good at. Their use of convolutional filtering with the network weights, combined with max pooling the results highlights the overall impression of the review very well.

LSTM networks, while good at remembering context and tracking specific meaning over sentences is not as well suited to classification tasks. LSTM networks are not any better or worse at feature extraction than a typical ANN so they struggle to get an overall impression of the humour of a review to the same degree a CNN does. LSTMs are more designed to pick up on previous words to help with understanding the context of later words, but this is not as useful as the convolution/pooling method of the CNN when trying to detect humour.

## 

## 5.0 Conclusions and Recommendations

Two different neural network models are constructed to try and understand human humour using Yelp’s dataset. To process the data, the review text is tokenized from a large generated word bank and the training data is normalized. The CNN achieves a peak accuracy of 80.12% and the LSTM achieves a peak accuracy of 79.6% on the entire Yelp dataset when intelligently considering the likelihood of any review being voted funny. On a evenly split dataset, the CNN achieves a 71.4% accuracy and the LSTM achieves a 64.9% accuracy. The accuracy results obtained are worse than previous studies but there are several reasons for this including a non-tailored word vector, and a small vocabulary.

The CNN is found to perform better than the LSTM network due to its superior performance in classification tasks despite LSTM network’s designed focus on text and language processing. CNN functions based on learning filters and applying them to extract the desired data. This method is found to be more effective at detecting humour and perceiving sentiments.

Recommendations include using hardware with more memory as the model is constrained by the allocations during the model generation. With more memory, GloVe can be used with denser models that have larger vector dimensions resulting in presumably more accurate humour prediction from a review’s text. Another improvement includes generating the word bank and tokens from word frequency lists generated from the training data. Lastly, LSTM networks have fallen out of favour to new, more performant attention networks [12] for language processing. It would be useful to analyze these networks and see if they outperform CNNs for this application.

## 

## 6.0 References

1. Yelp. Yelp Dataset Documentation. yelp.com, 2019. [Online] Available: <https://www.yelp.com/dataset/documentation/main>
2. Oliveira and Rodrigo. “Humour Detection in Yelp Reviews”. [Online] Available: <https://cs224d.stanford.edu/reports/OliveiraLuke.pdf?fbclid=IwAR2K9FCoVfVjbggl-UDS94bkZRiHLUBBGechne7DjGzLcw6TdF7yiyc9aGc>
3. Kate Allen. How a Toronto professor’s research revolutionized artificial intelligence thestar.com. April 17, 2015. [Online] Available: <https://www.thestar.com/news/world/2015/04/17/how-a-toronto-professors-research-revolutionized-artificial-intelligence.html>
4. Arun Mallya. Introduction to RNNs. [Online] Available: <http://slazebni.cs.illinois.edu/spring17/lec02_rnn.pdf>
5. N. Donges, *LSTM with Three Gates*. Medium.com, 2018. <https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5>
6. U. Karn. *A Simple ConvNet*. 2016. <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>
7. Andrej Kaparthy. Convolutional Neural Nets. [Online] Available: <http://cs231n.github.io/convolutional-networks/>
8. Jeffrey Pennington. Richard Socher, Christopher D. Manning. Global Vectors For Word Representations. [Online] Available: https://nlp.stanford.edu/projects/glove/
9. R.L.G. Lexical Fact. May 29, 2013. [Online] Available: <https://www.economist.com/johnson/2013/05/29/lexical-facts>
10. AWS. Splitting the Data into Training and Evaluation Data. [Online] Available: <https://docs.aws.amazon.com/machine-learning/latest/dg/splitting-the-data-into-training-and-evaluation-data.html>
11. Pavel Golik, Patrick Doetsch, Hermann Ney. Cross Entropy vs. Squared Error Training. [Online] Available: <http://books.jackon.me/Cross-Entropy-vs-Squared-Error-Training-a-Theoretical-and-Experimental-Comparison.pdf>
12. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. Attention Is All You Need. [Online] Available: <https://arxiv.org/pdf/1706.03762.pdf>