

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

CZ4034: Information Retrieval

Course Assignment Report

11 April 2021

Group Members:

Adrian Goh Jun Wei, U1721134D

Bhagwat Abhishek, U1722796C

Goh Puay Hiang, U1820306A

Low Yu Benedict, U1821762E

Somani Palak, U1822868F

Foreword	4
1. Crawling	4
1.1 Crawling Method Employed	4
1.1.1 Academic Dataset	4
1.1.2 Scraped Webpages before Parler's Shutdown	4
1.2 Data Storage	5
1.3 Sample Queries	6
1.4 Corpus Information	6
2. Indexing and Querying	6
2.1 Indexing	6
2.1.1 Score	7
2.2 Querying	9
2.2.1 Simple Search Interface	11
2.2.2 Advanced Search Interface	13
2.3 Innovations	22
3. Classification	24
3.1. Convolutional Neural Networks for Sentiment Analysis	24
3.2. Recurrent Neural Networks for Sentiment Analysis	26
3.3. BERT for Sentiment Analysis (State of the Art)	27
3.4. Pre-Processing	29
3.4.1. Stop Words Removal	29
3.4.2. Variants of Lemmatization	29
3.4.2.1. Pure Lemmatization	29
3.4.2.2. Lemmatization with POS Tag	29
3.4.3. Stemming	29
3.5. Subjectivity Classification	30
3.5.1. Labelling Subjectivity	30
3.5.2. Custom CNN Model Training and Performance	31
3.5.3. Custom RNN Model Training and Performance	33
3.5.4. BERT Fine-Tuning and Performance	34
3.5.5. CNN, RNN and BERT Evaluation Metrics Comparison	34
3.5.6. CNN, RNN and BERT Performance Metrics Comparison	35
3.6. Polarity Classification	36
3.6.1. Labelling Polarity	37

3.6.2. Custom CNN Model Training and Performance	37
3.6.3. Custom RNN Model Training and Performance	40
3.6.4. BERT Fine-Tuning and Performance	41
3.6.5. CNN, RNN and BERT Evaluation Metrics Comparison	41
3.6.6. CNN, RNN and BERT Performance Metrics Comparison	42
3.7. Negative, Neutral, Positive Classification	44
3.7.1. Labelling Negative, Neutral, Positive	44
3.7.2. Evaluation Metrics	46
3.7.3. Performance Metrics	46
3.8. Fine-Grained Sentiment Classification	47
3.8.1. Labelling Fine-Grained Sentiment	47
3.8.2. Evaluation Metrics	48
3.9. Emotion Classification	49
3.9.1. Labelling Emotion	49
3.9.2. Evaluation Metrics	50
3.10. Apply Model on Dataset	51
Appendix A: CNN Architecture	53
Appendix B: RNN Architecture	54
Appendix C: Convolutions and Pooling	55
Appendix D: LSTM Network	58
Appendix E: Recurrent Neural Networks	63
Appendix F: BERT	65
References	68

Foreword

This document serves as the Project Report for CZ4034, Information Retrieval.

The link to the video presentation can be found here:

<https://youtu.be/5jGU0wa8TFo>

The link to the compressed file with crawled data, queries, results and classification can be found here:

https://entuedu-my.sharepoint.com/:f/g/personal/c180065_e_ntu_edu_sg/ElpDGilYvPBIn3xGLVuOLE4BqzoWV4TpMX4irtuzxyf10g?e=GA7i11

Lastly, the link to the source codes and libraries can be found here:

https://entuedu-my.sharepoint.com/:f/g/personal/c180065_e_ntu_edu_sg/Etll8WntKCpGgBnoNdoEcbYBtF60Hqjon1qtiuh-f8fqVw?e=xqQmIH

1. Crawling

1.1 Crawling Method Employed

1.1.1 Academic Dataset

The dataset used in our project is from an academic source (Aliapoulios et al., 2021) from before the Parler shutdown. The dataset was scrapped from the period from August 2018 to 11 January 2021. The dataset contains **183M posts** and **13.25M user profiles** and associated metadata.

The **total file size exceeds 120GB** therefore we had to manually clean and organise the dataset by using custom defined Python scripts. Alternative command line utilities such as **jq** and **sed** needed a much larger magnitude of processing time to manipulate binary files. In our test experiments, we found that on a file of 28GB, a command line tool needed about 28 hours, while our custom defined scripts completed the same tasks in about 2 hours.

1.1.2 Scraped Webpages before Parler's Shutdown

Our secondary source of data is from alternative sources who scraped data from Parler after the Capitol riots (6 Jan, 2021) till the day it was shut down (Brian Fung, CNN Business, 2021). Our data source is a hacker donk_enby who was prominently featured in all news

outlets across the world (Whittaker, 2021). All the archived data was open-sourced and available freely for public consumption. The links for the whole dataset archive can be found in the Appendix.

The data archived consists of mixed content types including videos, images and text and is about 32.1TB. For our purpose, due to limitations in computing power, we had to choose only the text content which is approximately around 18GB and consists of around 1.6M posts.

This data was further preprocessed by parsing each individual HTML file and collecting into a ndjson file and then merging into our preexisting dataset from section 1.1.1.

1.2 Data Storage

The data schema for the data collected in section 1.1 consists of the following tables and corresponding fields respectively.

Table 1.1 Data Schema (Users)

Users				
badges	banned	bio	blocked	comments
datatype	followed	following	human	id
integration	isFollowingYou	joined	key	lastseents
likes	media	muted	pendingFollow	posts
private	profilePhoto	rss	state	user_followers
user_following	userid	username	verified	verified_comments

Table 1.2 Data Schema (Posts/Comments)

Posts / Comments				
article	body	bodywithurls	comments	createdAt
createdAtForma	creator	datatype	depth	depthRaw

ttd				
followers	following	hashtags	id	impressions
lastseents	links	media	parent	posts
preview	reposts	sensitive	shareLink	state
upvotes	username	verified	urls	

They are stored in HTML files and ndjson files, with the need for further processing.

1.3 Sample Queries

It's possible to extrapolate different kinds of information from the dataset such as sentiments of posts, changing trends in the crowds or other applications. From this data, we can figure out what keeps the crowd motivated. It is possible to figure out what exactly the crowd likes and whether it's influenced based on their clout (follower count) or because of the recommendation engine of Parler.

1.4 Corpus Information

Table 1.3 Corpus Information

Table	Records	Words	Types
Posts	82,668,000 (74.6 GB)	1,111,628,924	58,293,066
Comments	59,674,000 (62.1 GB)	541,537,874	29,633,483

2. Indexing and Querying

2.1 Indexing

We will be indexing by 2 set of fields:

- **Search-and-rank fields:** These fields will be available for searching from the frontend. Ranking of results will also be affected by the degree of match for these fields.

- body
- hashtags
- **Ranking-only fields:** These fields are non-searchable by users, and will only be used for the ranking of results
 - score
 - posted_at

```
def search_data
{
  body: preprocessed_body,
  hashtags: hashtags,
  score: creator_score.to_f * reach_score.to_f,
  posted_at: posted_at
}
end
```

2.1.1 Score

The “score” field refers to how relevant this particular post is, regardless of the degree of match based on the queries. This will be used to affect the relevance of the results. It is made up of 2 derived sub-scoring, known as “creator_score” and “reach_score”. These scores are derived from a couple of factors.

“creator_score”

This is the score given to the creator of the post, based on his/her popularity, and thus the relevance of his/her post.

Some assumptions that will determine the creator score

- The more followers a creator is, the more people are interested in following and being updated with his/her post, and thus the more relevant his/her post is.
- The higher the follower-to-following ratio that a creator has the more relevant a post is
 - A user with 10 followings, 10,000 followers, is likely to have more influential impact than a user who only has 10,000 followings and 10,000 followers, despite both of them having the same number of followers

- If the creator of the post is verified, he/she is more likely to be a figure with high authority and thus, the post is considered as more relevant.

As such, the 3 key attributes that will be used to determine the creator score are:

1. Number of followers
2. Number of followings
3. Whether it is a verified user

To calculate the “creator_score” of a particular user, we will be multiplying the normalized scoring for followers, following and verified together.

```
def calculate_creator_score(followers_histogram, following_histogram):
    normalized_followers_score(histogram_bins: followers_histogram) *
    normalized_following_score(histogram_bins: following_histogram) *
    normalized_verified_score
end
```

“reach_score”

This is the score given to the post, based on its popularity, and thus relevance.

Some assumptions that will determine the reach score

- The more impressions a post has, the more people the post has reached and thus, the more relevant a post is.
- The more upvotes a post has, the more people agreed with the post and thus, the more relevant a post is.
- The more reposts a post has, the more people agreed with the post and thus, the more relevant a post is.

To calculate the “reach_score” of a particular post, we will be multiplying the normalized scoring for impressions, upvotes and reposts together.

```
def calculate_reach_score(
    impressions_histogram, upvotes_histogram, reposts_histogram
):
    normalized_impressions_score(histogram_bins: impressions_histogram) *
    normalized_upvotes_score(histogram_bins: upvotes_histogram) *
    normalized_reposts_score(histogram_bins: reposts_histogram)
end
```


Normalized scoring

Normalized scoring is obtained by finding the position of its value in its histogram of size 100, that is derived from all the values.

Normalized scoring is used to rescale the original value by the minimum and range of the vector, to make all the elements lie between 0 and 1 thus bringing all the values of numeric columns in the dataset to a common scale. We do data normalization because the relations are more important and give a better and more accurate representation of the relative impact of the score.

For example, let's say we have 3 creators with the following statistics:

Creator ID	1	2	3
Followers count	100	10,000	1,000,000

While the difference in number of followers between each creator is 100 times respectively, it does not mean their relative relevance are 100 times apart. Creator 3 might be a lot more relevant than creator 1 and 2, because it is much more difficult to amass a million followers. As such, there could be a lot more creators in the range of hundreds and thousands, while the number of creators in the range of millions is very low. Therefore, with normalization (using a histogram in our case), we are able to understand their relative relevance better, and using their normalized value will give us a much better and accurate representation when trying to score and rank the post.

The same explanation applies for other attributes beside followers, such as followings count, impressions, upvotes and repost.

2.2 Querying

In our project, we can perform interactive queries using our minimalistic frontend UI built using **Streamlit**. Streamlit is a web application framework for Python built mainly for Machine Learning applications. Using a single framework we are able to wrangle the frontend and backends together. Interactive components such as Charts and filters allow users to leverage the power of search to obtain the required information as quickly as possible and also gain insights into the data in our dataset.

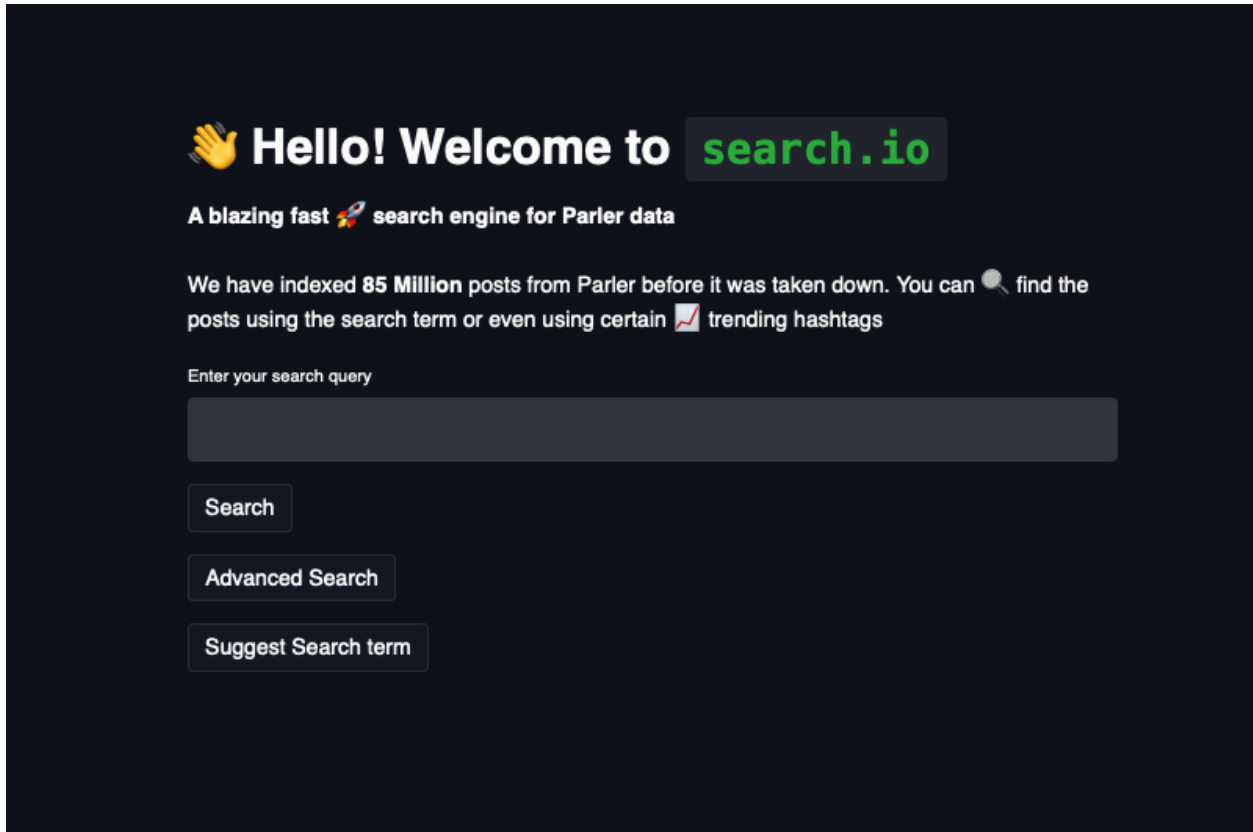


Fig-2.2.1: Search Interface of the app showing the search bar and search options

Figure 2.1.1 depicts the main interface of our app. You can observe a search bar where you can enter a search query in textual form. You then have three options. For more help on the search options and how you can filter your data in order to maximise impact from the data you have, you can refer to the sidebar of the app as shown in Figure 2.1.2.

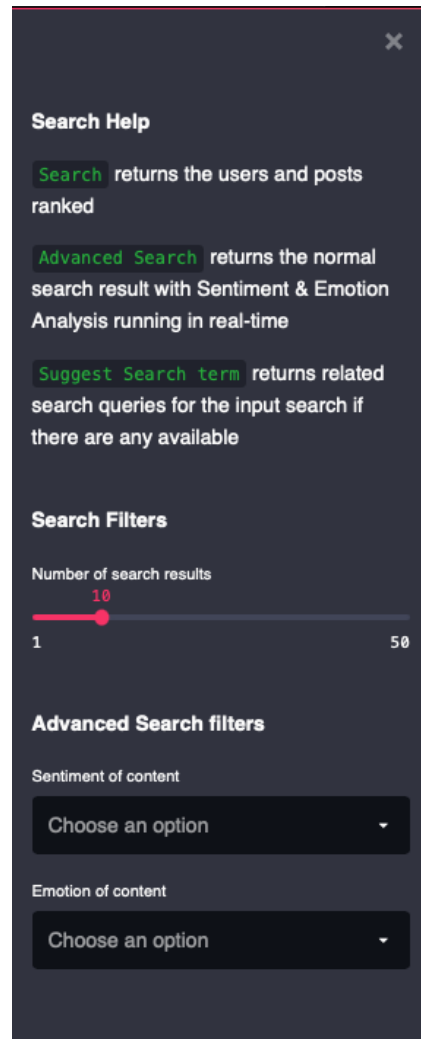


Fig-2.2.2: Sidebar of app showing search option descriptions and filters

Let us explore the three search options in the sections below.

2.2.1 Simple Search Interface

The simple search option queries the Elasticsearch engine which uses Lucene under the hood. Upon clicking the search button, an API call is made with a search query in the search bar and the number of search results to be returned set in the sidebar as the parameters. The API response is a JSON object containing a list of all the posts ranked from most relevant to least relevant. The response is nested as there is quite a lot of data returned, but for our particular use case, we flatten the JSON object, and store the data in a dataframe. Our frontend framework later renders a static table which stores the search result and username of the person who created that post.

Enter your search query

donald trump

Search

Advanced Search

Suggest Search term

Search Results

	_source_body	_source_username
0	Statement from President DonaldTrump	TeamTrump
1	DO THE MATH! President DonaldTrump had 20 to 30 thousand people at his rallies Joe Biden had 10 to 20 people at his backyard ralllies THERE IS NO WAY HE WON! 80 million votes? Whatever I DID THE MATH! IT DONT ADD UP Am I right or Am I right ????	RealTerrenceKWilliams
2	6 of 6: WATCH "Media Mistakes" #SLANTED #DonaldTrump @FullMeasureNews	SharylAttkisson
3	You could wear a normal mask & look like everybody else or you could choose Ghost Shield & look like yourself. ✓ Easy to Breathe in ✓ Light and Comfortable ✓ Reusable and Strapless ✓ Perfect for Glasses ✓ See smiles again ✓ Made in the USA 🇺🇸 >>> Click Image to Order! #usa #donaldtrump #thewhitehouse #presidenttrump #fraud #news #trump #counteverylegalvote #voterdtosaveamerica2020 #maga2020 #pennsylvania #nra #freedom #bluelivesmatter #2a	Theghostshield
4	The Battle for the White House: @SidneyPowell celebrates a court victory for President DonaldTrump in Georgia which stops Dominion voting machines in 3 counties from being wiped. #MAGA #AmericaFirst #Dobbs	LouDobbsTonight

Fig-2.2.3: Search results in the order of most related to least based on post metrics and scores

As discussed in the previous section, the ranking depends a lot upon the various post based metrics such as the number of **upvotes**, **followers** and post **impressions** and such. In order to give you a deeper insight into your search result, we have added interactive plots of these metrics for every post returned in the search results.



Fig-2.2.4: Plot of metrics for search results returned in the above search.

2.2.2 Advanced Search Interface

The advanced search option combines the simple search with Sentiment and Emotion analysis. Since our dataset is extremely huge, we are unable to perform these tasks on the data beforehand and store it in Elasticsearch as we do not have the computing power for that task. We instead perform both fine grained sentiment and emotion analysis in real

time on the frontend. In Section 3, we explore more on the models we use for these tasks but on the ML backend we make use of transformers from HuggingFace which provides a wrapper for these models. When you perform an advanced search on a certain query, an API call is made to the Elasticsearch backend to obtain the ranked results. The result body is then preprocessed and stored in a dataframe and then sent to the ML backend. Here we leverage multiple transformers to get the results for sentiment analysis and emotion analysis. Finally the results are aggregated on the frontend and a static table for the results is generated. This step takes a bit more longer than a normal search which is usually instantaneous as we are doing the inference of sentiments and emotions in real time. To further enhance the search experience and to gain better insights, you can filter the search results by multiple or single sentiments and emotions.

Enter your search query

donald trump

Search

Advanced Search

Suggest Search term

	_source_body	_source_username	_score	fg_sentiment	em_sentiment
0	Statement from President DonaldTrump	TeamTrump	10.7479	SOMEWHAT POSITIVE	SURPRISE
1	DO THE MATH! President DonaldTrump had 20 to 30 thousand people at his rallies Joe Biden had 10 to 20 people at his backyard rallies THERE IS NO WAY HE WON! 80 million votes? Whatever I DID THE MATH! IT DONT ADD UP Am I right or Am I right ????	RealTerrenceKWilliams	10.6881	SOMEWHAT NEGATIVE	ANGRY
2	6 of 6: WATCH "Media Mistakes" #SLANTED #DonaldTrump @FullMeasureNews	SharylAttkisson	10.6826	NEUTRAL	FEAR

Fig-2.2.5: Search results from Advanced Search including fine grained sentiments and emotions

You may refer to Figure 2.2.2 to see the options to filter search results by sentiments and emotions.

Incremental Indexing

As Parler is at the moment in a dormant state, the data is not available on the API level. We would therefore not be able to scrape for fresh data and would have to rely on alternative data sources. We are interested in gathering data from Gab as one of the alternative sources as both of them are “free-speech” oriented social networks. The Gab data dump is currently only available for academics for research(*GabLeaks*, n.d.).

Sample Queries and Execution times

Trigger Warning : All data shown is from the real app data and hence may contain inappropriate content such as profanities, racism, sexism, violence and related content. Please view at your own discretion.

For the sample queries, we will benchmark the results based on the number of posts returned by the Elasticsearch backend.

Enter your search query

coronavirus

Search

Advanced Search

Suggest Search term

Search Results

	_source_body	_source_username	_score
0	What Coronavirus? Gavin Newsom Dines Out With Large Group While Telling You to Lockdown	SeanHannity	10.7420
1	<p>**ASSOCIATED PRESS: "Some young children have forgotten how to eat with a knife and fork and others have regressed back into diapers as the coronavirus pandemic and related school closures take a toll on young peoples' learning."**</p>	SeanHannity	10.7366
2	<p>#Tucker let Gavin Newsom have for ignoring his own coronavirus restrictions. It's already making the far-left upset.</p>	DineshDSouza	10.7326
3	<p>1. Wall-to-wall media today on an increase in people testing positive for the coronavirus. They blame it on President Trump and Republican governors, which is what propagandists for the Democrat Party do. 2. As it takes about 10-14 days for the virus to kick in, virtually nothing is being said about the "mostly peaceful" rioters, most of whom are Democrats, and their contribution to any increase, or the politicians who cheered them on. Moreover, there's tons more testing. 3. Consequently, there's a lot more cases discovered. But what of the death rate? That is, why are the media virtually silent about it? The answer: it's .26 percent.</p>	Marklevinshow	10.7309
4	New York Gym Owner Says 'We Will Not Comply' – Rips Up \$15,000 Coronavirus Lock-down Fine.	DiamondAndSilk	10.7244

Fig-2.2.6: Query (1) results

Enter your search query

voter fraud

Search

Advanced Search

Suggest Search term

Search Results

	_source_body	_source_username	_score
0	will speak to @blocked tomorrow and @blocked Thursday about this. Mornings w maria. #voterfraud	Mariabartiromotv	10.7279
1	The 30 allegations made by @SidneyPowell are mostly based on witness & expert statements & relate to mail-in ballot #voterfraud & insecurities, #recount irregularities & deficiencies, as well as security hazards of #Dominion Voting Systems machines... story by @PetrSvabET	JanJekielek	10.7194
2	Join us today for the rebroadcast of Sunday morning futures on Fox News @ 3pm et. #breakingnews on the lawsuits coming tomorrow & this week from @blocked @blocked + @TedCruz @blocked @RepKevinMcCarthy #voterfraud	Mariabartiromotv	10.6868
3	Twitter on Wednesday suspended the account of #Pennsylvania Sen. Doug Mastriano, a Republican who called an #election oversight hearing that presented allegations and evidence of #VoterFraud. @Transparency2020 #Transparency2020	ntdnews	10.6711
4	#voterfraud #presidenttrump	BenStein	10.6664

Fig-2.2.7: Query (2) results

Enter your search query

impeach trump

Search

Advanced Search

Suggest Search term

Search Results

	_source_body	_source_username	_score
0	Just so I understand, I was out running errands. Former VP #JoeBiden is calling for the impeachment of #PresidentDonaldTrump for the very thing Biden bragged about doing as VP? Am I right? #ImpeachDonaldTrumpNOW #ImpeachTrump #BrainDead #Trump2020	JoePags	10.3564
1	Nancy Pelosi's impeachment inquiry of @realDonaldTrump will go down in history as one of the defining moments that led to his re-election in 2020. #Impeachment #ImpeachTrump	scottpresler	10.3407
2	THE PUSSIES CRYING FOR A CIVIL WAR THAT ORCHESTRATED THE TERRORIST ATTACK YESTERDAY ARE NOW TRYING TO BLAME ANTIFA – FUCKKK NO!! . #maga #georgia #whiteprivilege #bluelivesmatter #holdtheline #trumpmob #coup #washingtondc #amerikka #COVIDIOTS #impeachtrump #teamtrump #fightfortrump #sedition #deplorables #trumptreason #stopthesteal #riggedelection #voterfraud #kagpatriots #democracy #antifa #proudboys #traitortrump #pence #jan6 #presidentdonaldtrump #trumptapes #blacklivesmatter #stopthesqueal #georgia #gopbetrayedamerica #countryovertrump #americafirst #presidenttrump #stormthecapitol #thestormisuponus #qarmy #teamtrump #lawandorder #qcucksklan #americanhero #stopthesteal #Trump2020 #Maga2020 #Kag #Kag2020	TrumpTreason45	10.3159
3	I've walked the streets of West Baltimore. I've shaken the hands of homeless living in Los Angeles. I reject the democrat party that chooses illegal aliens over Americans & I reject their impeachment of a duly-elected President. #Impeachment #ImpeachTrump	scottpresler	10.3139
4	#Impeach #impeachtwice #impeachagain #impeachtrump	FuckTrumpToHell	10.3105

Fig-2.2.8: Query (3) results

Enter your search query

capitol riots

Search

Advanced Search

Suggest Search term

Search Results

	_source_body	_source_username	_score
0	The curtain is pulled back as a wall of propaganda hits, and the establishment hoodwinks people re; the capitol riot. (DM) #CapitolRiots #Capitol	Styxhexenhammer666	10.4401
1	I was there when #AshliBabbitt was gurneyed out. Videood 10 mins following her blood trail. THERE WAS NO RIOT. "NO RIOT: What National File Reporter A.J. Cooke Experienced First Hand While Covering The Capitol Protest" #CapitolRiots #CapitolHillMassacre	AJCooke1776	10.3167
2	The media is the enemy of the people. Never forget that.. #Trump #CapitolRiots #CapitolBuilding #Dc #media #facebook	RobertBeadles	10.3144
3	The One Minute News - January 8, 2020 -Daily News- -CLICK TO LISTEN AND SUBSCRIBE- Friday - Biden, Capitol Rioters, AOC, DOJ, Boston Marathon Bomber, FBI, Pipe Bombs, US Chinese Embassy, Uyghur, Kim Jung Un, Pakistan, Apple Store, History #news #breakingnews #dailynews #trump #joe Biden #history #comedynews #memes #comedy #truth #capitol #capitolriot #aoc #doj #boston #fbi #pipebombs #china #uyghur #kimjungun #pakistan #apple #podcats	Theoneminutenews	10.2932
4	The One Minute News - January 8, 2020 -Daily News- -CLICK TO LISTEN AND SUBSCRIBE- Friday - Biden, Capitol Rioters, AOC, DOJ, Boston Marathon Bomber, FBI, Pipe Bombs, US Chinese Embassy, Uyghur, Kim Jung Un, Pakistan, Apple Store, History #news #breakingnews #dailynews #trump #joe Biden #history #comedynews #memes #comedy #truth #capitol #capitolriot #aoc #doj #boston #fbi #pipebombs #china #uyghur #kimjungun #pakistan #apple #podcats	Theoneminutenews	10.2813

Fig-2.2.9: Query (4) results

Enter your search query

black lives matter

Search

Advanced Search

Suggest Search term

Search Results

	_source_body	_source_username	_score
0	If you listen closely, you can probably hear the sound of CNN turning off its cameras so people never hear this man's message to #BlackLivesMatter	DineshDSouza	10.6782
1	The Premier League are trying to back-peddle over #blacklivesmatter They went on bended knee to the thuggery. Against fans wishes. I hope they pay the price.	KTHopkins	10.6526
2	Brilliant. As a British football club take a knee to pander to #blacklivesmatter – some plucky soul flies a banner "white lives matter" across the stadium. Leftists are in MELTDOWN	KTHopkins	10.6499
3	Massive own goal by #blacklivesmatter in the U.K. Turns out that they need a rich, privileged white boy to build their statues for them #bristolstatue	KTHopkins	10.6454
4	RACE BAITING PIMP! Al Sharpton is a Race Baiting Pimp who only shows up when he can make money. He don't care about #BlackLivesMatter Only the #GreenDollarsMatter to him Raise your hand 🖐️ if you are sick of him spreading hate!	RealTerrenceKWilliams	10.6450

Fig-2.2.10: Query (5) results

Table 2.1 Sample Query Time

Text	Query Time (ms)
coronavirus	277
voter fraud	234
Impeach trump	270
capitol riots	222
black lives matter	257

2.3 Innovations

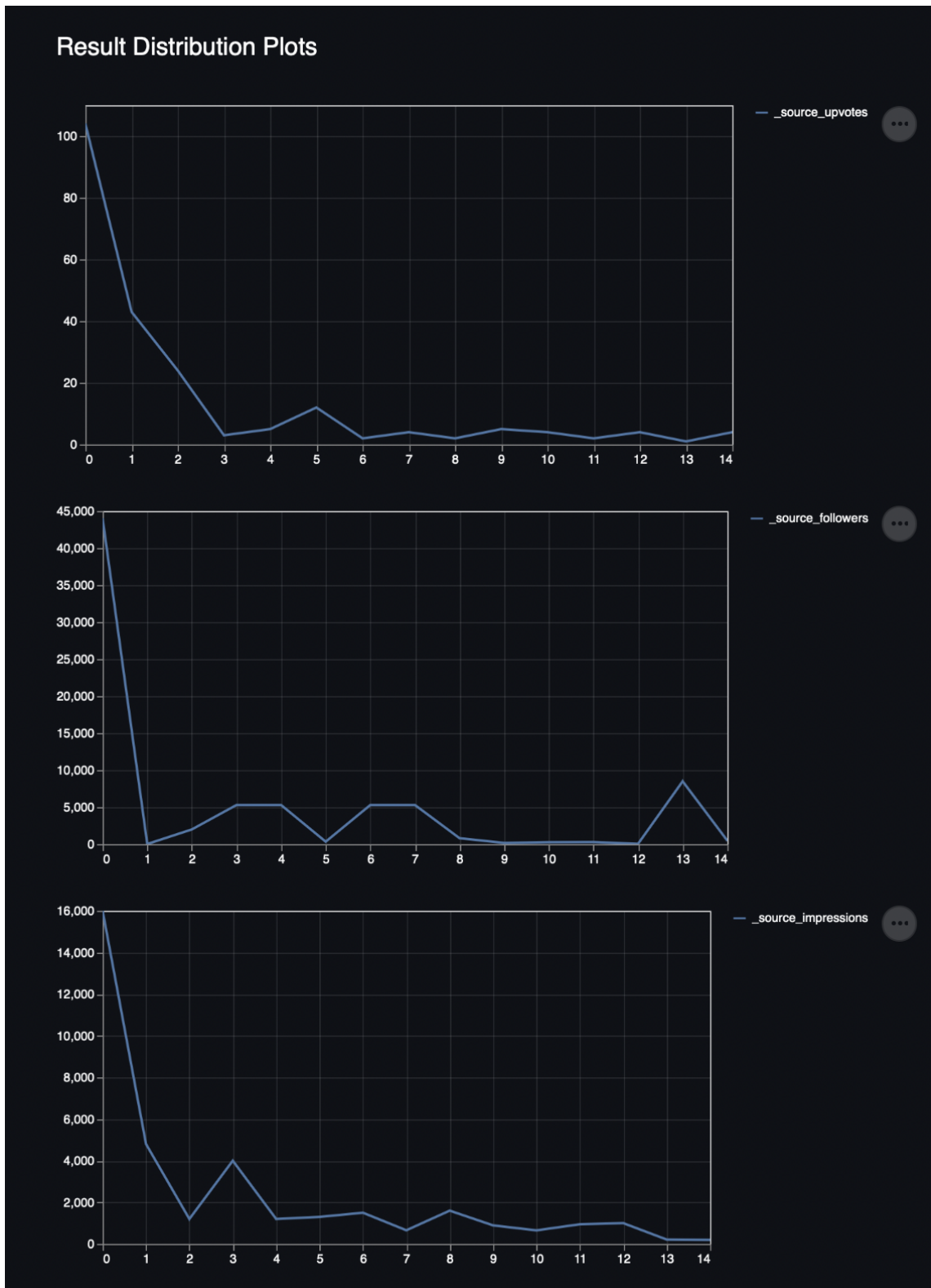
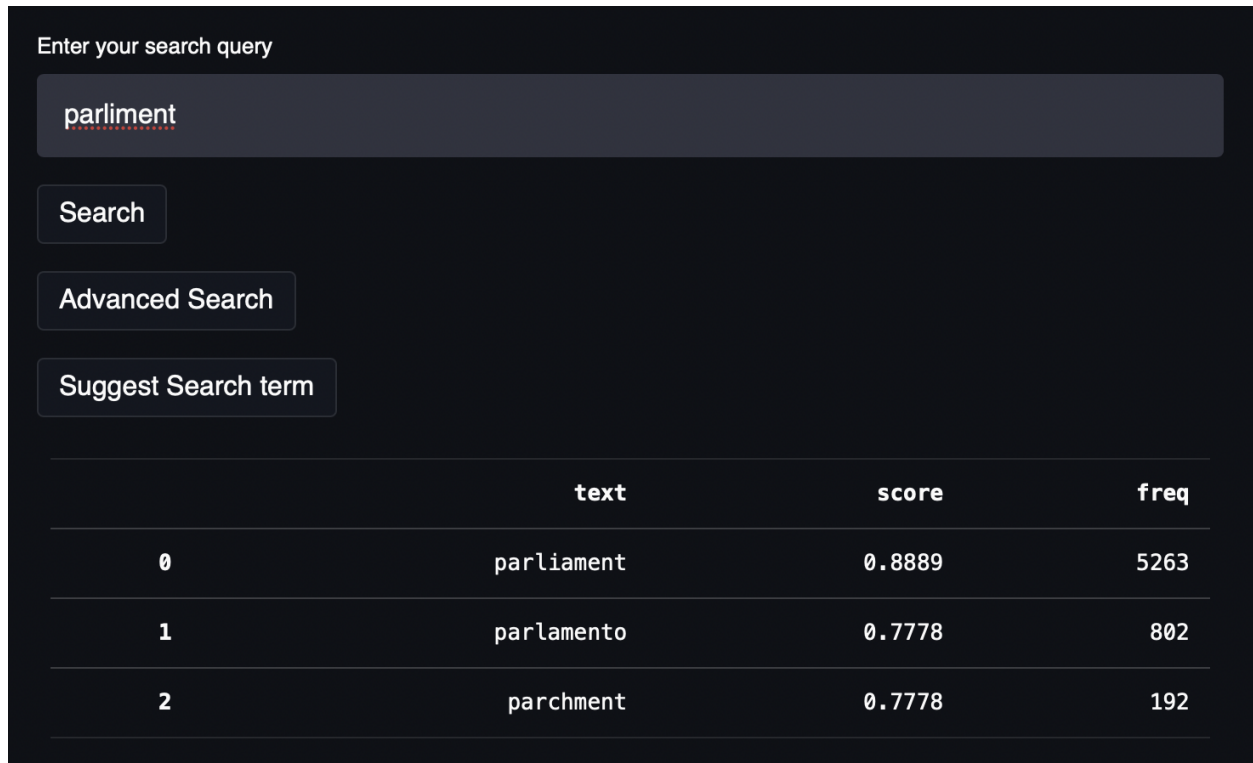


Fig-2.3.1: Interactive charts plotting the distribution of various metrics

Based on each query, we managed to generate the distribution of the impression, reposts and follower count of the tweet. Based on that, we are able to gather how popular the search term is and whether the influencers on parler are talking about it. We will also be able to find out if it's a ground up movement by looking at the distribution of the follower counts against the impression count.

Suggestions



Enter your search query

parliment

Search

Advanced Search

Suggest Search term

	text	score	freq
0	parliament	0.8889	5263
1	parlamento	0.7778	802
2	parchment	0.7778	192

Fig-2.3.2: Autosuggest or autocorrect feature displaying the nearest search terms, score and frequency in dataset

The web application will provide suggestions based on the frequency of terms and phrases using the query string.

All these innovations can be classified under:

- Interactive search - Based on user input, we are able filter or modify results
- Enhanced search - We provide interactive line plots to get the distribution of the various metrics based on our search result.

3. Classification

This section covers the classification portion of the project.

We begin with a brief introduction of the different models - Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and the state of the art Bidirectional Encoder Representations from Transformers (BERT) - we experimented with. We have also included details on how these models work under the hood in Appendix A to E.

After which, we will discuss the methods employed for various classification problems we tackled and the evaluation and performance metrics. The problems tackled are listed here:

1. Subjectivity Detection
2. Polarity Detection
3. Subjectivity with Polarity Detection
4. Fine-Grained Sentiment Analysis
5. Emotion Based Sentiment Analysis

Integral Libraries Used: Huggingface Transformers (TF/Torch), TensorFlow, PyTorch

3.1. Convolutional Neural Networks for Sentiment Analysis

3.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN's) are the backbone of several computer vision applications. Several tasks including object detection, image classification, and semantic segmentation can be effectively tackled by CNNs. It is a Deep Learning algorithm which takes in an image as an input, assigns importance (learnable weights and biases) to various aspects/objects in the image so as to eventually be able to differentiate one from the other with great precision.

A CNN usually consists of three layers: a convolutional layer, a pooling layer, and lastly a fully connected layer. (Fig-3.1 depicts the architecture of a CNN network). Refer to the Appendix-C to learn more about convolutions and pooling.

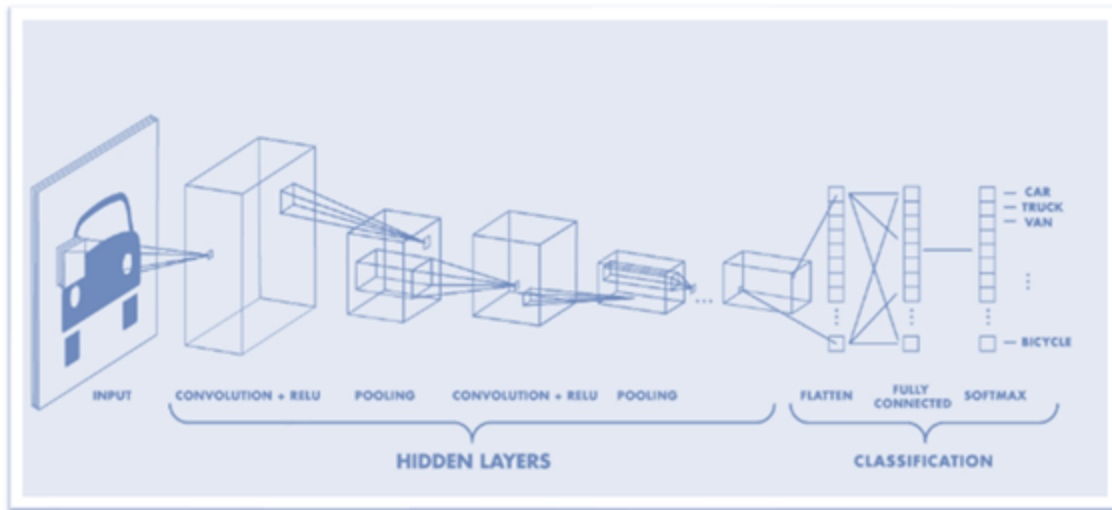


Fig-3.1: Architecture of a Convolutional Neural Network
Source: Adapted from [1]

3.1.1.1 CNN's for Sentiment Analysis

Images are just some points in space, just like the word vectors are. By representing each word with a vector of numbers of a specific length and stacking a bunch of words on top of each other, one can obtain an "image", and thereafter leverage CNN's for analysis.

In the case of sentiment analysis, the input consists of a 1-D array, and thus a 1-D convolutional layer has to be used. Similar to a two-dimensional convolutional layer, a one-dimensional convolutional layer uses a one-dimensional cross-correlation operation. For the case of a one-dimensional cross-correlation operation, the convolution window starts from the leftmost side of the input array and slides on the input array from left to right successively. When the convolution window slides to a certain position, the input subarray in the window and kernel array are multiplied and summed by element to get the element at the corresponding location in the output array. Fig-3.2 below illustrates an example of the same.



Fig-3.2: One-dimensional cross-correlation operation. The shaded parts are the first output element as well as the input and kernel array elements used ($0 \times 1 + 1 \times 2 = 2$)

The architecture of the custom CNN used in this project can be found in Appendix A. We also used a callback that reduces learning rate based on the validation data F1 Score. All hyper-parameters were adjusted through trial and error based on how the model performs and observing the behaviour (how long it seems to take till overfitting results in poorer results etc.).

Hyper parameters used that are not related to the model's architecture are outlined in the table below.

Table 3.1: CNN Hyper Parameters

Hyper Parameter	Value
Learning Rate	0.005
Epochs	30
Batch Size	1024
Learning Rate Decay Factor	0.1

3.2. Recurrent Neural Networks for Sentiment Analysis

3.2.1 Recurrent Neural Networks

A recurrent neural network (RNN) is a type of artificial neural network which uses time series or sequential data. They are ideal for use in temporal or ordinal problems, such as natural language processing (nlp), language translation and speech recognition. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, this is not the case for RNN's. In fact, the output of a recurrent neural network depends on the prior elements within the sequence.

While future events might also be useful in determining the output for a given sequence, unidirectional recurrent neural networks are not able to account for such events during prediction.

The architecture of the custom RNN used in this project can be found in Appendix B. Similarly, we also used a callback that reduces learning rate based on the validation data F1 Score. All hyper-parameters were also adjusted through trial and error.

Hyper parameters used that are not related to the model's architecture are outlined in the table below.

Table 3.2: RNN Hyper Parameters

Hyper Parameter	Value
Learning Rate	0.005
Epochs	30
Batch Size	1024
Learning Rate Decay Factor	0.3

3.3. BERT for Sentiment Analysis (State of the Art)

BERT (Bidirectional Encoder Representations from Transformers) employs Transformer, an attention mechanism that learns contextual relations among words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder and a decoder. The encoder reads the input in the form of text and the decoder generates a prediction for the task. Given that BERT's goal is to generate a language model, just the encoder mechanism is required. In contrast to directional models, where text is read from left to right or right to left (i.e. sequentially), the Transformer encoder reads the whole sequence of words together at once. Therefore it is considered bidirectional in some sense.

The input is processed in the following way to aid the model in distinguishing among the two sentences in training:

1. A [CLS] and [SEP] token is inserted at the beginning of the first, and the ending of each sentence respectively.
2. Sentence A or Sentence B is indicated by a sentence embedding (similar to token embeddings with a vocabulary of 2) which is added to each token.
3. The position of each token in the sequence is indicated by a positional embedding.

Fig-3.3 also illustrates the above process.

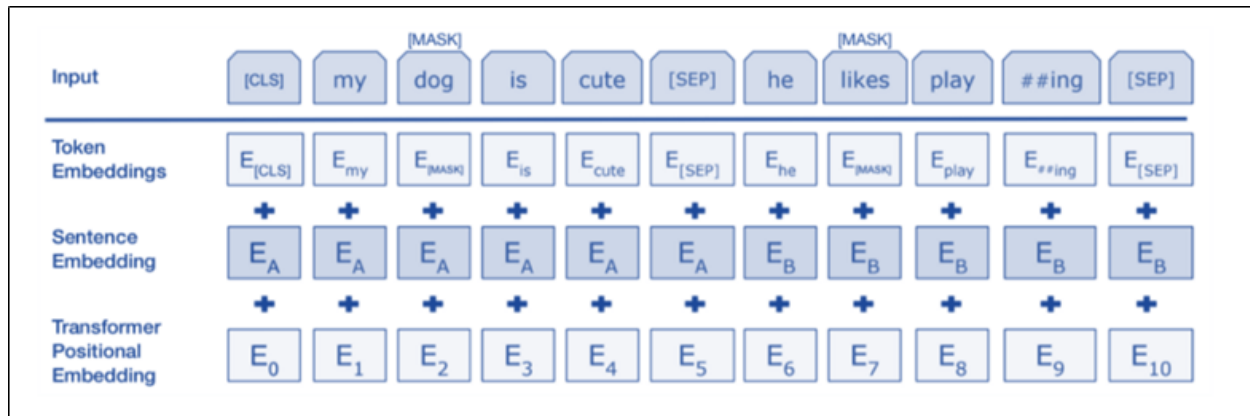


Fig-3.3: Bert with modifications

Source: Adapted from [8]

The following steps are then performed to determine whether the second sentence is indeed connected to the first:

1. The whole input sequence is fed through the Transformer model.
2. A simple classification layer (learned matrices of biases and weights) is used to transform the output of the [CLS] token into a vector of shape 2x1.
3. Softmax is used to calculate the probability of IsNextSequence.

Hence, Next Sentence Prediction and Masked LM are trained together (during the training of the BERT model) with an aim to minimise the combined loss function of the two strategies.

In this project, we used huggingface's transformer bert-base-cased. This particular transformer was used due to its excellent performance, short time required for fine-tuning and prediction (as compared to other transformers).

Furthermore, it was helpful that huggingface provided a simple API for fine-tuning with relevant documentations that were well written and easy to navigate. This was important as this was the first time some of the members in charge of machine learning have encountered BERT.

Bert-base-cased was used also because our dataset from Parler includes both upper and lowercase letters.

3.4. Pre-Processing

Pre-processing involves transforming the text into a form that is more analyzable and predictable to perform a task.

3.4.1. Stop Words Removal

The set of commonly used words in a language form the stop words of that language. “a”, “the”, “is”, “are” and etc. are examples of stop words in the English language. The intuition behind removing stop words is that, by removing low information words from text, we can focus on more important words instead.

3.4.2. Variants of Lemmatization

3.4.2.1. Pure Lemmatization

The process of converting a word to its base form is known as lemmatization. A word is converted to its meaningful base form keeping in mind the context of the word during lemmatization. The output of lemmatization is always a proper word of the language.

3.4.2.2. Lemmatization with POS Tag

The process of assigning a word to its grammatical category is known as Part-of-speech (POS) tagging. This is done in order to understand the role of a word within the sentence. Traditional parts of speech include verbs, nouns, conjunctions, adverbs, etc. POS tagging provides the contextual information that a lemmatizer requires when choosing the appropriate lemma of the word in a sentence.

3.4.3. Stemming

The process of converting inflection in words (e.g. troubled, troubles) to their root form (e.g. trouble) is known as stemming. It is important to note that unlike lemmatization, the “root” in this case might not be a word existent in the grammar, but rather just a canonical form of the original word.

A crude heuristic method that drops the ends of words in the hope of correctly transforming words into its root form is used during stemming. For example, the words “troubles”, “troubled”, and “trouble” may be converted to “troubl” rather than “trouble” since the last few characters are simply removed.

Several algorithms might be used to implement stemming with the “Porter's” algorithm being the most popular one.

The aim of both lemmatization and stemming is to reduce derivationally related or inflectional forms of a word to a common base form.

3.5. Subjectivity Classification

Subjectivity classification involves classifying a sequence as *Opinionated* or *Neutral*. It is often a first step before subsequently performing polarity classification. This section covers the pipeline performed for subjectivity classification in this project.

3.5.1. Labelling Subjectivity

It is worth noting that all labelling performed throughout this project was initially done using integer labels and subsequently processed into categorical wordings (i.e. initially labelled 0, 1 we then post-process it to ‘NEUTRAL’, ‘OPINIONATED’) to avoid confusion and improve readability. This labelling process of internally agreeing on an integer label and subsequently replacing integer labels with textual ones is repeated for all subsequent classification performed later. There were a total of two annotators for every labelling exercise, Palak and Benedict. Henceforth, we will refer to the Palak as *Annotator 1* and the Benedict as *Annotator 2*.

Early into our labelling process we realised that our dataset contained very few neutral statements. We started off with the intention to label 600 *Neutral* and 600 *Opinionated* but ended up labelling 4,097 rows before finally finding the 600th neutral row. As we had already intended to perform polarity classification, we also labelled all opinionated rows as *Positive* and *Negative*. The polarity labelling will be discussed in Section 3.6.1.

Of the 4,097 rows labelled, we had 3,866 labelled agreements. Annotator 1 labelled 3,368 rows as *Opinionated* and 729 rows as *Neutral*. Annotator 2 labelled 3,399 rows as *Opinionated* and 698 rows as *Neutral*.

Therefore, given $p_o = 0.944$,

$p_e = p_{opinionated} + p_{neutral} = (0.822 \times 0.830) + (0.178 \times 0.170) \approx 0.713$ and Cohen's kappa formula,

$$\kappa \equiv \frac{p_o - p_e}{1 - p_e}$$

We get Cohen's Kappa to be **0.805**. The comma separated values file for subjectivity labelling can be found in *raw_subjectivity.csv*. The subsequently cleaned and balanced dataset can be found in *cleaned_subj_polar_.csv*.

3.5.2. Custom CNN Model Training and Performance

For each type of pre-processing covered in section 3.4, we trained a CNN model ten times and averaged the values. This ensures a more reliable gauge of each pre-processing technique's impact on performance.

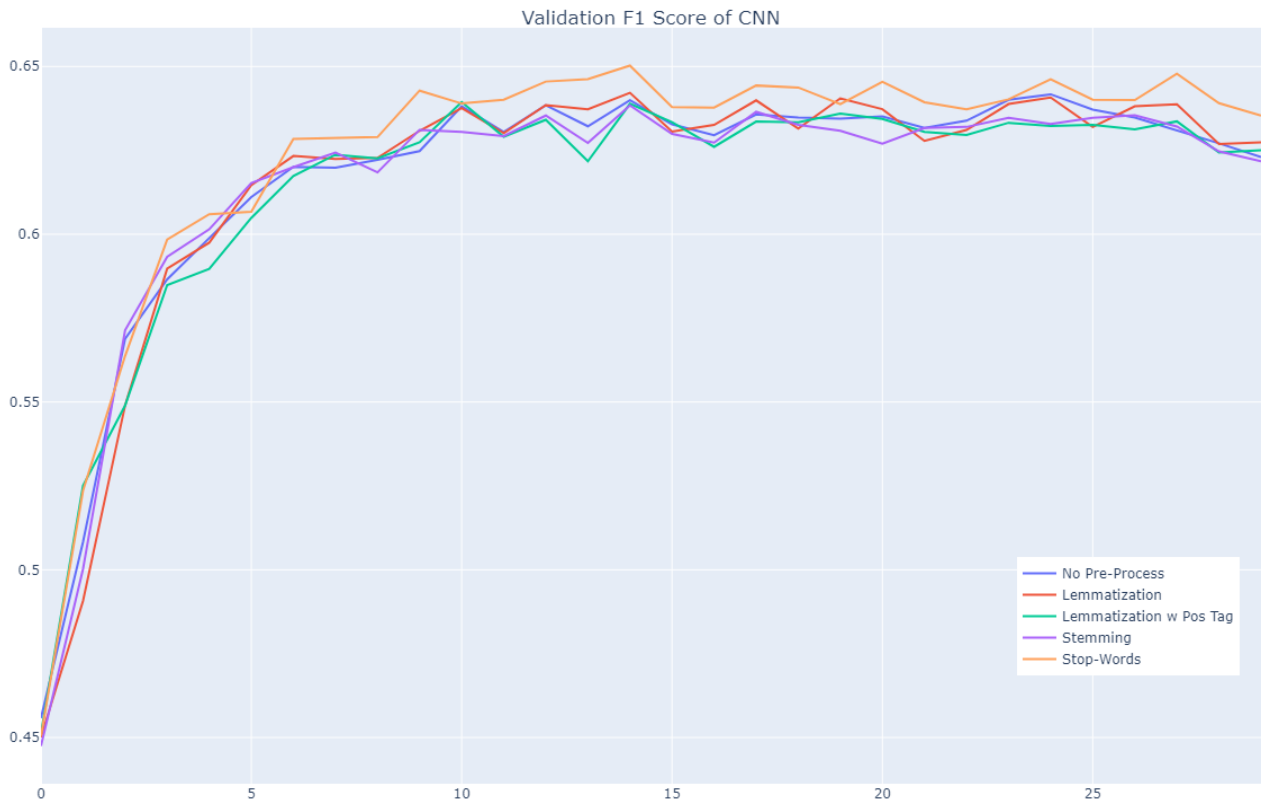


Fig-3.4 CNN Subjectivity Performance with Different Pre-Processing

From Fig-3.4 we can see that using stop-words removal pre-processing convincingly increased the F1-Score as compared to all other pre-processing. We believe that this could be attributed to how a CNN is trained. As stop words tend to yield little to no useful information, removing these stop words allows us to distill the sequences fed to our model to only the most important words.

For example, in the phrase “I am so angry at Biden” removing stop words will give “I angry Biden”. Suffice to say, stop-words removal effectively distills the sequence into the key words - and in this case the word “angry” is extremely helpful for determining the sequence is opinionated, whereas the words “so” and “at” might not be very helpful in this context. Furthermore, the use of pooling layers in our CNN model allows for our model means that removing stop-words can reduce the distance between essential words in the sequence, and allow for a better generalization overall.

On the other hand, lemmatization appeared to be a second option that could potentially improve the F1-Score. As such, we tried combining both stop-word removal and lemmatization to see if it would improve the overall performance of the model. However, as seen in Fig-3.5, combining both lemmatization and stop-words removal improved the performance as compared to just utilizing lemmatization only, but still fell short of the impact gained from only using stop-words removal. As such, we conclude that performing stop-words removal prior to training our CNN model will yield the best results.

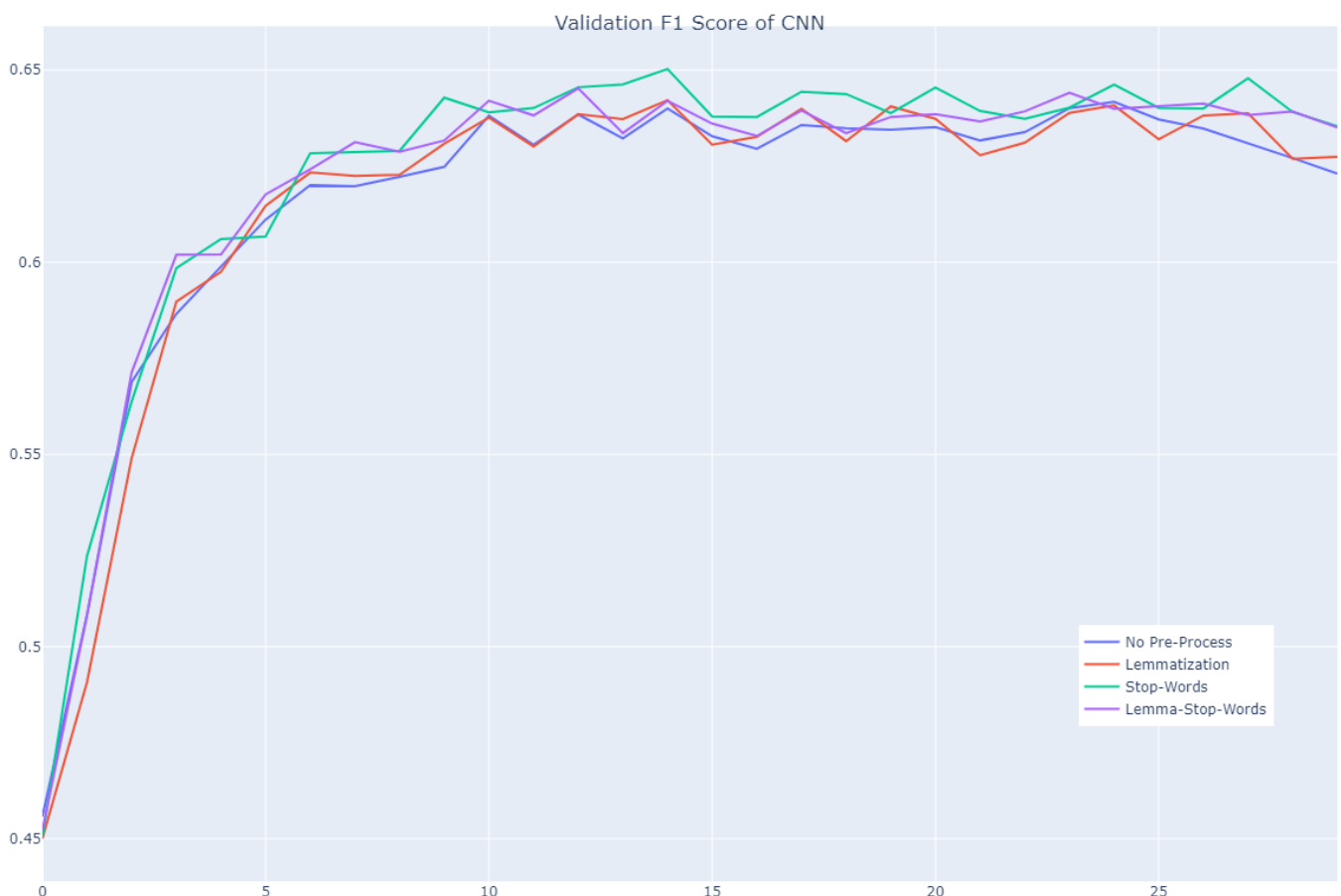


Fig-3.5 CNN Subjectivity Classification Performance with Hybrid Pre-Processing

3.5.3. Custom RNN Model Training and Performance

Similar to our CNN, we trained a new RNN model ten times using each pre-processing technique, and averaged the values. As the overall performance of each pre-processing type for our RNN is generally close, Fig-3.6 shows the graph zoomed in after epoch 6 where the F1-Score has mostly converged.

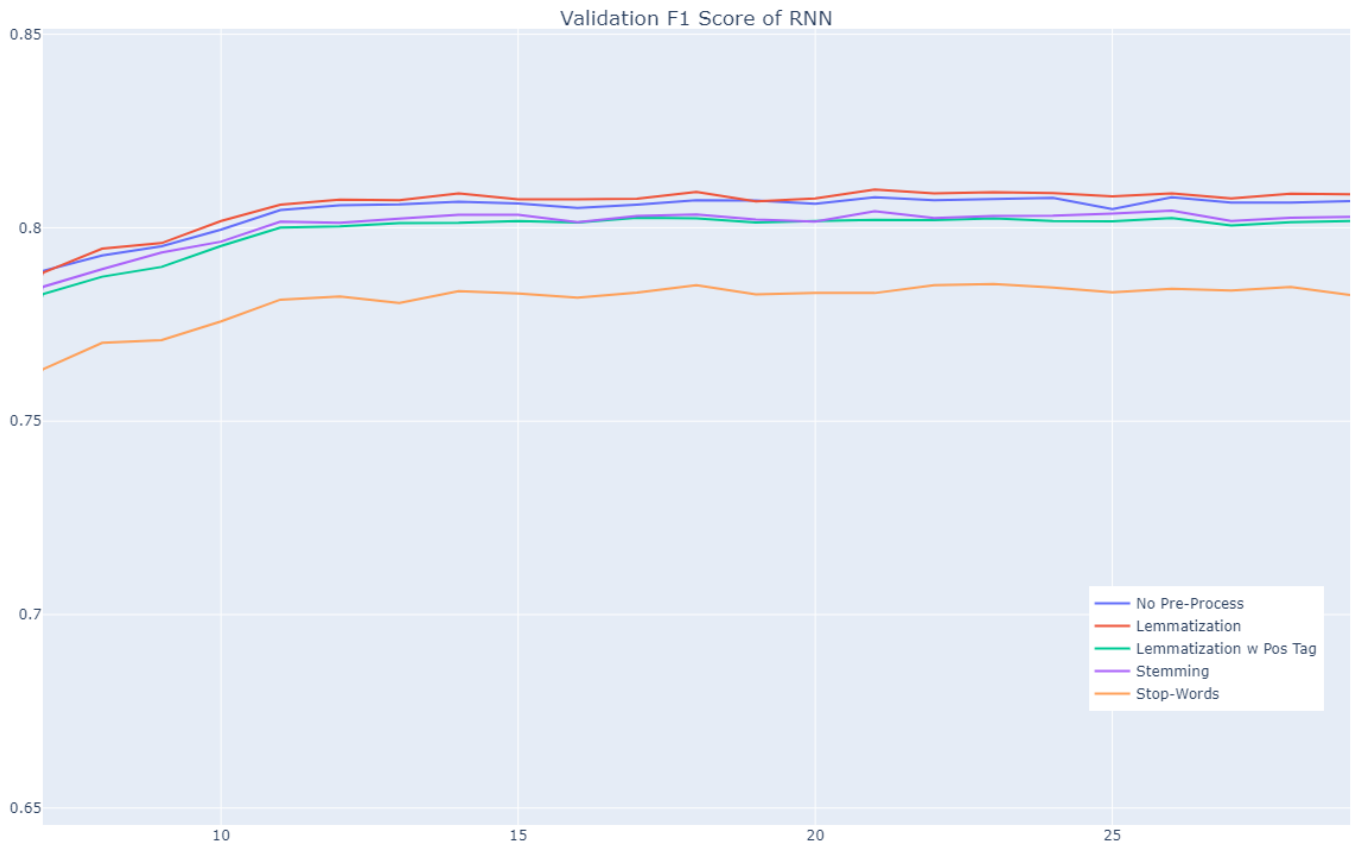


Fig-3.6 Zoomed in RNN Subjectivity Classification Performance for Different Pre-Processing

From Fig-3.6, we can see that lemmatization is most effective while performing no pre-processing comes as a close second. As the other preprocessing techniques performed inferior to lemmatization, we conclude that performing lemmatization prior to training our RNN model yields the best results.

We hypothesize that the reason stop-words removal performed significantly poorly compared to the other pre-processing techniques is because RNNs are already highly capable of utilizing all the words in a sequence to perform any classification. As such, stop-words removal may in fact serve as a hindrance for our model to utilize the positions of words to make a classification.

3.5.4. BERT Fine-Tuning and Performance

As mentioned in Section 3.3, BERT performs its own pre-processing when tokenizing any sequence. As such, we performed fine-tuning without any of the pre-processing techniques used in the CNN and RNN training. As seen in Fig-3.7, our BERT model converges at epoch 6 at an F1 Score of 0.894.

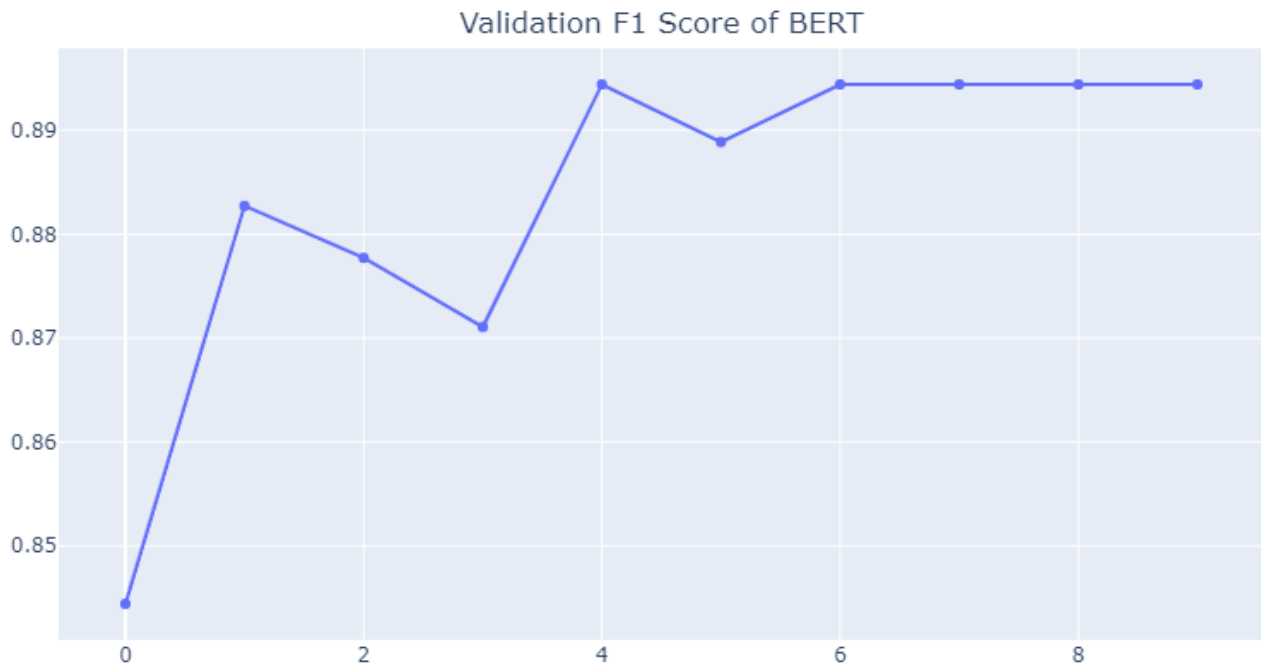


Fig-3.7 BERT Fine-Tuned Subjectivity Classification Performance

3.5.5. CNN, RNN and BERT Evaluation Metrics Comparison

Having identified the best preprocessing techniques for our CNN and RNN, we retrain the models and compare the evaluation performance of our CNN, RNN and BERT model on the testing dataset.

The figures below show the predictions in a confusion matrix for each model.

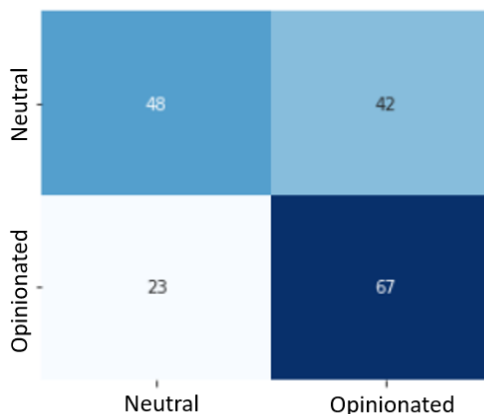


Fig-3.8 CNN Confusion Matrix

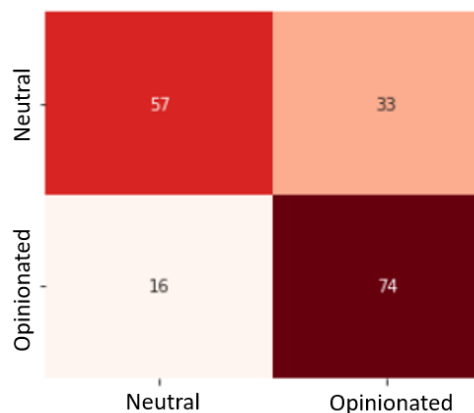


Fig-3.9 RNN Confusion Matrix

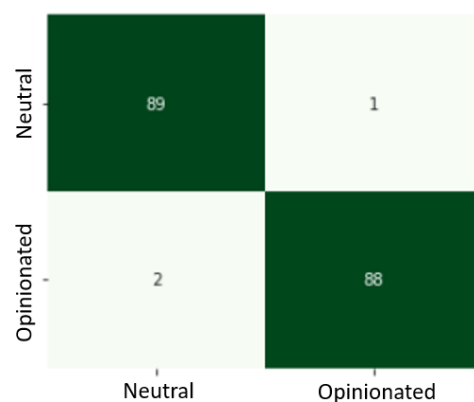


Fig-3.10 BERT Confusion Matrix

From the confusion matrices, we can compute the F1 Score and Accuracy. The results are shown in the figures below.

F1 Scores of CNN, RNN, BERT

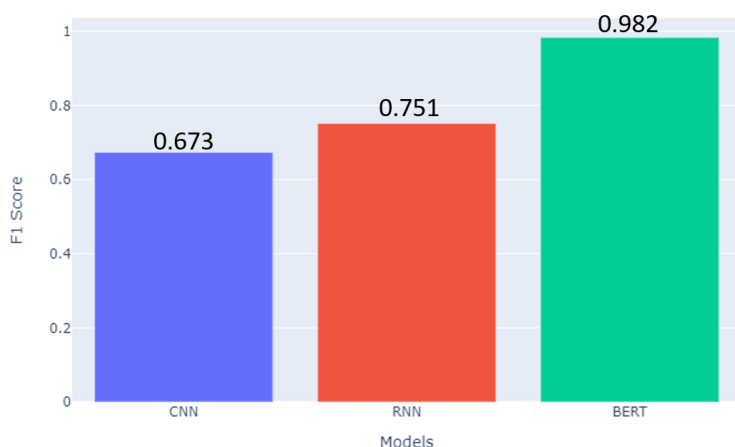


Fig-3.11 CNN, RNN, BERT F1 Scores

Accuracy of CNN, RNN, BERT

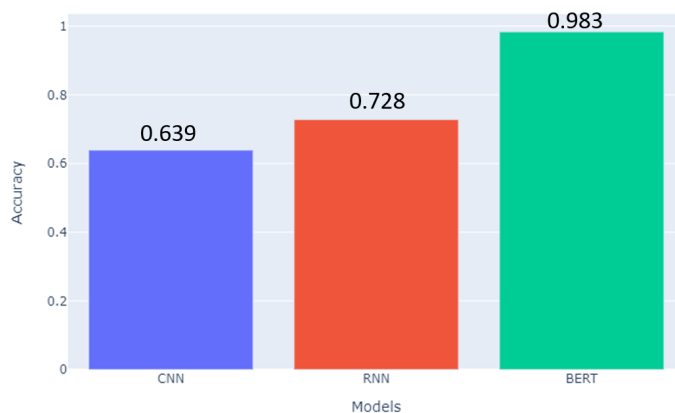
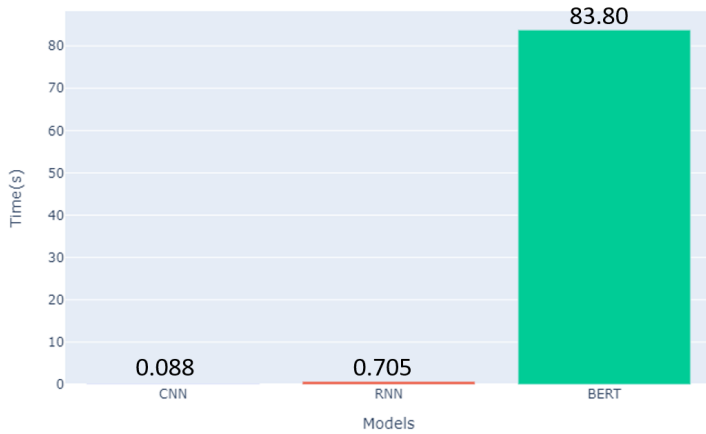


Fig-3.12 CNN, RNN, BERT Accuracy

3.5.6. CNN, RNN and BERT Performance Metrics Comparison

In addition to the evaluation metrics, we also compare the performances of our model. Namely, the time to predict 1,000 sequences averaged over ten iterations, and the average time to train each model per epoch. It is worth noting that for this project, a RTX 1060 GPU was used for training each BERT model. Better hardware may result in much quicker computation time.

Training Time per Epoch of CNN, RNN, BERT



Average Prediction Time of CNN, RNN, BERT for 1,000 Queries

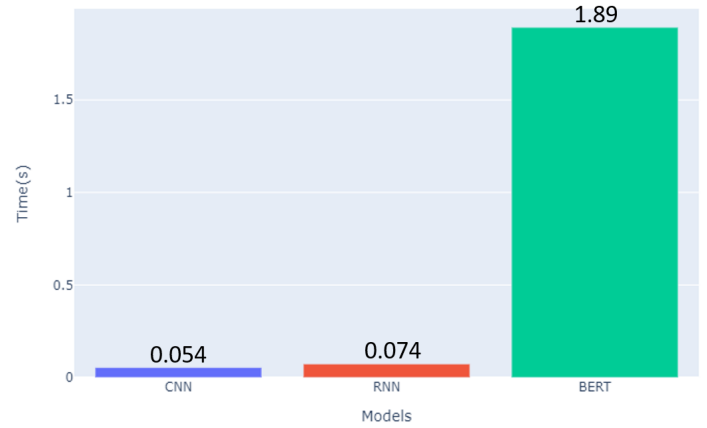


Fig-3.13 Average Training Time per Epoch of CNN, RNN, BERT

Fig-3.14 Average Prediction Time of CNN, RNN, BERT

The summary of, the evaluation and performance metrics for all three models are detailed in Table 3.3.

Table 3.3 Summary of Evaluation and Performance Metrics

Metrics		CNN	RNN	BERT
Evaluation Metrics	Accuracy	0.639	0.728	0.983
	F1 Score	0.673	0.751	0.983
Performance Metrics	Average Training Time per Epoch (s)	0.088	0.705	83.80
	Average Prediction Time (1,000 Sequences) (s)	0.054	0.074	1.89

3.6. Polarity Classification

Polarity classification involves classifying a sequence as *Positive* or *Negative*. In the pipeline of our project, polarity classification is done after subjectivity classification. That is, sequences classified as *Opinionated* are then passed forward to the next model in the pipeline to perform polarity classification. This section covers the pipeline performed for polarity classification in this project.

3.6.1. Labelling Polarity

As mentioned in Section 3.5.1., while we labelled the subjectivity of each sequence, we also labelled all *Opinionated* rows with polarity. For clarity, we first labelled subjectivity and polarity in `raw_subjectivity.csv` and then partitioned `raw_subjectivity` to only contain subjectivity labels, and `raw_polarity.csv` to contain polarity only. As such, please refer to `raw_polarity.csv` for the labelling of sequences on polarity.

Of the 4,097 rows labelled earlier in Section 3.5.1., we had a total of 3,266 *Opinionated* rows where both annotators agreed; and of the 3,266 labelled polarities, we had 3,175 rows where both annotators agreed. Annotator 1 labelled 1,500 rows as *Positive* and 1,766 rows as *Negative*. Annotator 2 labelled 1,486 rows as *Positive* and 1,777 rows as *Negative*.

Therefore, given $p_o = 0.972$,
 $p_e = p_{positive} + p_{negative} = (0.459 \times 0.455) + (0.541 \times 0.545) \approx 0.504$ and
Cohen's kappa formula,

$$\kappa \equiv \frac{p_o - p_e}{1 - p_e}$$

We get Cohen's Kappa to be **0.972**. The comma separated values file for polarity labelling can be found in `raw_polarity.csv`. The subsequently cleaned and balanced dataset can be found in `cleaned_subj_polar_.csv`.

3.6.2. Custom CNN Model Training and Performance

Similar to the method tried in Section 3.5.2, we trained our CNN ten times using each pre-processing covered in Section 3.4. We then average the values of all 10 model training to draw comparisons on the impact of each preprocessing technique.

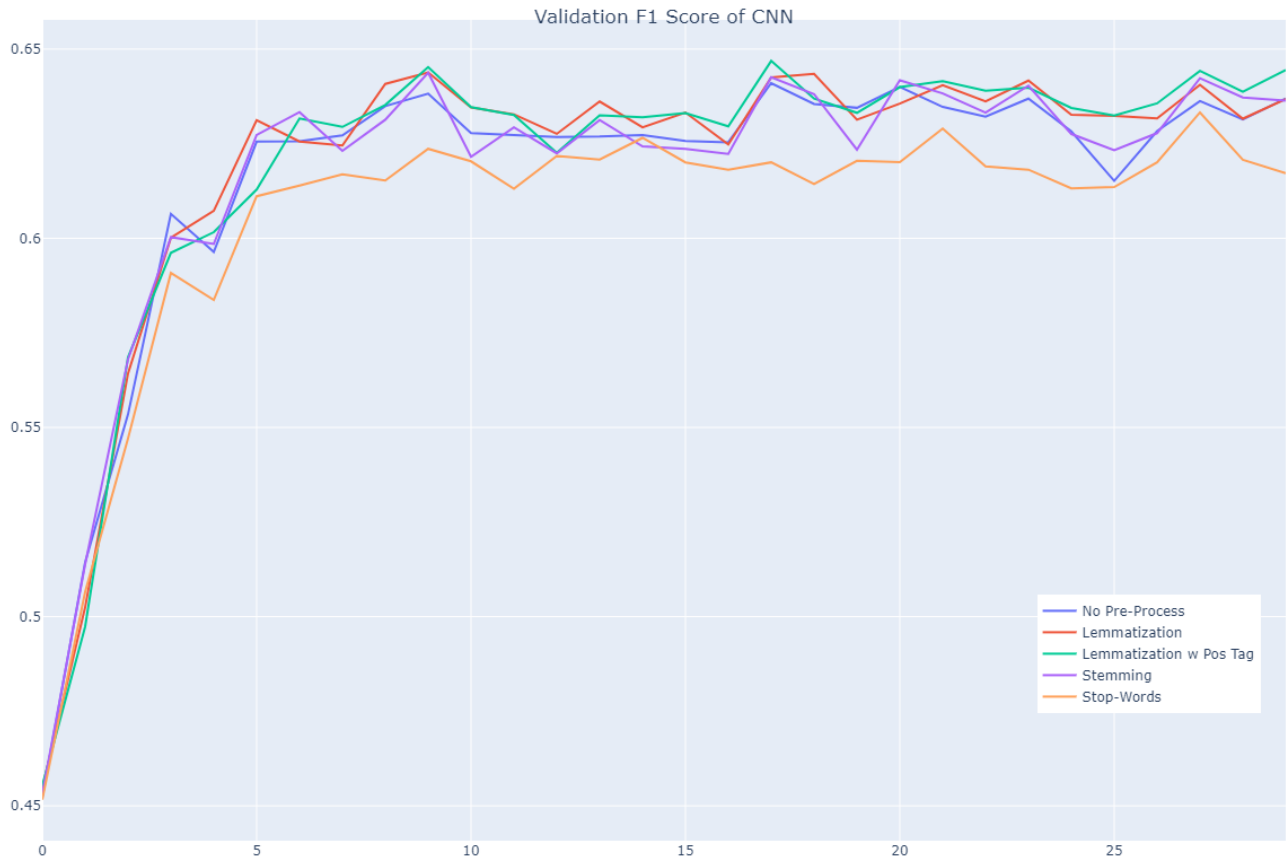


Fig-3.15 CNN Polarity Performance with Different Pre-Processing

As shown in Fig-3.15, we can see that lemmatization with POS tagging worked best, although the margin of improvements over the rest of the preprocessing is very small. Furthermore, it is worth noticing that stop-words performed very poorly on this CNN polarity classification task as compared to subjectivity classification.

This might indicate that there might be more to our hypothesis on how stop words could have resulted in the best performance in the CNN by a noticeable margin for subjectivity classification. A likely reason could be that stop words play a role in polarity classification. Words like “so” implies a more extreme extent of a sentiment. For example, the sentence “I am so done with him” should imply a negative sentiment, whereas “I am done with him” can be debated to be neutral or only borderline negative.

Ultimately, it is likely that with stop words removal, some additional pruning might be needed to determine the unnecessary stop words for the classification task. Due to time constraint and a lacklustre result of our CNN and RNN as compared to the state of the art

BERT (Section 3.6.5.) we did not further explore the impact of hand picking the stop words to be removed.

Also, as lemmatization with POS tagging as well as stemming showed similar improvements in performance over no preprocessing, we tried combining both preprocessing techniques. Stopwords were first removed, then lemmatization with POS tagging was performed.

Fig-3.16 shows the results. It is clear that lemmatization with POS tagging is still superior, although combining both lemmatization with POS tagging and stemming yields a marginally better result than having no preprocessing or just stemming alone. As such, we conclude that the best preprocessing for CNNs in polarity classification would be lemmatization with POS tagging.

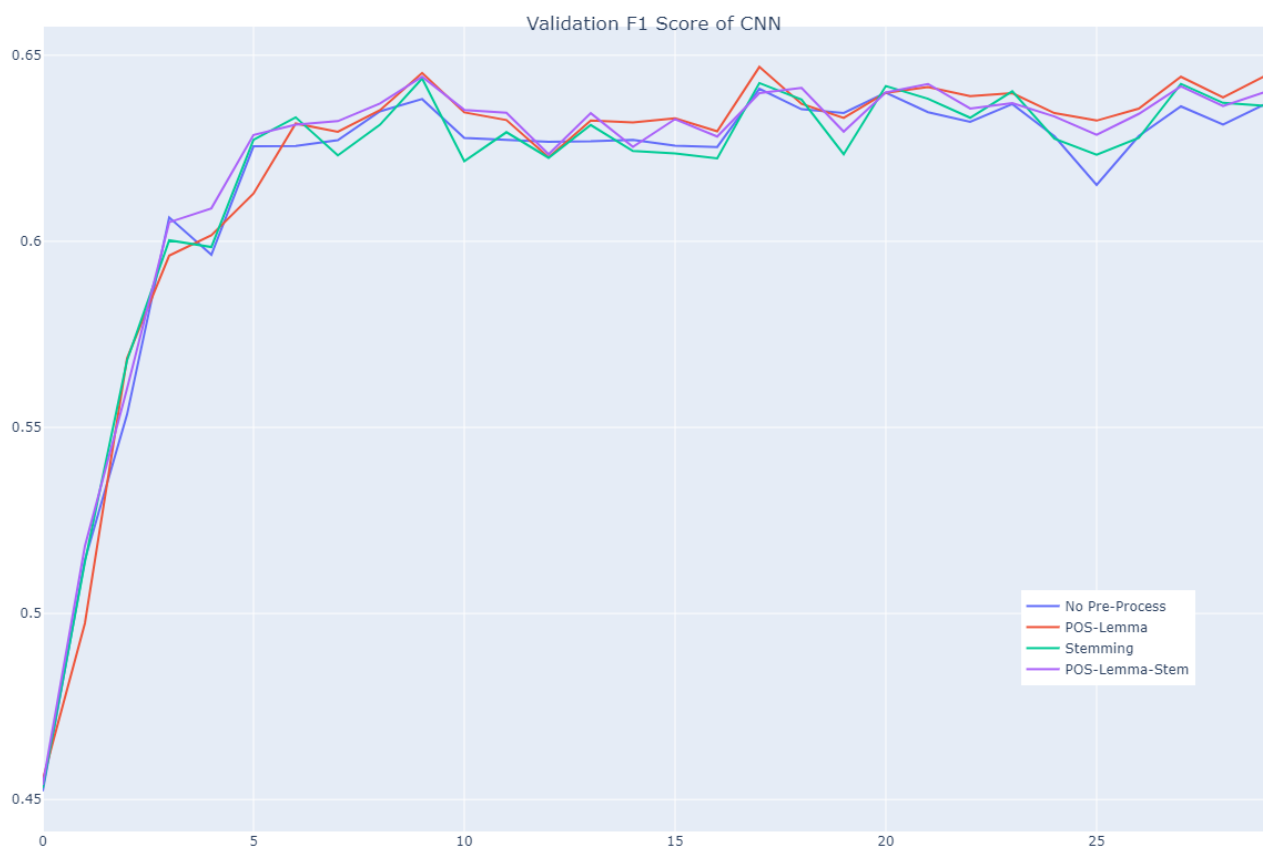


Fig-3.16 CNN Polarity Classification Performance Using Hybrid Pre-Processing

3.6.3. Custom RNN Model Training and Performance

We also trained an RNN to perform polarity classification. Similarly, the RNN was trained ten times for each preprocessing technique. We then averaged out the values to verify the average impact on performance of each preprocessing technique.

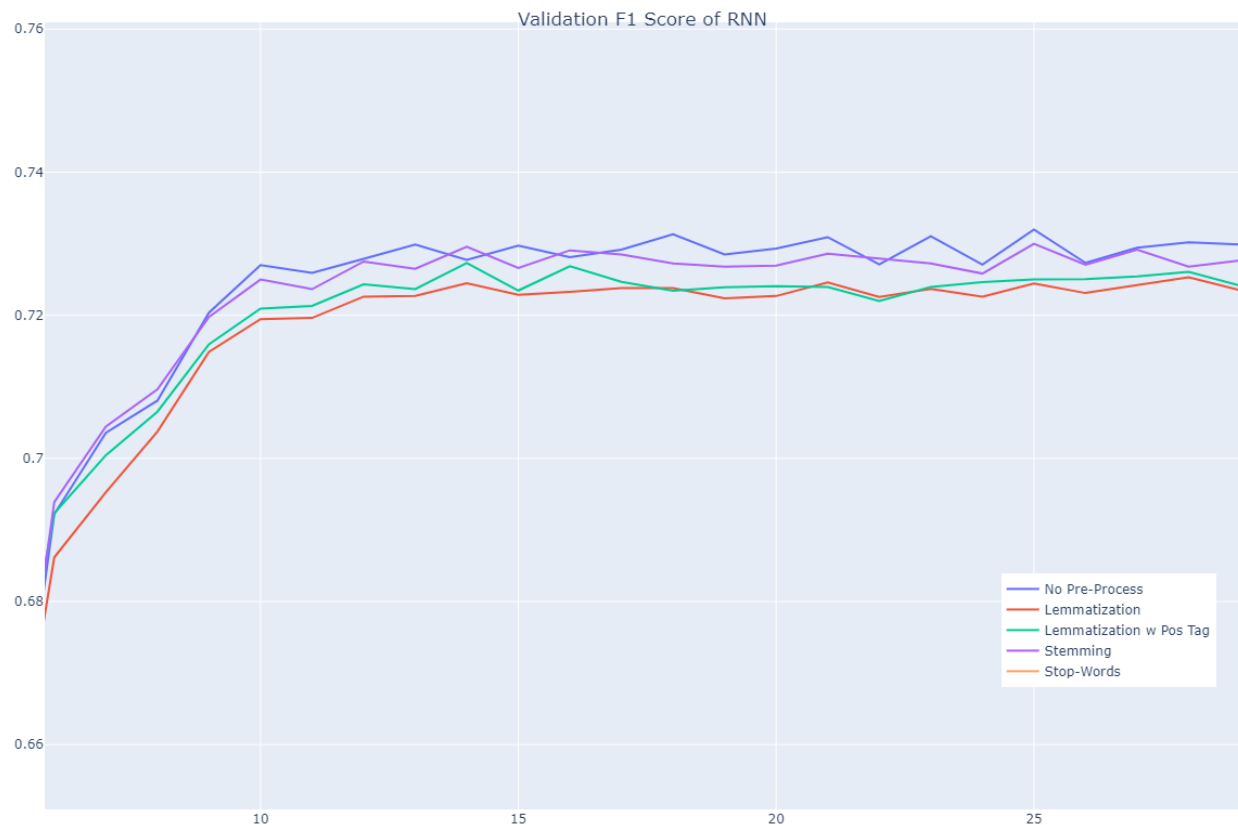


Fig-3.17 Zoomed in RNN Polarity Classification Performance for Different Pre-Processing

Much like the behaviour of our RNN in subjectivity classification, having no preprocessing performed seemed to yield good results. Also, the results were generally all in the same range for every preprocessing technique. In our case, no preprocessing was deemed the best action for preparing data for polarity classification.

As such, It is likely that our earlier hypothesis that RNNs are already highly capable of utilizing all the words in a sequence to perform classification stands true, and any form of preprocessing could instead degrade the performance as features that would have been present for the RNN to learn from would be removed.

3.6.4. BERT Fine-Tuning and Performance

The fine-tuned BERT model performed beyond our expectations and converged at a validation F1 Score of 0.944. Fig-3.18 shows the validation F1 score over 10 epochs of training.

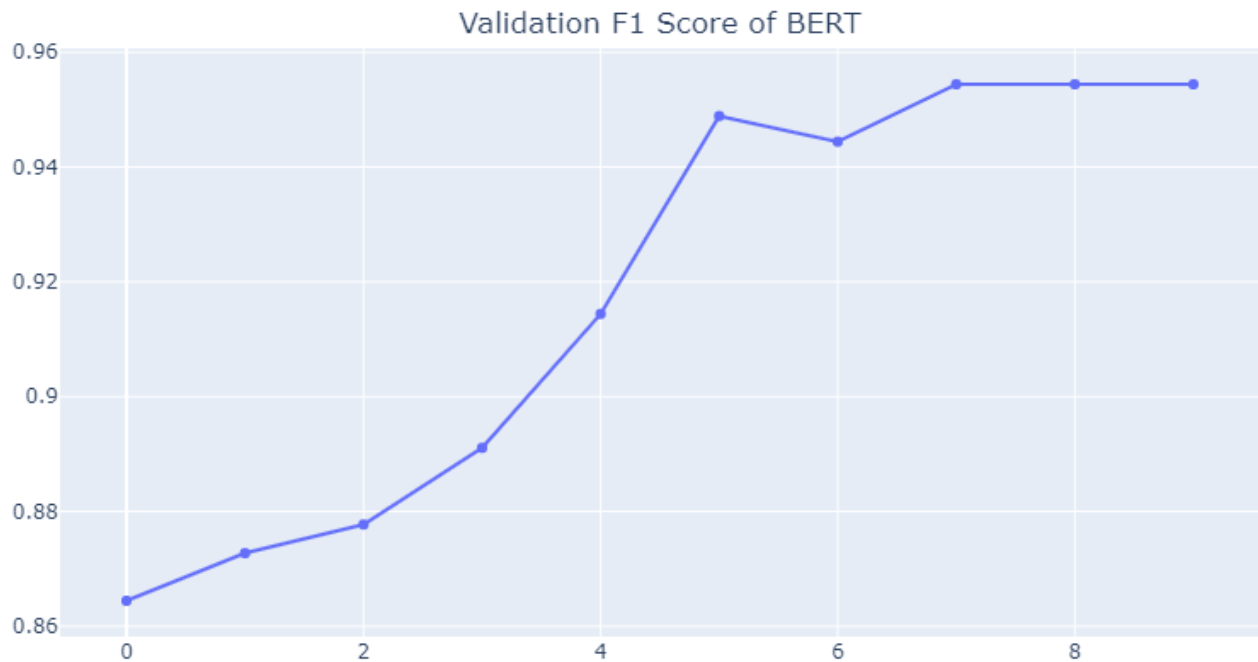


Fig-3.18 BERT Fine-Tuned Polarity Classification Performance

3.6.5. CNN, RNN and BERT Evaluation Metrics Comparison

With the most effective preprocessing techniques identified for both our CNN and RNN, we retrain our CNN and RNN using these preprocessing techniques and compare the evaluation metrics of our CNN, RNN and BERT model on the testing dataset.

The figures below show the predictions in a confusion matrix for each model.

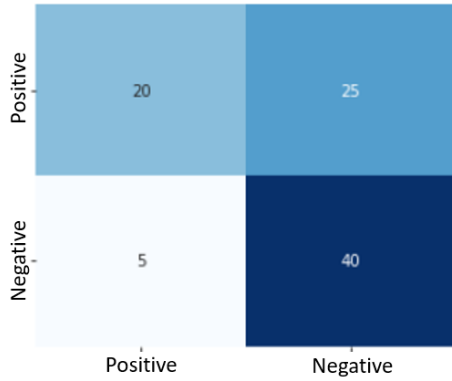


Fig-3.19 CNN Confusion Matrix

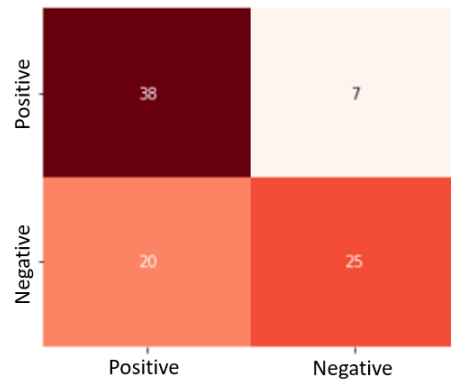


Fig-3.20 RNN Confusion Matrix

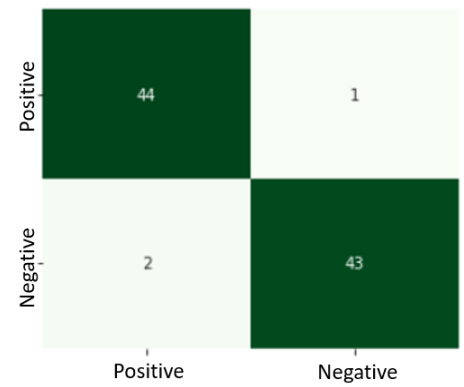


Fig-3.21 BERT Confusion Matrix

From the confusion matrices above, we can then compute the F1 Score and Accuracy. The results are shown in Fig-3.22 and Fig-3.23.

F1 Scores of CNN, RNN, BERT



Fig-3.22 CNN, RNN, BERT F1 Scores

Accuracy of CNN, RNN, BERT

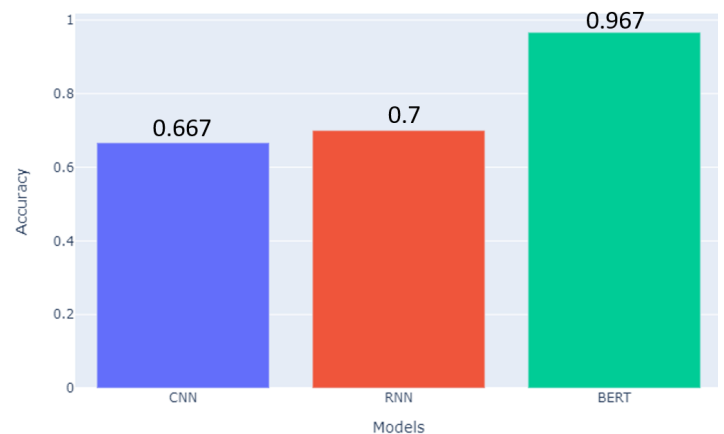


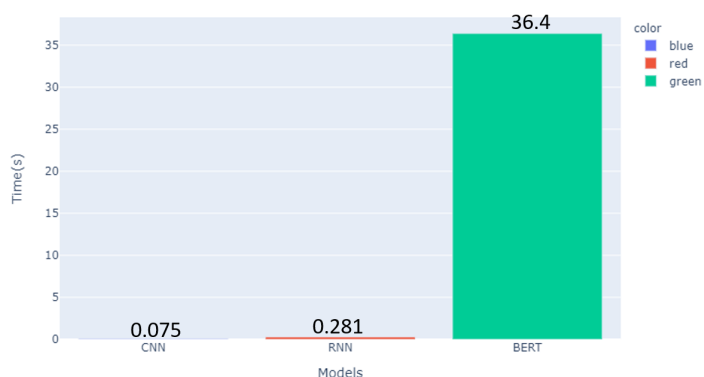
Fig-3.23 CNN, RNN, BERT Accuracy

3.6.6. CNN, RNN and BERT Performance Metrics Comparison

In addition to the evaluation metrics, we also compare the performances of our model. Namely, the time to predict 1,000 sequences averaged over ten iterations, and the average time to train each model per epoch.

It is worth noting that since our BERT model uses the same huggingface transformer for fine-tuning, the average prediction time is almost the same for subjectivity classification and polarity classification.

Training Time per Epoch of CNN, RNN, BERT



Average Prediction Time of CNN, RNN, BERT for 1,000 Queries

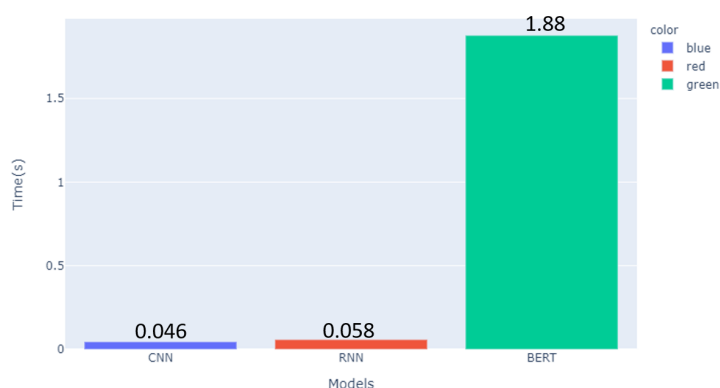


Fig-3.24 Average Training Time per Epoch of CNN, RNN, BERT

Fig-3.25 Average Prediction Time of CNN, RNN, BERT

The summary of, the evaluation and performance metrics for all three models are detailed in Table 3.4.

Table 3.4 Summary of Evaluation and Performance Metrics

Metrics		CNN	RNN	BERT
Evaluation Metrics	Accuracy	0.667	0.7	0.967
	F1 Score	0.727	0.649	0.967
Performance Metrics	Average Training Time per Epoch (s)	0.075	0.281	36.4
	Average Prediction Time (1,000 Sequences) (s)	0.046	0.058	1.88

Having accumulated sufficient empirical data on the evaluation and performance metrics of all three models for both subjectivity and polarity classification, we decided to carry on using only BERT for the rest of the project due to its vastly superior results.

This decision however, comes with its own trade off. Firstly, BERT models take a significantly longer time to train, accelerated only by hardware. As a group of students with very limited hardware, the cost of taking longer to train could result in tighter timelines for the project - thereby creating the possibility for other parts of the project (i.e. report, video) to be jeopardised.

Secondly, it is imperative that we have a quick information retrieval system. A system that takes several seconds to load up a query result will be a dealbreaker for most users. BERT treads a fine line as it takes 1.88seconds per 1,000 queries and it could be considered a little bit slow.

To counteract this issue, we decided to include a caching mechanism in our application. Caching helps with reducing the time to load as results previously searched are stored for use in future. Furthermore, it was decided that the information retrieval system was fairly good at giving results that are most relevant to a query. This means that it would be unlikely for 1,000 results to be required, and instead only up to the top 50 results may be needed for a user to likely be satisfied.

3.7. Negative, Neutral, Positive Classification

As a first step to multitask classification, we wanted to investigate the impact of training a BERT model on performing multitask classification with three classes - *Negative*, *Neutral* and *Positive*.

This would be a slightly more challenging task as it would attempt to accomplish both subjectivity and polarity classification in a single model. Should our model prove to perform reasonably well (i.e. F1 Score > 0.85), we can then consider other tasks like fine-grained classification and other kinds of multitask classifications.

3.7.1. Labelling Negative, Neutral, Positive

As we've a total of 600 labelled neutral sequences only, we decided to randomly sample 1,200 sequences from the much larger polarity labelled dataset with 600 positive and 600 negative rows. As seen in Fig-3.26, our BERT model performed exceedingly well in this respect too. This opened up doors for further innovation on useful classification tasks that can deliver greater insights for users of our information retrieval system.

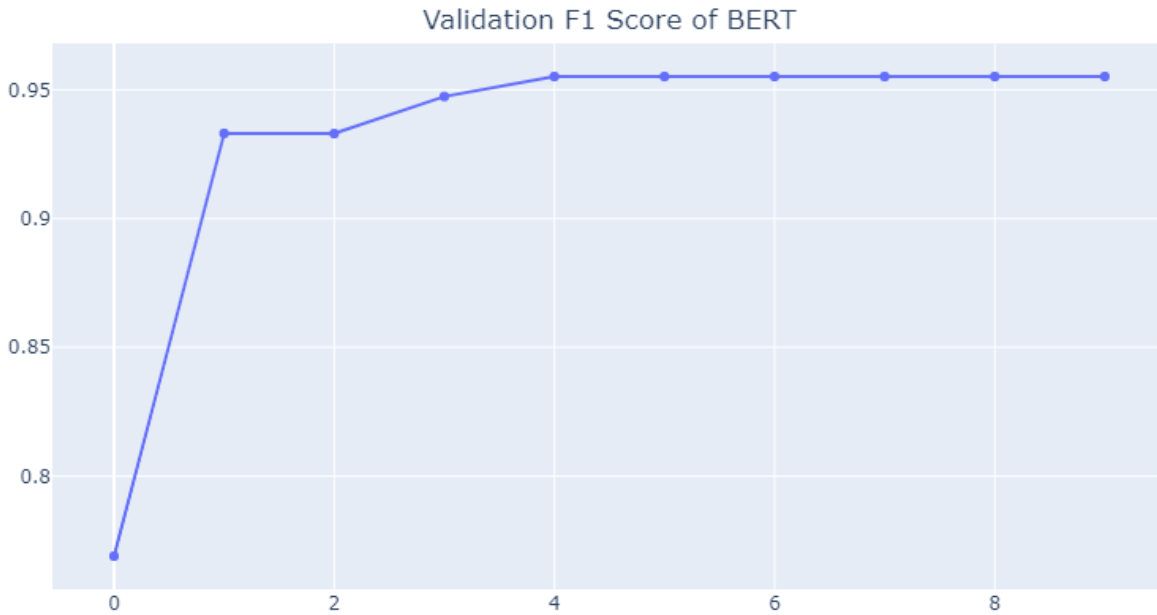


Fig-3.26 BERT Fine-Tuned 3-Class Classification Training F1 Score

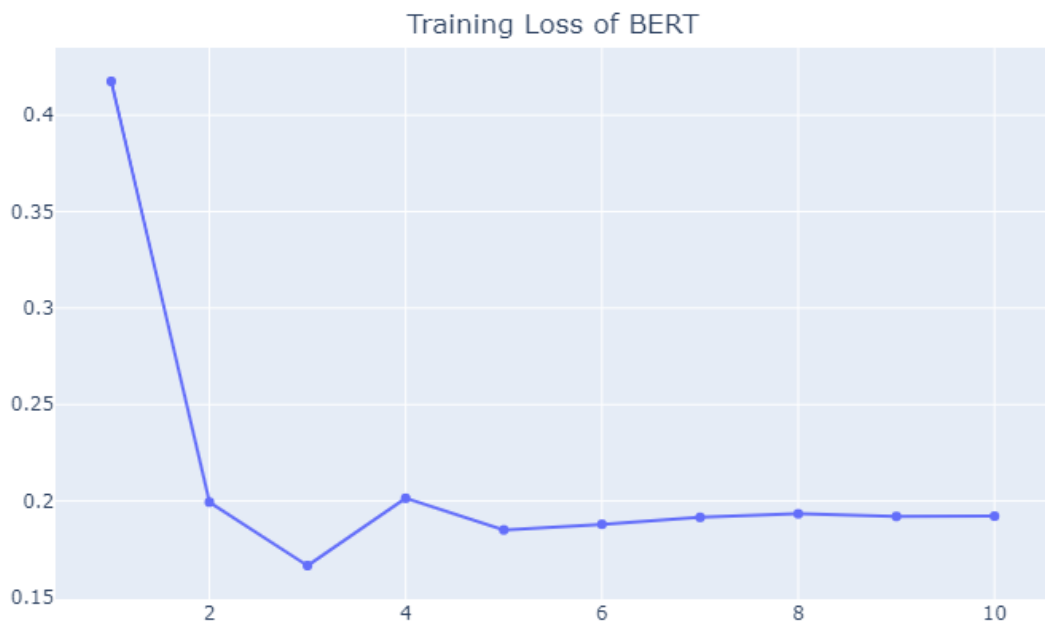


Fig-3.27 BERT Fine-Tuned 3-Class Classification Training Loss

From epoch 4 onwards, the model converged at its highest F1 Score of 0.955. We decided to take the model's checkpoint at epoch 5 as it maintained the highest F1 Score while having the lowest loss.

3.7.2. Evaluation Metrics

We used the weights of the model checkpoint from epoch 5 to then predict the labels of the test set. Fig-3.28 shows the confusion matrix.

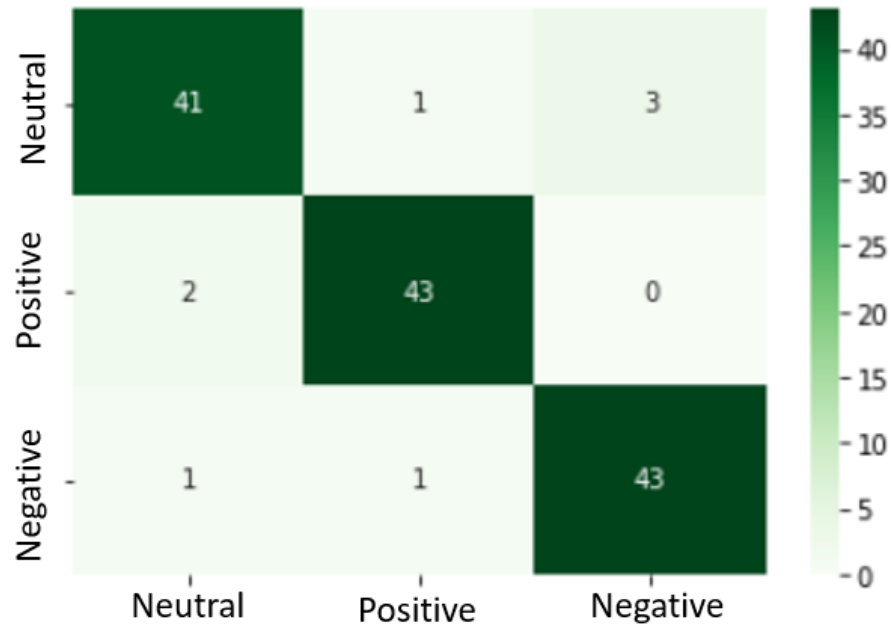


Fig-3.28 BERT Confusion Matrix on Subjectivity-Polarity Classification

From the confusion matrix, we can compute the following:

- **F1 Score (macro averaged): 0.985**
- **Accuracy: 0.941**

We use the macro averaged F1 Score as we had used random sampling with stratification for our test data.

3.7.3. Performance Metrics

Much like previous sections, we found the average time to predict 1,000 sequences and the average time to train per epoch after running ten iterations of each task. The results are shown in Table 3.5 below.

Table 3.5 Summary of BERT Evaluation and Performance Metrics

Metrics		BERT
Evaluation Metrics	Accuracy	0.941
	F1 Score	0.985

Performance Metrics	Average Training Time per Epoch (s)	67.3
	Average Prediction Time (1,000 Sequences) (s)	1.93

As seen in Table 3.3, Table 3.4 and Table 3.5, the performance metrics we are most interested in - average time to predict 1,000 sequences - is relatively similar. This is because our BERT models are all fine-tuned using the same model, huggingface's bert-base-uncased.

As such, going forward we will work under the assumption that the time taken to classify a sequence will not vary wildly regardless of the classification task. Section 3.8 and 3.9 performance metrics can be considered to be similar to Section 3.7.3.

3.8. Fine-Grained Sentiment Classification

Following the success of the multitask classification for subjectivity and polarity, we decided to attempt fine-grain classification. For our case, we perform fine-grained classification by decomposing negative and positive sentiments into *Somewhat Negative*, *Very Negative*, *Somewhat Positive* and *Very Positive* as well as adding some *Neutral* dataset. In total, we aim to have a balanced dataset of 1,000 samples with 200 rows for each class.

3.8.1. Labelling Fine-Grained Sentiment

For the labelling of fine-grained sentiments, we used the previously labelled dataset from *raw_polarity.csv*. As each row was already labelled *Positive* or *Negative*, we then added a new column for classifying the fine-grained sentiments.

We also added 200 randomly sampled rows of *Neutral* data from our earlier subjectivity classification tasks file *cleaned_subj_polar.csv*.

In total, we labelled 814 rows for fine-grained sentiment. We had a total of 805 rows where both annotators agreed. Table 3.5 shows the labelling done by both annotators.

Table 3.5: Labelling done by Annotator 2 and Annotator 1 for Fine-Grained Sentiment

Fine-Grain Sentiment	Annotator 1	Annotator 2
Very Negative	200	203
Somewhat Negative	205	200
Somewhat Positive	209	206
Very Negative	200	205

Given $p_o = 0.989$,

$p_e = p_{\text{somewhat-positive}} + p_{\text{somewhat-negative}} + p_{\text{very-positive}} + p_{\text{very-negative}} + p_{\text{neutral}}$
 $= (0.200 \times 0.246)$
 $+ (0.256 \times 0.200) + (0.261 \times 0.258) + (0.200 \times 0.256) \approx 0.219$, and Cohen's
 kappa formula,

$$\kappa \equiv \frac{p_o - p_e}{1 - p_e}$$

We get Cohen's Kappa to be **0.986**. The comma separated values file for emotion labelling can be found in *raw_fine_grained.csv*. The subsequently cleaned and balanced dataset including the 200 sampled *Neutral* data (total 1,002 labelled rows) can be found in *cleaned_fine_grained_.csv*.

3.8.2. Evaluation Metrics

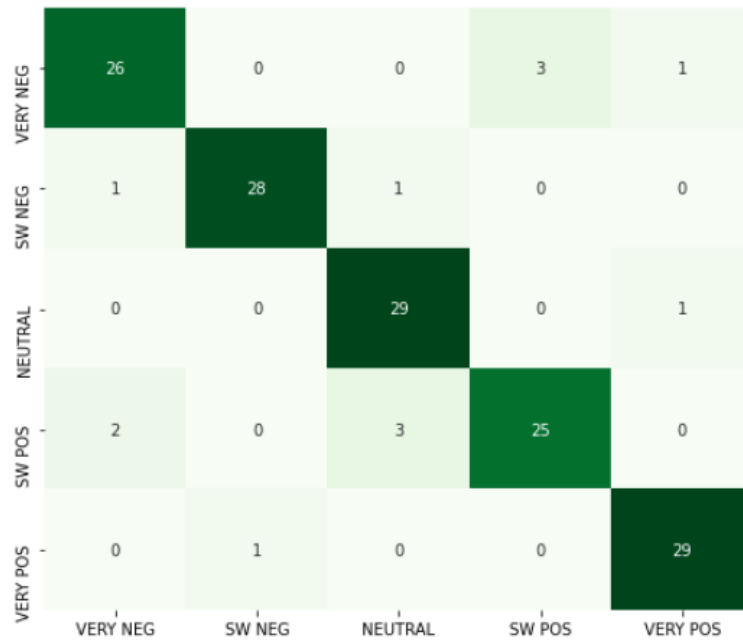


Fig-3.29 Confusion Matrix of BERT Fine-Grained Sentiment Predictions

From the confusion matrix, we can derive the following:

- **F1 Score (macro averaged): 0.937**
- **Accuracy: 0.913**

3.9. Emotion Classification

Yet another multi task classification we felt would be useful for our users was to include emotions of the query results. We decided to use the 6 basic emotions [9] in our labelling scheme:

1. Angry
2. Disgust
3. Fear
4. Happy
5. Sad
6. Surprise

We began labelling using a mix of neutral and opinionated rows that were labelled for subjectivity detection earlier. We included neutral rows to label as we intend for our information retrieval system to output the emotion regardless of subjectivity. An advertising statement like “Subscribe to our weekly newsletter for more news on the trainwreck that is called America” could be considered a neutral statement as it's a general comment asking for people to subscribe to their newsletter. The same statement can also

be classified as *Disgust* as it definitely exudes a subtle hint that the advertisement feels disgust at the state of America.

3.9.1. Labelling Emotion

With 6 classes we planned to label a total of 167 rows for each emotion, thereby having 1,002 rows for training. As there weren't too many rows we could classify as *Sad*, we ended up labelling 1,299 rows in total.

Of the 1,299 rows labelled, we had 1,177 labelled agreements. Table 3.6 shows the labelling results of both annotators.

Table 3.6: Labelling done by Annotator 2 and Annotator 1 for Emotions

Emotions	Annotator 1	Annotator 2
Angry	217	216
Disgust	230	232
Fear	232	229
Happy	194	196
Sad	183	184
Surprise	243	242

Given $p_o = 0.906$,

$p_e = p_{angry} + p_{disgust} + p_{fear} + p_{happy} + p_{sad} + p_{surprise} = (0.167 \times 0.166)$
 $+ (0.177 \times 0.179) + (0.179 \times 0.176) + (0.149 \times 0.151) + (0.141 \times 0.142) + (0.187 \times 0.186)$
 Cohen's kappa formula,

$$\kappa \equiv \frac{p_o - p_e}{1 - p_e}$$

We get Cohen's Kappa to be **0.887**. The comma separated values file for emotion labelling can be found in *raw_emotion.csv*. The subsequently cleaned and balanced dataset containing 1,002 labelled rows can be found in *cleaned_emotion_.csv*.

3.9.2. Evaluation Metrics

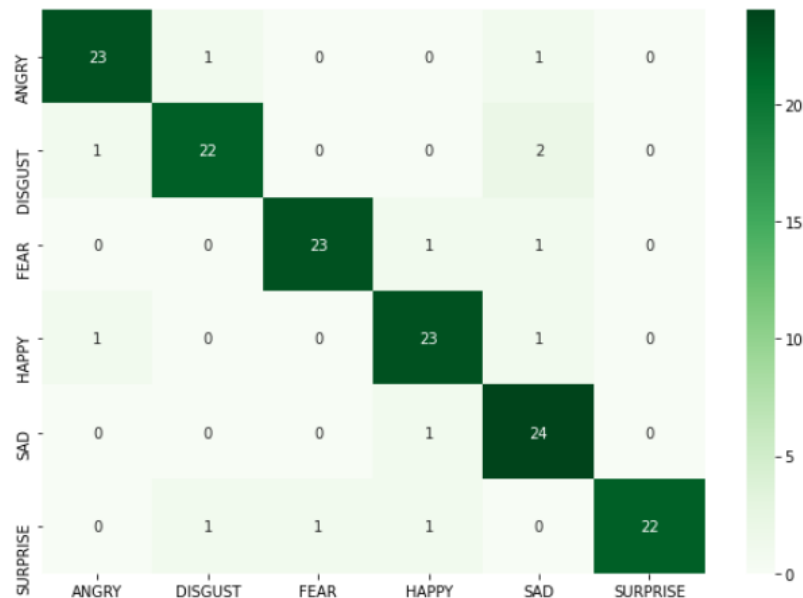


Fig-3.30 Confusion Matrix of BERT Emotion Predictions

From the confusion matrix, we can compute the following:

- **F1 Score (macro averaged): 0.914**
- **Accuracy: 0.913**

3.10. Apply Model on Dataset

Having trained several models, we then apply these models on the dataset. Due to time constraint, we only applied two models - emotion classification model and polarity-subjectivity multitask classification model.

As the dataset contains over a million rows, we ran the models over a subset of 75,000 rows. The findings are presented below.

Classified Subjectivity-Polarity

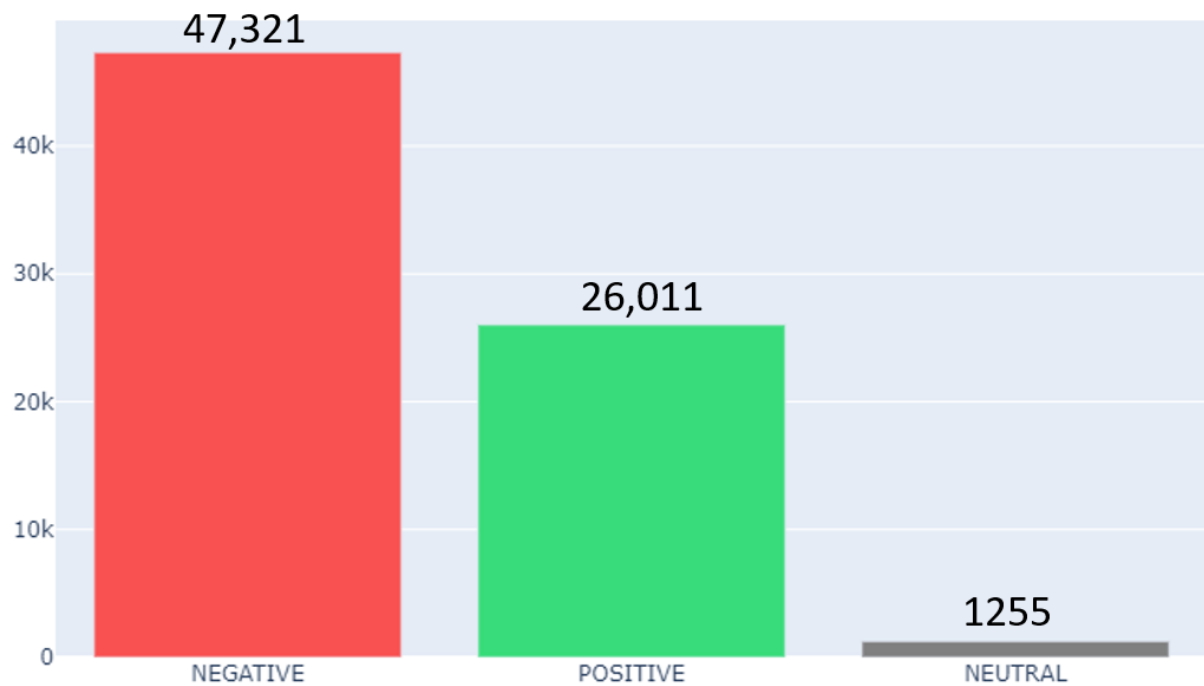


Fig-3.31 Classified Subjectivity and Polarity of Sample Data

From Fig-3.31, we can infer that the majority of posts are negative. This is in line with our own exploratory data analysis. Suffice to say it was likely because the dataset we have consists of many Parler posts just prior to the Capitol riots.

Furthermore, there are very few neutral posts. Most of these classified neutral posts are posts containing advertisements or news article headers.

Classified Emotions

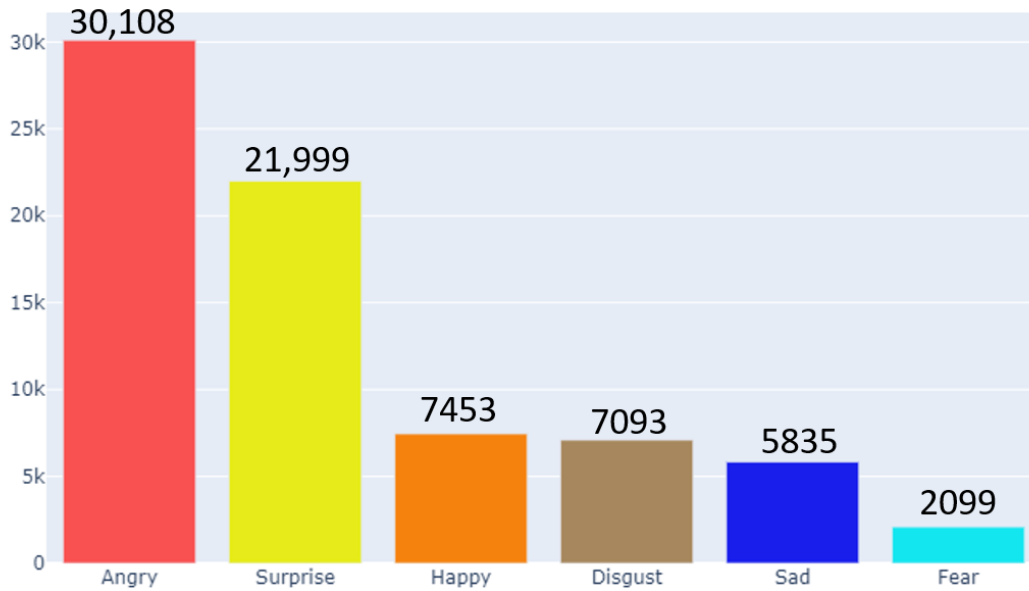


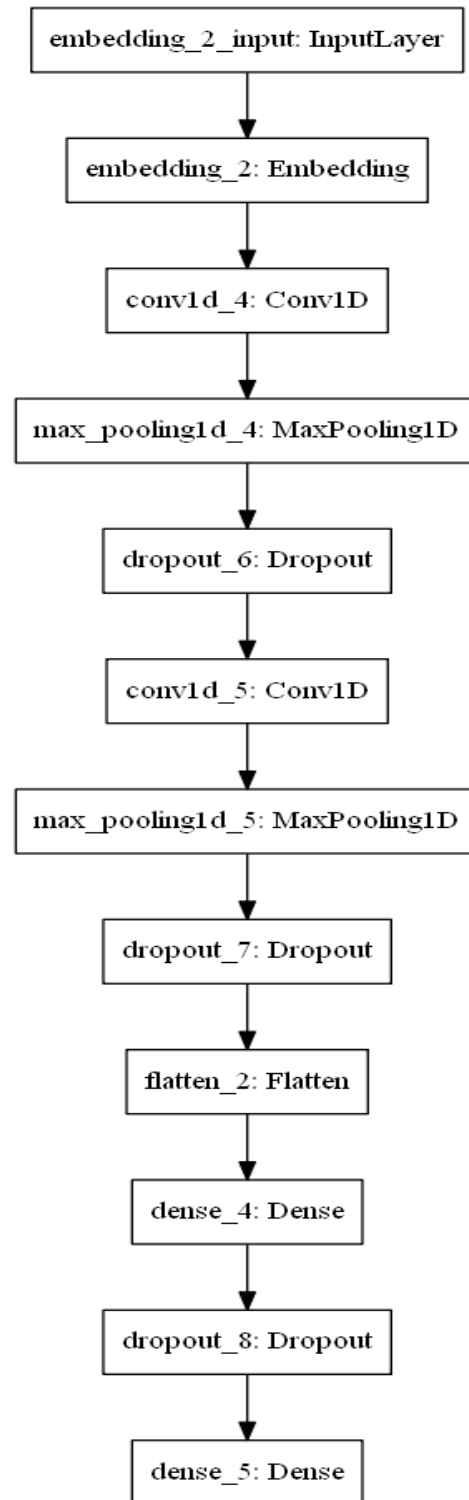
Fig-3.32 Classified Emotions of Sample Data

Similar to the classified polarity and subjectivity in Fig-3.31, we can see that most of the posts are classified as *Angry*. During our labelling we encountered the same heavily biased dataset. That is to say, while the other classes were generally encountered, the sad and fearful posts were harder to find.

Also, the reason there are so many *Surprise* classified posts is because oftentimes these angry or rebellious posts tend to be phrased in a manner that the annotators felt could be classified as *Surprise*. As such, it is likely that the model learnt the biases of our annotators and was able to successfully classify these posts as such.

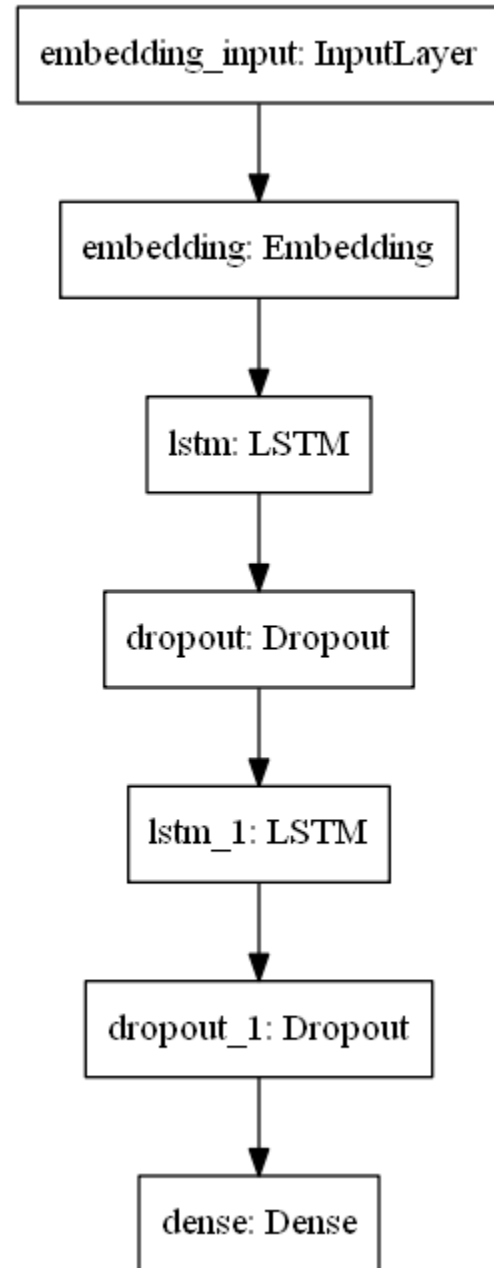
Appendix A: CNN Architecture

```
keras.layers.Embedding(em_dims, 30,  
input_length=max_len),  
  
keras.layers.Conv1D(40, 3,  
activation='relu'),  
keras.layers.MaxPooling1D(padding='same'  
),  
keras.layers.Dropout(0.3),  
  
keras.layers.Conv1D(5, 3, activation='relu'),  
keras.layers.MaxPooling1D(padding='same'  
),  
keras.layers.Dropout(0.3),  
  
keras.layers.Flatten(),  
keras.layers.Dense(20, activation='relu'),  
keras.layers.Dropout(0.5),  
keras.layers.Dense(1, activation='sigmoid')
```



Appendix B: RNN Architecture

```
keras.layers.Embedding(len(word_index)+1,  
em_dims, input_length=MAX_LEN),  
keras.layers.LSTM(100,  
return_sequences=True),  
keras.layers.Dropout(0.5),  
keras.layers.LSTM(100),  
keras.layers.Dropout(0.5),  
keras.layers.Dense(1, activation='sigmoid')
```



Appendix C: Convolutions and Pooling

Convolutions

Convolutions are sliding window functions which are applied to a matrix to achieve a specific result (e. g., edge detection, image blur). The sliding window is commonly referred to as a filter, kernel, or feature detector. A convolution is a linear operation involving the multiplication of a set of weights with the input, much like a traditional neural network. (This multiplication is performed between the array of input data and the kernel/filter). Furthermore, the filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, and is then summed, thereby yielding a single value. It is intentional to use a filter smaller than the input data such that the same filter (set of weights) can be multiplied with the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each filter-sized patch of the input data (overlapping part), left to right, top to bottom (as shown in Fig-3.11). The output from multiplying the filter with the input array one time is a single value. As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. The two-dimensional output array resulting from this operation is referred to as a “feature map”. After a feature map is generated, we can pass each value in the feature map through a non-linearity activation function. In other words, CNNs are basically several layers of convolutions with activation functions like leaky ReLU or Tanh that make it possible to capture non-linear relationships.

By applying this set of dot products, one can extract desired information from images, starting from edges on shallower levels to identifying the entire objects on deeper levels of neural networks.

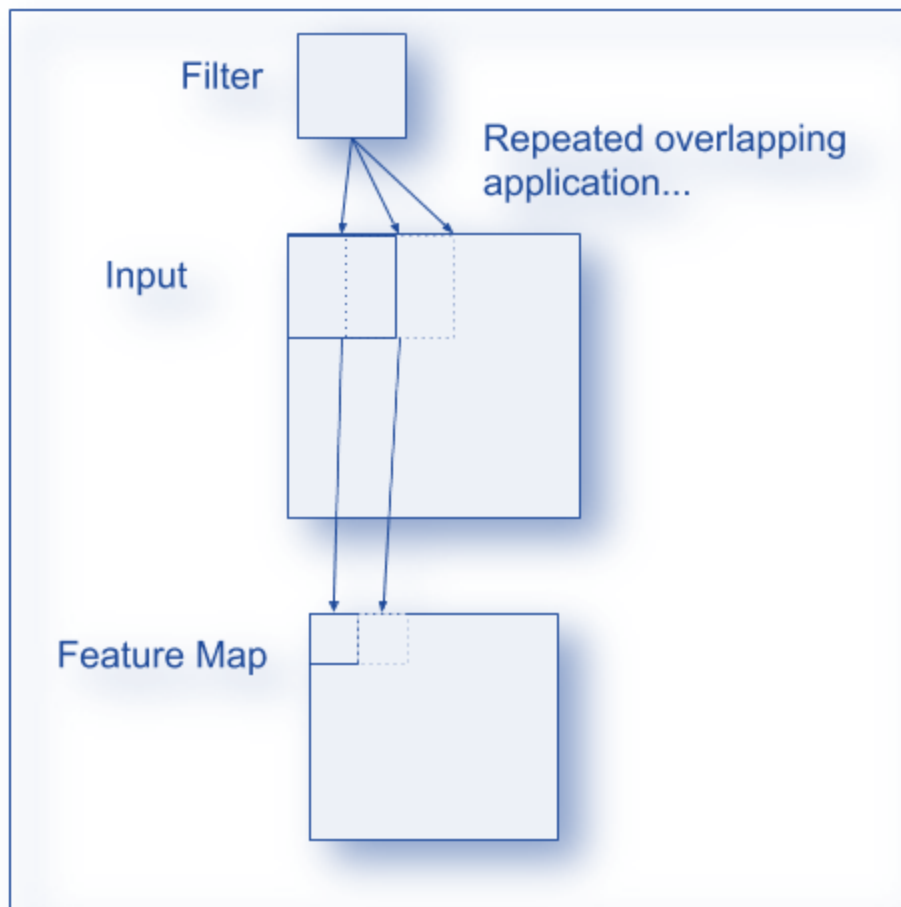


Fig-C.1: Example of a Filter Applied to a Two-Dimensional Input to Create a Feature Map
Source: Adapted from [2]

Pooling

A pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. The objective of this is to reduce the spatial size of the representation, thereby increasing performance by reducing computation. The pooling operation is processed on every slice of the representation individually.

There are several pooling functions such as the L2 norm of the rectangular neighborhood, a weighted average based on the distance from the central pixel, and the average of the rectangular neighborhood. The most popular one however is max pooling, which selects the maximum output from the neighborhood. Fig-3.12 shows an example of applying max-pooling with a 2x2 filter, given a stride of 2.

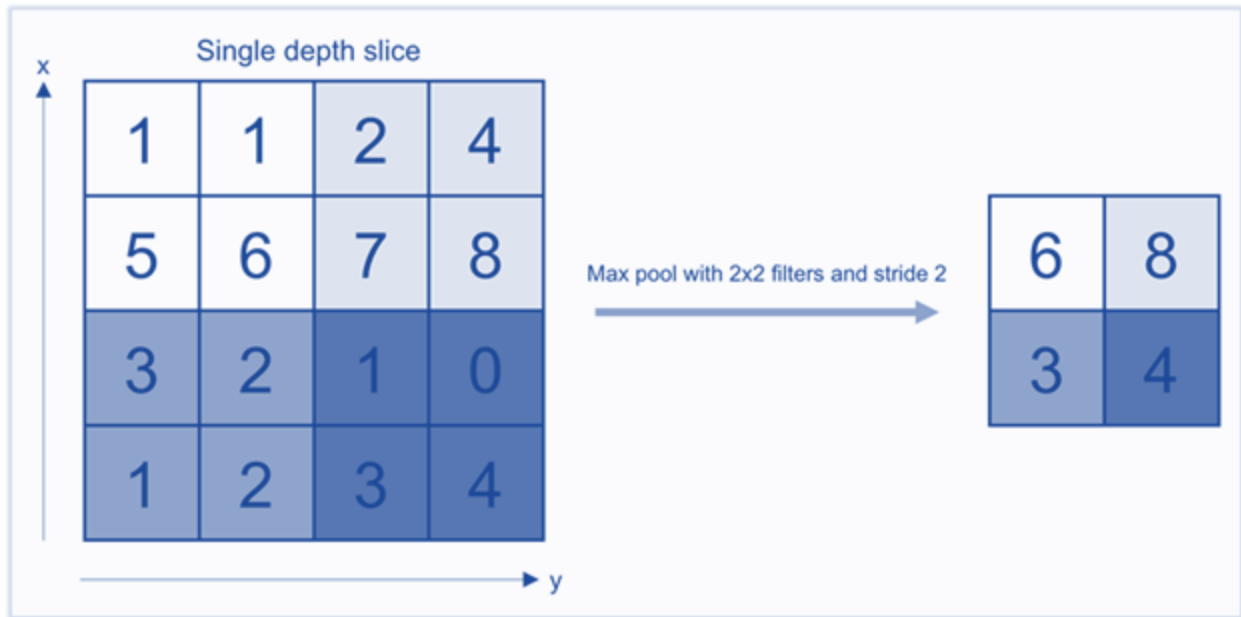


Fig-C.2: Applying Max Pooling
Source: Adapted from [3]

Pooling also provides a degree of translation invariance to some extent (i.e. An object can be recognized regardless of where it appears in the frame).

Appendix D: LSTM Network

Long short-term memory networks are the most popular extension for recurrent neural networks, which basically extends the memory. Therefore it is ideal to learn from important experiences that have extremely long time lags in between. LSTMs thus enable RNNs to remember inputs over a long period of time.

LSTMs contain information in a memory, much like the memory of a computer. The LSTM can read, write and delete information from its memory.

All recurrent neural networks are a sequence of repeating modules of neural networks. In a regular RNN, this repeating module has an extremely simple structure, such as a single tanh layer, as illustrated in Fig-3.5.

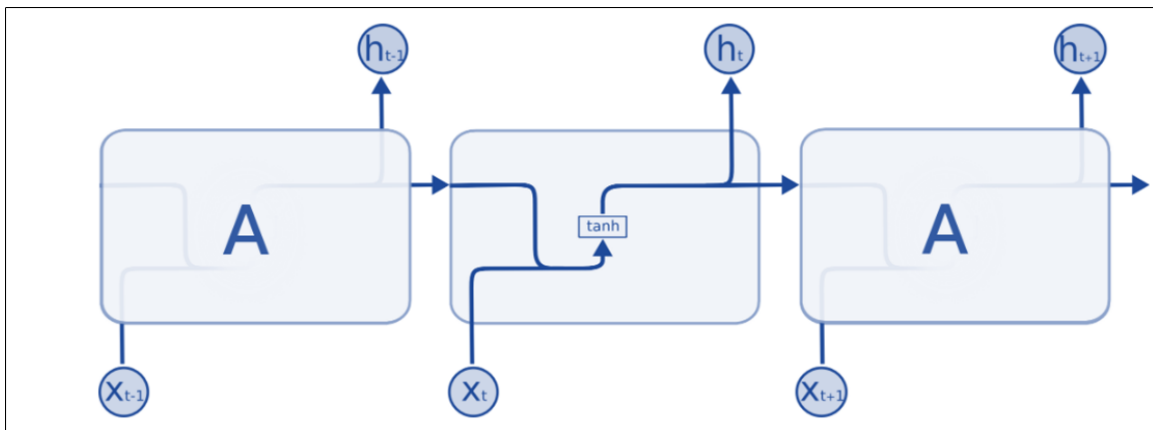


Fig-D.1: Standard RNN with Tanh layer
Source: Adapted from [6]

LSTMs are also composed of this chain-like structure, but the repeating module has a separate architecture. Rather than containing a single neural network layer, there are four, interacting in a very special way as illustrated in Fig-3.6.

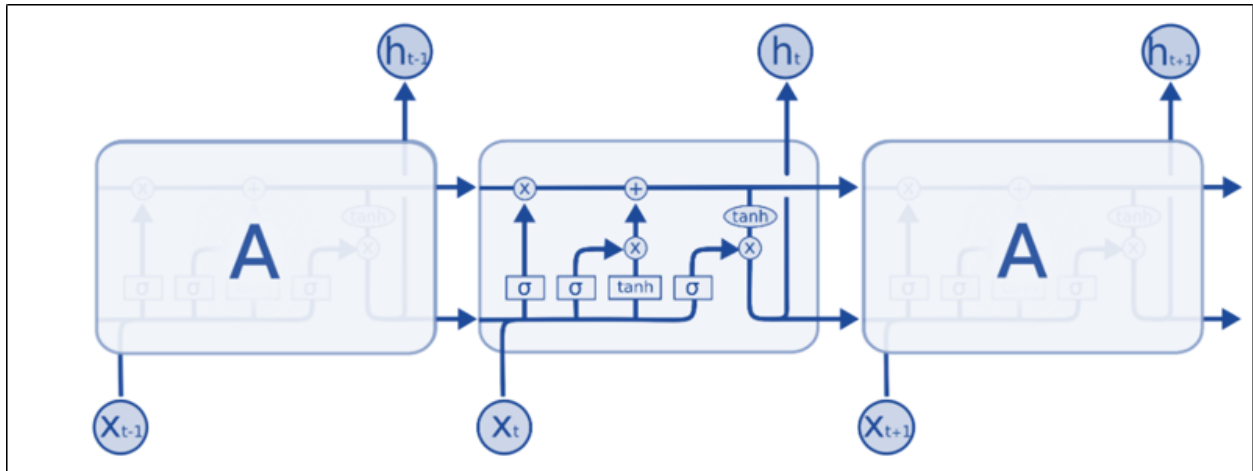


Fig-D.2: LSTM cell containing four interacting layers
Source: Adapted from [6]

Furthermore, each LSTM cell consists of three gates, namely - the forget gate, input gate, and output gate as illustrated in Fig-3.7.

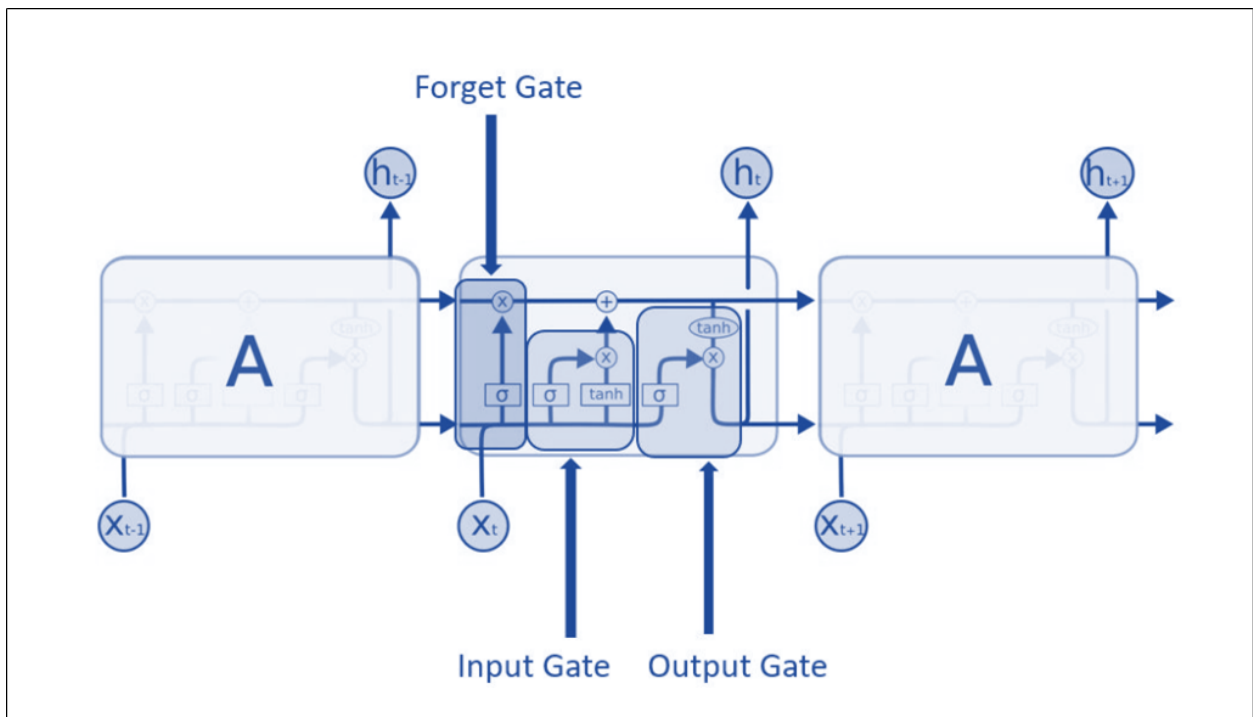


Fig-D.3: LSTM cell with its Gates
Source: Adapted from [6]

a) Forget Gate

In layman terms, this Gate is responsible for deciding on how much of the past should be remembered. This is done by determining which information is redundant and should be removed (forgotten) from the cell in the particular time stamp. This is done using the sigmoid function. It looks at the previous state (h_{t-1}) and the content input (x_t), and outputs a number between 0 and 1 for each number in the cell state C_{t-1} . A '1' represents "completely keep this" while a 0 represents "completely omit this."

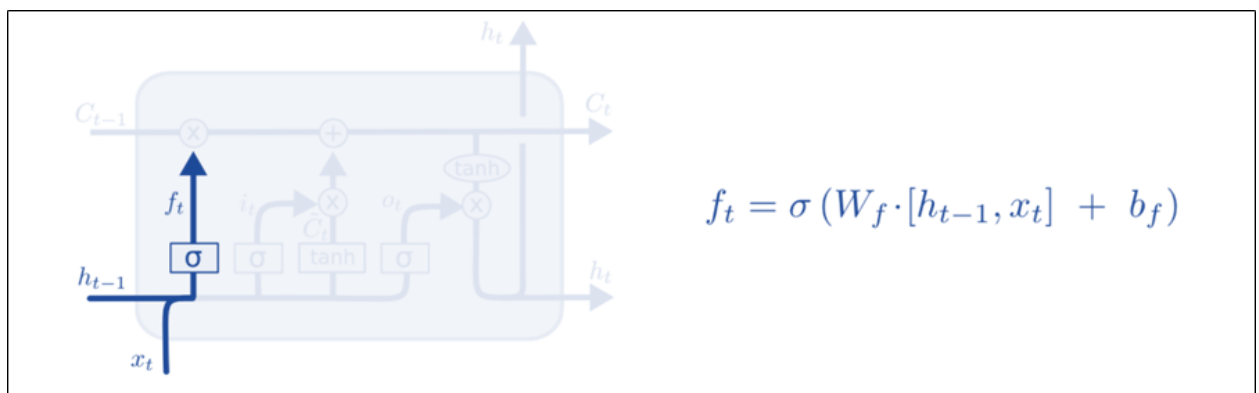


Fig-D.4: Forget Gate
Source: Adapted from [6]

Fig-3.13 illustrates the architecture of the forget gate. The gate multiplies the inputs (h_{t-1} and x_t) with the weight matrices, and a bias is then added. Following this, the sigmoid function is applied to this value. The vector output from the sigmoid function is then multiplied to the cell state.

b) Input Gate

The input gate is responsible for the addition of information to the cell state. This consists of two parts. First, a sigmoid layer also known as the "input gate layer" decides which values we'll update. Then, a tanh layer creates a vector of new candidate values, \tilde{C}_t , that could be added to the state. Fig-3.14 illustrates this process.

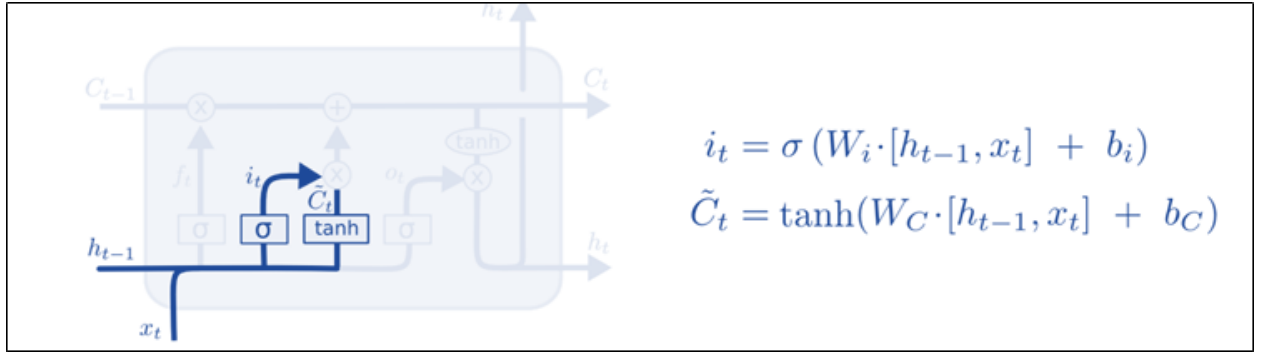


Fig-D.5: Input Gate(a)
Source: Adapted from [6]

The two steps performed are now combined to create an update to the state. The old state is multiplied by f_t , forgetting the things that were decided to forget earlier. $i_t * \tilde{C}_t$ is then added. This is the new candidate values, scaled by how much each state value was decided to be updated. Fig-3.15 illustrates this process.

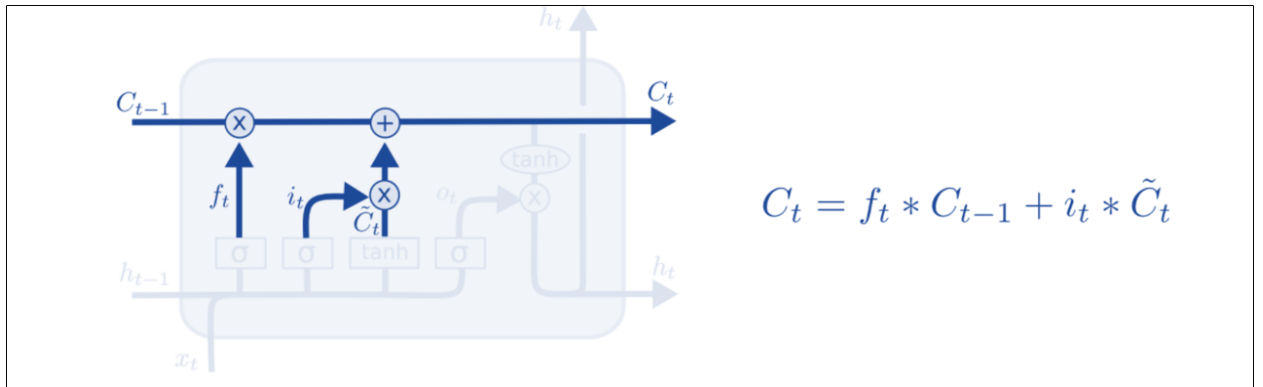


Fig-D.6: Input Gate(b)
Source: Adapted from [6]

c) Output Gate

The Output Gate selects useful information from the current cell state and displays it out as an output. First, a sigmoid layer is used to decide what parts of the cell state are going to be outputted. Then, the cell state is passed through tanh (to push the values to be between -1 and 1) and multiplied by the output of the sigmoid gate, so that only the parts that are decided are outputted. Figure-3.16 illustrates this process.

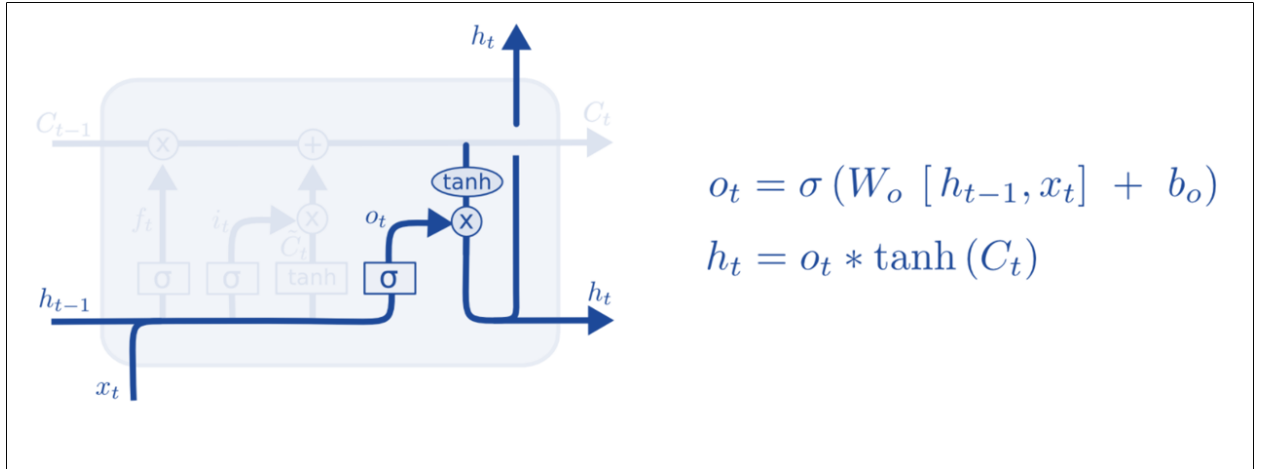


Fig-D.7: Output Gate
Source: Adapted from [6]

Appendix E: Recurrent Neural Networks

3.2.1.1 Recurrent Neural Networks vs Feed-Forward Neural Networks

Information moves only in one direction in the case of a feed forward neural network (i.e. from the input layer, through the hidden layers, to the output layer). The information moves straight through the network and never touches a node twice. Feed-forward neural networks do not hold memory of the input they receive and are thus not ideal at predicting what's coming next. Since a feed-forward network only uses the current input, it has no notion of order in time. It thus cannot remember anything about what happened in the past except its training.

On the other hand, in a RNN the information cycles through a loop. When a decision is being made, the current input as well as what has been learned from previous inputs is used. Fig-3.3 illustrates the difference in the flow of information between a feed-forward neural network and a RNN. As can be observed, a RNN has two inputs: the present and the past.

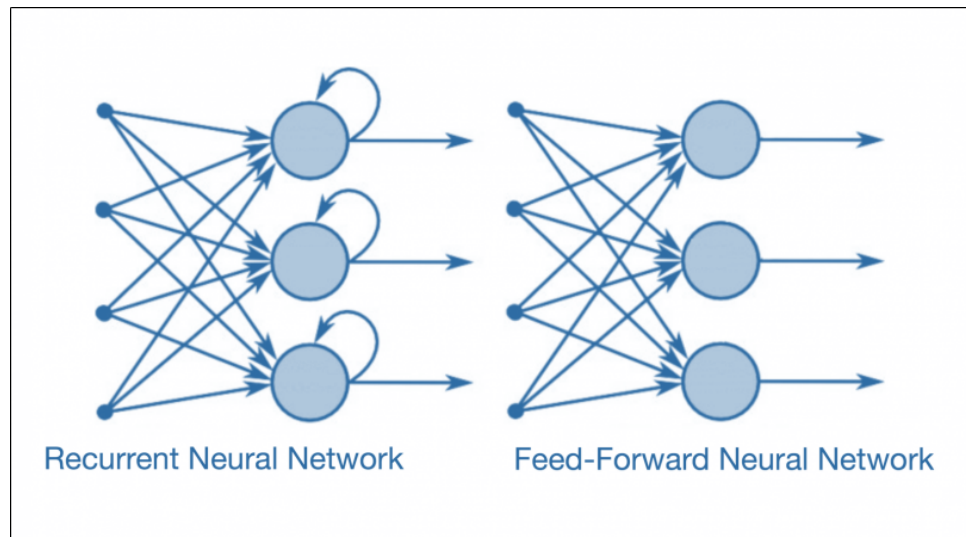


Fig-E.1: Flow of information in a RNN vs a feed-forward neural network

Source: Adapted from [4]

3.2.1.2 Issues with Recurrent Neural Networks

The gradient descent algorithm is commonly used to find the global minimum of the cost function that is going to be an optimal setup for a particular neural network. Information

flows through the neural network starting from the input neurons to the output neurons, while the error is calculated and propagated back through the network in order to update the weights of the model (This process is also called back-propagation). For the case of a RNN, Backpropagation Through Time (BPTT) is used while calculating the partial derivatives of the error with respect to the weights.

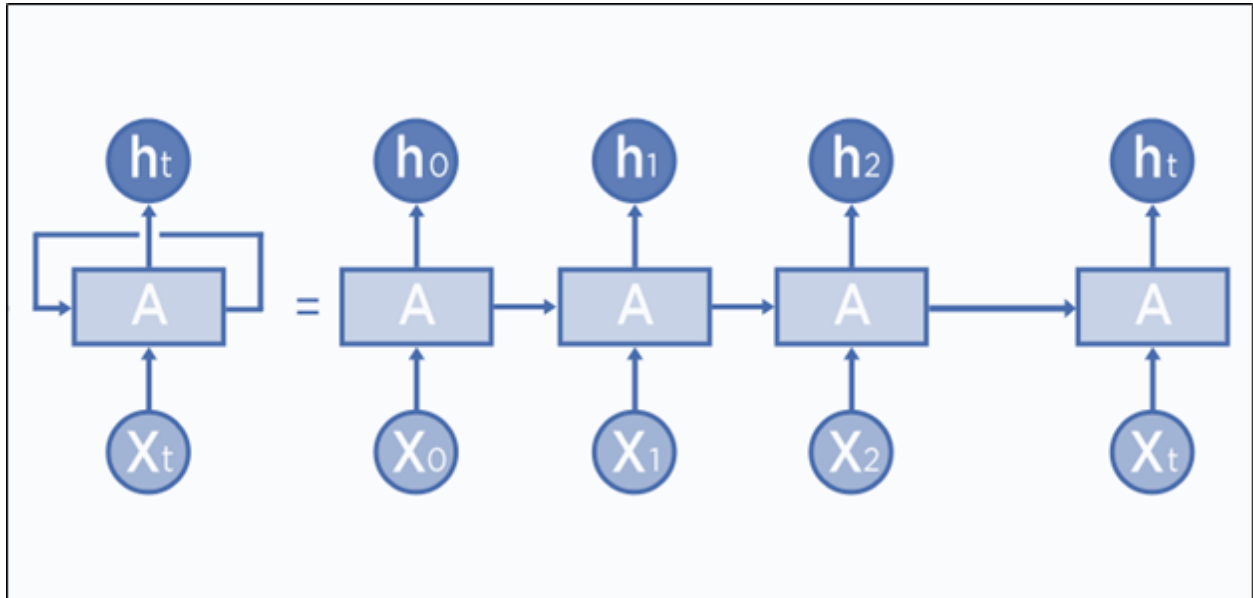


Fig-E.2: Unrolled RNN over time
Source: Adapted from [5]

The image above illustrates an unrolled RNN. Within BPTT the error is back-propagated from the last to the first timestep, while unrolling all the timesteps. This allows calculating the error for each timestep, which allows updating the weights. The algorithm can also be summarised as follows:

1. Present a sequence of timesteps of input and output pairs to the network.
2. Unroll the network then calculate and accumulate errors across each timestep.
3. Roll-up the network and update weights.
4. Repeat.

There is however an issue that arises when we back-propagate over many layers in time. If the weights are too big, the gradients grow exponentially, leading to the exploding-gradient problem. This problem however can be easily solved by squashing or truncating the gradients. On the other hand, if the weights are small, the gradients shrink exponentially. As a result, either the model stops learning or takes way too long, a problem referred to as vanishing-gradient. This problem can be solved with the use of a Long Short-Term Memory (LSTM) network. Refer to Appendix-D to learn more about LSTM networks.

Appendix F: BERT

3.3.1 Information Flow

Figure-3.8 illustrates the bert architecture. A word begins with its embedding representation from the embedding layer. A new intermediate representation is created by every layer which does multi-headed attention computation on the word representation of the previous layer. All these intermediate representations are of the same size. In figure-3.8, E_1 is the embedding representation, Trm are the intermediate representations of the same token, and T_1 is the final output and. In a 12-layers BERT model a token will have 12 intermediate representations.

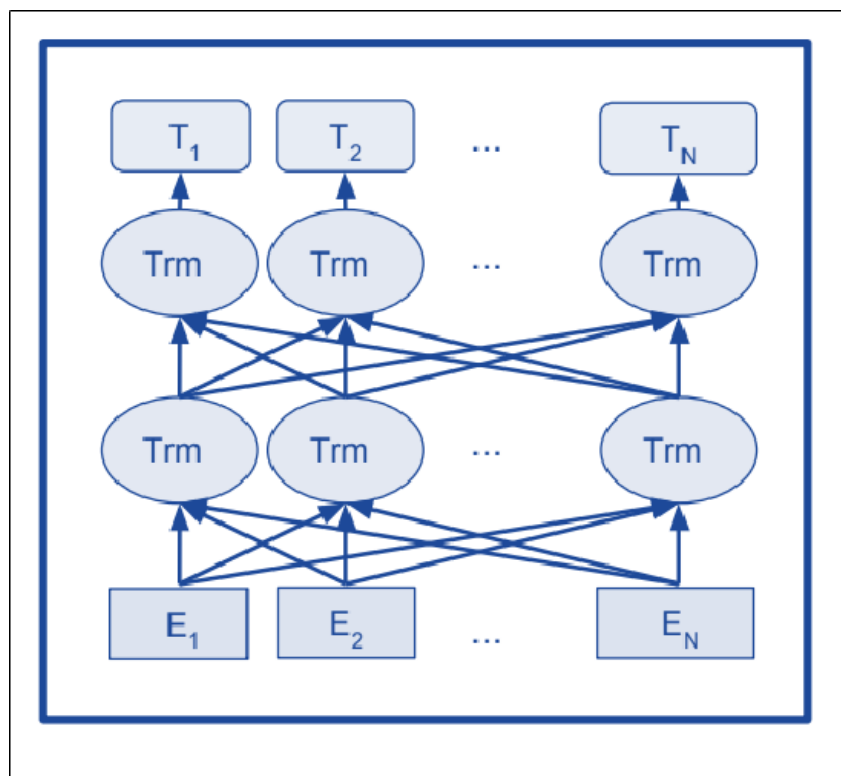


Fig-F.1: Bert Architecture

Source: Adapted from [7]

BERT makes use of two training strategies namely Masked LM, and Next Sentence Prediction

1) Masked LM (MLM)

Prior to when word sequences are fed into BERT, 15% of the words in every sequence are replaced with a [MASK] token. Next, based on the context provided by

the other, non-masked, words in the sequence, the model attempts to predict the original value of the masked words.

Determining the output word thus requires:

- On top of the encoder output, a classification layer is added.
- Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
- Softmax is used to calculate the probability of each word in the vocabulary.

Fig-3.9 illustrates this process. Furthermore, only the prediction of the masked values is taken into consideration, while non-masked words are ignored by the BERT loss function.

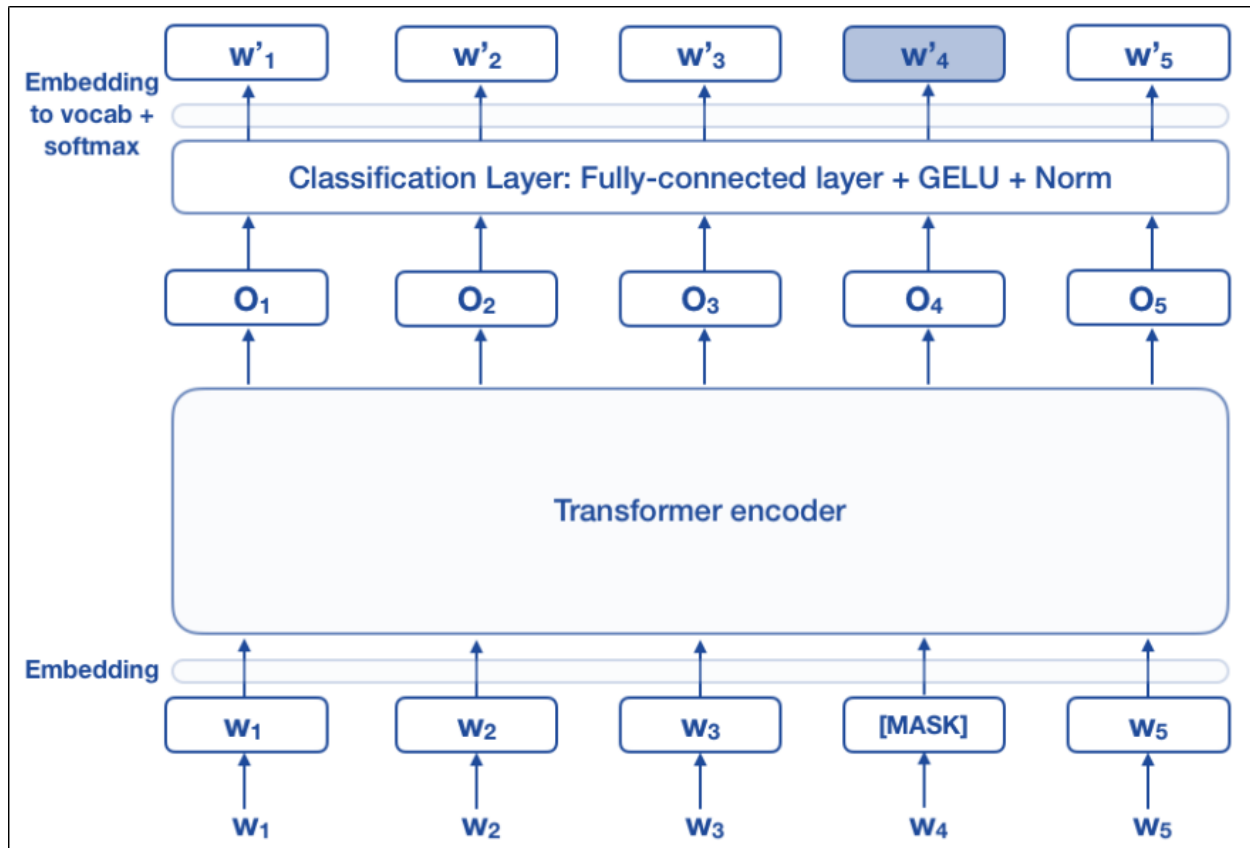


Fig-F.2: Masked LM
Source: Adapted from [8]

2) Next Sentence Prediction (NSP)

During the training process of BERT, pairs of sentences are fed to the model as input, and the model then learns to predict whether the second sentence in the pair is the subsequent sentence in the original document.

While training, $\frac{1}{2}$ (i.e. 50%) of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while for the other half a random sentence from the corpus is selected as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence.

References

- [1] Saha S. (2018, December 16). Medium. Retrieved April 04, 2021, from <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-easy-way-3bd2b1164a53>
- [2]. Jason Brownlee. (2019, April 17). Retrieved April 04, 2021, from <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>
- [3]. Arden Dertat. (2017, November 9). Retrieved April 04, 2021, from <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>
- [4]. Amit Shekhar. (2018, April 14). Medium. Retrieved April 04, 2021, from <https://medium.com/mindorks/understanding-the-recurrent-neural-network-44d593f112a2>
- [5]. Niklas Donges. (2019, June 16). Retrieved April 04, 2021, from <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>
- [6]. Christopher Olah. (2015, August 27). Retrieved April 04, 2021, from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [7]. Yashu Seth. (2019, June 12). Retrieved April 04, 2021, from <https://yashuseth.blog/2019/06/12/bert-explained-faqs-understand-bert-working/>
- [8]. Rani Horev. (2018, Nov 11). Retrieved April 04, 2021, from <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270#:~:text=How%20BERT%20works,%2Dwords.>
- [9] Ekman, P. (1992). Are there basic emotions? *Psychological Review*, 99(3), 550–553.
<https://doi.org/10.1037/0033-295X.99.3.550>
- [10] [Aliapoulios, M., Bevensee, E., Blackburn, J., Bradlyn, B., De Cristofaro, E., Stringhini, G., & Zannettou, S. \(2021\). An Early Look at the Parler Online Social Network. <http://arxiv.org/abs/2101.03820>](http://arxiv.org/abs/2101.03820)

[11] [Brian Fung, CNN Business. \(2021, January 10\). Parler has now been booted by Amazon, Apple and Google. CNN.](#)
<https://www.cnn.com/2021/01/09/tech/parler-suspended-apple-app-store/index.html>

[12] [GabLeaks. \(n.d.\). Retrieved April 11, 2021, from https://ddosecrets.com/wiki/GabLeaks](#)
[Whittaker, Z. \(2021, January 11\). Scraped Parler data is a metadata gold mine. TechCrunch.](#)
<http://techcrunch.com/2021/01/11/scraped-parler-data-is-a-metadata-goldmine/>