5th of June 2020

School of Computer Science and Informatics, Cardiff University

1920-CM3203 – One semester Individual Project – 40 credits

USING SENTIMENT ANALYSIS TO IMPROVE EMOJI PREDICTIONS IN TWEETS

Final report submitted as part of the Final Year Project for the degree of

BSc Computer Science

By Fani Krasimirova Noncheva

Supervisor: Luis Espinosa-Anke

Moderator: Matthias Treder

Acknowledgements

I want to thank Alexandra Nicholas and Brendan Rhys Lamb for supporting me in this time of global crisis, whilst I had to write this project on my own with no relatives to turn to, stranded a continent away from home, their support helped me stay strong and focus on work despite all the stressful events of this year. I am glad to have found a family overseas that I can rely on and cherish.

I would also like to thank my supervisor, Luis Espinosa-Anke, for providing guidance and insight, where needed, as well as being very understanding and supportive of me when I applied for extenuating circumstances due to personal reasons.

Notable mentions include Anastasia Ugaste, a friend and roommate of mine that assisted me with her profound knowledge of data science when I was first starting this project, Michael Stachowicz, my supportive partner, my family for the fiscal security they provided for me during the pandemic and to the authors and creators of machine learning mastery.

Table of contents

Table of figures

Abstract

This project is focused on exploring the relationship between sentiment analysis and emoji prediction. The data used within the project is a variety of tweets, collected from the USA at alternating time periods. Ultimately, this is an analysis of tweets perceived as natural language and how this language is connected to the use of emojis. The goal is to determine whether sentiment analysis can prove useful in the field of emoji prediction. To accomplish this natural language processing task, it employs a variety of neural networks, developed with TensorFlow, and more specifically Keras. The architectures explored vary between recurring neural networks and convolutional neural networks, as well as a few merged model structures.

Introduction

Emojis are essentially a set of ideograms and smileys that are becoming more and more commonplace as a way for people to express themselves online. They have even went as far as being recognized by Oxford Dictionaries as word of the year in 2015. [19] Considering the rapid adoption of emojis as a medium of expression, they hold their own implications, semantics and linguistic traits. That would imply their importance in text analysis is rising. As a relatively new type of data they require further and expanded research, since currently they are being somewhat neglected in the area of text analysis. [3] To understand these newly adopted forms of expression, we need to analyse their behaviour compared to more traditional means of communication such as written text.

For this project I will focus on the expression of emojis in social media. Emoji data is common to come across in online communication, and as such there is plenty of data sources to be explored. [2] However, it is of utmost importance to find data that has a healthy balance between text and emoji to assist in the task of comparing emoji to natural language. Seeing as emoji is commonplace in texts, social media posts and even making its way into certain articles online, the type of data I have chosen to focus on for this project is tweets that contain emojis.

I chose to analyse twitter data as by the end of 2019 there are over 330 million people are active on the platform monthly. [13] As such a widely used platform it can offer an up-to-date, realistic sample of emoji uses, as well as a variety of expression in the realm of sentiment analysis, resulting in perfect conditions for this analysis task. The nature of twitter data is quite favourable for analysis as it offers condensed and concise samples with just enough context and keywords to map and analyse, without any significant data noise or cluttering.

Data aside, emojis and sentiment seem to have a correlation that is frequently explored from the angle of sentiment. There has been past research pointing towards emoji and sentiment having a valuable correlation.[28],[23],[32] The focus of this project is to look at this relationship through a different angle that explores the relationship between sentiment and emoji, this time using sentiment to predict emojis, hopefully shedding light on those new and mysterious ways of expression. [16]

With all the above into consideration, the scope of this project lies mainly in the realm of natural language processing (NLP), which is a field tightly entwined not only with artificial intelligence (AI) but linguistics as well. Due to the broadness and complexity of the field, there is a vast disparity between the difficulty and requirements of the different subjects within it, hence why this project picks one of the better explored subjects – namely sentiment analysis, and a newer, less explored part of the field, the use of emojis in natural language exhibited online and mainly on social media.

Machine learning is not a new concept, yet it only recently started gaining traction among academics and the industry alike. Though since I have set my sights on a narrower application of machine learning – deep learning. In this project I chose to develop neural networks to accomplish my task, focusing on supervised learning. To achieve this, I employ the Keras API in Python to build and test my models, as it is one of the most popular and widely used APIs that are currently in use for machine learning projects.

Sentiment analysis has a long-standing history in the area of NLP as stated above. It is a well-researched topic with a variety of uses in academia and the advancement of tech that enhances day-to-day life. Since the broadness of complexity in sentiment analysis is wide, I chose to explore a rather simple variation focusing on the positive and negative sentiment shown through text. When it came to implement and understand the implications the task offered, there was outstanding research coverage on the most optimal techniques to analyse text data and determine sentiment as stated in some of the referenced studies. [20]

To accomplish the above I use deep learning models to explore both problems. The similarities mostly end here, as I use varying techniques and compare them to determine which is optimal for each problem and why.[1] The problems are similar yet, as emoji classification is a bigger categorical problem, that will require a more complex solution, determining the positive, negative or neutral sentiment of a text sample is a much simpler classification process. Both problems are categorical, and both can run into some issues later regarding sarcasm and finer ways of expression. [7]

Background

There is a variety of research papers on the topic, covering different implementations and several solutions to emoji prediction and sentiment analysis. For one, there is an in-depth analysis of the different techniques used for sentiment analysis in the realm of AI [20], however, my primary interest lies with machine learning implementations, specifically neural network ones. For NLP there seems to be a consensus that the most optimal types of models around are based on either Long Short-Term Memory (LSTM) based neural networks and convolutional neural networks. [8], [27]

There seems to be a wider variety of LSTM-based models that are implemented for this specific task, mostly due to LSTM's ability to analyse the given tensors using a 'timestep' allowing it to have a more in-depth grasp of the context of the provided sequential data. The type of LSTM layers used varies greatly, from Bi-sense, to attention-based and the ordinary LSTM that can also prove very useful in analysing sequential data.

The Bidirectional recurrent neural networks boil down to the concatenation of two independent RNNs together. The structure allows both networks to have access to backward and forward information about the sequence at every given timestep. Whereas the attention LSTMs focus on implementing an encoder-decoder dynamic in order to increase the accuracy of the model providing 'context vectors' that enable the LSTMs to more adequately recognize patterns in sentences or tweets. A more in-depth showcase of its impact on sentence data can be found in the referenced studies. [31] There are two types of attention: hard attention and soft attention their focus being crisp making of decisions via context vector (hard) and implementing the context vector in the context of weights (soft), respectively explored in more detail within 'Show attend and tell'. [39]

Apart from that there are studies of models made with convolutional neural networks, despite their more common use in image processing. The main idea behind this approach being to implement them on a 1D convolutional space. The reason for this is the way the convolution treats the matrices parsed in the model. The simplest way I can explain convolution is from a mathematical standpoint: it is a mathematical operation that takes two functions and produces a third function, which demonstrates how one affects the other. In NLP convolution is usually used in its 1D variation, as the dimensions refer to the way the data is being convolved and processed. [40]

The further in-depth description aside, they show quite promising results regarding classification problems such as detecting emotion in tweets [30] and the overall more classical problem of approaching sentiment analysis of tweets. [26] The findings of them show that they can be a useful tool in sentiment analysis yet are much more prone to overfitting and require a much more steady and experienced approach to deliver satisfactory results in metrics and performance compared to their RNN counterparts.

It is important to note that this project is inspired primarily from a SemEval task during the year of 2018 regarding emoji prediction in tweets. [4] The information on the competing teams and their approaches is quite intriguing and I have attempted to emulate some of the models suggested to be the most successful. The best performing models when observing the task of emoji prediction seem to be based primarily on attention and bidirectional LSTM layers in varying configurations. [6],[14] Those solutions explore not only the efficiency of using embeddings for the purpose of emoji classification but also offer a glimpse into the use of n-grams and present their solution to be closer to support vector machines (SVMs). [14]

With that information considered I try to explore both simpler variations of LSTM recurrent neural networks, as well as taking a closer look at the SVM approach of handling the problem of emoji prediction. A choice to try out and compare the most basic approach with convolutional networks has also been made on my part within the scope of the project, that can be seen in further depth below.

Data sources

For this project I have used two resources that are readily available online for academic use, so before carrying on, I will describe them to give context. The sentiment dataset I used is from the Sentiment 140 project related to twitter sentiment analysis.[25] It is contained within archive consisting of two csv files, the training one, that offers 1,600,000 tweets with corresponding labels of negative (0), positive (4) and neutral (2), plus, a testing set for evaluation with the same labels.

The second external resource for NLP that I use is expanded upon in the implementation section and that is Glove embeddings. [18] There is a range of pre-trained versions for different purposes available in a variety of dimensions depending on the user's needs.

The last remaining emoji dataset is courtesy to the researchers behind the SemEval task of 2018, [4] one of which agreed to provide me with the dataset they used for their task.

Approach & Design

The approach to the problem involved research into the basics of the project's structure since I am new to the type of problem I chose to take on. I spent the initial two weeks researching what was available on the library search regarding my chosen field and the ways it were implemented. It helped me better understand the constraints and possible blocks I could encounter throughout the process.

Thanks to the initial first week of research I was capable of promptly planning out my time and setting achievable goals in the initial report I submitted. The second week was focused on the practicality of the implementation and what or how to use. At the time, unfortunately most available resources operated in TensorFlow 1.x, meaning that most of the research and tutorials I came across were in that said version. That will become relevant later in this report as I discuss the issues, I ended up having with that.

In the remaining weeks I finally got down to implementing and tweaking the models for both sentiment analysis and emoji prediction. My approach to the problem involved three simple steps: to find the easiest possible solution to my problem, adapt it to my problem and then investigate ways to improve it.

The data is to be analysed in two ways, the simpler of which is sentiment analysis. Sentiment analysis is accomplished in natural language processing by using artificial intelligence to analyse text and after doing so, determine the text's sentiment in simple baseline versions, that often means 'teaching' the AI to distinguish between positive, negative and neutral sentiment. More complex variations of sentiment analysis can try to capture polarity [36] and other more sensitive differences in text input like irony, comedy etc.

The topic is extensively researched with a variety of papers and past studies to investigate on clues in how to develop, and even better existing analysis structures.[15] With that said, there is quite a variation of models available depending on the area of research and the goals that they are looking to achieve through it. With that said, for the current problem at hand, there have been several iterations in trying to better sentiment via the use of emojis, less so the other way around. [24]

Due to the complexity of emoji prediction and the sheer number of variables to consider and tweak, I chose sentiment analysis as the anchor of this NLP project, as it has a rather well-established formula for development. With that said, it usually involves several steps that boil down to the gathering of data, clearing it from noise, building a model to analyse the data and lastly, tweak it to improve its predictions.

Turning towards the emoji prediction section of the project, it possesses much less related literature due to the topic being relatively new. Especially considering emojis have only been introduced as commonplace in Unicode 5.2 in 2009, by which point 722 of them were already in wide use by the Japanese public. Currently in Unicode 8.0, there are over 1.6 thousand emojis present. [33],[5] While emojis are widely covered by the media, they are only recently gaining the attention of linguists and analysts alike. [22]

Due to the sheer number of individual emojis listed above, the project focuses on the twenty most popular ones. The emoji prediction problem can be reduced to a text classification problem. Except rather than the categories being traditional text genres or text patterns revealing sentiments, but instead focused on determining the individual characteristics of tweets that are related to a specific emoji. Such a classifier is significantly more difficult to implement than sentiment analysis due to the number of variables (words) to be classified within a range of over a dozen categories (emojis).

Given that knowledge I chose to attempt both convolutional and LSTM architectures to tackle the problem of emoji prediction. Speaking of the convolutional approach, after a few attempts at constructing a more efficient and adequate variation of the base convolutional model I faced greater challenges and the results of my work seemingly didn't pay off when it came to performance and metrics (see graphs in implementation for reference).

Another issue I faced was the lack of tutorials on CNNs apart from their primary use in image processing. There seemed to be an overwhelming amount of CNN tutorials and guides, but most of them were focused on the application of CNNs in object recognition, image processing and classification. There was rarely mentions of using them for bigger classification problems, so I chose to attempt using other options in my search for a solution.

Due to this I chose to focus on the RNN approaches available to me to tackle solving the problem. There were several avenues to explore, though SimpleRNNs were quickly ruled out as a long-term solution due to their lossy performance and overall literature pointing against using SimpleRNN for most problems, as it essentially is just a worse variation of the long short-term memory RNN.

Once I moved on to RNNs, I found it easier to understand the reasoning behind the way they interpret text and retain its semantic properties over several timestep iterations. That prompted me to move on to LSTM layers, since a high proportion of the literature in references and background sections point towards them as the solution with highest likelihood of success apart from CNNs.

CNNs gave a competitive result to my initial LSTM model, however, the loss metric had improved steadily, as listed below. That prompted me to choose to focus on developing an emoji prediction model using LSTM layers. With LSTMs I found it easier to envision the model's possible ways for improvement as well as the way the weights would be distributed.

My final attempt includes bidirectional LSTMs that have been proven to provide good results when it comes to working with text sequences towards solving a text classification problem. My results with them show a similar trend compared to the previously mentioned variations of layers.

When it comes to implementation, a similar approach to the one generally used in sentiment analysis is valid for the emoji prediction part of the project. However, it is important to note that its categorical nature complicates the process of building and tweaking the model.[10] To combat this, I investigated different methods of optimizing categorical queries that relate to NLP as well as a competition task that was related to predicting emojis based on given tweet's text. [4]

Despite the difference of the use and context of the data, the sentiment analysis model and the emoji prediction, the structure of these models is very similar. Considering the emoji data provides many categories, that increases the computational complexity, requiring more in-depth understanding, as well as humble tweaks to the structure, to achieve satisfactory results. [21]

Data processing

The start to any successful machine learning project is the proper processing of data. To achieve this, I use a separate python file to pre-process the text initially given in the datasets. Since I use two separate datasets to train my sentiment analysis model and the emoji prediction, I have had to work around the way I store and extract the data. I have chosen to keep the original datafiles intact for security purposes, as I do not want to damage the data via altering and/or processing it incorrectly.

For the sentiment analysis dataset, which is contained within a CSV file I utilize the python package Pandas, to adequately extract and process the data by rows and columns. It is more efficient than the standard python libraries as it allows for fewer lines of code with higher efficiency. As for the emoji prediction dataset, I use functions that I have written myself. I have split the processing in two sections: one that is done to create a more simplified txt file version of the data to be parsed, and a second section done right before running through the model.

Why focus on data processing?

Since I'm using deep learning neural network models to accomplish my project, it's of vital importance to carefully moderate the number of variables used to predict emoji and sentiment alike, in order to ensure that the predictions can be as accurate as possible and find the most important links between words and certain emojis. To achieve this, I use several data processing tools to reduce noise and the text corpora to minimise the impact of redundant data that is only essential due to the constraints of human communication.

Initial pre-processing

Due to the types of data involved and their different nature, the tweets data requires more attention. In lieu of looking for pre-existing processing, I focused on creating something that was simple enough to cater to my needs, hence putting research into a variety of tools within the programming language to ensure I do not reinvent the wheel. [29] Each of those tools has been specifically chosen to aid with the data's ease of processing by a machine learning algorithm and ensuring its integrity. Listed in order: lowercasing, multiple space removal, removal of punctuation, special characters and numbers, plus lemmatization and removal of stop words.

The simplest two functions that I use, are the bare minimum when it comes to textbook machine learning data processing: removing excess spaces and using lowercase. While lowercasing has debatable impact on the semantics [9], its benefits outweigh its risks, especially when it comes to processing tweets. In social media, proper capitalization is often scarce, while often it is used as more of an emphasis rather than clear distinction between separate entities (e.g. the word 'apple' and the company 'Apple'). Tweets can also be referred to as informal language, lacking complexity of more elaborate and structured texts such as scientific papers, articles, essays etc. Hence, capitalization losing its nuanced role, and rather becoming more of an obstacle towards the finding of specific correlations.

The second function is regex-based and focused on clearing whitespace from the tweets by replacing large quantities of whitespace with a single space. Apart from being handy regarding reducing the size of the tweet stored in the string, it is also used to aid in the splitting of the text data into tokens for later processing. Since the Keras tokenizers detect spaces and determine where to 'split' a string into individual tokens based on that. In older versions multiple spaces were found to create 'empty' tokens that had high frequency, skewing the model's ability to focus on more important tokens.

The third and most crucial function, regex based as well, is concerned with the removal of all characters that are not alphabetical or spaces, including punctuation and special characters. That holds high importance as it keeps the data clear of any external symbols that might alter the resulting tokens and it prepares the data for the following stage of lemmatization. As it only leaves in words that the lemmatizing function can simplify and adequately process.

Before I run the lemmatization on the data, I employ the use of the NTLK toolkit's stop words. The stop words include an array of prepositions, articles, modal verbs, pronouns, and other words of high frequency in language that can prove redundant when looked at through a text-analysis prism.[28] Due to the nature of tweets and their informal nature mentioned above, they can contain certain characters that don't belong in the NTLK Stop words set, so there have been additions that I've noticed such as the frequency of the word 'user', and common shortened, alternatively misspelled variations of modal verbs and pronouns. So that more tokens of low importance and high occurrence can be cleared, shown to improve the model's ability to focus on more vital tokens provided that may lack a high rate of occurrence.

Sadly, later in my research I discovered that this function was in fact hindering my models' performance, so I had to cut it out of the data processing pipeline to ensure better results. My hypothesis on why that is, is the high likelihood that it interferes with the sequences' similarities by removing the words that bind the sentence structure together, and thus abstracting the sequences to the point of there being very little correlation between the individual tweet sequences.

Due to the many categories that are implied in the machine learning algorithm, I tried to keep the data relying on as few variables as possible to improve its predictions, hence why I was seeking to remove the redundant words that would pop up frequently and lack any serious contribution to the predictions. In other cases, it may be beneficial to keep that part of the data (such as text generating via machine learning), but in this specific case, it proves redundant.

Then, the process of lemmatization appears in the pipeline. It's essentially grouping words, or 'lemmas', together if they are inflected forms of a word so that they can be analysed as a single entity. That is especially useful when trying to find patterns in language, as it simplifies the different forms of verbs to a more accessible and simplified version, allowing the model to recognize patterns more easily. It is also a good tool to use against misspellings of words, as they are also included in the NTLK toolkit for lemmatization.

The above concludes the pre-processing section of the pipeline I have made for this project. I have chosen to separate my script for data processing from the model training and model evaluating files, to ensure easier access and reading of each component, as well as the ability to run them independently, depending on the occasion.

Tokenization

The secondary processing is essentially loading the data from the simplified text files into the one of the model scripts and adding final preparations before passing them to the model as training, evaluating or predicting arguments. Firstly, the tweets are separated by a single newline character, so after loading the data, the file gets split into a list of tweets, each of which characterized by a new line in the original file.

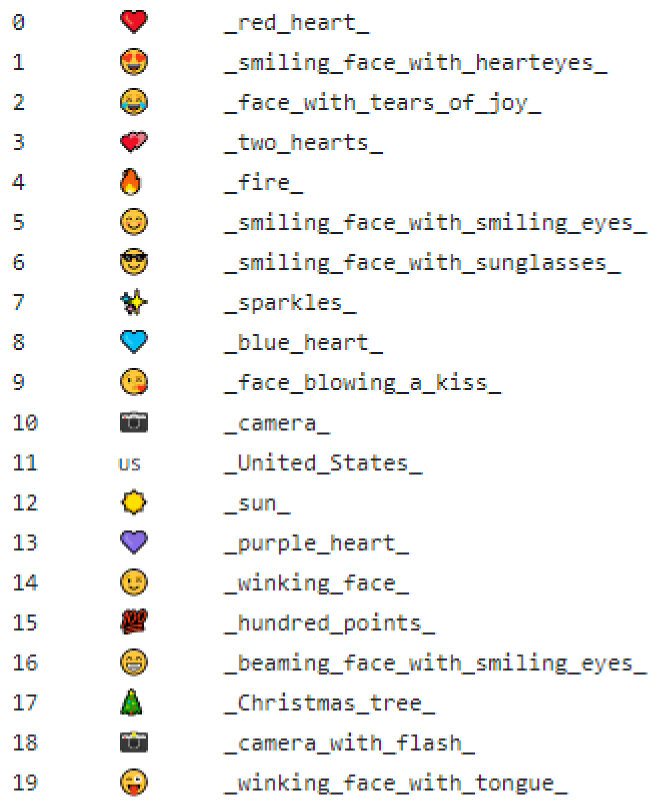
Following the completion of this step, one of the last preparations prior to passing the data is executed: tokenization. A tokenizer is declared, then used to fit to the data from the tweets, encoding every word in the text corpus with a unique value of its own. After encoding the data with a tokenizer, I have written a function of my own to find the largest tweet, which I use as standard to set my input size to. It's essential to ensure the uniform size of the tweets, prior to passing them into the embedding layer of the neural network, hence why after finding the maximum, before I specify it as the input size of the training model, I pad all sequences to the required dimensions, using a built-in function provided in Python's sequences library.

Once the dimensions are standardized and all prior actions have taken place, the data gets passed onto the embedding layer of my machine learning model, representing the words parsed as multi-dimensional vectors for the machine learning model to analyse. I experimented with several embedding sizes throughout, however, their further relevance is thoroughly covered in the section regarding implementation and performance.

Numerical data

While the numerical data, representing the emojis, is initially read from its designated file. See figure one for how they are stored initially. After that it runs a swift function that converts the individual strings of each number to an integer type, then maps those numbers to their appropriate indexes within the realms of a NumPy array. Once this essential step is covered, the data is converted to categorical before being passed onto the model, creating a 20-dimensional vector. Which in turn represents each emoji as its own separate category, allowing the neural network to adequately predict the individual emojis with better clarity and optimise loss more efficiently.

Regarding sentiment I had to change the original dataset's annotations due to a certain bug in the sparse categorical cross entropy loss function. The labels needed to be consecutive, which meant I had to pinpoint and relabel every single instance of the neutral denoted by 2 and positive, denoted by 4, into respectively neutral – 1 and positive – 2.



0	❤️	_red_heart_
1	😍	_smiling_face_with_hearteyes_
2	😂	_face_with_tears_of_joy_
3	💕	_two_hearts_
4	🔥	_fire_
5	😊	_smiling_face_with_smiling_eyes_
6	😎	_smiling_face_with_sunglasses_
7	✨	_sparkles_
8	💙	_blue_heart_
9	😘	_face_blowing_a_kiss_
10	📷	_camera_
11	us	_United_States_
12	☀️	_sun_
13	💜	_purple_heart_
14	😉	_winking_face_
15	💯	_hundred_points_
16	😁	_beaming_face_with_smiling_eyes_
17	🎄	_Christmas_tree_
18	📸	_camera_with_flash_
19	😜	_winking_face_with_tongue_

Figure 1, emoji mapping

There is a slight difference in how I treat the two types of data. Emoji-related tweets are processed prior to running the script that trains and initializes the model. This solution provides much faster testing in comparison to the function used for the sentiment analysis data that is being processed in the same script as the model initialization and training. The main reason for this difference is the file formats used for the two datatypes: .txt for emoji and .csv for sentiment.

In both cases I find it essential to preserve the original unfiltered data so it can be available to improve upon if the data processing proves to be detrimental or there's room for improvement on it later down the line.

Implementation

The implementation was difficult to envision at first – yet with enough tutorials, practice, and insight – by the end of the research phase it seemed clear what needed to be accomplished for this project to be a success. The initial choice was to use TensorFlow v1.x, since most of the available tutorials and pre-existing models that I found on the internet were based around it. In conjunction with python 3.6, as there are several built-in packages that refer to data processing and are essential to the data preparation section of the project pipeline. Later due to issues listed below, the v1.x was replaced with v2.x.

There were several online resources such as Towards Data Science, Medium, Kaggle and many more that were extremely helpful in grasping the basis of how to build and optimize models.

The approach I used initially for the emoji model, was to try to tackle it as a categorical classification problem with simple convolutional and recurring neural network (RNN) layers. [34], [12] In order to be prepared, I chose to develop the easiest possible solution and build on top of it. The first issue I encountered was using fresh embeddings, which I soon realised were not going to yield good results.[35],[10]

After further investigating the field of natural language processing, I realised how long it would take to train and develop adequately performing embedding structures, so I opted to use the pre-existing Glove embeddings, to try to relate the tweets better and give the embedding layer a starting push towards the right direction.[18] I employ the 27 billion tokens twitter version with fifty dimensions, for which I have provided links in the GitHub repository, should it be needed for download.

Emoji prediction model

I investigated two possible variations as being most successful at predicting emojis: multiple or single–layer LSTM models [11], combined with recurring dropouts set in the layers, as well as convolutional architectures. There was an issue with overfitting, where the model performs nearly perfectly on training data, however, it severely underperforms when introduced to new test data. In simpler words, it memorises the training dataset and in doing so underperforms on other data sets used for testing. Comparison drawn from an average of 24.33% on testing categorical accuracy, versus an average of 30.17% on training categorical accuracy. (Calculated from data over 20 epochs.) Loss sustains itself on both training and testing data at roughly 2.3 on average. The conclusion was drawn from experiments on training and testing data for accuracy and loss listed in the figures below.

My reasoning behind this was that recurring neural networks (RNNs) perform better with sequential data, thus they would excel at analysing text and retaining the sentence meaning. That, of course relates to LSTM exclusively, as it has the groundwork for spotting longer-term patterns. That knowledge would prove useful when it comes to learning the patterns of each of the categories within the categorical problems and theoretically performing better due to that. [12]

The final model for emoji prediction involves an input layer for the tokenized sequences, an embedding layer using Glove embeddings and three bidirectional LSTM layers, with decrementing number of neurons, can be seen in figure 2. The main idea behind this structure is to capture the broadest amount of features on the top layers of the model, and gradually narrow down to a specific prediction by limiting the size of the available neurons and decreasing the recurrent dropout of each layer. The final layer is a fully connected Dense layer with 20 neurons, that employs a SoftMax activation. That means that after getting the final vector from the last Bi-LSTM layer, the activation function sums it into a probability distribution consisting of several probabilities that are proportional to the exponentials of the input numbers. (Reference emoji_vanilla_pred.py for code.)

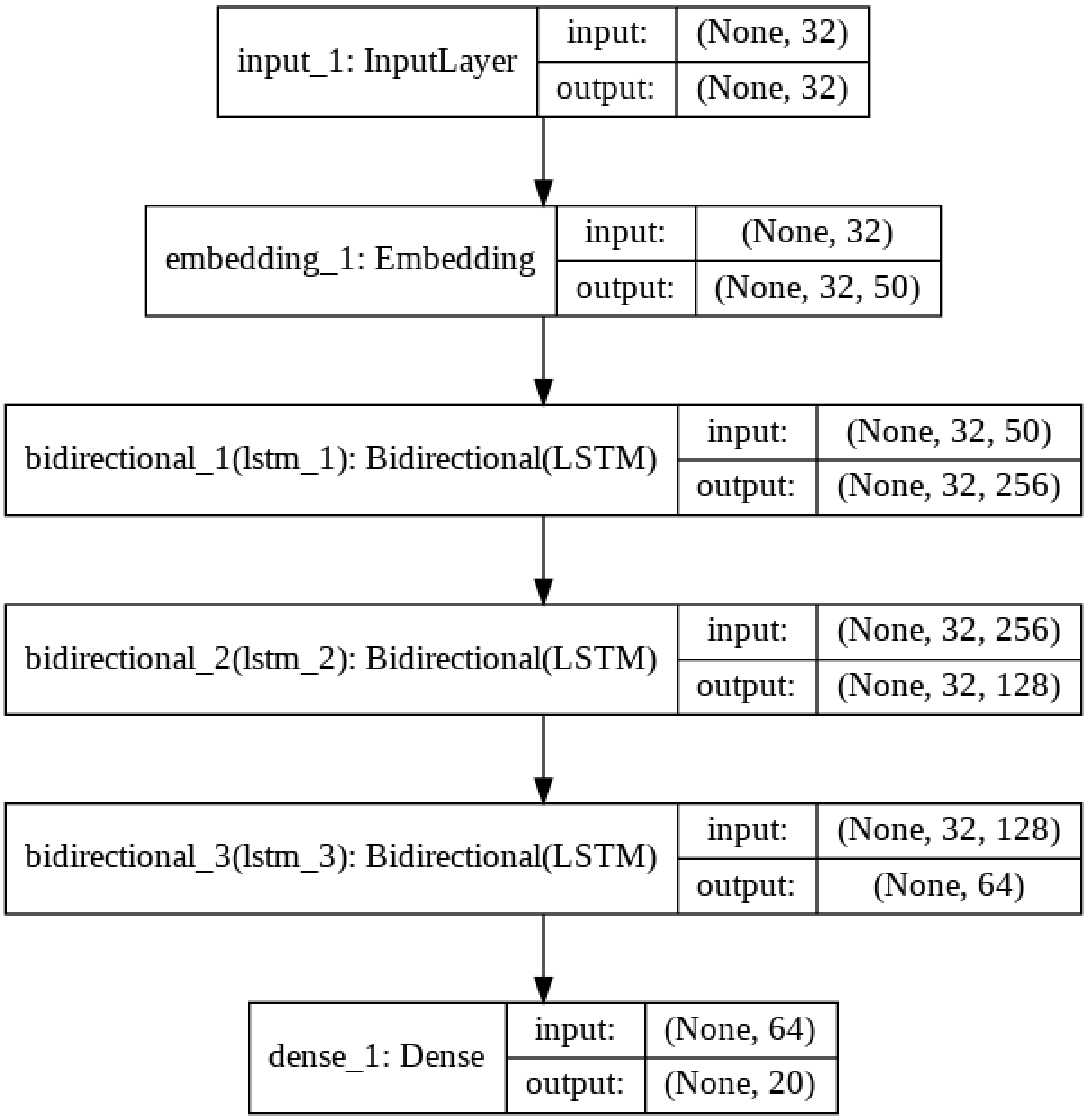


Figure 2: Structure of emoji model

Prior to reaching this final architecture, three other options were tested: a convolutional model with global maximum pooling, a simple RNN architecture, and finally an LSTM-based model. They performed with varying success, usually demonstrating a similar degree of success on incremental epochs. I provide results from the initial benchmarks below.

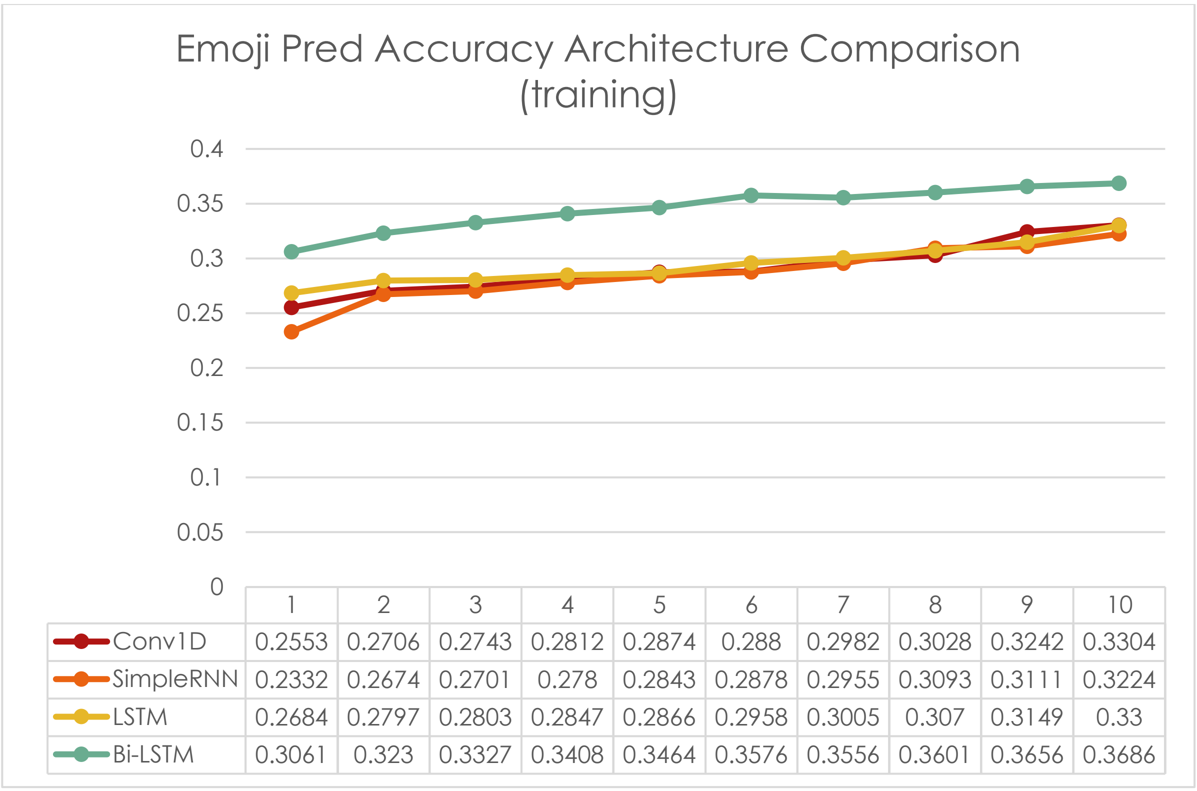
There were several advantages to this architecture that made me choose it over the aforementioned ones. Firstly, while it took longer per epoch It shows a much higher accuracy than its counterparts. That is due to the sequence being scanned in both directions and providing more contextual understanding of the data to the model. That allows for more accurate predictions and a more in-depth understanding of the tweets' token arrangement and contextualizes the semantic structures better compared to an ordinary LSTM network. Comparisons to the other models' performances can be seen in figures 3 and 4.

When it comes to compiling the model there are several parameters involved in the process: loss, optimizer and metrics. For each of the emoji models I have chosen to use categorical cross entropy as my loss function. Notably I chose categorical instead of sparse categorical since the performance of the models that predicted emoji was good as is, there was no need to implement the sparse variation that skips hot encoding in favour of execution speed.

Different emoji can follow a specified tweet, which makes the task even more difficult as the categories are theoretically not mutually exclusive. For clarity's sake, I've chosen to treat them as such due to the fact that a greater part of the emojis in this dataset, as seen above, convey vaguely similar emotions, meaning that the task should be focused more on spotting the finer differences between them. A notable mistake that haunted me throughout the early days of developing this project was picking the wrong optimizers for my task, as well as simply declaring metrics as 'accuracy'.

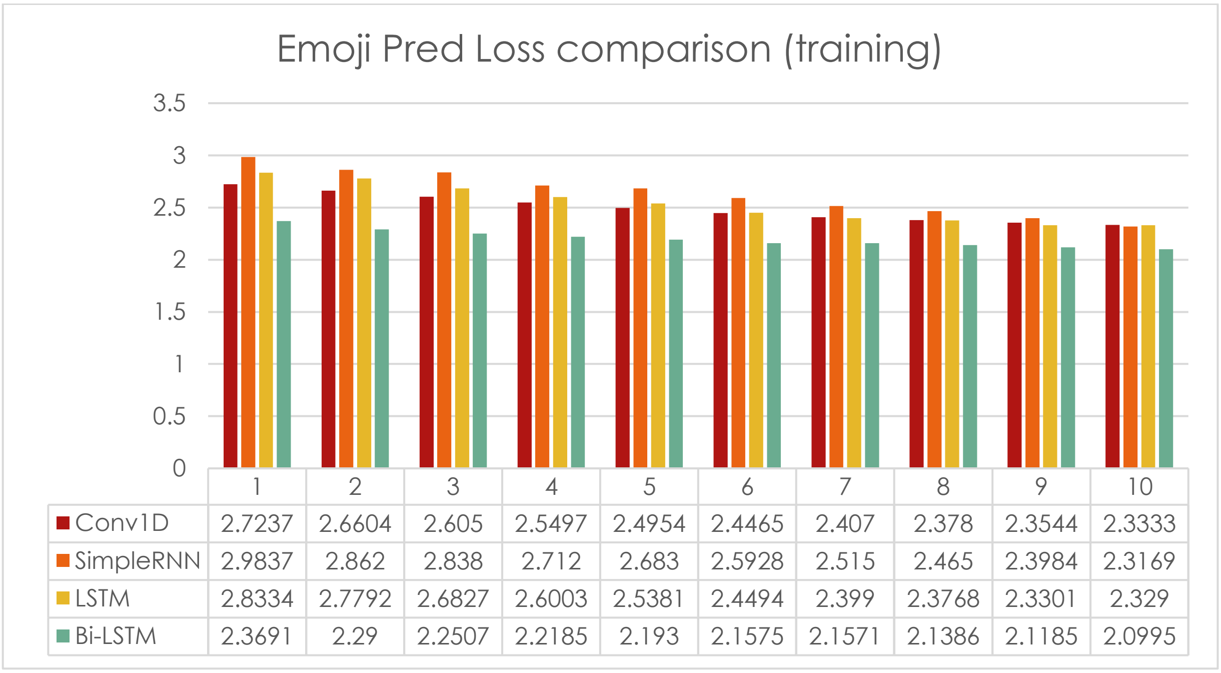
With more research and time, I realised that as a matter of fact the metric for using categorical cross entropy is consequently categorical accuracy. That was key in properly monitoring the accuracy I was getting, since the same architectures performed vastly different in both training and validation after this minor change.

With that cleared up, I tested different optimizers, as well as read up on similar issues that others faced with Keras to pick RMSprop as my optimizer of choice. Compared to most other optimizers I tested, RMSprop and SGD performed best with the given architectures. RMSprop's ability to adequately perform with larger embedding structures has proven invaluable in comparison to ADAM or other famous picks.



	1	2	3	4	5	6	7	8	9	10
Conv1D	0.2553	0.2706	0.2743	0.2812	0.2874	0.288	0.2982	0.3028	0.3242	0.3304
SimpleRNN	0.2332	0.2674	0.2701	0.278	0.2843	0.2878	0.2955	0.3093	0.3111	0.3224
LSTM	0.2684	0.2797	0.2803	0.2847	0.2866	0.2958	0.3005	0.307	0.3149	0.33
Bi-LSTM	0.3061	0.323	0.3327	0.3408	0.3464	0.3576	0.3556	0.3601	0.3656	0.3686

Figure 3: comparison of the emoji-only models (accuracy)



	1	2	3	4	5	6	7	8	9	10
Conv1D	2.7237	2.6604	2.605	2.5497	2.4954	2.4465	2.407	2.378	2.3544	2.3333
SimpleRNN	2.9837	2.862	2.838	2.712	2.683	2.5928	2.515	2.465	2.3984	2.3169
LSTM	2.8334	2.7792	2.6827	2.6003	2.5381	2.4494	2.399	2.3768	2.3301	2.329
Bi-LSTM	2.3691	2.29	2.2507	2.2185	2.193	2.1575	2.1571	2.1386	2.1185	2.0995

Figure 4: comparison of the emoji-only models (loss)

Sentiment analysis model

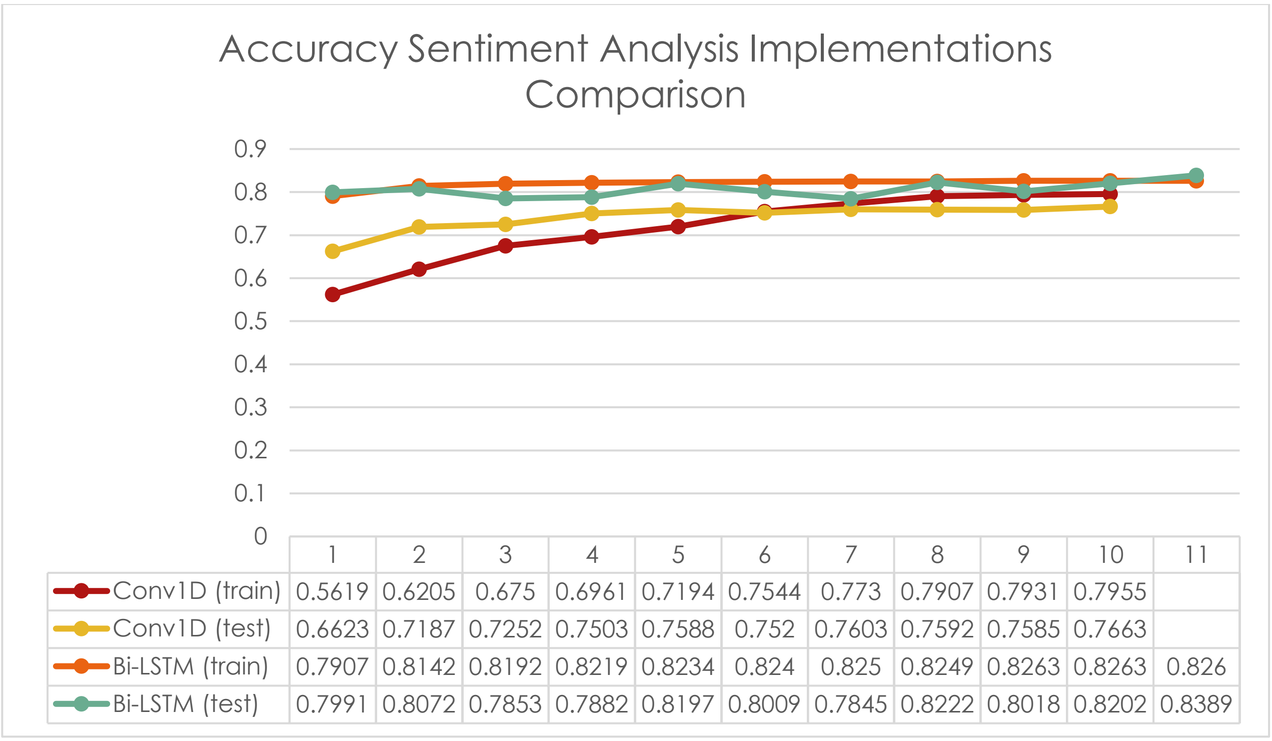
For sentiment analysis, I tested the same three architectures as listed above for emoji. Though, pointers suggested that LSTM was going to be the prevalent architecture again. [36],[3] The results were similar to the outcomes of the emoji tests, whereby Bidirectional LSTMs outperformed the other models in both loss and accuracy. My initial sentiment analysis model was built to only recognize the highest polarity of the dataset. To start off with a simpler model I used binary cross entropy as loss function, along with the ADAM optimizer.

Once I found an architecture that I found worked well for the binary classification problem I built upon it with the introduction of neutral sentiment and converting the problem to categorical. Due to the simpler nature of the sentiment analysis classification problem I chose to give the bidirectional layers smaller dimensions, both to help with the performance and reduce the possibility of overfitting down the line of training. [20]

The final model includes the embedding layer discussed above, a bidirectional LSTM layer, followed by another bidirectional LSTM layer with less parameters to narrow down the semblances as listed above in the emoji structure. The fewer categories require less specifics, and thus can be optimized by using smaller and fewer layers for better time performance and overall metric results. The prediction layer is a fully connected layer with a SoftMax activation function. (Reference sent_LSTM.py for full code.)

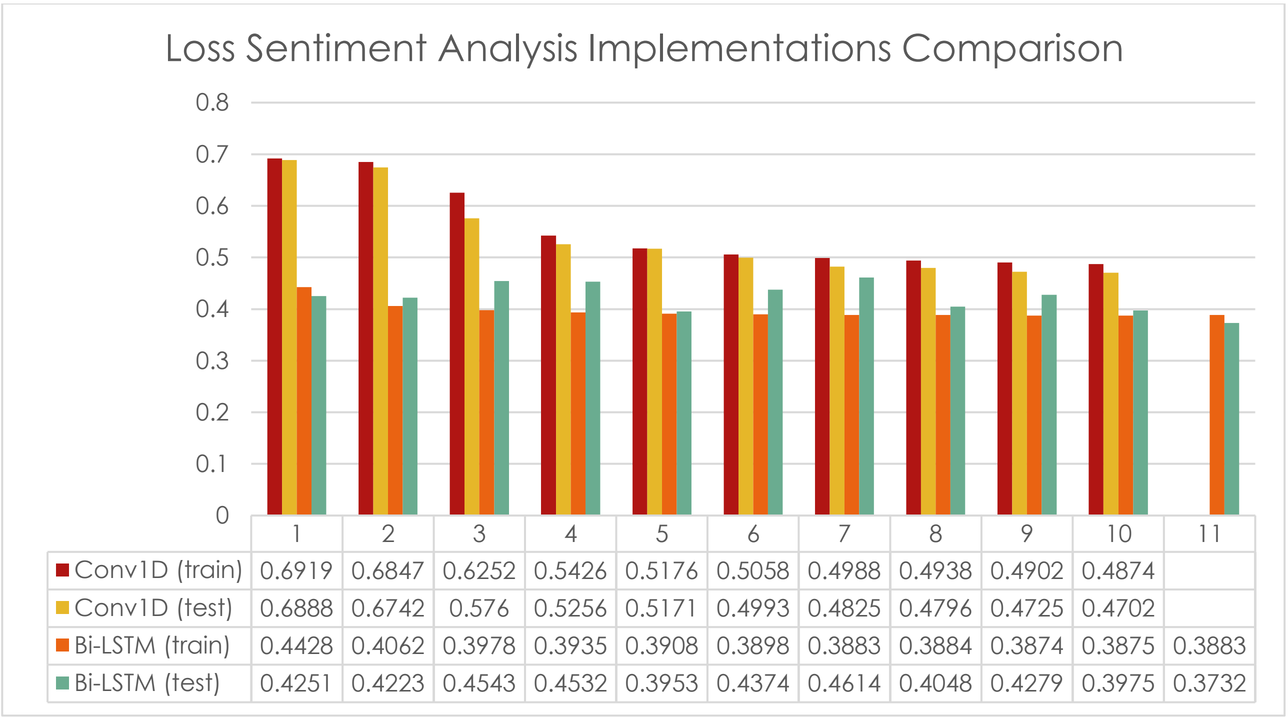
Unlike the emoji classification problem, sentiment analysis had a larger dataset, hence required more optimization and it came with a bigger embedding. I read up on individual sources and experiments that others had attempted on this dataset, and while most of them used ADAM, I seemingly kept achieving better performance using RMSprop. That provided some significant improvements regarding accuracy and overall performance. I chose sparse categorical cross entropy to try and further save on computation time, as categorical by itself performs a fraction slower and requires further label preparation. (See figures 5 & 6.)

However, it is notable to mention that in the initial version of Keras (v1.x) the 'accuracy' metric did not calculate the outcome of sparse categorical cross entropy correctly, either. Much like it's non-exclusive and hot encoded variation (plain categorical), it has a specific accuracy function to reliably calculate the accuracy, as the API had employed different calculation functions if the label wasn't specified to 'sparse_categorical_crossentropy'.



	1	2	3	4	5	6	7	8	9	10	11
Conv1D (train)	0.5619	0.6205	0.675	0.6961	0.7194	0.7544	0.773	0.7907	0.7931	0.7955	
Conv1D (test)	0.6623	0.7187	0.7252	0.7503	0.7588	0.752	0.7603	0.7592	0.7585	0.7663	
Bi-LSTM (train)	0.7907	0.8142	0.8192	0.8219	0.8234	0.824	0.825	0.8249	0.8263	0.8263	0.826
Bi-LSTM (test)	0.7991	0.8072	0.7853	0.7882	0.8197	0.8009	0.7845	0.8222	0.8018	0.8202	0.8389

Figure 5: Sentiment analysis accuracy



	1	2	3	4	5	6	7	8	9	10	11
Conv1D (train)	0.6919	0.6847	0.6252	0.5426	0.5176	0.5058	0.4988	0.4938	0.4902	0.4874	
Conv1D (test)	0.6888	0.6742	0.576	0.5256	0.5171	0.4993	0.4825	0.4796	0.4725	0.4702	
Bi-LSTM (train)	0.4428	0.4062	0.3978	0.3935	0.3908	0.3898	0.3883	0.3884	0.3874	0.3875	0.3883
Bi-LSTM (test)	0.4251	0.4223	0.4543	0.4532	0.3953	0.4374	0.4614	0.4048	0.4279	0.3975	0.3732

Figure 6: Sentiment analysis loss comparison

Merged and combined models

That concludes the finalized simple models that I offered to solve the two individual problems that this task consists of. The remainder was focused around trying to combine the two existing structures into one. I tried several methods: a model with shared structure with two inputs and two outputs, a model with two inputs, shared embedding, separate structures and two outputs, and my simplest attempt which was employing the pre-existing models and making a pipeline out of them.

Following what I said in the approach section, I chose the simplest approach first. I created a new and altered version of the emoji algorithm. The initial version just had the sentiment analysis result appended to the end of the tweet sequence and fed into the model. As I anticipated this did not alter the accuracy in any way, shape or form. My second attempt with this approach was to create a separate input and pad it to the same size as the emoji sequence input. Sadly, as I anticipated that did not alter the results.

My final attempt with option one included a change of the model structure. That did showcase some change in the initial results I was noticing. I replaced my additional Bi-Sense layers with a simpler structure relying on a global pooling layer, followed by a fully connected layer, dropout layer and a prediction layer. The performance of this model is outstanding and computation-wise it shadows the rest when it comes speed and performance. Figure 7. (Reference emo_pred_plus_sent.py for full code.)

An important note on the implementation of this model is that in contrast to the solo training models it does not use zero masking within the embedding layer. Due to its differing structure that requires pooling, it needed to have a different approach to the embedding layer and how it handles data. The zero padding is removed in this case to provide the model with more context regarding the hot encoded nature of the sentiment values included.

The pipeline created for it is as follows: prepare the tweets needed using the data preparation file, then run them through the sentiment predictor file, and once these two steps are done, the required files for the concatenated output are ready. Once loaded into the model, they can be used for prediction of emoji based on the data provided.

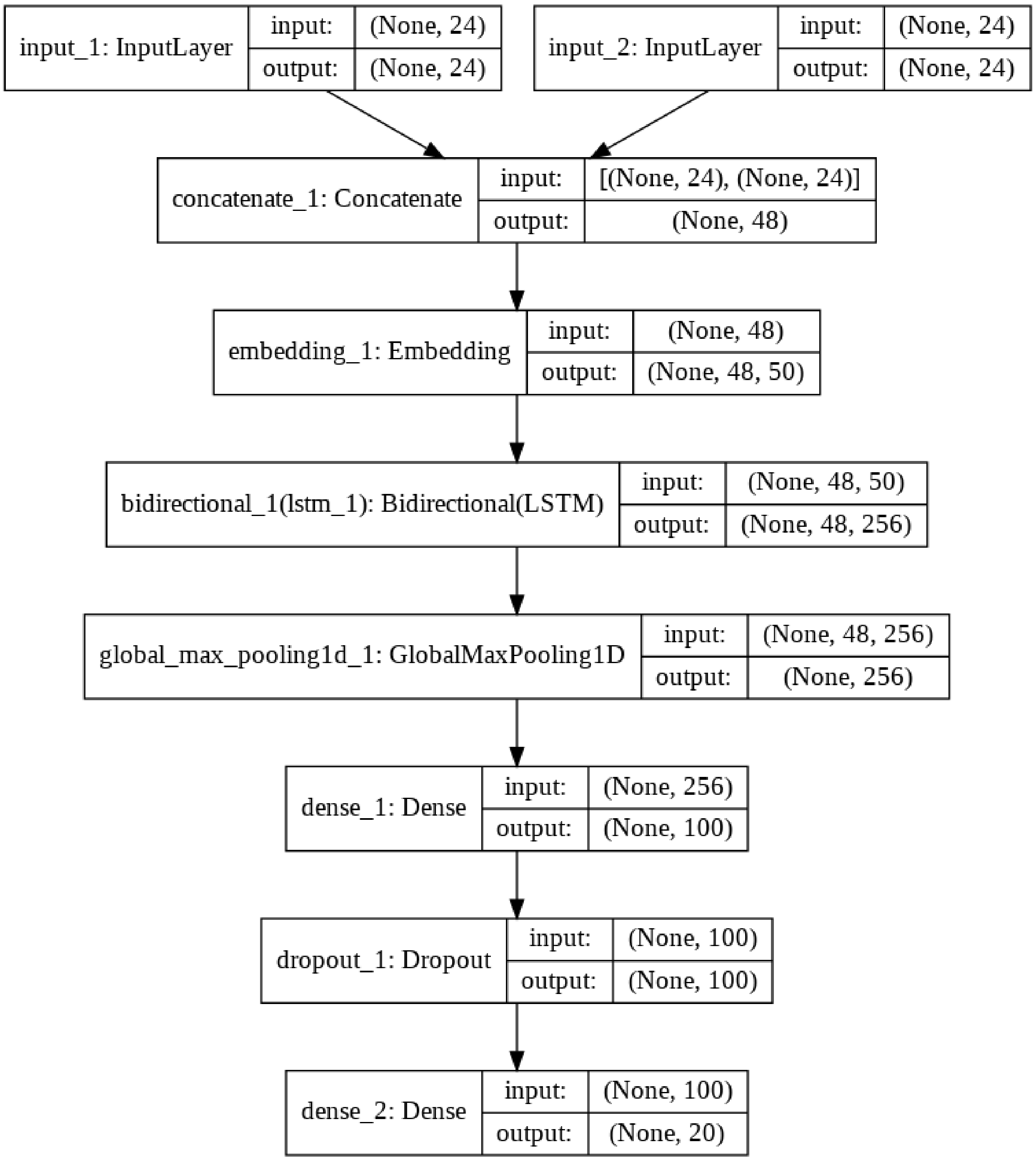


Figure 7: Simple Sentiment + Emoji model structure.

The next model I built was going to be a merged model of the two that I had built previously. My plan was to have two inputs that connect to a shared embedding layer. To keep the sequences' semantic characteristics intact I figured that I'd first apply analysis of each of the sequences through individual Bidirectional LSTM layers, the outputs of which I'd concatenate before passing onto the shared layers and making predictions. With two separate fully connected layers.

This model did not perform very well, despite retaining most of the characteristics of its standalone predecessors. I assume that its added complexity took a toll on both its performance and metrics. Below I draw a 10-epoch comparison of the sentiment analysis standalone training data versus the complex model's in the findings section.

Due to its complexity, the performance was quite lacking in most areas, including metrics as mentioned below. It tends to be quite computationally taxing as well due to the number of parameters required for it to be able to carry out its predictions. I have tried forming concatenations on different parts of the layers and experimented with several different methods of concatenation, but most of them only multiplied the number of parameters and slowed the model greatly.

After many attempts to train it and tweak its performance I eventually moved on to another idea as I noticed this may not be a feasible solution. I decided to use a global max pooling layer after the concatenation and mimic the previous model. My theory behind it was that it would benefit the model more than following up with two LSTM layers as shown below in figure 8.

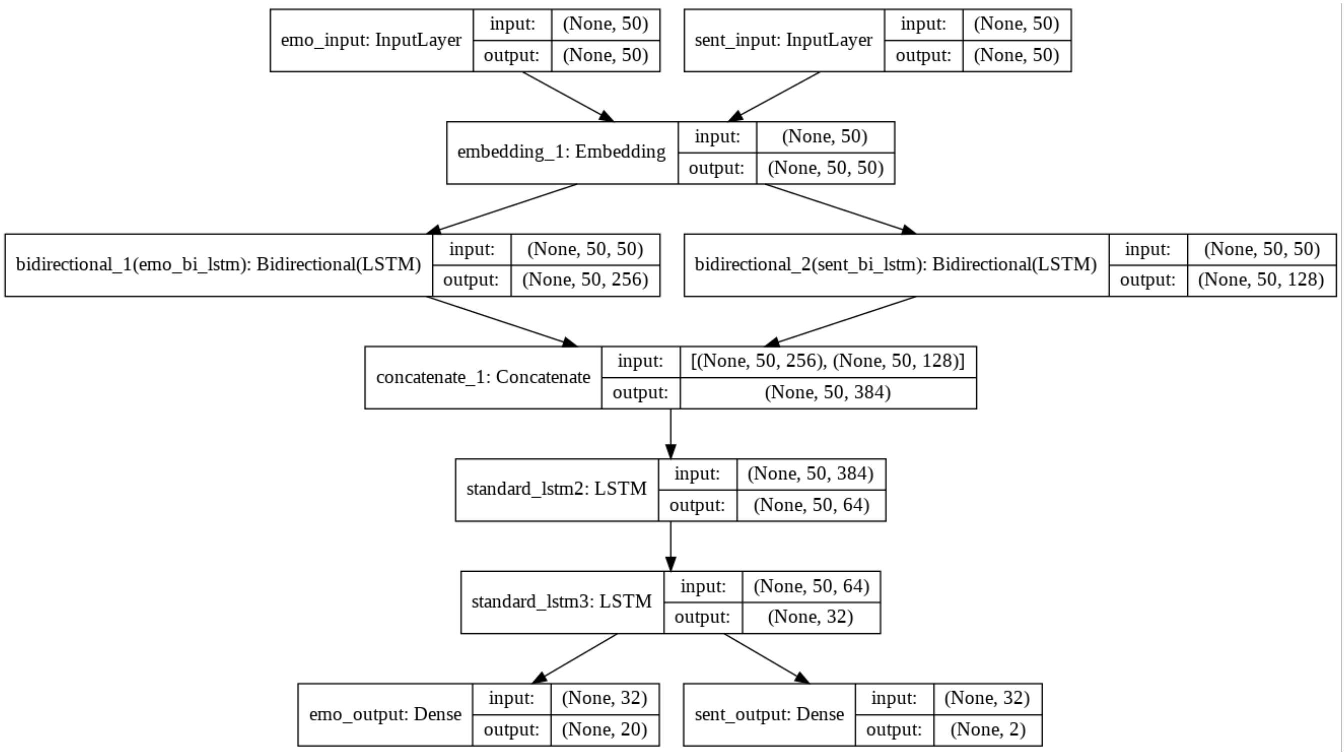

emo_input: InputLayer | input: (None, 50) | output: (None, 50)
sent_input: InputLayer | input: (None, 50) | output: (None, 50)
embedding_1: Embedding | input: (None, 50) | output: (None, 50, 50)
bidirectional_1(emo_bi_lstm): Bidirectional(LSTM) | input: (None, 50, 50) | output: (None, 50, 256)
bidirectional_2(sent_bi_lstm): Bidirectional(LSTM) | input: (None, 50, 50) | output: (None, 50, 128)
concatenate_1: Concatenate | input: [(None, 50, 256), (None, 50, 128)] | output: (None, 50, 384)
standard_lstm2: LSTM | input: (None, 50, 384) | output: (None, 50, 64)
standard_lstm3: LSTM | input: (None, 50, 64) | output: (None, 32)
emo_output: Dense | input: (None, 32) | output: (None, 20)
sent_output: Dense | input: (None, 32) | output: (None, 2)

Figure 8: Merged model: combined LSTMs structure.

Upon later consideration of the above architecture, an idea came to mind, to investigate if it were possible to improve the model's performance by implementing the concatenated sentiment + emoji input's architecture into the merged model. To accomplish this, I thought of replacing the shared LSTM layers with a global max pooling layer and a fully connected layer.

The idea behind this was to apply a more computationally light approach to how the layers are handled post-concatenation. Consequently, I chose to use sparse categorical cross entropy for both problems, to speed up the formulae computation. The details on the architecture can be seen in figure 9. (Reference merged_structure.py for full code.)

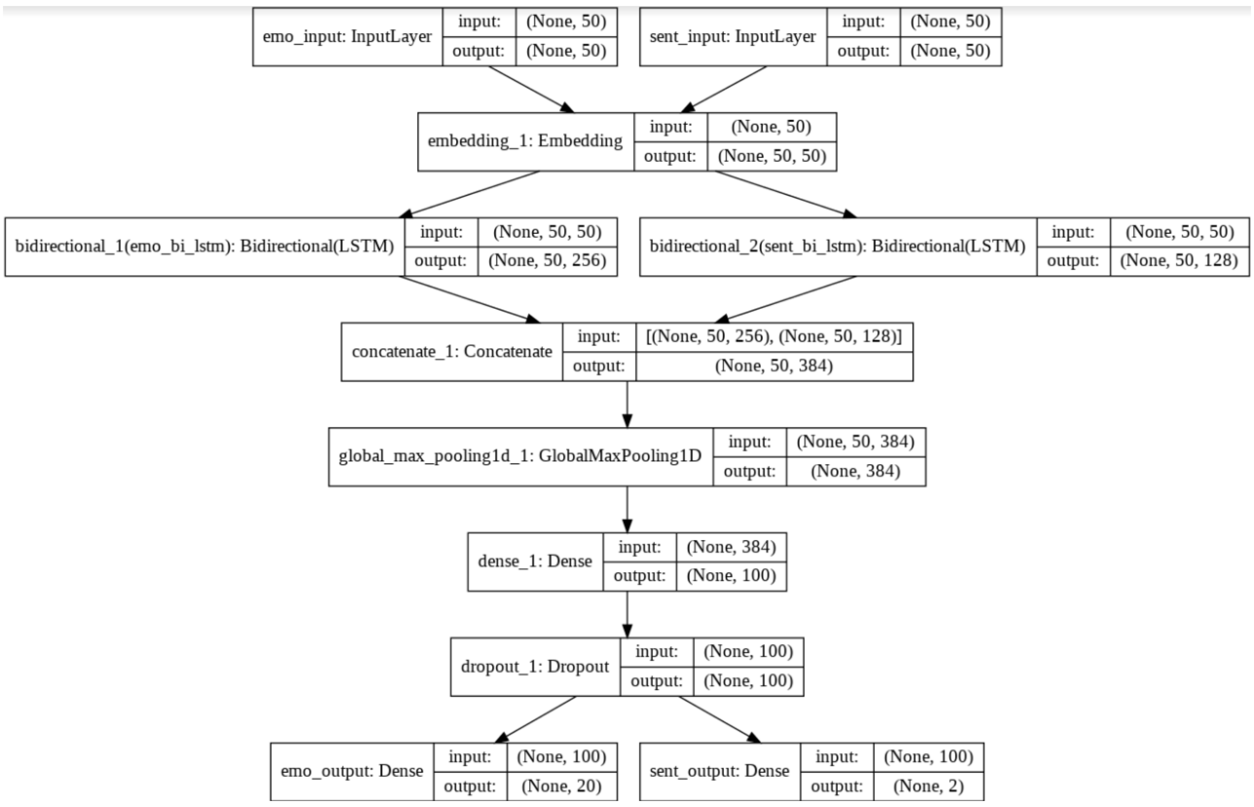

emo_input: InputLayer | input: (None, 50) | output: (None, 50)
sent_input: InputLayer | input: (None, 50) | output: (None, 50)
embedding_1: Embedding | input: (None, 50) | output: (None, 50, 50)
bidirectional_1(emo_bi_lstm): Bidirectional(LSTM) | input: (None, 50, 50) | output: (None, 50, 256)
bidirectional_2(sent_bi_lstm): Bidirectional(LSTM) | input: (None, 50, 50) | output: (None, 50, 128)
concatenate_1: Concatenate | input: [(None, 50, 256), (None, 50, 128)] | output: (None, 50, 384)
global_max_pooling1d_1: GlobalMaxPooling1D | input: (None, 50, 384) | output: (None, 384)
dense_1: Dense | input: (None, 384) | output: (None, 100)
dropout_1: Dropout | input: (None, 100) | output: (None, 100)
emo_output: Dense | input: (None, 100) | output: (None, 20)
sent_output: Dense | input: (None, 100) | output: (None, 2)

Figure 9: Updated merged model architecture

Unfortunately, this idea struck as I write this report so I can only show very little results for it in a graph comparison between the older version of this model and the newer version of it. The first impressions of this improved model are quite different to those of its predecessor – it outperforms the older version in every conceivable way. The average loss for version one is roughly 2.9477, whereas the average of version two 2.8159, which shows an impressive improvement in only the first five epochs. The table provided below shows the improvements in accuracy on the first five recorded epochs of each of the models. (v.1 referring to the old model, v.2 referring to the new one.

Train v.1	Train v.2	Test v.2	Metric: Accuracy
0.1783	0.2308	0.2485	Emoji prediction
0.1857	0.2516	0.2548	
0.1869	0.2567	0.2583	
0.1888	0.2597	0.2608	
0.1895	0.2617	0.2624	
0.6142	0.7247	0.7607	Sentiment analysis
0.6298	0.7700	0.7765	
0.6311	0.7810	0.7846	
0.6346	0.7875	0.7900	
0.6360	0.7922	0.794	

Table 1: Accuracy on merged model variations one and two.

This offers insight into how the layers interact from the concatenation onwards. While the Bidirectional LSTM picks up the timesteps and ways natural language progresses, once concatenated, the pooling layer can extract important features from the LSTM layer, to then analyse in a fully connected layer. You can see a rather rushed evaluation of the epochs that I managed to train within this short period of time in a graph of the values in the findings section, as well as the graph above, showcasing the difference in metrics.

This merged model doesn't compare all that greatly to the emoji-only model or the emoji prediction plus a sentiment vector model, but it shows promise that the initial idea behind the merge can perform better, perhaps with a more convolutional approach.

That brings me to the latest model created for this task that simply employs a shared embedding layer and two separate structures to do the predictions for each of the sentiment and emoji parameters. It essentially combines the two models into one by retaining the original architectures of each of them, except it only uses a shared embedding. As suspected, it outperformed its concatenated counterpart but only slightly. Due to time running out I was unable to conduct and record this model's overall performance and evaluation.

This model was abandoned due to having additional complexity and not really being related to the task in many ways, as in the end apart from extended embeddings it did not share other data between the layers involved in later predictions. (Reference diverged_structure.py for full code.)

Note: all the code is available on GitHub for reference and review: https://github.com/Marielene/sentiment-emoji

Findings

The goal of this project being exploring the relationship between emoji and sentiment, has reached several conclusions. While there is certainly a dependence between the two, it would seem my project has not managed to find an efficient way to utilize this correlation. To plot all graphs the metric used for evaluation is primarily accuracy. However, the evaluation findings give hope that this project's hypothesis is on the right track.

Note that for optimal results the data must be cleared from noise, while retaining the sentences' structures. That includes just the use of lemmatization, removal of extra spaces, special symbols and lowercasing. Initial models with heavily processed data (using both lemmatization and removal of stop words), ended up giving poorer results and higher loss overall compared to the less processed data that retained more words (lemmatization only). Note: this table refers to testing loss monitored by categorical cross entropy for each of the models. As shown in table below, over roughly 10 of their best epochs each (20 total, best referring to metrics-wise):

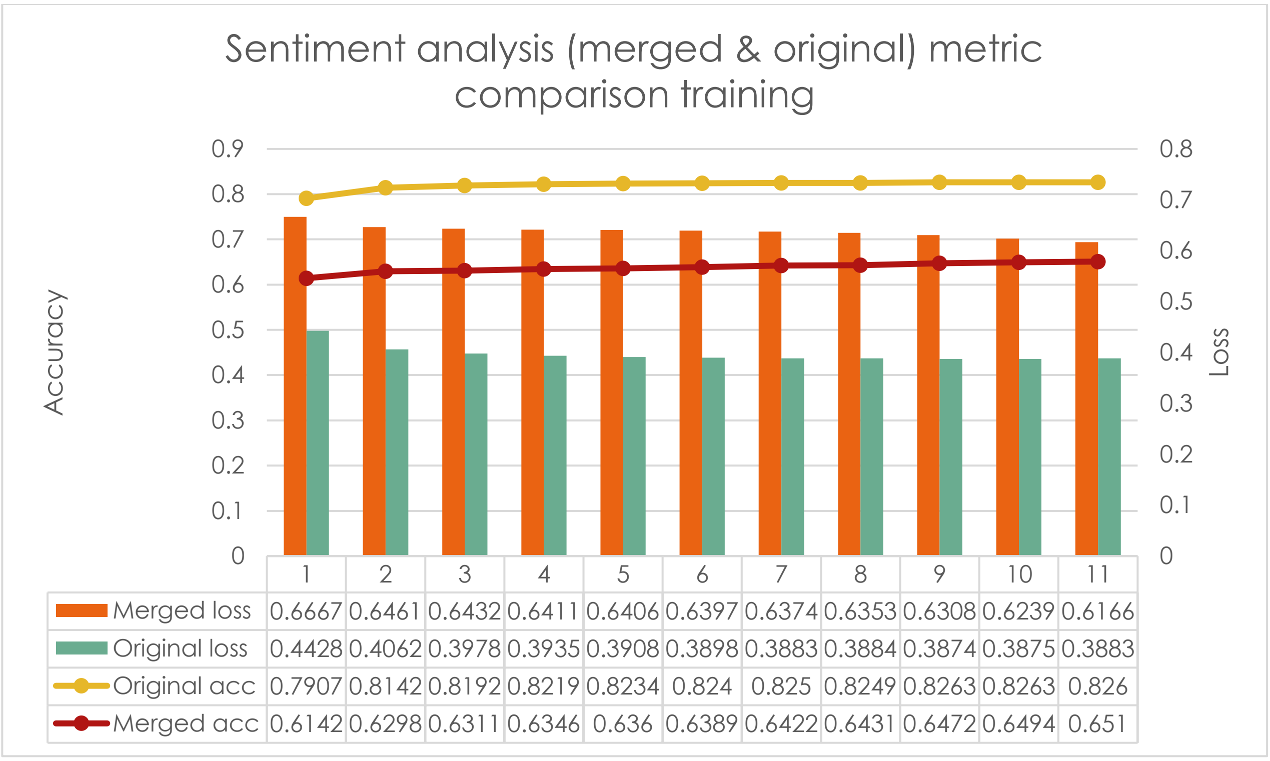
Models	AVG loss with light pre-processing (LPP)	Accuracy AVG LPP	Loss AVG heavy pre-processing (HPP)	Accuracy AVG HPP
Conv1D	2.4831	0.2847	2.6333	0.2431
SimpleRNN	2.573	0.2633	2.6942	0.2333
LSTM	2.2819	0.2955	2.5663	0.2597
Bi-LSTM	2.2195	0.3106	2.4861	0.2866

Table 2: Averages of testing data with different data processing approaches

The single merged model's performance was lower than the individual on both emoji and sentiment analysis. (Figures 6 and 7.) Showcasing an average of 15~16% worse performance on part of sentiment analysis and a similar decrease on the emoji prediction frontier, dropping to a 21.9% validation accuracy from nearly 37.5% on the Bi-Directional LSTM model with sentiment tokens. See figures 10 and 11. The individual drops are presented below in two comparison graphs.

Model (Emoji predict)	Average Accuracy	Average Loss
Bi-LSTM	0.3456	2.1992
Bi-LSTM + Sent vector	0.3460	2.2099
Merged model v.1	1.9477	2.9477
Merged model v.2	0.2521	2.8159

Table 3: Averages of test data on emoji prediction models over 20 epochs each.



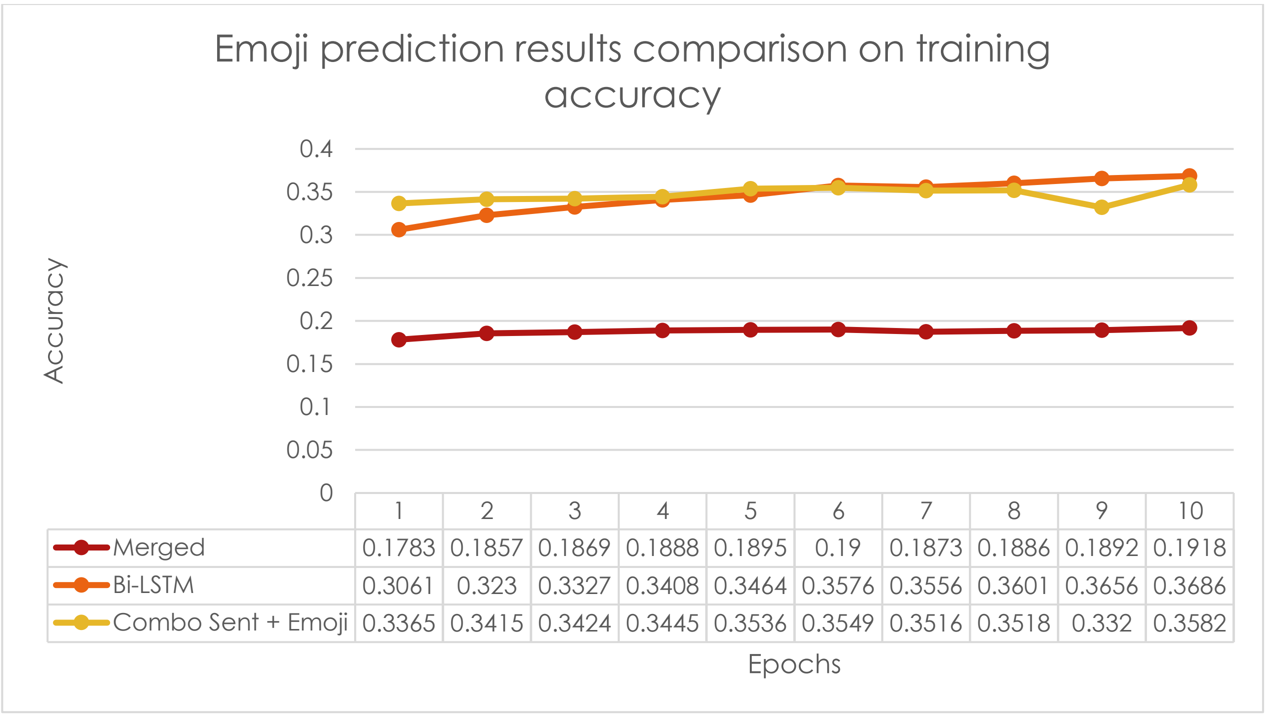
	1	2	3	4	5	6	7	8	9	10	11
🟧 Merged loss	0.6667	0.6461	0.6432	0.6411	0.6406	0.6397	0.6374	0.6353	0.6308	0.6239	0.6166
🟩 Original loss	0.4428	0.4062	0.3978	0.3935	0.3908	0.3898	0.3883	0.3884	0.3874	0.3875	0.3883
● Original acc	0.7907	0.8142	0.8192	0.8219	0.8234	0.824	0.825	0.8249	0.8263	0.8263	0.826
● Merged acc	0.6142	0.6298	0.6311	0.6346	0.636	0.6389	0.6422	0.6431	0.6472	0.6494	0.651

Figure 10: Sentiment analysis metric comparison to merged model.

The reason for this, is likely due to underfitting on my part when creating the model and possible poor layer optimization on my part, as a way to improve it would be to convert the LSTMs to Bi-LSTMs and add dropout layers after concatenation. Another reason is the overcomplicating of the problem, without adequately optimizing as stated above. If there were a finer level of optimization to be applied to the structure, it would probably improve both metrics and performance-wise. As can be seen in figures 10 to 12.

There is a lot more to improve in that direction, as I am certain there must be a simpler way to implement this variation and achieve adequate results. Since the simpler variation with sentiment does seem to have a closer performance to the original. But as showcased in the graphs below it could not be referred to as an improvement on the overall accuracy and loss of the model.

The positive results with further training would likely be from the separated model that included a shared embedding layer. However, it too performed poorly with its initial epochs, though due to its complex structure, I believe it can prove useful with more tweaking and training.



	1	2	3	4	5	6	7	8	9	10
Merged	0.1783	0.1857	0.1869	0.1888	0.1895	0.19	0.1873	0.1886	0.1892	0.1918
Bi-LSTM	0.3061	0.323	0.3327	0.3408	0.3464	0.3576	0.3556	0.3601	0.3656	0.3686
Combo Sent + Emoji	0.3365	0.3415	0.3424	0.3445	0.3536	0.3549	0.3516	0.3518	0.332	0.3582

Figure 11: Emoji prediction models comparison of accuracy (validation)

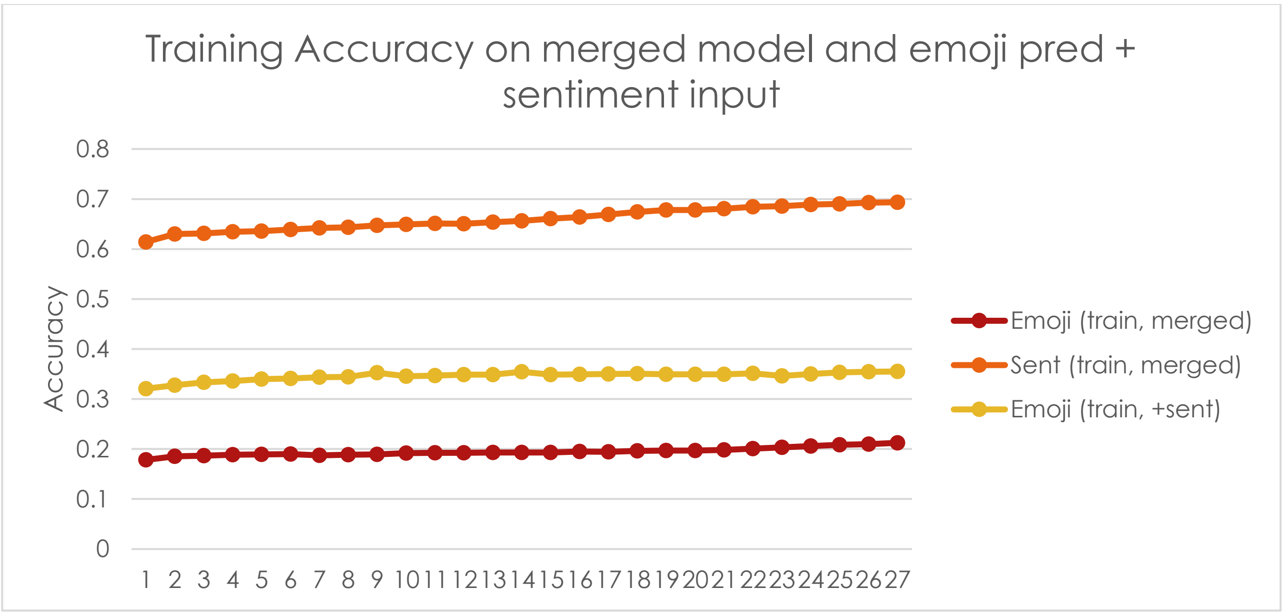
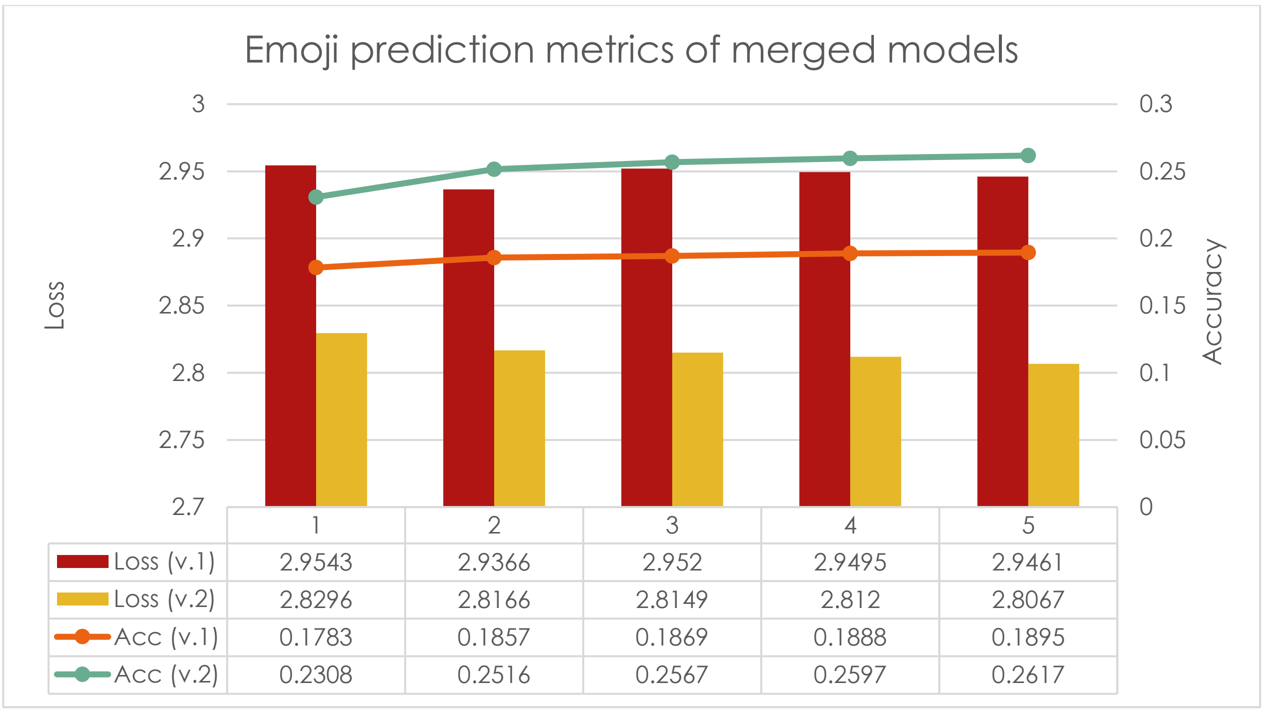


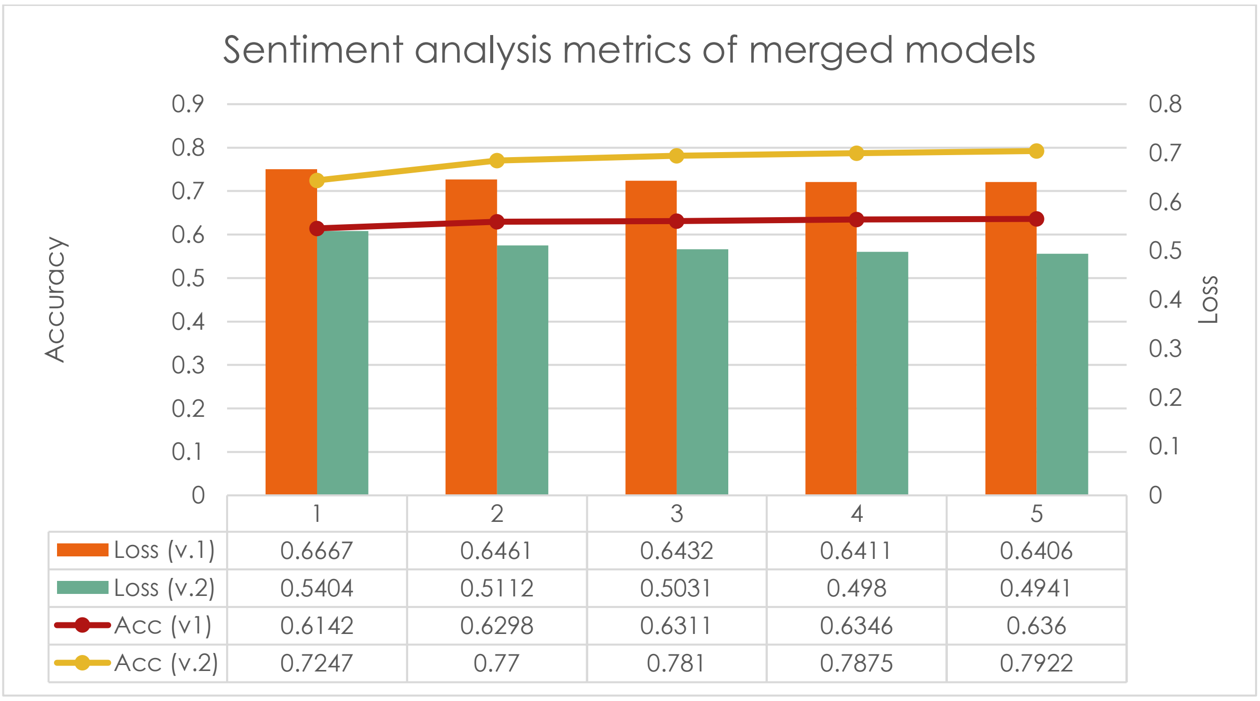
Figure 12: Emoji accuracy and Sent accuracy refer to the accuracy from the merged model, whereas emoji + sent refers to the model with the added sentiment tokens.

Since the update of the merged model, the new architecture shows more promise. Sadly, due to inefficient time the comparison is made with little data. A vital improvement is that on the first epoch the updated model achieves higher accuracy than its predecessor does in thirty epochs. That applies to both tasks presented in the problem, respectively sentiment analysis and emoji prediction. I present figures 13 and 14 to visualise the data.



	1	2	3	4	5
🟥 Loss (v.1)	2.9543	2.9366	2.952	2.9495	2.9461
🟨 Loss (v.2)	2.8296	2.8166	2.8149	2.812	2.8067
● Acc (v.1)	0.1783	0.1857	0.1869	0.1888	0.1895
● Acc (v.2)	0.2308	0.2516	0.2567	0.2597	0.2617

Figure 13: Merged model emoji data. Note: v.1 refers to LSTM only, v.2 refers to improved mode



	1	2	3	4	5
🟧 Loss (v.1)	0.6667	0.6461	0.6432	0.6411	0.6406
🟩 Loss (v.2)	0.5404	0.5112	0.5031	0.498	0.4941
● Acc (v1)	0.6142	0.6298	0.6311	0.6346	0.636
● Acc (v.2)	0.7247	0.77	0.781	0.7875	0.7922

Figure 14: Merged model sentiment data. Note: v.1 refers to LSTM only, v.2 refers to improved mode

Upon evaluation the model with additional sentiment vector achieves a better accuracy compared to its emoji-only counterpart. That leads me to believe that the minor addition of this vector improves the ability of the model to distinguish between certain emojis based on the context for the tweet that the sentiment provides.

The difference in evaluation being miniscule of around 1%. The combined model achieves a loss of 2.285 and accuracy of 34% on a final evaluation. Whereas the emoji only evaluation achieves an accuracy of 33% and slightly bigger loss of 2.2983. Which demonstrates that the addition of sentiment presents a positive impact on the predictions, regardless of how little it might be. That proves the hypothesis has merit for future development and investigation.

Reflection

There are several issues to touch upon in this section, all of which focused on deeper understanding and adequate planning of the project. Looking back on it now, I lacked both at the beginning of this task, and barely acquired them a few weeks prior to submission. There is a lot to unpack behind this statement, so I will get into details below.

Firstly, when I started this project, I had no experience with data science ort machine learning itself. That resulted in a rather ambitious outlook regarding how fast and adequately things would work and be produced. The technicalities and the level of understanding required quickly felt overwhelming. I learned to incrementally deepen my understanding of a subject instead of jumping into the deep headfirst.

My research consisted of three individual phases: the initial first week of research to give me an overall idea of the scope and the amount of time it will take to complete the project, the second week based entirely around implementation and learning to utilize the tools within the Keras library and get a basic understanding for the API, and lastly, the serious academic research that was based around the details and technicalities of how to improve and adequately build a model that I deem satisfactory.

The initial research delved into basics and the overall idea of the project. Understanding the scope was crucial in order to come up with a manageable timeline that could be fulfilled in the given circumstances. I looked up available projects on Kaggle that were focused around using twitter data and analysing it in various ways, usually sentiment analysis of different kinds. It gave me a rough idea of what to expect and how my initial models will look.

The second research phase was far more practical involving following in-depth tutorials and carefully reading the Keras documentation to attain more in-depth understanding of the API I was going to use. There were varying levels of recommendations regarding the options on which API would seem to be best suited for the task. While Keras and TensorFlow provided a good starting foundation for novices in the area of deep learning, PyTorch allows for more creative freedom and building way more complex models in an accessible package. In my case, I opted for Keras due to being inexperienced in the field.

The third and most eye-opening phase of research involved focusing on more detailed and methodical approaches to the problem such as fine-tuning models and understanding them from a more academic standpoint. That meant reading dozens of papers, keeping track and comparing their findings in hope of finding something that works for my scenario. With that in mind, I spent a long time trying to filter out the applicable insights from the inapplicable, which led to certain complications.

Due to most of the papers being about creating those models from scratch, working with an API both reduced the amount of statistics and in-depth theory I needed to know, yet it limited my options when it came to developing high rates of accuracy. With this knowledge, I started looking into ways to refine the options that the API gave me control over, so rather than papers, I had to turn towards forums and developer insights. The result of that is the amalgamation of knowledge of both developers in non-academic and academic context coming together for the development of this project.

By the aforementioned, I mean that by the time I had realised how much literature I needed to promptly read and classify was a rather considerable task in itself, it felt very overwhelming at first. To combat this for future reference I made a system of incremental learning that would prevent me from feeling overwhelmed, as before reaching for the scientific papers, I read through most tutorials online on basic Keras building techniques and models.

Regarding implementations there were a few situations where things did not go as planned and resulted in slowing the overall progress of the task. One of those peculiar situations was choosing the amount of processing the data should go through. I thought this issue could be resolved by delving into research and looking at similar projects, however, in the end it boiled down to seeing what works best for the structures I had.

Keras and Google Collab have been both a blessing and a curse at times, due to dependencies and updates that often stirred up the development process. Around the 27th of March Google Collab updated its TensorFlow version and thus, my old code was performing in a much clunkier fashion. Rather than continuing to pursue efficiency and reusing my existing code, I chose to rewrite and retrain most of the prototypes I had. Luckily, the update was a blessing in disguise as it allowed for more features to be added, thus improving the code's overall performance.

If I had to redo this project, I would spend a lot more time researching CNNs, so I could have multiple advanced models to have benchmarks on. That would help me develop a more a more competitive structure that would ultimately provide better results. Since the method I ended up choosing for the sentiment vector emoji predictor, would work much better with a convolutional structure compared to my current implementation.

Future Work

There are several ways to build upon this foundation if given more time. Firstly, this system given a few tweaks could be the groundwork for an emoji embedding to be developed and trained, though that could classify itself as an entirely separate project, given the amount of further reading and time it would take to develop.[38], [17]

While I have not succeeded in this endeavour, I am certain that with a bigger sample size of emojis to predict, this project's idea would be much more feasible to implement and see in action. There is room for improvement in the finer elements of this project, as well as space for experimentation with different architectures and more complex tweaking.

For example, adding attention to the emoji prediction model to improve its ability of understanding the tweets and developing a deeper understanding for their contents. The model's bidirectional layers combined with an attention dynamic would theoretically improve greatly based on past research in the area of NLP with neural networks. [39]

Regarding the merged models, a lot can be expanded on in future recreations in terms of architecture and approach. Research points towards much more complex methods of NLP being feasible for the task and due to its complexity, it seems difficult for it to be achieved with just Keras by a person with little experience as myself.

Overall, while Keras has definitely been the correct choice for me as a beginner in the field, I am certain that this project would require a much more in-depth level of tweaking and control over the layers, so for anyone inclined to pursue a project of this scope and achieve successful results, I'd recommend them to rack up experience prior to the task, as that will definitely be of aid to reaching the deliverables in a reasonable time.

In the near future, as emojis grow to become an even more essential part of communication, I am certain that the endeavour of predicting and delving further into the semantics of emoji will become its own part of sentiment analysis itself. I hope this project proves useful enough for others in the field to build upon.

Conclusion

While the yield of these final models does not present the most convincing results, the concept behind the project can be explored further. LSTMs can be built upon with attention to improve the results presented in this report. [37], [11] However, the alternatives are also tempting to consider. Especially so with the use of convolutional networks that could provide much better insight when it comes to the concatenated models. Due to the nature of the task and the way the data is presented during concatenation, convolution is likely to be outperforming LSTMs in a bigger sample space. So, the use of it would be more beneficial in this kind of task if the goal is to create a combined model that uses concatenation.

Bibliography

[1] Aliza Sarlan, Chayanit Nadam, and Shuib Basri. 2014. *Twitter sentiment analysis. In Information Technology and Multimedia* (ICIMU), 2014 International Conference on. IEEE, 212–216.

[2] Alshenqeeti Hamza. 2016. *Are emojis creating a new or old visual language for new generations? A socio-semiotic study.* Advances in Language and Literary Studies 7, 6 (2016), 56–69.

[3] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. 2011. *Sentiment analysis of twitter data.* In Proceedings of the workshop on languages in social media. Association for Computational Linguistics, 30–38

[4] Barbieri, Francesco and Camacho-Collados, Jose and Ronzano, Francesco and Espinosa-Anke, Luis and Ballesteros, Miguel and Basile, Valerio and Patti, Viviana and Saggion, Horacio *(2018) SemEval-2018 Task 2: Multilingual Emoji Prediction, Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)*

[5] Barbieri Francesco, Kruszewski Germán, Ronzano Francesco, and Saggion Horacio. 2016. *How Cosmopolitan Are Emojis?: Exploring Emojis Usage and Meaning over Different Languages with Distributional Semantics.* In Proceedings of the 2016 ACM Conference on Multimedia Conference, MM 2016, Amsterdam, The Netherlands, October 15-19, 2016. 531–535.

[6] Baziotis, C., Athanasiou, N., Chronopoulou, A., Kolovou, A., Paraskevopoulos, G., Ellinas, N., Narayanan, S. and Potamianos, A., 2018. Ntua-slp at semeval-2018 task 1: Predicting affective content in tweets with deep attentive rnns and transfer learning. arXiv preprint arXiv:1804.06658.

[7] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. 2017. *Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm.* In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017. 1615–1625

[8] Britz, Denny. Understanding convolutional neural networks for NLP. URL: http://www. wildml. com/2015/11/understanding-convolutional-neuralnetworks-for-nlp/ (visited on 11/07/2015) (2015).

[9] Camacho-Collados J, School of Computer Science and Informatics Cardiff University, Taher Pilehvar M, School of Computer Engineering Iran University of Science and Technology Aug 2018 *On the Role of Text Pre-processing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis.*

[10] Chen, C., Gao, S. and Xing, Z., 2016, March. *Mining analogical libraries in q&a discussions--incorporating relational and categorical knowledge into word embedding.* In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)* (Vol. 1, pp. 338-348). IEEE.

[11] Chen, Yuxiao, Jianbo Yuan, Quanzeng You, and Jiebo Luo. *Twitter sentiment analysis via bi-sense emoji embedding and attention-based LSTM.* In Proceedings of the 26th ACM international conference on Multimedia, pp. 117-125. 2018.

[12] Christopher Olah, *Understanding LSTM Networks*, Posted on August 27, 2015 URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[13] *Twitter: number of monthly active users 2010-2019*, Published by Clement J., Aug 14, 2019 https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

[14] Çöltekin, Ç. and Rama, T., 2018, June. Tübingen-Oslo at *SemEval-2018 task 2: SVMs perform better than RNNs in emoji prediction.* In Proceedings of the 12th International Workshop on Semantic Evaluation (pp. 34-38).

[15] Efthymios Kouloumpis, Theresa Wilson, and Johanna D. Moore. 2011. *Twitter Sentiment Analysis: The Good the Bad and the OMG!* In Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011

[16] EISNER, Ben, et al. *emoji2vec: Learning emoji representations from their description. arXiv preprint arXiv:1609.08359*, 2016.

[17] Eisner Ben, Rocktäschel Tim, Augenstein Isabelle, Bosnjak Matko, and Riedel Sebastian. 2016. *emoji2vec: Learning Emoji Representations from their Description. In Proceedings of The Fourth International Workshop on Natural Language Processing for Social Media*, SocialNLP@EMNLP 2016, Austin, TX, USA, November 1, 2016. 48–54

[18] Pennington Jeffrey, Socher Richard, and Manning Christopher. 2014. *GloVe: Global Vectors for Word Representation* https://nlp.stanford.edu/projects/glove/

[19] Katy Steinmetz NOVEMBER 16, 2015 2:08 PM ET *Oxford's 2015 Word of the Year Is This Emoji* https://time.com/4114886/oxford-word-of-the-year-2015-emoji/

[20] Kharde, Vishal, and Prof Sonawane. *Sentiment analysis of twitter data: a survey of techniques.* arXiv preprint arXiv:1601.06971 (2016).

[21] Kun-Lin Liu, Wu-Jun Li, and Minyi Guo. 2012. *Emoticon Smoothed Language Models for Twitter Sentiment Analysis.* In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.

[22] Ljubešić, Nikola, and Darja Fišer. *A global analysis of emoji usage*. In Proceedings of the 10th Web as Corpus Workshop, pp. 82-89. 2016.

[23] T. LeCompte and J. Chen, *Sentiment Analysis of Tweets Including Emoji Data, 2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, 2017, pp. 793-798.

[24] Mohammed O. Shiha, Serkan Ayvaz, *The Effects of Emoji in Sentiment Analysis* 1 Department of Computer Engineering, Bahcesehir University, Besiktas, Istanbul, Turkey. 2 Department of Software Engineering, March 1, 2017. DOI: 10.17706/ijcee.2017.9.1.360-369

[25] *Sentiment 140*, datasets and additional information http://help.sentiment140.com/for-students

[26] Severyn, Aliaksei, and Alessandro Moschitti. *Unitn: Training deep convolutional neural network for twitter sentiment classification.* In Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015), pp. 464-469. 2015.

[27] Sepp Hochreiter and Jürgen Schmidhuber. 1997. *Long short-term memory.* Neural computation 9, 8 (1997), 1735–1780.

[28] SHIHA, M.; AYVAZ, Serkan. *The effects of emoji in sentiment analysis.* Int. J. Comput. Electr. Eng. (IJCEE.), 2017, 9.1: 360-369.

[29] SRIVIDHYA, V.; ANITHA, R. *Evaluating pre-processing techniques in text categorization.* International journal of computer science and application, 2010, 47.11: 49-51.

[30] Stojanovski, Dario, Gjorgji Strezoski, Gjorgji Madjarov, and Ivica Dimitrovski. *Emotion identification in FIFA world cup tweets using convolutional neural network.* In 2015 11th International Conference on Innovations in Information Technology (IIT), pp. 52-57. IEEE, 2015.

[31] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. *Sequence to sequence learning with neural networks.* In Advances in neural information processing systems, pp. 3104-3112. 2014.

[32] Tianran Hu, Han Guo, Hao Sun, Thuy-vy Thi Nguyen, and Jiebo Luo. 2017. *Spice Up Your Chat: The Intentions and Sentiment Effects of Using Emojis.* In Proceedings of the Eleventh International Conference on Web and Social Media, ICWSM 2017, Montréal, Québec, Canada, May 15-18, 2017. 102–111.

[33] Unicode Consortium: https://unicode.org/emoji/charts/emoji-list.html

[34] Vukotić, Vedran, Christian Raymond, and Guillaume Gravier. *Is it time to switch to word embedding and recurrent neural networks for spoken language understanding?* 2015.

[35] Wang, Peng, et al. *Semantic expansion using word embedding clustering and convolutional neural network for improving short text classification.* Neurocomputing 174 (2016): 806-814.

[36] Wilson, T., Wiebe, J. and Hoffmann, P., 2009. *Recognizing contextual polarity: An exploration of features for phrase-level sentiment analysis.* *Computational linguistics*, *35*(3), pp.399-433.

[37] Wu, Chuhan, Fangzhao Wu, Sixing Wu, Yongfeng Huang, and Xing Xie. *Tweet emoji prediction using hierarchical model with attention*. In Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, pp. 1337-1344. 2018.

[38] Xiang Li, Rui Yan, and Ming Zhang. 2017. *Joint Emoji Classification and Embedding Learning.* In Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data. Springer, 48–63.

[39] Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. *Show, attend and tell: Neural image caption generation with visual attention.* In International conference on machine learning, pp. 2048-2057. 2015. https://arxiv.org/abs/1502.03044

[40] Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.