Bsc(Honours) Degree in Computer Science

Final Year Project

By
Sam Toguri

Project unit: PJE40
Supervisor: Ella Haig

May 2020

1. Abstract

This paper will outline the planning, development and testing of a sentiment analysis application that is trained on twitter data. We propose a Long Short-Term Memory (LSTM) recurrent neural network that is trained on GloVe embeddings of twitter data. The model will then be deployed onto a Node.js server application and be accessible in the browser. This means users will be able to take their datasets that they want to be analysed, simply upload them through the client application, wait a few minutes (depending on the size of the dataset) and view the results. This is an extremely powerful application because if successful, it would give business without any experience in deep learning or programming at all, the option to quickly analyse the data that they have gathered on their business in order to gather a general public opinion on whatever it is in their dataset.

2. Contents

3. Introduction

The advent of online social networks has produced a crescent interest on the task of sentiment analysis for short text messages (Go et al., 2009). However, sentiment analysis of short text data, such as Twitter messages (tweets), is difficult because of the limited amount of context in this type of data. Therefore, to effectively solve this task it requires methods that extract deeper and more detailed information from the sentence/message than models such as bag-of-words or even continuous bag of words. In order to compensate for the lack of contextual information in the tweets, it is more suitable to use methods that can exploit prior knowledge from large sets of unlabeled texts.

3.1. Node.js & Express framework

In 2009, Node.js came along. Node.js took V8, Google Chrome's powerful JavaScript engine, out of the browser and enabled it to run on servers. However, with the vanilla Node.js APIs, developers often have to write a lot of boilerplate code. Express exists to cut down on this boilerplate code by simplifying the APIs of Node.js and adding helpful new features (Hahn, 2016). The V8 JavaScript engine that Node.js is based on is fast and Node.js makes asynchronous coding easier, meaning you get faster execution of code while avoiding errors due to multithreading.

3.2. GloVe & Feature Extraction

Feature extraction is a vital part of the pre processing stage because you cannot simply input a word into a classification algorithm, it needs to be represented numerically first. GloVe, short for Global Vectors for Word Representation, is a method of feature extraction developed at Stanford in 2014. It is a weighted least squares model that trains on global word-word co-occurrence counts and is therefore using the available data in a more efficient way. Models like bag-of-words do not deal with context in a very efficient way because it does not maintain the order of the words and their proximity to each other. The model produces a word vector space with meaningful substructure, as evidenced by its state-of-the-art performance of 75% accuracy on the word analogy dataset (Pennington et al, 2014).

3.3. Long Short-Term Memory Recurrent Neural Network

A recurrent neural network (RNN) is an extension of a conventional feedforward neural network, which is able to handle a variable-length sequence input. The RNN handles the variable-length sequence by having a recurrent hidden state whose activation at each time is dependent on that of the previous time (Chung et al., 2014). LSTM is, again, an extension of RNN that allows it to store more memory about previous and future iterations by introducing the cell state variable, this is very useful as it means LSTM are capable of learning long term dependencies, which vanilla RNNs are not. In scenarios such as the sentiment analysis of this sentence "I lost my phone last night …(random text in between)... now im slightly better." from the second part of the sentence you would think the sentiment was positive, but the first part is definitely negative. Therefore, we need the context of the first part of this sentence to determine the polarity of later parts and it is often the case that the relevant previous information is quite far behind where it is needed.

3.4. Web Based Sentiment Analysis Application

This report details the design and implementation of a web-based sentiment analysis application for Twitter interactions that achieves the categorisation of sentiment polarity of tweets into three categories: positive, neutral and negative, through the use of the Global Vector (GloVe) feature extraction method and a RNN based on the LSTM architecture. Users will be able to upload CSV files that will then be input into the sentiment analysis system and the results of the algorithm will be

returned to the client and visualised for the user to view. Twitter data is inherently lacking in usable and valuable data to discern the context of tweets. However, by improving upon other feature extraction models and inputting GloVe embeddings into the LSTM we should already see an improvement in the accuracy of predictions made by the model. Furthermore, LSTM are recurrent models and are therefore inherently good at making predictions on sequential data which is why it has been chosen for this project. Finally, there are no open-source, online applications for performing sentiment analysis on large datasets, in particular social media interactions. Due to the insight that can be gained from the immense amount of data generated by social media

4. Aims

To produce an implementation of a sentiment analysis system for classifying tweets as positive, negative or neutral and that has the added properties of being web-based and employing contemporary deep learning techniques through the following steps:

- The solution should be able to be accessed via internet connection through the use of Node.js technology with the Express framework and should provide users with a clear and intuitive user interface.
- The solution should achieve the preprocessing required of the data in an efficient manner while still retaining enough meaning and context necessary for the feature extraction stage to produce rich and meaningful vector representations of the words.
- The solution should achieve the classification of data into three categories, positive, negative and neutral through the use of an LSTM recurrent neural network. This way I will be utilising the property of recurrent neural networks that perform very well with sequential data, which text data is.

5. Objectives

5.1. Web App

This objective is to produce a web application by using the Express (Expressjs.com, 2020) framework on top of the Node.js (Node.js, 2020)technology to implement a simple application program interface (API) and server that will allow the user to get sentiment predictions from a pre-trained LSTM model. The web app system will be based on the client-server architecture, with the actual sentiment analysis program being run on the backend.

5.2. Preprocessing & Feature Extraction

The objectives for the preprocessing stage are to remove all meaningless and noisy data from the corpus and to represent each word in the corpus as a vector using GloVe embeddings, so that the final vectors that are produced are the best possible representations of the words that they are. This will be done by removing stopwords, emojis, numbers, urls and consecutive characters of more than three e.g. ‘eeee’, then the data will be part-of-speech (POS) tagged and lemmatised. Finally, once the lemmas have been produced the text data will be vectorised using GloVe feature extraction.

5.3. Long Short Term Memory RNN

The classification algorithm itself will be an LSTM, implemented using Tensorflow, an end-to-end open source platform for machine learning (TensorFlow, 2020). Other types of popular classification algorithms, such as Naive-Bayes and MaxEnt, obtain a maximum accuracy of ~85% and it is believed that the closest you can ever get to 100% accuracy is 93% because of the intrinsic ambiguity of text

data. Therefore, the aim is to achieve accuracy >80% using my methods of using GloVe embeddings as the input to the LSTM.

6. Literature Review

Sentiment analysis, or opinion mining, can be defined as automating the process of extracting attitudes, opinions, views and emotions from text through Natural Language Processing (NLP). The fundamental problem in sentiment analysis is categorization of sentiment polarity (LK-W, J-C, Y-L & K, 2015), this is categorising a piece of text into either positive, negative or neutral sentiment polarity. Sentiment analysis is replacing traditional and web based surveys conducted by companies for finding public opinion about entities like products and services (Indurkhya & Damereau, 2010). Sentiment analysis is of great value for business intelligence applications, where business analysts can analyze public sentiments about products, services, and policies (Funk, Li, Saggion, Bontcheva & Leibold C, 2008).

Text data must be preprocessed to remove incomplete, noisy and inconsistent data. People often use a typing vernacular such as repeating letters, '...looooonng', to add emphasis to their meaning. (Hemalatha, Saradhi Varma & Govardhan, 2012) performed a study into finding the necessary information through preprocessing of Twitter reviews in order to categorise the sentiment of each review. They performed this preprocessing by: removing urls, filtering out words with more than 3 repeated letters, removing questions words such as 'what' or 'who', removing special characters like '{}/?' and the removal of retweets which is when someone reposts another users post on their own page. They concluded that this was an efficient method for preprocessing twitter data in preparation for any machine learning algorithm. However (Asghar, Khan, Ahmad & Kundi, 2014) show that higher accuracy is achieved when you lemmatise the text as well. Lemmatisation uses vocabulary and morphological analysis of the given word and tries to remove inflectional endings, thereby returning words to their dictionary form (Balakrishnan & Lloyd-Yemoh, 2014). Also, due to the fact that lemmatisation takes into account the POS tag attached to each word it much more accurately returns words to their dictionary form and removes inflectional endings than stemming can.

Feature extraction process takes text as input and generates the extracted features in any of the forms like Lexico-Syntactic or Stylistic, Syntactic and Discourse based (Das et al, 2008). There are many methods of feature extraction, the most generic is the bag-of-words (BOW) model. This involves creating a list of unique words in the text corpus called vocabulary. Then we can represent each sentence or document as a vector with each word represented as 1 for present and 0 for absent from the vocabulary ("Machinew Learning — Text Processing", 2018). Another, more recent, method of feature extraction is called Global Vectors for Word Representations (GloVe). This method was invented in Stanford by Pennington et al. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space (Pennington et al, 2014). GloVe feature extraction combines ideas and methods from multiple styles of feature extraction, namely window based methods like the skip-gram model and also that it is dependent upon co-occurrence probabilities.

Vanilla recurrent neural networks are plagued by vanishing gradients problem during training which is when long term components go exponentially fast to norm 0, making it impossible for the model to learn correlation between temporally distant events (Pascanu, Mikolov & Bengio, 2013). LSTM is designed to overcome these error back-flow problems. It can learn to bridge time intervals in excess of 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag capabilities (Hochreiter & Schmidhuber, 1997). The most important elements of LSTM cells are

– cell state – the state of the cell passed in sequence after to the next steps,
– forget gate – the gate that decides what information should be omitted,
– input gate – a gate that decides what should be forwarded to the next activation (Nowak, Taspinar & Scherer, 2017).

(Pang & Lee, 2004) present a very wide range of existing work in sentiment analysis. The authors describe existing techniques and approaches for an opinion-oriented information retrieval. However, they do not describe many studies on opinion mining that consider blogs and even much less addressed microblogging. Conversely, there have been more recent studies that look much closer at social media interactions, in particular Twitter. (Pak & Paroubek, 2010) proposed a model to classify tweets as neutral, positive and negative. They created a corpus of tweets that used emoticons using Twitter API. Using that corpus, they created a sentiment classifier based on the multinomial Naive Bayes method that employs features such as Ngram and POS-tags. However, the training set they used only used emoticons so it was less efficient. (Go and L.Huang, 2009) proposed a solution for sentiment analysis for Twitter data by using distant supervision, in which their training data consisted of tweets with emoticons which served as noisy labels. They build models using Naive Bayes, MaxEnt and Support Vector Machines (SVM). Their feature space consisted of unigrams, bigrams and POS. They concluded that SVM outperformed other models and that unigram were more effective as features.

7. Methodologies

7.1. Project Development Model

Choosing a software development model is an important process while designing a software artifact as it is this model that will guide you through your deliverables and implementation of the whole application. This section will analyse the pros and cons of implementing the most prominent of these in relation to the software artifact that I will be building. Agile development has not been considered as a model that is strongly geared towards benefiting people on team based projects and I will be conducting the planning, implementation and testing alone.

7.1.1. Chaos model

The Chaos model is a software development strategy based on the chaos theory. The main strategy is to prioritise and complete tasks based on their importance, which is measured using the following guidelines:

- A task is an incomplete programming task.

- The highest priority issue is a combination of big, urgent, and robust.
    - Big tasks are the ones whose solution provides working functionality.
    - Urgent tasks are time dependent and may have adverse consequences if not completed as soon as possible.
    - Robust tasks are ones able to be tested and their completeness measured when resolved. Developers can then safely focus their attention elsewhere.
- To resolve means to bring it to a point of stability.

The chaos strategy resembles the way a developer would work through a project and focuses heavily on the actual development of the software and not on managing schedules and documentation. Therefore, due to my project being a solo undertaking this could be a very useful model because the organisational aspects that focus on team projects do not apply to the development of this software. Furthermore, this model gives a structured way to measure the importance of each part of the software implementation down to individual lines of code which should result in better structure and efficiency during the implementation stage.However, the lack of organisational structure and importance on scheduling might cause problems while meeting some of the preliminary deadlines, e.g. the February progress report.

7.1.2. Waterfall Models

The waterfall model is dependent on following steps in a linear order that ensure the project is always moving forward. If your project has followed the steps of the model, making substantial progress with the development and is then faced with an unexpected issue, of which there are many in real life scenarios, that requires a change in the scope or goals of the software being developed, going back and updating previous steps won't be easy. This is one of the biggest disadvantages of the waterfall method, that it lacks flexibility. Another property of the waterfall method that is usually thought of as a drawback is that it doesn't focus massively on the end user of the software. However, for the purposes of this project it could be seen as a strength of the waterfall model because of the nature of the software artifact being produced the end user does not need to be involved with the development process.

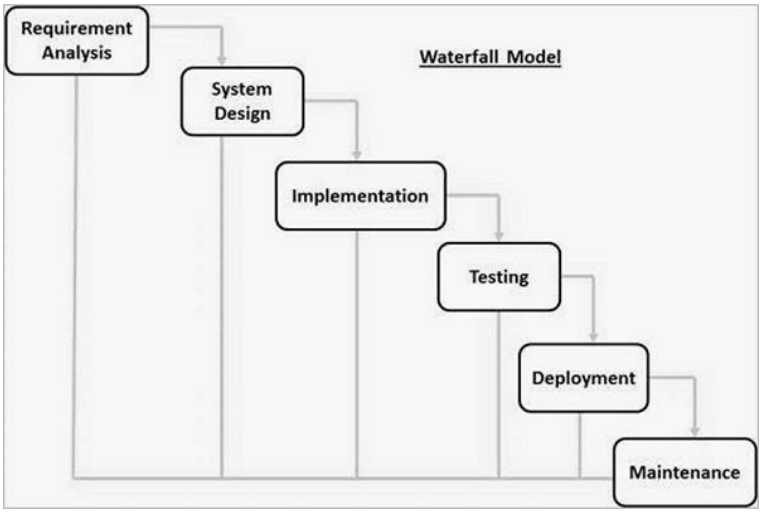

Fig 1. The Waterfall Model (ResearchGate, 2020).

7.1.3. Iterative Models

The iterative model starts with the design and implementation of just a small part of the software, and you then complete multiple iterations, hence the name, with each iteration adding a small amount of

functionality until you have a finished software artifact. This means that the design stage happens iteratively too and not before you start implementing the software. Therefore, you can easily make the necessary changes to accommodate unforeseen problems. However this model requires rigorous scheduling and management to ensure that all deployables are delivered on time. This could cause the quality of the code being implemented to suffer due to spending too much time on keeping a strict and updated schedule. Another strength of this model is that it will result in you creating basic prototypes of the software very early on in the development process, which can give you a lot of information going into future iterations.

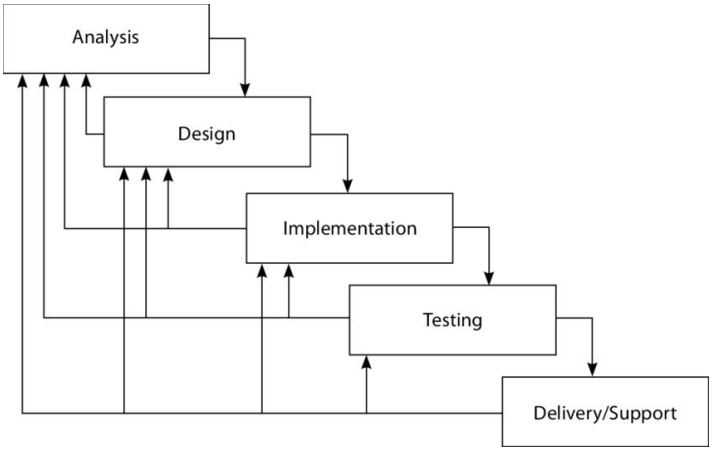

Fig 2. Iterative Model (ResearchGate, 2020)

7.2. Decision and justification

The chaos model provides structure and effective planning for the actual implementation of individual component pieces of the software artifact, which could result in higher quality code. However, it doesn't take into account the planning and time management that is required for the creation of design documentation, which is an equally important task. Especially for this project, where producing a good report is arguably the most important element. Therefore, the chaos model does not provide enough benefit to outweigh the potential pitfalls in its design and it will not be the selected model.

The waterfall model solves the problem of the chaos model, in that it does not offer enough structure outside of the implementation process, by making the members of a development team follow a linear and rigid set of instructions from project inception until deployment and maintenance. Ironically, this rigidity produces its own set of issues that seriously hinder the viability of the waterfall model. It is extremely difficult to go back and make changes to the requirements or other design documents because you will have already come a long way and it can mean having to alter everything you have implemented. For this reason the waterfall method will not be selected as although it provides clear organisational structure, for the intent of this project I will most likely need to go back at points and make alterations to the design documentation which would be made much more difficult and disruptive using this model

An iterative model provides the best parts of both of these prior approaches. It gives developers the option to freely alter and update all elements of the design documentation as the software evolves into its final form throughout each iteration. Also it helps to ensure that the code is

as high quality as possible because you are only focusing on implementing small parts of the software at a time so you can be more meticulous. For this project in particular this model allows me to plan my time flexibly which is important because of all the other projects I will have running concurrently with this. Although it will require that I stay very organised and on top of the deliverables, this model will be the best fit for this project and offers me the best chances at producing a good software artifact.

8. Requirements

The requirements have been based on what is required of a sentiment analysis system aimed at analysing tweets and what we can add to it by implementing it in a web based platform. The MoSCoW method has been chosen to prioritise the requirements, as this will provide clear direction and idea of the finished system, given the limited development time.

8.1. Functional Requirements

8.1.1. Must Have

- **Frontend -** The client interface must send Http requests to the backend.
    - This must be done using the Node.js technology.
    - This function must send the submitted file to the server for processing.
    - This function must also handle the response from the server, containing the results of the sentiment analysis.

- **Backend -** An API must be implemented so the frontend can interact with the backend.
    - This API must listen for HTTP requests on only one path, *'/submit_data'*.
    - It must validate the format of the file received from the client, and if not valid then it must respond with an error.
    - Upon validation of data the API must control the execution of functions that will prepare and analyse the sentiments in the data.
    - The API response must contain the results of the function call to the LSTM.

- **Preprocessing -** The system must be able to process the raw data, and make it ready for input into the classification algorithm.
    - Emojis must be removed anywhere that they occur within the text of a tweet.
    - Urls must be removed anywhere they occur within the text of the tweet.
    - Each word in the corpus must have its part-of-speech (POS) identified and paired with the word such that for every tweet that is input, each token within that tweet is pos-tagged and placed back into the tweet. A pos-tagged tweet must look like below:
        - $\{(token_1, POS_1), (token_2, POS_2),_{...}, (token_n, POS_n)\}$
    - For every $(token, pos)$ pair in the corpus, the token must be returned to its dictionary form, and have inflectional endings removed based on their pos-tag, via lemmatisation. For every $(token, pos)$ input, the output must be a single corresponding lemma.

- **Feature Extraction** - The tweets must be converted from text into feature vectors so that the data can be interpreted and processed by the classification model.

    - The GloVe, aggregated global word-word occurrence matrix method for obtaining word embeddings must be used.
    - This process must occur only after the preprocessing has taken place and before the sentiment analysis algorithm begins.

- **Sentiment Analysis** - The system must contain a method for identifying the sentiment polarity of tweets.
    - This must be done through the use of long short term memory recurrent neural network.
    - The LSTM must classify the tweets into one of three sentiment polarities:
        - Positive,
        - Neutral,
        - Negative.
    - The input must be the vectors obtained by the Feature extraction stage.
    - The output must be a prediction of the sentiment polarity for every input tweet.

8.1.2. Should Have

- **Data visualisation -** The backend application should be able to create a graph that will display the categorical accuracy metric over time of the whole analysis as well as the loss over time. The loss should also be able calculated and displayed in a graph.

- **Preprocessing** - requirements that are important but not essential to the preprocessing stage.
    - Should be able to remove all numbers from the tweets.
    - Should be able to shorten tokens with characters that are repeated more than three times, to only three repetitions. For example, “juuuuust” should become “juuust”.

- **Sentiment analysis** - requirements that are important but not essential to the functionality of the LSTM.
    - Multiple, more than an embedding, LSTM and output layer to improve the accuracy of the predictions.
    - The model should contain at least 2 LSTM layers.
    - The output layer (densely connected) should be wrapped in a time distributed layer that will apply the output layer to each temporal slice of input data (time steps), this has the potential to greatly improve the accuracy of predictions.

8.1.3. Could Have

- **Dockerisation** - the whole project could be split into several docker containers that would package everything necessary to run the project: dependencies, software versions etc., into containers that can be run on any machine.

- **Sentiment analysis** - the RNN that will be implemented could be a bidirectional LSTM, meaning that the first recurrent layer in the network could be duplicated so that there are now two layers, then providing the original input as input to the original first layer and providing a

reversed copy of the input sequence to the copy of the layer allowing you to traverse a tweet from both ends of the sentence.

8.2. Non-Functional Requirements

- **Training Data -** The training data must be in a certain format before it is input into the system, this is as follows:
    - Files that are input into the system must have '.csv' file type.
    - The data inside the file must have the following columns in the specified order:
        - Each row = (ID, sentiment polarity, Tweet)
    - Sentiment must be one of 3 values 'positive', 'neutral' or 'negative'.
    - Id must be a unique integer identifier.
    - Tweet must be a string containing the text of the tweet. This value can include emojis, numbers and urls.

- **New User Data -** Any data that is inputted into the system by the user must follow the same format as the training data in order for the system to be able to read the correct columns.

- **Frontend -** The frontend application must have a GUI that allows users to upload files and to view the results .
    - The application must alert the user that only .csv file types are accepted if a non csv file is attempted to be uploaded.
    - The website will be responsive(Usable on multiple different devices). Testing will be done using the responsive tool built into Google Chrome.

- **Backend -** input validation and data visualisation must happen in the backend/server application.
    - If there are not 3 columns in the correct order (ID, sentiment, tweet) then the server must respond with an error message for the user.
    - If the sentiment column does not contain only the values 'positive', 'negative' or 'neutral' then the server must respond with an error message for the user.
    - If each row in the ID column is not a unique integer value then an error message must be returned to the user.
    - The server should present the results of the sentiment analysis to the user in a graph using the Chartjs library.
    - The server should have minimal downtime and try to be up 24/7.

9. Design

9.1. Use cases

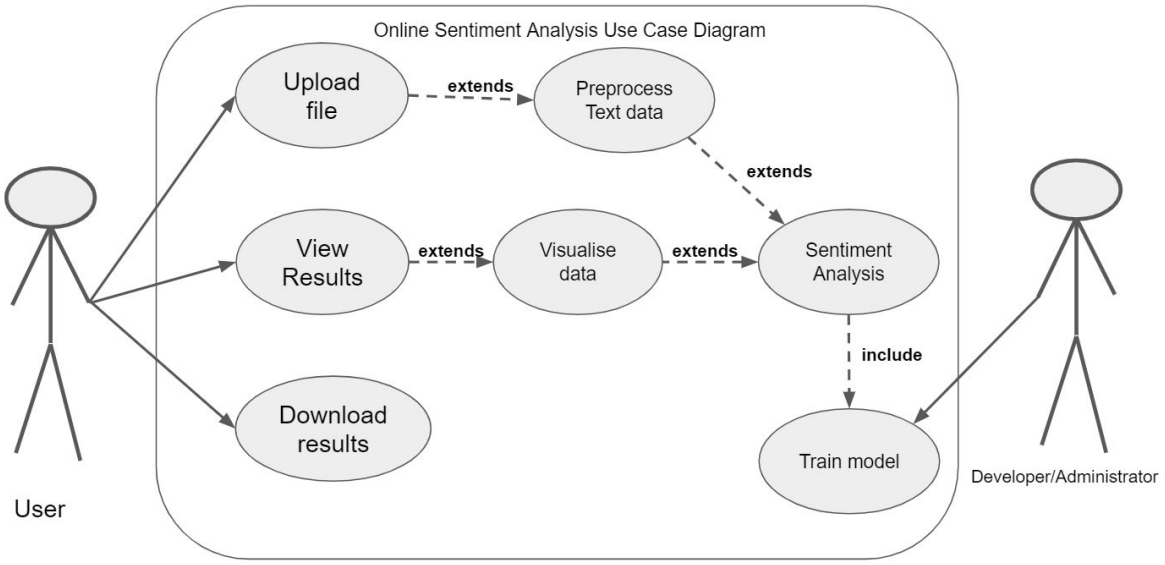

Fig 3. Use case diagram for this project

Upload File:	
Actors:	App User
Description:	The user selects a file through the provided file browser and submits the form.
Data:	**Input:** single .csv file **Output:** Object containing result of sentiment analysis
Pre-Condition:	The file is in .csv format. The data within the file has columns in the order {tweet ID, tweet} A previous file is not currently being processed.
Post-Condition:	The file has had its sentiment predicted and the predictions have been gathered in the server, ready for display to the user.
Valid Case:	The user clicks on the file input button and selects the file they want to be analysed. The file will then be sent to the server and the preprocessing stage will begin. Progress messages will be displayed to the user during all stages as the entire process can take several minutes to complete. After it has completed the results will be collected in the server ready for data visualisation.
Error Case:	The User uploads a file that is either not in the .csv file format or does not have the columns organised correctly within the file. The server will respond with an error message prompting the user to correct the possible errors and try again.

View results :	
Actors:	App User
Description:	Return the data to the user in the form of a graph
Data:	**Input:** Results from sentiment analysis **Output:** Graph in the client window displaying the results

Pre-Condition:	The user has uploaded a file and the sentiment analysis has been completed
Post-Condition:	The user has been shown a graph displaying the results of the sentiment analysis
Valid Case:	The results of sentiment analysis have been received by the node server. The data points will then be plotted on a graph with the amount of loss and uncertainty in the predictions. Finally the graph is displayed in the client window so that the user can analyse the results
Error Case:	There are no errors to be made at this step because the LSTM output data (i.e input data for the view results use case) is in the same shape every single time, so if there were errors in the data they would have been caught and handled before being entered into the LSTM.

Download results :	
Actors:	App User
Description:	Download the results of the sentiment analysis in a .txt file
Data:	**Input:** Results from the sentiment analysis **Output:** downloads results as a .txt file
Pre-Condition:	The user has uploaded a file and the sentiments of the tweets within the file have been predicted by the model
Post-Condition:	The results have successfully downloaded onto the users device
Valid Case:	The user clicks on the 'download results as file' button and the browser begins to download the results as a .txt file. Once this has completed the file will appear in the users 'Downloads' directory.
Error Case:	Network errors can occur during the download for example a loss of internet connection, which will cause the download to fail.

Preprocess text data :	
Actors:	App User
Description:	Remove all non UTF-8 characters, lemmatisation and obtaining GloVe embeddings.
Data:	**Input:** The file that was uploaded by the user **Output:** GloVe word embeddings
Pre-Condition:	The user has uploaded a file and and it has passed validation in the upload file use case
Post-Condition:	Every word within the corpus of tweets has been assigned a GloVe word vector.
Valid Case:	The file has been validated. The data is then cleaned and GloVe word vectors have been produced ready for insertion into the LSTM.
Error Case:	If the file is empty or there is no data in the tweet columns then the user will be shown an error message telling them that there must be at least 1 complete entry in the dataset

Sentiment Analysis :	
Actors:	App User
Description:	Input the tweets in their GloVe form into the LSTM and predict the sentiment of those tweets
Data:	**Input:** tweets in the form of GloVe word vectors **Output:** prediction made for each tweet.
Pre-Condition:	Every word has been assigned a GloVe vector and the corpus has been reassembled using those word vectors.
Post-Condition:	The sentiments of the tweets have been predicted by the LSTM.

Valid Case:	The data is input into the LSTM and the LSTM is able to perform all 50 epochs and return an object containing an array that holds the results of the sentiment analysis.
Error Case:	The device running the sentiment analysis model (the server) can run out of memory when processing very large datasets and in this case the sentiment analysis stage will fail.

Visualise data :	
Actors:	App User
Description:	The predictions made by the LSTM are plotted on a graph and displayed to the user for analysis.
Data:	**Input:** Results from the sentiment analysis **Output:** A graph in the client window
Pre-Condition:	The LSTM successfully finished predicting the sentiment of all tweets in the corpus and the user has clicked the 'view results' button.
Post-Condition:	The graph has been successfully displayed to the user in the client window.
Valid Case:	The user is notified of the completion of the sentiment analysis stage and continues by clicking the View Results button which will cause the graph to be retrieved from the server and rendered on their screen.
Error Case:	The user is notified that results are ready and attempts to view them. However if there are network connection problems such as a loss of connection then the rendering of the graph will fail.

Train model :	
Actors:	Developer/Administrator
Description:	Train the model using a dataset of your own and save the model for later use.
Data:	**Input:** Training data set in the form {tweet id, tweet sentiment, tweet} **Output:** HDF5 file containing weights and final states of all nodes and parameters in the trained model
Pre-Condition:	The actor attempting to train the model must have access to the source files and therefore must be one of the developers or an administrator level user. There must be a file named 'training.csv' in the './data' directory and the contents of the file are formatted correctly
Post-Condition:	The LSTM has been trained and the trained model saved to the actors device.
Valid Case:	There is a file in the python code for this project called 'training.py', which takes no arguments, but when run it will open the file 'training.csv' and begin the training process. The preprocessing is the exact same for predicting sentiments on an already trained model. The sentiment mode will take the sentiments this time in order to calculate the loss, weights and biases for each connection within the network based on the accuracy of the previous predictions. Depending on the size of the training set and the computing power of the device running the server application, training can take anywhere from a few minutes to a few hours or days. Once it has completed the trained model will be saved in an HDF5 file to the './data' directory.
Error Case:	If there is no file called 'training.csv' within the './data' directory then the training process will fail. Also if the file is present but it is empty then the process will also fail. Training an LSTM is an extremely compute intensive operation and requires a very large amount of RAM, and often requires you to leverage the computing power of your GPU through the use of the CUDA and cuDNN technologies. Without at least 16GB RAM or using the GPU and its VRAM to accelerate training then this process will fail for datasets of ~10,000 samples.

9.2. System Architecture

Several factors were considered when choosing the architecture for this system. There will be only limited resources and computing power available during the development process. The task of sentiment analysis is very computationally intensive as it requires complex computations with vectors and therefore, to be able to offload this computation to a more powerful machine would be a great benefit. Furthermore, the software that is being developed is a web app, meaning a client-server architecture is likely the best choice. However, two other architectures were also considered: Cloud computing architecture and peer-to-peer.

9.2.1. Client-Server Architecture

Client server architectures typically consist of a client PC, Database server and Application server (Oluwatosin, 2014). The client PC sends HTTP requests to the application server which, depending on which path the request was received, will make its own request to a separate database server to retrieve specific information. However, due to the nature of the application that is being presented in this report there is no need for a database server, which can be replaced by a Python server that will run the sentiment analysis and deep learning. One benefit of an architecture such as this is that it helps to free up the resources on the Python server that will be running the sentiment analysis because that machine will not have to handle any other computations, they will all be done in the application server. This can help to provide a better user experience by reducing the server response times, in turn reducing the wait times for the user to receive their analysed data. Conversely, this is a simple architecture which is likely to get overloaded with the amount of network traffic there is on the internet today. This would lead to congestion in the network, lost packets and a negative user experience. Another negative to this type of architecture is that it creates a single point of failure in the application server. If the application server crashes, then the entire application will go down with it and users will be unable to access the service.

9.2.2. Cloud Computing Architecture

Most of the current clouds are built on top of modern data centers. It incorporates Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), and provides these services like utilities, so the end users are billed by how much they used (Tsai, Sun & Balasooriya, 2010). Cloud computing allows users with devices that, by themselves, would not be able to perform certain processes or services, to be able to access the service from cloud servers. This gives small mobile devices the appearance of having greater computing power by being able to perform computationally intensive tasks in the cloud. This would be an ideal architecture for this application to be offered as a cloud service because it would give users the best and most current user experience. There are however some large drawbacks to this model, especially for the development and scale of this project. Firstly, there are costs involved with building a cloud server as it typically requires large amounts of infrastructure, such as at least one server although the number of servers that make up the most popular cloud services today are in the thousands. Also, cloud computing is still a new technology that would require taking a lot of time to learn how to develop applications as cloud services and due to the time constraints of building this project it is not realistically possible.

9.2.3. Peer-to-Peer Architecture

In a P2P networking model, each of the computers on the network could act as both servers and clients thus distributing the network load and overhead across all the devices on the network, forming a network of individual peers with ability to search the network for shared resources (Abiona et al., 2009). P2P architecture offers benefits such as reliability and fast one-to-one communication, which make it a suitable architecture for a mobile application. However, this project is not aimed at being used from a mobile device and would see greater benefits from having a centralised infrastructure, such as a server. This is due to the fact that the server will likely be more powerful than the device that the user is using so the user is able to utilise the extra computing power of a server.

9.2.4. Decision and Justification

Adopting a client-server architecture is the best option for this project for several reasons. Firstly, it would allow the quick development and deployment of a scalable and working server side application provided you have a server to run it on, where as with cloud computing it is much more complicated to set up and would require the developers to have to learn new skills which, given the time frame of this project, would make it much more likely to not meet the deadline. Also client server architectures are the most popular in the world and there are many javascript frameworks such as Express for node.js that are specifically designed for quickly building scalable server side APIs. However, given a longer period of time the chosen option would have been cloud computing as it would have allowed many other features such as cloud storage for a users datasets and for the results of their analysis. As for P2P architecture it is not feasible for this project because any computation offloading that occurs from client to server is computed at the server, while in a wireless P2P network this computation

would be shared among other nodes in the network, which means other users of the application would see the performance of their devices affected. Also there is no access to a centralised data storage for the P2P architecture. Fig 4 displays a high level representation of the system architecture and the flow of data through the different components of the application. The dotted arrow between train model and preprocess data is to show that, that is an administrator level action.

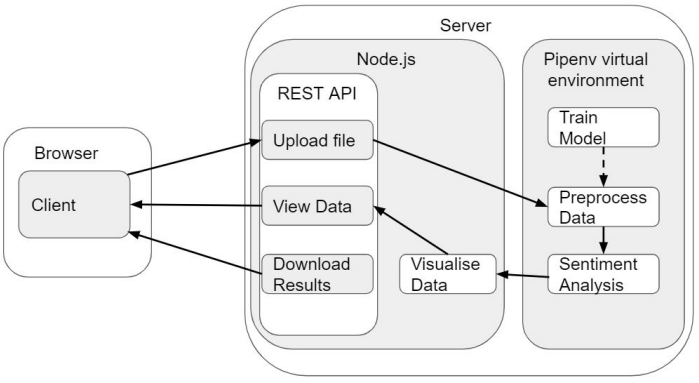

Fig. 4 Architecture of this project and flow of data through the different systems

10. Implementation

This section will outline the decisions made around the implementation of each of the systems of this project. The dataset that we are using to train our model is available for download or inspection at https://www.kaggle.com/crowdflower/twitter-airline-sentiment. It contains 14,427 tweets labelled as positive, negative or neutral as well things like the topic that is being discussed, and the confidence in the sentiment polarity (How strongly positive, neutral or negative a tweet is).

10.1. Project Platforms, Environments and Technologies

The entire project will be split into 2 applications; one for the Node.js application and one for the Python application. The node app will handle all of the web functionalities from handling file uploads to making predictions on a pre trained model. The python app is what will be used to compile, train and test the LSTM as well as training the GloVe embeddings, you can also make predictions with the python application. Each application has a 'README.md' file in their main project directory with instructions for how to run each application.

Javascript, HTML and css are all extremely well supported languages with countless libraries to aid in the development of all kinds of web production. These are the main reasons why they have been chosen for the client side application for this project. Also javascript is a language that the developer of this project is proficient with, which means not having to learn a new language. Creating a Java web app with JSF was also considered however the Java web technologies do not have as extensive cross-language support as Node.js offers javascript. This is important because Python files, containing the deep learning code, need to be run from the API which either way is going to be written in Javascript or Java, so it makes sense to go with the server language with the best support for running python code. Node.js allows javascript to be run outside of the browser which means that frontend and backend applications can all be written in the same language.

Python has the largest and most extensive list of third-party libraries aimed at performing all aspects of machine learning and deep learning which are the main domains covered by the GloVe embedding stage (Adaptive gradient descent machine learning) and the LSTM RNN (deep learning). The syntax and grammar in Python is also one of the most human-like and easy to read, which makes it ideal for trying to model and make predictions on complex structures and mechanisms, such as language, because it can help to simplify and make clear the logic of the code. Furthermore, Python has lots of useful methods for ensuring the correct dependencies and versions are installed on the device. For this project we will be using Pipenv, which packages all required dependencies into a Pipenv virtual environment and then when the code is run on a different device the 'pipfile.lock' file will ensure that all the necessary dependencies are installed. The most popular python library for working with vectors and matrices is NumPy which provides amazing functionality for these data structures and therefore different types of NumPy arrays have been used extensively throughout the development of this project. One of the largest Python libraries for developing deep learning models is the tensorflow end-to-end machine learning platform that has functionality to aid with the preprocessing, such as removing punctuation, obtaining word embeddings and using the Keras Tensorflow library, build, train and deploy custom neural network models. While Tensorflow comes with the ability to create word embeddings and shape the resulting vectors correctly, which can be an excruciating and confusing process, it will only be used for the construction, training and testing of the LSTM model. This is because the GloVe embeddings are going to be trained on the same dataset that is being used for training the LSTM and the pre trained GloVe embeddings from https://nlp.stanford.edu/projects/glove/ will not be used in this project. Secondly, since twitter data is very unique and the way that people write on twitter, and other popular social media, is unlike

anywhere else and therefore custom and twitter-tailored preprocessing must be written in order to obtain the highest possible accuracy in the LSTM.

10.2. Preprocessing

Several methods and strategies for preprocessing the data have been tried in order to find an optimal process for preparing the text. There is no definitive way to find the optimal processes to ready the data due to the uniqueness and unpredictability of language. Immediately after the user uploaded file is received by the Node.js server it needs to have all of the tweets preprocessed and turned into GloVe word vectors. The server application will use the "child_process" module that comes packaged with Node.js to run the python code and pass the necessary arguments (the file). The python control file, "main.py", controls the order of execution of the whole sentiment analysis process and the communication with the Node.js server. The code for the preprocessing stage can be found in "./source/sentiment_analysis/preprocessing.py" file. Preprocessing starts by first opening and reading the lines of the "stop_words.txt" file into a python list. The entire corpus of tweets is then iterated over and any intersections between each tweet and the set of stopwords are removed from the corpus. In the same iteration over the corpus we are using a regular expression to recognise and remove punctuation, hash tags, @'s including other twitter users and emojis. It is important to remove these because such characters can confuse the GloVe embeddings and LSTM and produce negative results.

Secondly in the preprocessing stage we need to remove consecutive duplicates of characters within words. Twitter data uses a lot of slang and is very informal and you therefore see a lot of samples that include elongations of words which represent a users attempt to add emphasis to their tweet. For example, in the training data set used for this project there is an elongation of the word 'guys' into 'guyyyys' (4 consecutive 'y') which would be shortened into 'guyyys' (3 consecutive 'y'). We do not remove all consecutive duplicates because some letters are supposed to come after each other, such as 'hello', and the duplicates also contain a lot of information to be garnered in themselves. Therefore, to completely remove them would be removing potential insight and knowledge that could be learned by the GloVe embeddings and offer better predictions in the LSTM.

Finally, the tweets must be lemmatised. Lemmatisation is the act of returning the word to its dictionary form, removing inflectional endings. This is a better option than stemming because lemmatisation takes into account the part of speech (POS) of each word and you therefore get more accurate results. As stated, this process takes into account the POS of each word and we therefore need to implement a POS tagger as well. Both of these operations are made easy through the use of Natural Language Toolkit (NLTK) which provides both a POS tagger and a lemmatiser. Every tweet is tokenised and each token given a POS tag, like ("running", "v"). This tuple is then fed into the lemmatizer which outputs a single lemma, continuing on from the running example, the lemma would be 'run'. This concludes the preprocessing and the data is now ready to be turned into GloVe word vectors.

10.3. GloVe embeddings

We have implemented our own GloVe embeddings following the methods outlined in Pennington et al. (2014). The process starts with building a vocabulary from the corpus, which is the set of all words in the corpus. We then build a word cooccurrence matrix, X, containing rows of vectors showing how often the word j appears in the context of the word i and the value of the cooccurrence is given by X_{ij}. In order to achieve this we implement a sliding n-gram window of length 10 over every tweet in the corpus and we take the centre word in the window to be the main word, i. The amount of increment that will be applied to X_{ij} is calculated as the inverse proportion of the number of words, d, between i and j,

$$X_{ij} = X_{ij} + 1/d \tag{1}$$

This is done to ensure a larger increment for X_{ij} when the distance between words are small, potentially meaning that they are more strongly related, and the increment decreases as i and j grow further apart. We end up with a list of tuples representing the matrix value for all words as both i and j. The length of the list will be $|Corpus| * 2$ because of that and each tuple in the cooccurrence matrix is structured as (I_{id}, J_{id}, X_{ij}).

Before beginning the training we set up a new word vector matrix, W, as a 2-dimensional matrix with shape $(2V, d)$ where V is the vocabulary size and d is the vector dimensionality. This is because we have a vector for each word as both a main and context word. We then set up three arrays to hold the biases for the word vectors, the sum of all previous squared gradients, and the biases associated with those gradients. All of these arrays have the shape $2V$ because they contain 1 value for each word vector. Finally, all of these values are stored in an array, 'data'.

Now we apply adaptive gradient descent (AdaGrad) to these cooccurrences to obtain the final, trained, GloVe embeddings. AdaGrad is a version of stochastic gradient descent optimisation algorithm that *adapts* the learning rate of each iteration to the parameters of the previous iterations, performing smaller updates to the vectors in W (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data and it is also the algorithm used by Pennington et al. (2014) to train the original GloVe embeddings at Stanford. The weighting function for each iteration of this algorithm is,

$$f(X_{ij}) = \begin{cases} \frac{X_{ij}}{X_{max}} \ if & X_{ij} < X_{max} \\ 1 & otherwise \end{cases} \tag{2}$$

The maximum weight allowed to be added per iteration is 1 because for word pairs that have a very high X_{ij}, e.g. $>X_{max}$, we will also see the cost associated with that word to be too high, which results in less accurate embeddings. This is since the cost function that is implemented for this algorithm is,

$$J = \sum_{i,j=1}^{V} f(X_{ij})((w_i \cdot \tilde{w}_j) + b_i + \tilde{b}_j - log(X_{ij}))^2 \tag{3}$$

J is the sum of the cost of all word vectors per iteration. $f(X_{ij})$ is the weight function Eqn. (2), w_i and b_i are the main word vector and the bias associated with it, and $\tilde{w}_j$ and $\tilde{b}_j$ are the context word vector and context word vector bias respectively. The cost function shown in Eqn. (3) takes the weight and multiplies it by the gradients of the word vectors, plus their biases, all squared and then repeats and sums the value for every $(i, j) \in vocab$. The algorithm is trying to find a local minima, therefore it is very useful to have adaptive learning rates so that we can vary the amount of gradient descent with each step making our algorithm much more efficient at finding a local minima.

We use the inner component (cost-inner) of the cost function, $(w_i \cdot \tilde{w}_j) + b_i + \tilde{b}_j - log(X_{ij})$ to calculate the gradients updates. The gradient sums of the biases are simply the value of the cost inner value. However, the gradient sums of main and context word vectors are calculated by:

$$grad_{main} = ((w_i \cdot \tilde{w}_j) + b_i + \tilde{b}_j - log(X_{ij})) * \tilde{w}_j \tag{4}$$

$$grad_{context} = ((w_i \cdot \tilde{w}_j) + b_i + \tilde{b}_j - log(X_{ij})) * w_i \tag{5}$$

Using the newly updated gradient sums we calculate the updates to the word vectors and to their respective biases with the following equations, which happen once per iteration:

$$w_i = w_i - \frac{\eta * grad_{main}}{\sqrt{gradsq_{main}}} \tag{6}$$

$$\tilde{w}_j = \tilde{w}_j - \frac{\eta * grad_{context}}{\sqrt{gradsq_{context}}} \tag{7}$$

$$b_i = b_i - \frac{\eta * gradbias_{main}}{\sqrt{gradbiassq_{main}}} \tag{8}$$

$$\tilde{b}_j = \tilde{b}_j - \frac{\eta * gradbias_{context}}{\sqrt{gradbiassq_{context}}} \tag{9}$$

Finally, the squared gradients sums are calculated by squaring the previously updated gradients: $grad_{main}$, $grad_{context}$ and the gradients of the bias for both of those words, then adding that value to their respective global totals. That is the process for a single iteration to train our the GloVe embeddings and we have initialised the algorithm to go through 50 iterations in order to encode enough meaning into the word vectors. (Pennington et al., 2014) suggest using 50 iterations for word vectors with dimensionality less than 300, and the word vectors we are producing will have dimensionality of 200. However, during the testing stage all of these hyper parameters will be altered to find more optimal configurations. Before the GloVe embeddings can be handed over to the LSTM for classification, we have to do something with the context vectors. Some suggestions are to simply remove these context vectors, however we do not want to throw away their potential meaning so instead we have calculated the mean vector between the main and context vectors for each word. The final step before starting the sentiment analysis process with the LSTM is creating a new list and inserting a tuple for every word in the vocabulary, containing a word and its associated vector.

10.4. LSTM

Our GloVe word vectors need to be inserted back into the corpus of tweets in the exact same position that the words they represent occur in. This leaves us with the exact same corpus except the words are now word vectors. Secondly, the tweets are of different length and need to all be padded to the same length because the LSTM is expecting every input sample to be the same shape. For this reason we pad the beginning of all tweets vectors with matching 200 dimensional vectors initialised with 0's because the LSTM will ignore any 0 vectors. Each tweet is padded until it is the length of the longest tweet in the corpus and we now have a 3 dimensional tweet vector array that is ready for input to the LSTM. Now we move onto preparing the target data for our LSTM. There are three possible output labels for each tweet, positive, negative or neutral and we need to encode this into a vector for the LSTM to be able to do computations with that data. We have used one-hot encoding to represent the target data labels as positive = $[0, 0, 1]$, neutral = $[0, 1, 0]$ and negative = $[1, 0, 0]$. The final process in preparing the data is to split it into 3 sets for training (80%), validating (10%) and testing (10%).

To implement the LSTM itself we are using Google's Tensorflow platform with the keras library. The LSTM that is being implemented has 5 layers: two LSTM layers each followed by a dropout layer, and then a single densely connected output layer wrapped in a keras time distributed

layer. This is displayed in Fig. 5, however the values you see for the output shape and number of parameters are not representative of the actual tests as this screenshot was taken during development. In reality the LSTM layers will have the output shape $(1000, tweet_{max}, H_n)$ where 1000 is the batch size (number of samples input per update), $tweet_{max}$ is the number of vectors per tweet and H_n is the number of hidden nodes in the layer.

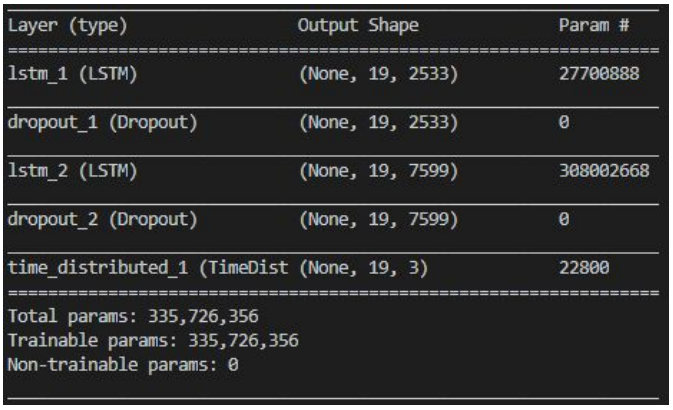

Fig. 5 Structure of LSTM RNN model

Keras LSTM layers are implemented with the architecture shown in Fig 6. X_t is the input word vector at each time step, h_{t-1} is the output from the previous hidden node and c_{t-1} is the output of the previous cell state. The cell state is what allows LSTM's to remember long term dependencies because it is a continuous value that is maintained and updated throughout each time step. Another point here is that keras LSTM allows you to set a property, 'return_sequences=True', on an LSTM layer that will make the current hidden state value, h_t, get output to the next layer at every time step and not only the final timestep of a sample, which is why the output of the two LSTM layers have shapes with 3 dimensions.

Dropout is a generalization method used to combat overfitting in neural networks. It consists of multiplying neural net activations by random zero-one masks during training (random masks are not used in test time, but are approximated by a fixed scale). The dropout probability p determines what proportion of the mask values are one (Cheng et al., 2017). In Tensorflow they have support for this through the dropout layer. As you can see in Fig. 5, we implement a dropout layer after every LSTM layer to minimize the overfitting as much as possible. We set the dropout rate, p, to be 0.2 initially, although different numbers will be tested.

Finally the output layer is configured as a Time Distributed wrapper around a densely connected layer with softmax activation. This wrapper is a Keras resource that applies a layer to every temporal slice of an input ("Layer wrappers - Keras Documentation", 2020). We use the Time Distributed wrapper to apply the densely connected output layer to, in the case of Fig. 5, the 19 time steps in every sample. This is because the default behaviour of the densely connected layer is to only be applied to the output of the final timestep in the previous layer.

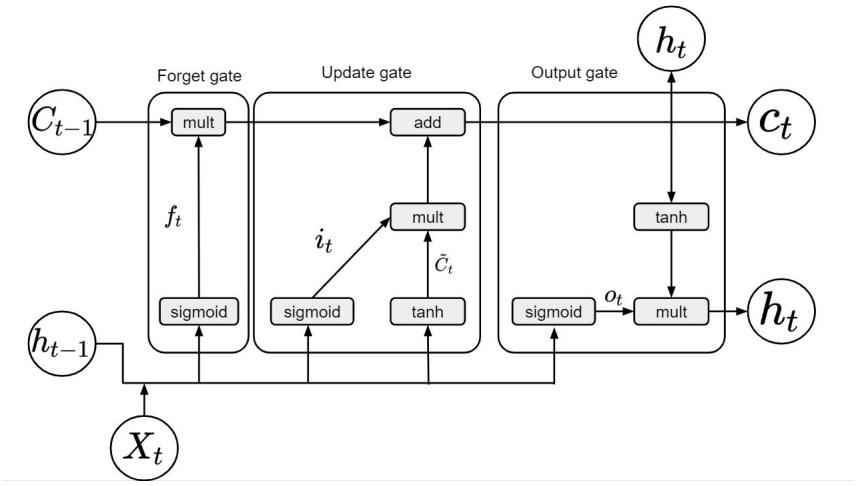

Fig. 6 LSTM architecture

There are several hyper parameters that need to be established before fitting the LSTM to the training data. These include:

- the batch size which we have initially set to 1000 is the number of samples that are inputted per update,
- The number of epochs to perform, which is the number of passes over the entire corpus. We set this to 10 initially.
- the dropout factor that will be given to the dropout mask and used to set a proportion of neurons to switch off,
- the weight decay for the l2 (least squared errors) regularization of the weights added between connections we set equal to $1x10^{-8}$,
- we are using the Adam, adaptive gradient descent, algorithm to optimise our results and we therefore need to set a learning rate which we initialise as $5x10^{-4}$.

There are many hardware constraints that govern the hyperparameters here, for example the amount of memory (RAM) that we have available to us for training impacts the choices we are going to make massively. This is because training the LSTM is incredibly computationally intensive and if there are too many parameters to train our machine will run out of memory and the training will fail. However, we will also be leveraging GPU acceleration technology to speed up the training process which can take hours, depending on lots of things. The hardware of the machine that is going to be running the training has a single Nvidia GeForce RTX 2070 GPU, single Intel i5-6500 CPU and 16GB RAM. This poses a potential problem as most deep learning is performed on machines with multiples GPUs and multiple CPUs, therefore I might have to scale down the number of samples, the GloVe vector dimensionality or the number of hidden nodes in the LSTM layer if the hardware available to me is insufficient. This will be discovered during the testing stage because while developing we are not using the full dataset, merely a very small portion of the dataset to see that the functionality is correct before we try the full dataset.

10.5. Node.js and Express framework API

This project is aimed at offering fast and readily available access to sentiment analysis services and therefore we need to set up a server application that will allow the service to be accessed 24/7 and from anywhere with an internet connection. Node.js is the javascript runtime which we will be using together with the Express framework to implement a REST API that will handle HTTP requests and serve the results of the sentiment analysis to and from the javascript client application. Due to the fact

that we do not have access to an actual web server and I do not want to set up my own PC as a web server for security reasons we are running the server on 'localhost:8080', that is on the localhost port 8080. The run command for the server is 'node server.js'

Our API requires only one endpoint, '/submit_file'. To do this we use Express to set up a request handler function that behaves similarly to javascript event listeners in that it listens to all 'POST' requests on the given path. Express request handlers make handling these requests very intuitive by creating two objects, 'request' and 'response'. 'Request' holds all of the data that was sent to the request handler by the client application, in the case of this project the 'request.file' property will hold the file that the user wants to be analysed. 'Response' is the object that will handle sending a response back to the client. This object has several methods for responding with specific data types, such as JSON and strings and specific actions like page redirects or file downloads. The response methods we are using for our project are 'response.download()' and 'response.JSON()' because we have two operations that must be implemented: View the results as a graph so we will transfer the graph data as a JSON object and we also need download the results as a txt file. Request handlers also need to validate the data, ensuring that the file data is formatted correctly with the following columns, (Tweet id, Tweet text).

There will be a file in the './data' directory that will hold every word and word vector pair obtained from the training of the GloVe embeddings in the python code. The server application iterates over every word in the user input file and searches the GloVe embeddings, that are stored on the server, for the corresponding word vector. Once a match is found the the word in the user input file is replaced by the word vector in the GloVe file. This leaves us with an array of tweets represented by GloVe word vectors that is ready to be prepared such that it can be used as input for the LSTM model.

Now that the GloVe embeddings have been obtained we need to prepare the data so that it can be input into the LSTM. We start by iterating over the GloVe embeddings and finding the longest one, the one with the most vectors. We then set a variable, 'tweet_length', which holds the length of that embedding and is later used for padding. Since we are using a Keras model on the Tensorflow platform we need to convert the embeddings, which are currently stored in a javascript array, into a tensorflow tensor object. This is achievable by first standardising the length of all the GloVe embeddings by zero padding them until their length is equal to 'tweet_length'. This involves pushing additional vectors initialised all with 0's and with the same number of dimensions as the GloVe vectors to the end of the GloVe embeddings. Then we set up new tensors with the 'tf.tensor(values, shape)' function which takes as arguments a javascript array and the shape that you want the vector to have. The shape argument must be three dimensional and have the following value $[t, w, v]$, where t is the number of tweets that are going to have their sentiment predicted, w is the value of 'tweet_length' that we found earlier and v is the dimensionality of the word vectors.

10.6. Client Application

The client side application consists of HTML, CSS and Javascript files working together to display and gather data to and from the user as well as sending HTTP requests to the server. There is only one HTML page for the website, which contains a form that allows users to select a file using their file browser, and also to enter a name if they wish to rename the file. After the user uploads the file to the backend application it will receive one of two responses, either there was something wrong with the format or file type of the submission, in which case the user will be prompted to try again and made aware of possible errors or the client will receive the results of the sentiment analysis. Depending on the size of the dataset that the user wants sentiment predictions to be made on and the hardware of the server machine, obtaining GloVe embeddings and then using the model to make predictions can take anywhere from seconds to hours. Therefore, the user will be shown update messages to advise them

on the progress of the sentiment analysis process. These messages will take the form of Javascript alerts which render pop up alert boxes on the users page that you can give a custom message.

As stated previously the server will run on 'localhost:8080' which means to be served the files to your browser from the node server, you first need to open a command shell or a terminal in your IDE and cd to the project directory. Once there, execute the run command for the Node server, you will be notified in the terminal when it is running, open your browser (only tested on Google Chrome) and navigate to 'localhost:8080'.

11. Testing

The testing for this project was performed in 4 stages or areas. Those were the:

- web application, encompassing the functionality of both the front and backend applications,
- Preprocessing of the user data including, removing noise and lemmatisation,
- GloVe embeddings will be tested against one-hot encoded vectors by seeing the results when they previously go through the same preprocessing and subsequently are input into the same LSTM model
- LSTM accuracy will be tested by comparing its results against baseline predictions to see how the model produced in this project performs.

The web application will be tested by ensuring that the server starts when the run command is executed and that it is accessible by navigating to 'localhost:8080' in a web browser. The file upload will also be tested ensuring that the file that the user uploads is properly received by the server, this will be done by simply uploading a test file, printing the contents of the file when it arrives in the server and then comparing this to the original file that was uploaded. The file validation will be tested by uploading a file which does not have .csv type, another file where the contents are not formatted correctly and a third file which has both a .csv file type and the contents are formatted correctly. The validator must reject the first two files and accept the third files. Upon rejection the server must respond with an alert message prompting the user to try again.

The preprocessing must remove a prescribed list of stopwords that are stored in 'stopword.txt' file. The success of this will be tested by using the same list of stopwords and dummy data in he place of input data, but this time searching for any intersections between the two lists, if any are found then the test has failed. The same test will be done for the regex used to remove punctuation, numbers, @'s, emojis or URLs, using the same regex patterns to search the resulting corpus. Finally we will test the lemmatisation process by simply inspecting the resulting corpus. The words should all be root words and the lemmatisation should also have rebuilt the tweets using the lemmas in the correct order. If these conditions are not met then the test will fail.

To test the effectiveness of the GloVe embeddings on the accuracy of the predictions made by the LSTM we will test the results against those obtained by using the Bag-of-Words (BOW) algorithm, this will give an idea of the added value that the GloVe embeddings bring. However, if the BOW vectors outperform the GloVe embeddings then the test will be considered a failure. The preprocessing and the LSTM model that are used by both methods of word embedding will be identical so that the only differences in the two tests is the algorithm used to obtain the vector representations of the words.

The final stage of testing will be done on the LSTM which will be tested by recording the two metrics (cross entropy loss and mean accuracy) and analysing the resulting graphs to ascertain the quality of predictions made by the LSTM. Comparisons will be made between the loss after each epoch which represents the deviation of the predicted outputs from the expected outputs, also the

predicted probability distribution of the classes will be compared which will show which model made more correct or incorrect classifications.

11.1. Web functionality testing

Test id	Description	Pass/Fail conditions	Result
1	The server must start when the run command 'node server.js' is executed in the project directory	**Pass**: the server notifies the user of a successful start **Fail:** the server fails to start	Pass
2	After starting the successfully starting the server we navigated to 'localhost:8080' in a web browser.	**Pass:** the website is accessible at 'localhost:8080' **Fail:** the website is inaccessible at 'localhost:8080'	Pass
3	A file was uploaded to the server using the form in the client application	**Pass:** the REST API receives the file on the '/submit_file' endpoint. **Fail:** the api does not detect any requests being made to that endpoint.	Pass
4	The server must recognise when a file is not in .csv format and reject this file.	**Pass:** the server rejects any file that does not have .csv suffix **Fail:** the server accepts files that do not have .csv suffix	Pass
6	The server must check that the file has 2 columns. Column 1 must contain unique integer id and column 2 must contain String data	**Pass:** the server reject any file that does not meet the conditions laid out in the description **Fail:** the server accepts files that do not meet every condition laid out in the description	Pass
5	The server must recognise when a file is empty	**Pass:** the server rejects any file is empty **Fail:** the server accepts files that are empty	Pass
6	Python code in the './source/sentiment_analysis/main.py' file must be executed from the Node.js javascript application.	**Pass:** A child process is spawned and python code begins to run **Fail:** The child process fails to begin	Fail

11.2. Preprocessing functionality testing

Test id	Description	Pass/Fail conditions	Result
8	The Input file must have the stopwords found in the './data/stop_words.txt' removed anywhere they appear in the corpus.	**Pass**: searching the output of the preprocessing stage for intersections between the stopwords finds no matches.	Pass

		Fail: searching the output of the preprocessing stage for intersections between the stopwords does find matches	
2	Iterate over the test corpus containing dummy data and apply the regex pattern used to remove @'s, hashtags, emojis and numbers to every tweet.	**Pass:** searching the output of the regex patterns for every tweet for occurrences of hashtags, emojis, number or @'s finds no results **Fail:** searching the output of the regex patterns over every tweet for occurrences of hashtags, emojis, numbers or @'s finds results.	Pass
3	The test corpus will then be subjected to the lemmatisation process, in which we feed the words one by one along with their POS tag, into the lemmatiser and observe the output.	**Pass:** Every word in the text corpus must be a root word and they must no longer have their POS tag attached to them **Fail:** If there are any words in the test corpus after lemmatisation with inflectional endings e.g. '-es', '-ing', '-ed' etc.	Pass

11.3. GloVe embedding results

We tested the effectiveness of the GloVe embeddings in terms of how closely encode the relationship between words in twitter data in comparison to Bag-of-Words (BOW)embeddings, which will be a variation of one hot encoding that produces dense vectors by assigning words in the vocabulary a unique index value to represent it, we will then encode the tweets using the index value for each word. This will allow us to if the euclidean distance between a word vector space being closer to the words that it appears in the context of frequently (GloVe embeddings), can improve the accuracy of an LSTM RNN over embedding methods that do not try to do this with the word vectors, in this case BOW. The BOW implementation used for testing can be found in the file './source/sentiment_analysis/BOW.py'. Both types of word vectors were tested over 20 epochs to speed up the process a little bit.

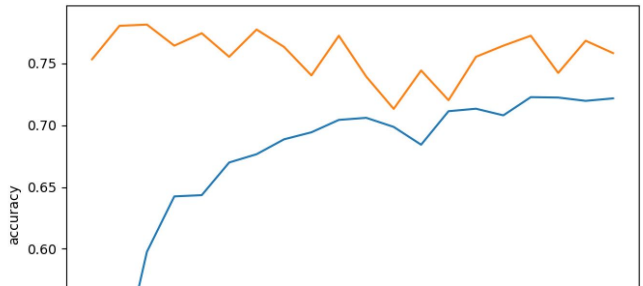

Fig 7. LSTM results with BOW vectors *Fig. 8 LSTM loss with BOW vectors*

Fig. 7-8 display the results from the LSTM when the BOW vectors were used. As you can see BOW achieved an average accuracy of ~75% in the validation sets, which is the equivalent of testing the model on unlabeled data after every epoch. However, the accuracy is also fluctuating up and down while the loss in Fig. 8 is decreasing. This shows the BOW embeddings contain noise that the model is learning. Fig 14 shows that the GloVe embeddings achieved roughly the same accuracy but with much more consistency, while comparing Fig. 8 with Fig. 12 we see that apart from the early spike the model has fit the data better.

11.4. LSTM RNN Results

Several large issues were discovered during the testing stage of the LSTM. The dataset that was originally being used to train this model had 14,427 unique tweets, we were training GloVe vectors with 200 dimensions and the first LSTM layer had 2533 neurons while the second layer had 7599. This created more parameters than there were bytes of RAM on the machine that the training was being run on, resulting in out-of-memory (OOM) errors. Due to the Coronavirus pandemic I was not able to access more powerful machines at the University and my tests and model therefore had to be scaled down so that they were able to be executed on the only machine that was available. The TimeDistributed layer created extra trainable parameters for the weighted connection between every temporal slice of the output of the final intermediate layer and the densely connected output layer. This needed to be removed as our machine did not have enough memory to store the values of that many vectors. This meant that the output layer was only being applied to the output of the final time step in the previous layer. The training set was reduced to 5,000 tweets which is still large enough for some more complex patterns and nuances to be learned. However, the GloVe embeddings have been reduced to 100 dimensions which in turn reduces the number of neurons in the LSTM layers and therefore reduces learning capabilities.

Using this model we were able to train and record metrics for an LSTM RNN. The metrics that we recorded were the cross-entropy loss after each epoch for both the training and validation sets, showing how closely the model got its predictions. We also recorded the mean accuracy after each epoch for the training and validation sets, showing the proportion of predictions that the model has got correct regardless of if it only predicted positive with, for example, 51% certainty.

Figures 7-8 show the first round of testing done with the LSTM. Fig. 7 displays the loss after each epoch of the training and validation loss. Both curves are smooth and do not rise which is an indication of a model that has fitted the data well. However, by looking at Fig. 8 you can clearly see that the model is not learning the data very well and the predictions that it is making are not better

than chance, the highest accuracy over all epochs was 68%. This is evidence that the model is becoming more certain of the predictions it is getting right, but the amount of right predictions is not increasing. In fact Fig. 8 shows drastic decreases and increases across epochs in the accuracy of its predictions, which might mean that it is learning the training data too well, and overfitting. There are several reasons for this and the solutions that were attempted to fix this problem can be found in section 12 - Evaluation.

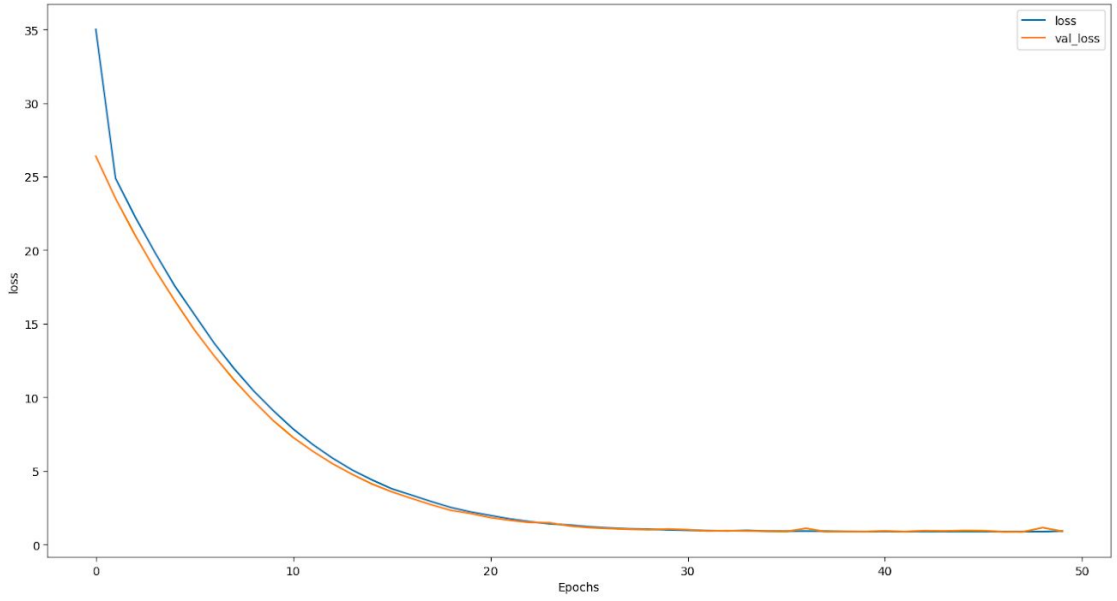

Fig. 11 shows the training vs validation loss

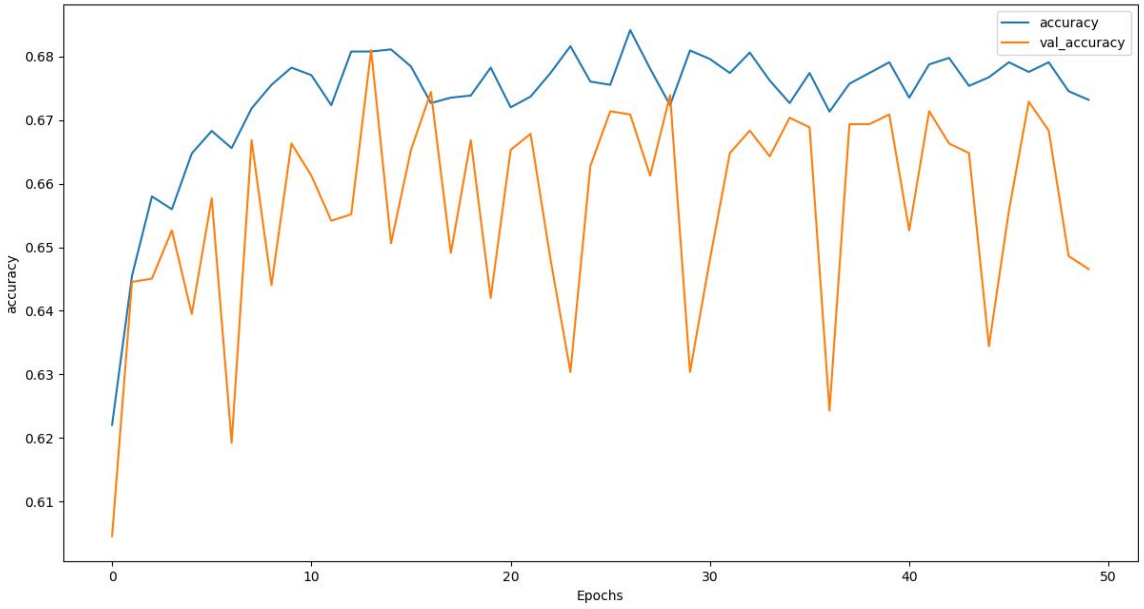

Fig 12. Training vs Validation accuracy

12. Evaluation

12.1. Web Application

The web application successfully allows and validates file uploads, even allowing users to upload files which do not have a '.csv' file extension. The api endpoint '/submit_data' then writes the contents of the user uploaded file in the '/data' directory in a file called '/user_file.csv' on the server.

The web application failed in being able to create the tensorflow datasets that are required to input data to an Tensorflow model in Layers Format (JSON model). The reasons for this are many. Javascript is not a language that is aimed at performing mathematical operations or for easily working with and representing vectors. Our original plan was to have the Python and Nodejs environments running on the server at the same time so that, using Nodejs 'child_process' module, we could spawn a child process that would run the python code. However, we were not able to activate the Pipenv environment that has the dependencies for the Python code installed in it. This meant that we had to switch approaches. Tensorflow offers a javascript and Nodejs compatible version that makes it able to load a model into the server and prepare the data and everything else you can do in python however the API is quite different to the Python version. This meant that while trying to learn how this Tensorflow.js application worked we ran out of time and were not able to finish this part. Although we were not able to convert the user data from javascript array to Tensorflow tensors the trained LSTM RNN model was successfully loaded into the server and can be used to make predictions if you hardcode a Tensorflow tensor with dummy values.

However the server was successful in everything up to that point including all of the preprocessing tasks required for the project. Those include removing all hashtags and preceding text; all @'s and preceding text; all URLs with any protocol (http, htpps, www); all words that contain numbers and removing stopwords; obtaining lemmas. A test file with 10 sample tweets was used to test this, the file is available to view in './data/tesing.csv'. The test file contained occurrences of all of the aforementioned test patterns that needed to be removed. After subjecting the test file to the process of removing noise the results could be gathered by simply printing the output of the remove noise function to the terminal and observing. The same file was used to test the lemmatisation where we iterate over the test file and insert the words one by one into the lemmatiser. The javascript lemmatiser did not have as many words in its dictionary as the one used for the python application, and as such lemmas could not be found for every word. The web server is also capable of using the pre-trained GloVe vectors to replace the words in the corpus, so that the users file now consists of only GloVe vectors and not words.

12.2. Preprocessing

Preprocessing for twitter data is extremely difficult as we have found and leaving yourself with clean data that has as little useless noise as possible is a challenge, particularly for twitter data. The reasons for this are several fold. Firstly, there are large amounts of slang in twitter data and the usage and variety of this slang different depending on your data things like the geographical location of where your dataset comes from and how recent your dataset is. For example, the dataset used in this project is all tweets about US airlines, so the slang used in this dataset is mainly American English slang and the data was gathered in February 2015 and slang, especially online slang, rapidly evolves and slang words become outdated. Therefore, you cannot remove all occurrences of these slang words because

there is no way to know what all of them might be. This itself would be an interesting problem to investigate, using machine learning to recognise slang words. Furthermore, the presence of these slang words means that when the words are input to the GloVe process, both the word that is spelled correctly, along with its potentially many variations such as, ‘thx’ or ‘thnx’ which are both online slang for ‘thanks’, are all assigned different vectors. This negatively impacted the effectiveness of the word embeddings, and in turn LSTM, because all of the places in the corpus where a word for example ‘thanks’ appears in the context of other words, there might instead be ‘thx’ or ‘thnx’. Consequently, the vector that represents ‘thanks’ is not as close to the vectors of the words that appear in the context of its slang variations as it should be, meaning that the embeddings are not encoding as much information as they could.

Secondly, there is a very high amount of vocabulary used in twitter data that most would consider to be stopwords, e.g. ‘to’, ‘and’ or ‘what’, which made it very difficult to decide which words should be in our list of stopwords. Initially, we were using NLTK’s (Natural Language Toolkit) list of stopwords but this resulted in removing large amounts of the text that was in the training tweets. When we got to testing the LSTM for the first time we saw really terrible results, less than 50% accuracy, and one of the tried solutions to fix this bad performance was not removing the stopwords. We immediately saw more encouraging results, ~20% higher accuracy was recorded. From this point we explored custom stopwords lists and ended up on what you can see in the ‘stop_words.txt’ file. We found that removing single letters and two letter words was effective for twitter data.

12.3. GloVe embeddings

The GloVe embeddings did not provide any significant increase in the accuracy of the predictions that were being made by the LSTM. You can see from the results of testing a BOW model on the exact same preprocess and LSTM achieved nearly the same accuracy and nearly the same loss. However the LSTM did not fit the BOW vectors as closely during the initial stages, the loss of both methods levels out and remains constant towards the last epoch.

One of the reasons for the GloVe embeddings not having as much impact on the predictions as we had hoped is because we had to use a dataset with only 5000 samples due to the hardware constraints. For example the original GloVe embeddings were trained on a 4 billion line corpus with 400,000 words. We don't need 4 billion tweets but the results would have definitely been better had the developers been able to train the GloVe embeddings with more data.

In the same vein the quality of the GloVe embeddings depends highly on the quality of the NLP tasks that were performed before the GloVe training begins. Without having a very effective method for removing the noise from the corpus your tweets will have too much information that does not provide the GloVe algorithm with useful features. This is the case with the GloVe embeddings since they are assigning vectors to slang words and spelling errors that could not be removed and is one of the reasons why they did not outperform the BOW vectors.

12.4. LSTM

The tests that were conducted on our LSTM showed that improvements needed to be made and Fig. 11 and Fig. 12 gave us ideas about where we should start looking to fix them. First of all, the dropout method of combating overfit brings no positive effect beyond a drop mask ratio of 0.5, so we alter our model to use the same dropout rates – 50% dropout for all hidden units and 20% dropout for visible units (Hinton et al., 2012). This correlates to dropout rate for the first LSTM layer to be 0.5 and the second LSTM layer to be 0.2, thus getting the most out of the dropout method without needlessly shutting off neurons. We also modified the learning rate of the Adam optimiser that we are using from 0.0005 to 0.05, this resulted in much bigger steps being taken during the gradient descent process of

Adam. We found that changing the value of the weight decay used for the least sum squares regularization did not have any effect on the results and as we approached 0 we saw results get worse and conversely as we got below 1e-9 we also saw the results get worse. After these alterations we performed the testing again to see what results our changes made, Fig. 13 - 14 shows these results.

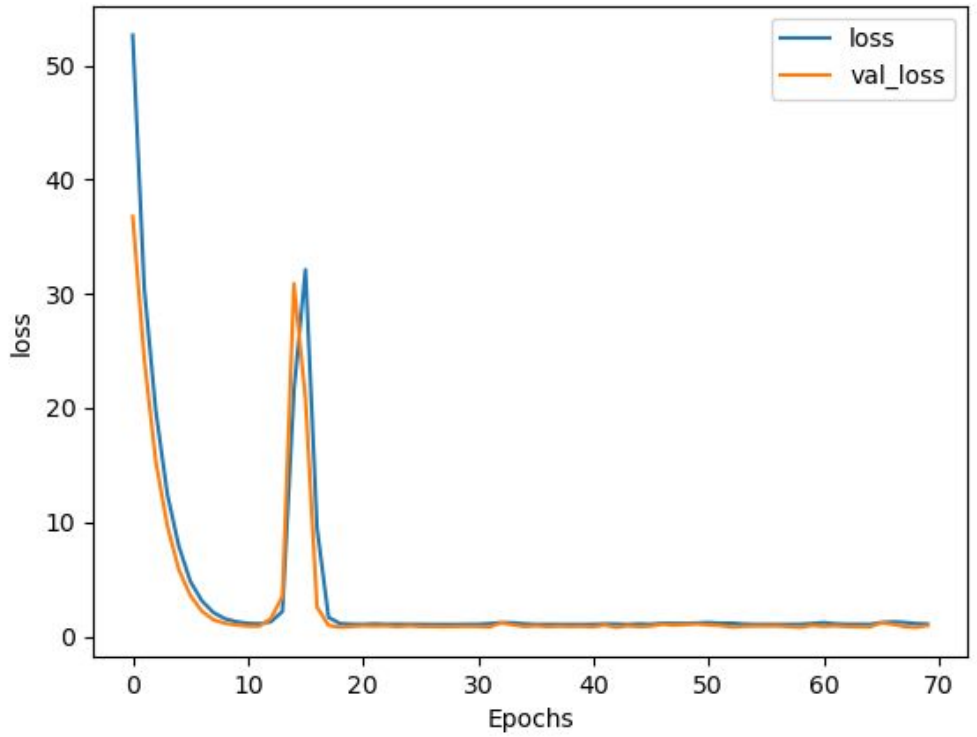

Fig. 13 training vs validation loss after improvements

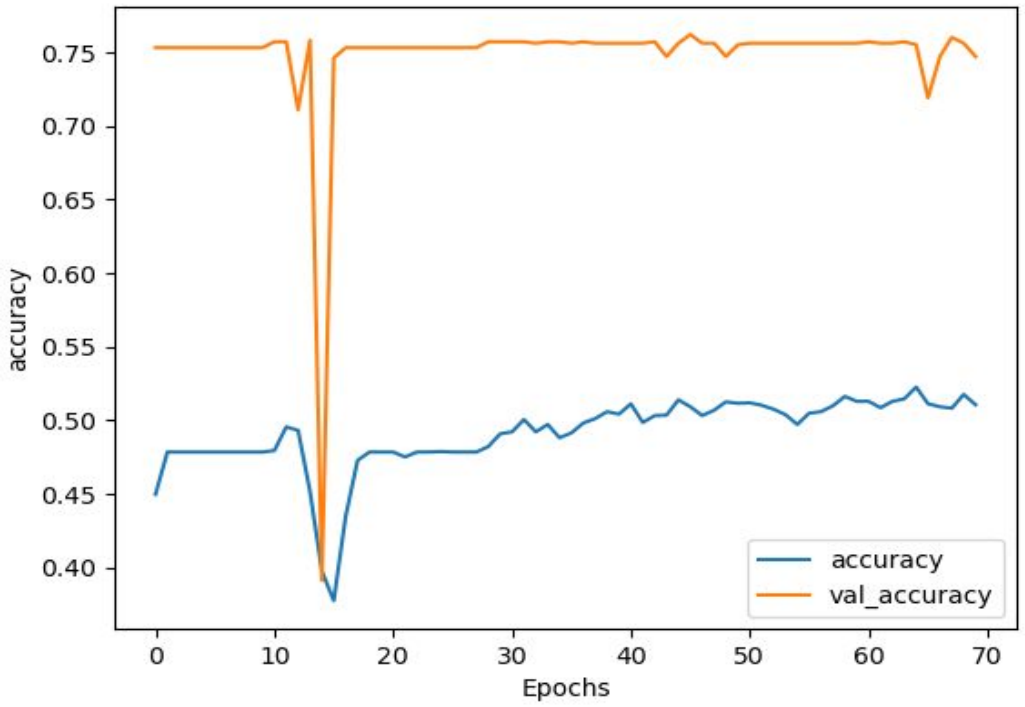

Fig. 14 training vs validation accuracy after improvements

By comparing Fig. 13 to Fig. 11 we can see that the changes made cause the loss to drop twice as rapidly (half as many epochs) as the initial testing. Therefore the number of epochs we are running could have been reduced because if the loss is no longer decreasing towards 0, then the model is no longer learning new information and by running more iterations you give it more chance to overfit. There is a spike/trough in between epochs 10-20 in both the loss and accuracy graphs and I believe this to be down to the amount of noise that is left by the preprocessing stage, causing some epochs to pay too much attention to the noisy data, while during most epochs it is not as impactful. By comparing Fig 8 and 10 we can see drastic improvements in the mean accuracy of the predictions being made on the validation data set.

13. Conclusion

Although there were shortcomings of the artefact, in terms of making predictions through the web browser, considering the design and implementation time allotted as well as the scope of the project. The solution demonstrates the promise of having modern sentiment analysis using GloVe embeddings with a Long Short-Term Memory recurrent neural network deployed on a server and available from the browser. A fully functional Python program that will train its own GloVe embeddings from the provided corpus then train an LSTM model to make predictions of the sentiment of the tweets that the GloVe embeddings represent. The trained LSTM model has then been saved and converted to a JSON file containing all the parameters necessary to make predictions with the model, later deployed into the servers './data' directory and loaded into a Nodejs server application with the developer having little to no knowledge of python, natural language processing or deep learning.

GloVe embeddings were chosen for their proven ability to work better than other embeddings for capturing the semantics in text. Pennington et al,. (2014) found that they achieved ~8% higher accuracy over 30 iterations than Continuous Bag of Words algorithm and ~4% better than skipgram after the same amount. However, the embeddings that were produced in this project performed

marginally better, in terms of their effect on the accuracy of an LSTM, than the standard bag of words model and this is mainly due to the lack of data provided to train the GloVe embeddings, for example Pennington et al,. (2014) used a 6B token corpus (Wikipedia 2014 + Gigaword 5) that had a 400,000 word vocabulary. Given more data, tens of thousands or even hundreds of thousands of tweets, would provide the GloVe embeddings with many more examples to improve the accuracy of its encoding and in turn the accuracy of the LSTM.

By observing Fig. 13 and Fig. 14 we can see that the model does not seem to be overfitting which is one of the largest problems that hinder the generalisability of recurrent neural networks. Apart from the large spike around epoch 15 the loss does not increase, meaning that the prediction made by the LSTM that were correct never got less correct, as in if the model predicted a tweet was positive with 0.80 probability, then this never got worse over epochs. However despite the loss decreasing we do not see the accuracy increase above 76%. This shows that the model has learned to predict on too much noise in the data because the loss decreases but the number of actual predictions it gets right, regardless of if the model predicted 51% or 100% probability, does not increase.

In conclusion, had the develops had access to more powerful machines so that larger amounts of data could be processed then we would have seen more promising results from the GloVe, although the results we did obtain were not a failure, we achieved a validation accuracy for both BOW and GloVe that were insignificantly different, however the results from the GloVe vectors provide much more stable and therefore generalisable results. As for the LSTM, after the improvements we made to the regularisation methods and hyper parameter values we saw a significant increase in its accuracy, validation accuracy increased by ~10%. Given much more data and also finding better NLP processes to reduce noise this model could reach much higher accuracy levels and be good enough to deploy in production. Finally the web application does everything it needs to do except for being able to structure the tweets for input to the LSTM, we were even able to load the pre-trained model into the Node application. After the contents of the file has been converted to GloVe embeddings they then need to be converted to Tensorflow tensor objects in order for the LSTM to be able to use them. This is possible to do however the developers were not able to find a solution in time. Given the numerous problems causes by the closure of the University due to Coronavirus and all of its facilities the production and training time of my project grew exponentially. Also given the scope of the project the developers produced an LSTM model, trained on GloVe embeddings for predicting the sentiment polarity of twitter data that is fully functional and shows the potential of having this type of technology readily available through a web browser.

14. Future Work

For future works the following would be interesting tasks to complete:

- An algorithm to recognise slang or online abbreviations during the preprocessing stage and convert the words to their proper form.
- Rebuilding the Server application as a Flask application written in python so that both application, training the model and using the model are written in the same language and can be used together.
- Deploying this model onto a cloud server with the Google Cloud Platform, since this the model is developed on the tensorflow platform. This would enable training the model on much larger datasets.

15. References

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. Technical report, Stanford University.

Asghar, M., Khan, A., Ahmad, S., & Kundi, F. (2014). *A Review of Feature Extraction in Sentiment Analysis*. Institute of Computing and Information Technology, Gomal University.

Balakrishnan, V., & Lloyd-Yemoh, E. (2014). *Stemming and lemmatization: A comparison of retrieval performances*. IACSIT.

Cheng, G., Peddinti, V., Povey, D., Manohar, V., khudanpur, s., & Yan, Y. (2017). *An exploration of dropout with LSTMs*. Retrieved from https://www.researchgate.net/profile/Gaofeng_Cheng/publication/319185476_An_Exploration_of_Dropout_with_LSTMs/links/59b68e2e0f7e9bd4a7fc4233/An-Exploration-of-Dropout-with-LSTMs.pdf

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. Universite de Montreal.

Das et al, A. (2008). *Topic-Based Bengali Opinion Summarization*. Coling.

Expressjs.com. (2020). *Express - Node.js web application framework*. [online] Available at: https://expressjs.com/ [Accessed 24 Feb. 2020].

Funk, A., Li, Y., Saggion H., Bontcheva K., & Leibold C. (2008). *Opinion analysis for business intelligence applications*. In First international workshop on Ontology-Supported Business Intelligence

Go, R. Bhayani, L.Huang. *Twitter Sentiment ClassificationUsing Distant Supervision.* Stanford University, Technical Paper,2009

Hahn, E. (2016). *Express in action*. Shelter Island, NY: Manning Publications.

Hemalatha, I., Saradhi Varma, D., & Govardhan, D. (2012). *Preprocessing the Informal Text for efficient Sentiment Analysis*.

Hu, M., & Liu, B. (2004). *Mining and summarizing customer reviews*. KDD.

IBM Cognos Content Analytics Information Center, available at http://publib.boulder.ibm.com/infocenter/analytic/v2r1m0/index.jsp?topic=%2Fcom.ibm.discovery.es.ta.doc%2Fiiysalgstopwd.htm

Indurkhya N., Damereau F.J. (2010). *Handbook of Natural Language Processing. 2nd Ed.*, Chapman & Hall/CRC

Layer wrappers - Keras Documentation. (2020). Retrieved 1 May 2020, from https://keras.io/layers/wrappers/

Limsopatham, N., & Collier, N. (2016). *Bidirectional LSTM for Named Entity Recognition in Twitter Messages* Retrieved from https://www.repository.cam.ac.uk/bitstream/handle/1810/261962/Limsopatham_and_Collier-2016-WNUT2016-VoR.pdf?sequence=1&isAllowed=y

LK-W, T., J-C, N., Y-L, T., & K, C. (2015). *Sentiment analysis using product review data* (p. 3). Retrieved from https://link.springer.com/article/10.1186/s40537-015-0015-2

Node.js. (2020). *Node.js*. [online] Available at: https://nodejs.org/en/ [Accessed 24 Feb. 2020].

Nowak, J., Taspinar, A., & Scherer, R. (2017). Conference PaperPDF Available LSTM Recurrent Neural Networks for Short Text and Sentiment Classification.

Oluwatosin, H. (2020). *Client-Server Model*. IOSR Journal of Computer Engineering. Retrieved from https://www.researchgate.net/profile/Shakirat_Sulyman/publication/271295146_Client-Server_Model/links/5864e11308ae8fce490c1b01/Client-Server-Model.pdf

O. O. Abiona *et al*., "Architectural model for Wireless Peer-to-Peer (WP2P) file sharing for ubiquitous mobile devices," *2009 IEEE International Conference on Electro/Information Technology*, Windsor, ON, 2009, pp. 35-39.

Pak, A., & Paroubek, P. (2010). *Twitter as a Corpus for Sentiment Analysis and Opinion Mining*.

Pang, B., & Lee, L. (2004). *A sentimental education: Sentiment analysis using subjectivity analysis using subjectivity summarization based on minimum cuts*. ACL.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). *On the difficulty of training recurrent neural networks*.

Pennington, J., & Socher, R. (2014). *GloVe: Global Vectors for Word Representation*. Stanford University, Stanford.

Raccoon, L. (1995). *The Chaos Model and the Chaos Life Cycle*

ResearchGate, (2020). [image] Available at: https://www.researchgate.net/figure/terative-Waterfall-model-Advantages-I-Much-better-model-of-software-process-II_fig1_305863548 [Accessed 29 Feb. 2020].

ResearchGate (2020). [image] Available at: https://www.researchgate.net/figure/The-iterative-development-model_fig1_236170589 [Accessed 29 Feb. 2020].

TensorFlow. (2020). *TensorFlow*. [online] Available at: https://www.tensorflow.org/ [Accessed 13 Feb. 2020].

Tsai, W., Sun, X., & Balasooriya, J. (2010). *Service-Oriented Cloud Computing Architecture*. Beijing: IEEE.

16. Appendices

School of Computing final year project

Sam Toguri

PJE40

Project Initiation Document

Detecting Sentiment in Twitter text data

Project Initiation Document

1. Basic details (mandatory)

Student name:	Sam Toguri
Draft project title:	Detecting Sentiment in Social Media Interactions
Course:	Computer Science

Client organisation:	University of Portsmouth
Client contact name:	
Project supervisor:	Mihaela Cocea

2. Degree suitability

I am studying Computer Science as my degree and I have chosen to create a program that will detect the sentiment in social media interactions such as Tweets and comments. This project will focus heavily on Natural Language Processing (NLP) techniques which are currently at the forefront of computer science, data analytics, human-computer interactions, machine learning, and many other topics. It will also focus heavily on deep learning through the implementation of a recurrent neural network. It will give me a chance to display my programming abilities in new and exciting ways using a variety of technologies, and will also give me the opportunity to learn more about the previously stated topics which are all extremely relevant in today's software development industry. It is for these reasons that I believe my project is going to be challenging, yet is very suitable to the degree I am enrolled in.

3. Outline of the project environment and problem to be solved

This type of software would be aimed at anyone who needs to analyse large amounts of social network interactions on a particular, or a range of topics to gain a better understanding of the wider public opinion on said topic/s. Companies such as Facebook, Google and Youtube who provide targeted ads to consumers based off of their online interactions make extensive use of this technology to accurately target users with relevant ads that they would prefer to see. Analysing this data manually introduces a whole host of issues such as human errors and time-consumption which results in less accurate analysis of the data, and if the dataset is extremely large then it would take a very long time to do and would be very boring for a person. This makes it very hard, if not impossible, for marketing agencies and other ad providers to generate ads that are tailored to each individual without making use of some form of sentiment analysis. The technology allows businesses to be proactive, instead of reactive, to public opinions. For example, the Obama campaign in 2012 used sentiment analysis to gauge public opinion on things like policy announcements and campaign speeches. This gives businesses a huge competitive advantage

4. Project aim and objectives

This project aims to categorise social media interactions based on the sentiment in the text. The program will use a Long Short-Term Memory (LSTM) recurrent neural network to classify social media interactions into three categories: positive, neutral and negative. There are several objectives in meeting this aim. Firstly I will identify what research has been conducted using in related areas and use that information to identify aspects that I can improve. After this has been completed, the functional and non-functional requirements will be derived to inform the design and architectural aspects of the system. The system will then be built according to the architecture and other design aspect, finally being tested against a

criteria. Finally the system must be tested to ensure a suitable level of accuracy in the predictions that it will make on whether comments are positive, neutral or negative.

5. Project deliverables

I will produce requirements and design specifications to set out clearly a model of what I'm going to build and why I'm going to build it that way. Test strategies, clearly outlining the testing that i'm going to do, will also be produced. A project report detailing what difficulties I encountered and the aspects that could be improved will

The program will have to pre-process the data to get it ready for the LSTM. This involves denoising the noisy raw data, and turning it into binary vectors that can be plotted on a graph. This follows several steps:

- Word tokenization to split each string into its component words.
- Lemmatization and stemming to remove inflectional endings and derivational forms and to return the base or dictionary form of a word, which is known as the 'lemma'. For example: am, are, is => be and dog, dog's, dogs => dog, such that a sentence like 'I am taking my dogs for a walk' outputs something along the lines of 'I be take my dog for walk'.
- Removing stop words from the dataset. Stop words are generally the most common words that are used and ones that tend to have less relevance than others such as 'and', 'the', 'a'.

Now that the data has been denoised, the strings can be turned into binary vectors that can then be used and interpreted by the ML algorithm. Feature extraction is performed using bag of words model. This turns the strings/sentences into binary vectors, based on how often they occur. Using logistic regression algorithm, the program can now be trained.

6. Project constraints

Constraints include my knowledge and ability with NLP techniques. Although I have previously implemented other machine learning algorithms I have never done anything with NLP so there will be some things that I am going to need to teach myself, however there are plenty of online resources that mean it won't be too much of an issue. There are also time constraints affecting whether or not I finish everything in time, including all the testing. These include progress reports and other deadlines. Another constraint that I have encountered is finding enough background research regarding the dataset that I am going to use to train my algorithm.

7. Project approach

- I will conduct research on the type of work that has already been conducted using the dataset from https://www.kaggle.com/crowdflower/twitter-airline-sentiment.
- I am going to identify the requirements by looking at what people are currently doing, research wise and why they are doing it that way. After I have established that I will start to consider how to improve their methods and this will form the basis of my requirements for this project.
- The skills required for this project are as follows:
    - Python programming.
    - Knowledge of NLP preprocessing techniques.
    - Knowledge of deep learning and recurrent neural networks.
- I am going to need to teach myself more about the preprocessing techniques currently being used and also what is being researched e.g. feature extraction models like *GloVe,* and what should my program consider 'noise' in order to produce the most accurate predictions and also to perform research on new techniques.
- The production stage will follow an agile approach. This will help to evolve the requirements and will reduce the rigidity of the development. It will allow me to create a better structure around my software deliverables, which will hopefully result in a better, more cohesive program.

8. Log of risks

The main risk of this project will be in implementing deliverables which might break the system. However I will be using Git to provide source control for all of my code, so if I merge a production branch into the master branch and it breaks the copy stored on the master branch, then i can revert to the previous version and debug the production branch.

9. Starting point for research.

There is a plethora of data on the internet detailing how sentiment analysis is used in industry and how to perform sentiment analysis. I have read many articles on medium.com e.g. (https://medium.com/datadriveninvestor/sentiment-analysis-on-customer-tweets-nlp-c0eeaeffd19a)
That detail the exact process of sentiment analysis. There is also a very interesting and informative paper in the "Journal of Medical Internet Research" by "Felix Greaves[1,2], MBChB; Daniel Ramirez-Cano[2], PhD; Christopher Millett[1], PhD; Ara Darzi[2], MD; Liam Donaldson[2], MD" on "Use of

Sentiment Analysis for Capturing Patient Experience From Free-Text Comments Posted Online”

10. Breakdown of tasks

1. Define system requirements ((non)functional requirements).
2. Produce system design and architecture document.
3. Build the artefact:
    a. preprocessing tasks involving removing the noise in the data and preparing it for input into the analysis algorithm. This will use lemmatization and stemming techniques, the removal of stop words and finally feature extraction, like *GloVe*.
    b. Implement a linear regression algorithm to make predictions on what category a feature set belongs to.
    c. Produce user interface to upload new data, and display analysed data. Users will be able to upload CSV files.
    d. Create connection to Neo4j database to store user inputted data.
    e. Dockerize the system so that it can be deployed and behaves the same on any device that it is run on.
4. Test software artefact for adequate accuracy levels of predictions and for bugs.

11. Project plan

What are you going to do when? (This may be an attached output from MS Project etc.) What risks to the success of the project have you identified? What steps can you take to minimise them? Note that plans can change over the course of the project, so this plan should be maintained.

12. Legal, ethical, professional, social issues

With regards to the dataset that I will be using to train my algorithm, I found it online on an open source website, so as long as I properly reference where the data came from in my report, then there is no legal or ethical issue.

Certificate of Ethics Review

Project Title: Online sentiment analysis for twitter data

Name: Samuel Makoto Toguri **User ID:** 781439 **Application Date:** 12-Apr-2020 11:04 **ER Number:** ETHIC-2020-458

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative for the School of Computing is Carl Adams

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **SOC**
What is your primary role at the University?: **UndergraduateStudent**
What is the name of the member of staff who is responsible for supervising your project?: **Ella Haig**
Is the study likely to involve human subjects (observation) or participants?: **No**
Are there risks of significant damage to physical and/or ecological environmental features?: **No**
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: **No**
Does the project involve animals in any way?: **No**
Could the research outputs potentially be harmful to third parties?: **No**
Could your research/artefact be adapted and be misused?: **No**
Does your project or project deliverable have any security implications?: **No**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor signature: **Date:**

A. GloVe training command line output (last 10 iterations)

```
        Beginning iteration of training  40 ...
                Done (cost =  53.54272219799444 )
        Beginning iteration of training  41 ...
                Done (cost =  52.369500110445905 )
        Beginning iteration of training  42 ...
                Done (cost =  50.78874541076703 )
        Beginning iteration of training  43 ...
                Done (cost =  49.8743704109003 )
        Beginning iteration of training  44 ...
                Done (cost =  48.590919079754386 )
        Beginning iteration of training  45 ...
                Done (cost =  47.569045079345 )
        Beginning iteration of training  46 ...
                Done (cost =  46.439321595709394 )
        Beginning iteration of training  47 ...
                Done (cost =  45.61945918418753 )
        Beginning iteration of training  48 ...
                Done (cost =  44.572369893622295 )
        Beginning iteration of training  49 ...
                Done (cost =  43.74459989295695 )
        Beginning iteration of training  50 ...
                Done (cost =  43.040092061436006 )
GloVe embeddings trained, number of vectors =  5291.0
```

B. Output during training of the model

```
Training model...
Train on 2989 samples, validate on 997 samples
Epoch 1/20
2989/2989 [==============================] - 501s 167ms/step - loss: 51.6267 - accuracy: 0.4165 - val_loss: 36.7955 - val_accuracy: 0.7533
Epoch 2/20
2989/2989 [==============================] - 490s 164ms/step - loss: 30.5014 - accuracy: 0.4888 - val_loss: 24.5268 - val_accuracy: 0.7663
Epoch 3/20
2989/2989 [==============================] - 479s 160ms/step - loss: 20.0932 - accuracy: 0.5025 - val_loss: 15.5902 - val_accuracy: 0.7533
Epoch 4/20
2989/2989 [==============================] - 476s 159ms/step - loss: 12.8441 - accuracy: 0.5239 - val_loss: 9.8841 - val_accuracy: 0.7663
Epoch 5/20
2989/2989 [==============================] - 481s 161ms/step - loss: 8.0011 - accuracy: 0.5313 - val_loss: 5.9850 - val_accuracy: 0.7663
Epoch 6/20
2989/2989 [==============================] - 477s 159ms/step - loss: 4.9290 - accuracy: 0.5537 - val_loss: 3.9035 - val_accuracy: 0.5948
Epoch 7/20
2989/2989 [==============================] - 481s 161ms/step - loss: 3.1764 - accuracy: 0.5547 - val_loss: 2.3769 - val_accuracy: 0.7653
Epoch 8/20
2989/2989 [==============================] - 481s 161ms/step - loss: 2.2038 - accuracy: 0.5450 - val_loss: 1.6297 - val_accuracy: 0.7683
Epoch 9/20
2989/2989 [==============================] - 479s 160ms/step - loss: 1.7296 - accuracy: 0.5657 - val_loss: 1.2086 - val_accuracy: 0.7623
Epoch 10/20
2989/2989 [==============================] - 482s 161ms/step - loss: 1.3621 - accuracy: 0.5785 - val_loss: 1.0542 - val_accuracy: 0.7653
Epoch 11/20
2989/2989 [==============================] - 487s 163ms/step - loss: 1.2313 - accuracy: 0.5741 - val_loss: 1.0695 - val_accuracy: 0.7533
Epoch 12/20
2989/2989 [==============================] - 482s 161ms/step - loss: 1.3621 - accuracy: 0.5785 - val_loss: 1.0542 - val_accuracy: 0.7653
Epoch 11/20
2989/2989 [==============================] - 487s 163ms/step - loss: 1.2313 - accuracy: 0.5741 - val_loss: 1.0695 - val_accuracy: 0.7533
Epoch 12/20
2989/2989 [==============================] - 482s 161ms/step - loss: 1.2018 - accuracy: 0.5781 - val_loss: 0.9787 - val_accuracy: 0.7643
Epoch 13/20
2989/2989 [==============================] - 479s 160ms/step - loss: 1.2690 - accuracy: 0.5835 - val_loss: 1.0439 - val_accuracy: 0.7583
Epoch 14/20
2989/2989 [==============================] - 509s 170ms/step - loss: 1.1875 - accuracy: 0.5795 - val_loss: 1.1083 - val_accuracy: 0.6720
Epoch 15/20
2989/2989 [==============================] - 500s 167ms/step - loss: 1.2038 - accuracy: 0.5708 - val_loss: 0.8872 - val_accuracy: 0.7703
Epoch 16/20
2989/2989 [==============================] - 486s 163ms/step - loss: 1.2245 - accuracy: 0.5778 - val_loss: 0.9719 - val_accuracy: 0.7653
Epoch 17/20
2989/2989 [==============================] - 510s 170ms/step - loss: 1.1753 - accuracy: 0.5851 - val_loss: 0.9635 - val_accuracy: 0.7513
Epoch 18/20
2989/2989 [==============================] - 501s 168ms/step - loss: 1.1700 - accuracy: 0.5798 - val_loss: 0.9287 - val_accuracy: 0.7683
Epoch 19/20
2989/2989 [==============================] - 499s 167ms/step - loss: 1.2374 - accuracy: 0.5868 - val_loss: 1.0436 - val_accuracy: 0.7553
Epoch 20/20
2989/2989 [==============================] - 509s 170ms/step - loss: 1.2413 - accuracy: 0.5945 - val_loss: 1.0347 - val_accuracy: 0.7492
997/997 [==============================] - 48s 48ms/step
```