Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Rory How

Measuring Political Bias in British Media:

Using Recurrent Neural Networks for Long Form Textual Analysis

Master's Thesis
Espoo, 24th May 2020

Supervisor: Professor Aristedes Gionis, Aalto University
Advisor: Daryl Weir Ph.D.

Aalto University
School of Science

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Rory How		
Title: Measuring Political Bias in British Media: Using Recurrent Neural Networks for Long Form Textual Analysis			
Date:	24th May 2020	**Pages:**	80
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Professor Aristides Gionis		
Advisor:	Daryl Weir Ph.D.		

In this thesis we aim to explore methods of determining political bias in the traditional British print media. It can be shown that much of the British public perceive there to be explicit political biases in many of the UK's most popular media outlets. It is also known that people are inherently prone to political influence from their sources of news. Due to this reason, there is motivation to seek a means to formalise political bias in British media outlets.

In our study, we took the 2016 UK referendum of EU membership as the source to identify a political bias. We sought to find a means in which to determine on a sentence level, whether a newspaper identified with a pro-leave or pro-remain philosophy. For this, we used the newspapers explicit endorsements for a certain referendum outcome that are provided by the newspapers themselves as a ground truth.

Recurrent neural networks have been shown to be useful when working over data of varying sizes, such as encoded textual data. Recurrent neural networks have also been used to perform classification tasks over short form textual data in the scope of determining political bias. However, little work has been done on processing more long-form textual data for classification tasks within a political bias domain. Here, we sought to determine if recurrent neural networks would be a viable approach for solving this problem, compared to more traditional and simple approaches, such as a Naive Bayes model.

In our study, we were able to determine that recurrent neural networks are successfully able to determine a political bias in British media outlets. Our models also indicated slight biases in supposedly unbiased outlets, such as the BBC.The generated models were also able to transfer their learnings into new domains, such as determining EU membership political bias in more recent news articles. However, due to a lack of input data, and ground truths applied in a broad manner, traditional methods such as the Naive Bayes were able to achieve similar results to the recurrent neural networks, with much less compute power required.

Keywords:	Politics, Bias, Media, Recurent Neural Networks, LSTM, GRU
Language:	English

Acknowledgements

I wish to thank my friends and family for constantly moaning at me about the status of my education, and I'd also like to thank the British government and population as a whole for giving me something worthwhile to write about.

Helsinki, 24th May 2020

Rory How

Abbreviations and Acronyms

ANN	Artificial Neural Network
RNN	Recurrent Neural Network
BiRNN	Bidirectional Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
BiLSTM	Bidirectional Long Short-Term Memory
BiGRU	Bidirectional Gated Recurrent Unit
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
EU	European Union
UK	United Kingdom of Great Britain and Northern Ireland

Contents

Chapter 1

Introduction

1.1 A Marriage of Media Bias and Machine Learning

As members of society, it is unavoidable that we consume media on a daily basis, even if we are not explicitly going out of our way to find it. This gives an inherent amount of power to those who are responsible for generating such media, as there is a capability to influence and sway opinion. Furthermore, many people actively seek out media in which to delegate their own opinions. That is, they consider the media sources to be more well informed than they are, so look to this in order to find an opinion in which to share.

Many members of the British public hold strong opinions on the partisan nature of the print media. For example, when polled by YouGov [8], out of those who held opinion of the political bias on the popular newspaper *The Daily Mail*, 81% believed that it was leaning towards the right-wing, with a total of 44% going as far as to say that they believe the outlet to be "very right-wing". However, these opinions are purely subjective and hold little weight in a formal context. But, the apparent unity of the British public on topics such as these hints that there could be some way in which this could be formalised, of which we will try to pinpoint as a research focus of this thesis.

These opinions that the British public believes the newspapers' to possess is not without impact: In a study conducted by Wanta, Golan and Lee of the US media [42], they found that for the countries that were more commonly covered in the US media, the American public widely considered these to be more important to the "American interest" as a whole. Furthermore, within the countries that received negative attention from the media, the public followed suit and also employed a negative opinion of the country. Perhaps more interestingly, countries that received positive attention from the media did not necessarily receive more positive support from the American public. Whilst this is not within the UK specifically, this showcases the general receptive nature of the public when given opinions about nations (and their ideologies) as news.

The 2016 referendum of the UK's membership in the EU serves as an ideal base for our studies. During this time, many of the mainstream British newspapers gave an explicit endorsement to a certain campaign [15] [27] [31] that can be used as a ground truth for predicting political bias in articles around this topic. This is also quite a relevant point of interest to study, as the referendum results were very close (51.89% voting to leave, 48.11% voting to remain), we can see that even minor forms of influence from the media on public opinion has the possibility to sway the outcome of an entire vote.

With more recent developments in textual analysis, such as the advent of the Word2Vec collection of algorithms created By Mikolov, Chen, Corrado and Dean in 2013 [29], we are able to able to represent words in a vector space whilst preserving the word relationships with a high level of accuracy. Furthermore, there exists high-dimensional "pre-trained" datasets implementing these word embeddings over billions of existing news articles. These complex models accurately represent the highly nonlinear relationships between terms in news articles, and perhaps these could existing embeddings could transferred over effectively into a new problem domain — modelling term relationships in British news articles covering the lead up to the EU referendum vote on the 23rd June 2016.

There have also been waves of developments utilising recurrent neural networks for classification tasks, with the advent of "gates" in model cells that regulate the quantity of information travelling through a network. This is shown by the creation of the Gated Recurrent Unit by Chung et al. in 2014 [14], and the LSTM cell by Hochreiter et al. in 1997 [24]. Recurrent neural networks have traditionally been hampered by the issue of maintaining the influence of long term dependencies in the learned weights of a network (known as the vanishing gradients problem), but the utilisation of gates helps to mitigate this issue. When modelling long form text, such as entire news articles, we could capitalise on these more recent developments in such a way that we can classify entire documents into a political ideology, whilst maintaining important learned weights that may be formed at any point in the document. However, there are multiple different approaches to be able to classify political bias using recurrent neural networks, and we would like to be able to compare and contrast different approaches to be able to give a "best" model that generates the most accurate classification predictions for articles surrounding a given topic.

1.2 Research Questions

With this in mind, our research questions are as follows:

RQ1: Are we able to find a way in which to determine political bias in the traditional British media, by utilising ground truths around certain events such as the European Union referendum endorsements made to affirm political bias in the way that articles are framed?

RQ2: Are we able to find a way in which to predict a political bias in supposedly unbiased outlets, such as the BBC, by utilising the ground truths surrounding events such as the EU referendum endorsements made by the traditional print media?

RQ3: Which machine learning models produce the highest amount of accuracy and shortest training time in which to make effective predictions of the political biases of news articles?

1.3 Structure of the Thesis

1.3.1 Background

1.3.1.1 Political Bias Background

In this section we will more accurately define our problem domain. We will give background to the political climate in the UK, through the eyes of the print media. We will talk about how the print media has influence over the British population, showing examples where they have had impact. We will also give background into the semantics of our problem domain; we will define what types of bias there are, and what kinds of bias we will be looking to determine using with our own model. Finally, We will talk about the exact problem domain that we are looking to work over — the 2016 European Union referendum. Here we will set the scene for how the newspapers acted during this time, and what questions are left open that we could aim to answer.

1.3.1.2 Sequence Classification Background

In this section we will approach the problem from a technical standpoint. We will summarise neural networks in simple terms, and talk about recurrent neural networks, from their inception to more recent developments. We will also talk about ways in which to convert sequences of words to numerical vectors that can be parsed by these recurrent neural networks. We will talk about the different methods that we will be looking to utilise in our methods.

1.3.2 Methods

Here we will propose a solution in which to determine political leaning in news articles. We will defend this proposal and argue why we aim to take this approach over other ones. We will also showcase similar studies that have resorted to similar methods to solve this problem.

1.3.3 Environment & Implementation

In this section we will describe the environment surrounding the study and how this was implemented. With the methods described in the previous section, we will here describe what was done in concrete terms in order to make this a reality. We will then talk about real-world limitations of implementing a political bias classifier in a real world situation. We also talk about potential flaws in the methods that were taken on, and how they might adversely affect the results that were generated.

1.3.4 Evaluation & Discussion

In this section we will talk about different means of gauging success and use this to argue why a certain model and set of hyperparameters produces the best results. We will make some provisional conclusions on the implementation, and point towards a single best result that solves the problem of performing a binary classification of political bias over newspapers articles covering the European Union referendum campaign. Here we will also discuss what caused the results to be as they are, and provide brief insight on how we could possibly improve upon these results with some small improvements.

1.3.5 Conclusion

Finally, we will summarise the work done. Here, we will talk about whether we were able to answer the question of concretely determining political bias in news articles, more specifically surrounding the 2016 EU referendum campaign. We will also answer each of the research questions proposed in the introduction. Here, we will also talk about what could be done in the future to build upon and improve the work laid out in this document.

Chapter 2

Background

2.1 Background of Political Bias in British Media

According to a 2019 study by YouGov [2], 53% of adults in the UK hold little to no trust in the information provided by national broadsheet newspapers. This information is amplified even further in online-only news outlets, where 60% of adults believe have little to no trust in the information provided. This sets the baseline for motivating a study into bias in media outlets in the UK — we would like to be able to formally denote bias in certain types of articles from certain outlets, as it is clear that trust in media outlets from the British population is especially low.

We can also refer to a study conducted by YouGov in 2017 for a general insight into the British population and how they consider the newspapers to be biased (shown in Figure 2.1). It is clear that the populous consider certain newspapers to be closely affiliated with certain ideologies (for example, *The Daily Mail* being considered right-wing, and *The Guardian* being considered left-wing), but this is purely subjective: people are only able to gauge a political opinion of a news outlet based on their own feeling. With this intrinsic subjectivity in mind, would it be possible to use the explicit endorsements that newspapers provide surrounding certain events (such as the 2016 European Union referendum) in which to deterministically define the political bias of certain articles? Using this philosophy we could take a more concrete approach into labelling political bias in the media.

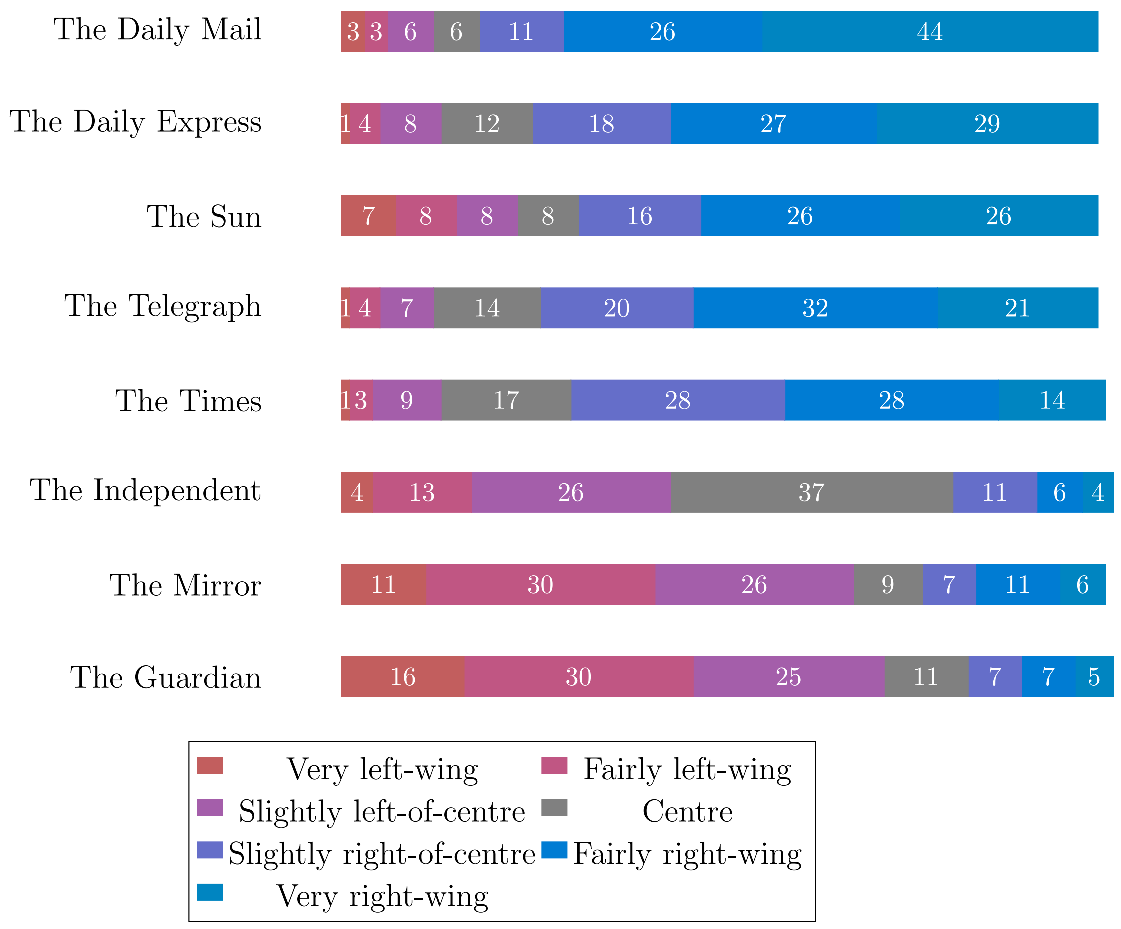


Figure 2.1: A stacked chart showcasing the opinions of 2040 British adults when asked the following question: “Some people talk about ‘left’, ‘right’ and ‘centre’ to describe parties and politicians. With this in mind, where would you place each of the following?” [8], where each value is shown as a percentage. Please note that *The Daily Express*, *The Telegraph* and *The Times* have “very left-wing” values of 1, rather than having “slightly left-of-centre values” of 14 or 13. Results exclude those polled who said that they didn’t know a newspaper’s political ideology.

2.1.1 Media in the United Kingdom

Within the UK, many of the newspapers showcase a political affiliation to certain parties explicitly surrounding given events [38]. This is most apparent during general elections, when each newspaper will traditionally publicly back a certain party, and urge their readership to vote in this direction.

Newspaper	2010 GE	2015 GE	2017 GE	2019 Circulation
The Sun	Conservative	Conservative	Conservative	1 371.19
The Daily Mail	Conservative	Conservative	Conservative	1 199.76
The Mirror	Labour	Labour	Labour	499.82
The Times	Conservative	Conservative	Conservative	406.28
The Telegraph	Conservative	Conservative	Conservative	335.74
The Express	Conservative	Conservative	UKIP	312.77
The Financial Times	Conservative	Conservative	Conservative	168.55
The Guardian	Lib Dem	Labour	Labour	134.57
The Independent	Lib Dem	Lib Dem	None	N/A

Table 2.1: Newspaper endorsements given for general elections (denoted in table as GE) in 2010 [44], 2015 [45] and 2017 [46]. ‘None’ denotes that the paper made no endorsement for that election. The final column denotes the 2019 circulation for that paper, in thousands [3]. Here we use the statistics for the weekday edition of the newspaper: For example, We use *The Sun*’s daily readership figures as opposed to *The Sun on Sunday*’s.

This partisan approach to print media allows for us to be able to extrapolate party-based influence at a population level. If we refer to Table 2.1 it becomes clear that in the past three general elections, the more popular newspapers often back the Conservative party (with the notable exception of The Mirror). This shows that a large portion of the British public are exposed to pro-conservative media, rather than a more balanced source. However, we would like to be able to further prove that these newspapers follow a certain philosophy not only through their explicit endorsements, but through the way in which they word their articles. This kind of subtle bias is something that people are less likely to pick up on, but still have the potential to be influenced by.

2.1.1.1 The BBC

The BBC is a unique case in the UK, as it is state owned (and so is less likely to have intrinsic bias based on the political leanings of its ownership). Furthermore, the BBC has an official guideline [1] that formalises this impartiality.

“ The BBC is committed to achieving due impartiality in all its output. This commitment is fundamental to our reputation, our values and the trust of audiences. The term ‘due’ means that the impartiality must be adequate and appropriate to the output, taking account of the subject and nature of the content, the likely audience expectation and any signposting that may influence that expectation. ”

Figure 2.2: The first paragraph of the BBC editorial guidelines, section 4.1: Impartiality [1]

This means that the BBC has a duty to report all sides of a story, without bias. However, the tricky nature of this means that it is likely that the BBC could have a minor political bias in the way that its articles are phrased. Here, we will aim to utilise the impartiality of other news sources to either confirm the BBC’s own impartiality or preference to certain political parties and ideologies.

2.1.2 Types of Bias

We can divide the slant of the media into several definitions (which we will specify here). Whilst all media has a responsibility to report correct news (and for the sake of this study, we assume that they do), there can be nuances in how the news is reported that can have a significant impact in how this is processed by the reader. A portion of political bias can be determined by what news is reported, and how this is presented to the reader (aside from the content of the article itself). Because this is often a human-based activity, we can assume that there will be some level of subjectivity in how the news is presented to the reader, and what is prioritised.

Another area of political bias is called *selection bias* [33]. Some stories may not be covered by some outlets at all, but may be considered vitally important to others. This is a human-based process of how important a news outlet considers an issue to be. This can often by swayed by the demographic of the newspapers’ readership, their target audience, or even the owner of the newspaper itself. The ability to not cover a story at all can be a manner in which to shape public opinion surrounding certain issues, without explicitly being biased in the news text itself.

If a topic is covered by a newspaper, they may choose to omit certain parts of the story, or they may choose to have the story occupy more physical space on the page to grant additional exposure to it. This is known as *coverage bias* [4]. When there are many breaking stories surrounding a large issue (such as in the lead up to the European Union referendum in the UK in

2016), only covering certain parts of a story can be an effective way in which to mould an audiences opinion in a certain direction.

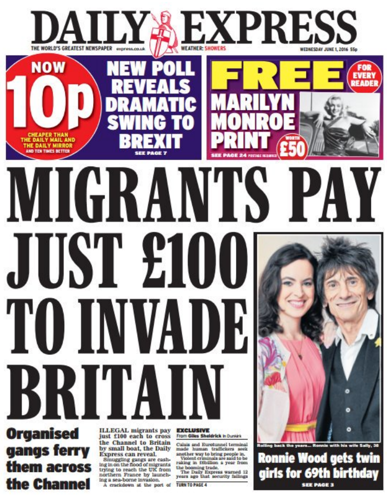
If we then assume that all relevant aspects of a story are covered by a news outlet, the newspaper is still able to present facts in such a way to point the audience in a certain direction. This is known as *framing bias* [18]. Often, this is down to the individual journalist and their writing style. Here, the way that facts are presented are often aligned with the journalists own beliefs, which makes it difficult for a reader to create their own opinions from the facts presented. At an even more nuanced level, it is possible that the specific wording that is used to convey a fact can have micro-effects in how a user responds to them. Framing bias can be often very subtle but when a news outlet has an extensive reach to a large audience, this can have a large real-world impact in political opinion.

Lastly, *statement bias* [37] is the perspective of the individual who writes the article. This is when the author will make opinionated comments in an article that represents their own beliefs surrounding a topic. This will often be separated into opinion columns for various outlets, but others will intertwine statement bias with factual news reporting. This often makes for better reading for someone who already agrees with the opinion of the author, however, for audiences who are unopinionated on a topic, this makes it more likely that the user will adopt the same opinion as the author.

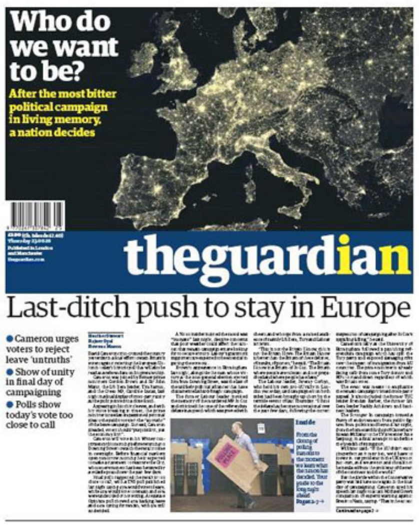
The combination of these subtle (and obvious) ways in which to convey a certain bias can mean that it can be somewhat straightforward to push an audience towards a certain ideology surrounding certain issues. Next, we will cover studies of these biases in a real-world scenario, namely the 2016 United Kingdom referendum of its membership in the European Union.

2.1.3 The 2016 European Union Referendum

In the lead up to the EU referendum on the 23rd of June 2016, many of the leading newspapers in the UK diverted the brunt of their attention towards the topic, with 77% of articles published on the leading news outlets in the UK providing coverage relating to the EU referendum [31] in the final week preceding the referendum vote. However, much of the reporting was done so giving focus to the strategies taken by both sides, which lead to many detailed issues being left under the radar. In a study by Oxford University [31], 41% of referendum-related articles were labelled as pro-leave, 27% were labelled as pro-remain (with the remaining articles being considered to have no significant referendum-based leaning). Yet conversely, in a study conducted by Deacon et al. at Loughborough University [15], many papers were shown to present a balanced or neutral view in their referendum related articles,


DAILY EXPRESS
NOW 10p
NEW POLL REVEALS DRAMATIC SWING TO BREXIT
FREE MARILYN MONROE PRINT
MIGRANTS PAY JUST £100 TO INVADE BRITAIN
Organised gangs ferry them across the Channel
Ronnie Wood gets twin girls for 69th birthday

(a) A headline from the pro-leave newspaper The Daily Express


Who do we want to be?
After the most bitter political campaign in living memory, a nation decides
theguardian
Last-ditch push to stay in Europe
Cameron urges voters to reject leave 'untruths'
Show of unity in final day of campaigning
Polls show today's vote too close to call

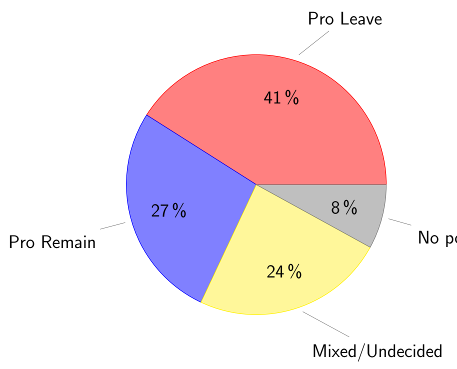
(b) A headline on the day of the referendum vote from the pro-remain newspaper The Guardian

Figure 2.3: A side-by-side contrast of headlines published by a pro-leave and pro-remain outlet relating to the referendum vote, respectively.

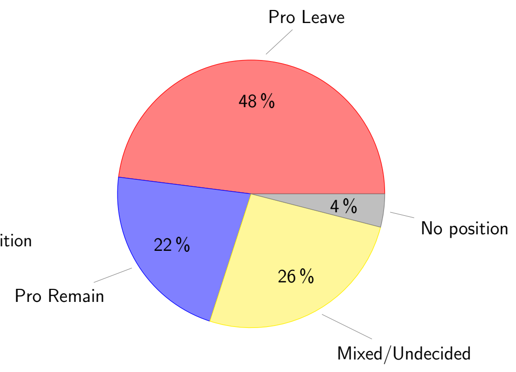
with 7 out of 10 newspapers studied presenting a balanced view in over 40% of articles printed.

Within the national newspapers, the majority of the more popular outlets explicitly endorsed either the pro-leave or pro-remain campaigns. The pro-leave campaign was supported by The Sun, The Daily Mail, The Daily Express, The Sunday Express, The Daily Telegraph, The Sunday Telegraph, The Sunday Times, and The Spectator. Eight prominent newspapers supported Remain: The Guardian, The Times, The Financial Times, The Independent, The Mail on Sunday, The Mirror and The Observer [31]. Whilst many of these newspapers declared their endorsements in the final week of the referendum vote (with the exception of The Daily Express), some of the newspapers expressed euro-sceptic sentiment prior to the referendum announcement.

According to the study conducted by Moore and Ramsay [31], immigration was presented to be a large issue by the pro-leave newspapers, making a front-page appearance on 78 occasions, with only 12 in total being published by largely pro-remain outlets. Here, we could assume that the newspapers publishing more articles related to immigration issues, are targeting audiences that have more divisive opinions regarding immigration. Whilst some of these articles were not directly relating to the UK's membership in the EU directly, this might be an issue that pushes the audience in a certain direction (as immigration was a topic covered extensively by the leave campaign itself [10]). However, this higher level of exposure for immigration coincides with data from the study conducted at Oxford University, where a sample of 1,000 people were asked "What is the most important issue facing Britain today?". The results from this showed that concern regarding immigration was highest in the second half of 2015, where 56% considered it to be the most important issue. Whilst this doesn't directly align with a pro-leave philosophy, the leave parties' utilisation of immigration as a cornerstone of their campaign seems to hint at their capability to convert uncertain voters over this issue. Furthermore, the pro-leave newspapers printing more far more headlines on this subject than the pro-remain ones would seem to confirm this.



(a) Unweighted article biases



(b) Weighted article biases by newspaper reach

Figure 2.4: A pie chart showcasing the the classification of biases in 2378 studied articles by Oxford University [31]

Whilst immigration has always been an important issue to the British public (being considered as the most important issue in the UK by at least 20% of people regularly from 2003 onward), the EU/Common Market has not raised the same level of concern. In fact, less than 10% of the public considered the EU to be the most important issue in the final months of 2015. However, this concern spiked soon after, recording a peak of 33% saying is was most the most important topic just before the date of the referendum vote. This hints towards the British public not holding particularly strong opinions towards the EU until very close to the day of the vote. This then gives the opportunity for the newspapers to be able to sways opinions of undecided voters more easily.

With this in mind, we could say that the leave campaign had not seen so much more exposure in the newspapers vs the remain campaign (with 41% articles being pro leave vs 27% respectively). However, this figure becomes amplified still if we apply a weighting for the reach of newspapers (conduced by Prime research). After this weighting is applied, we can impact of the pro-leave media increases to 48%, whilst the pro-remain media decreases to only 22% (as shown in Figure 2.4). This shows that the leave campaign has the potential to influence a larger audience with their biases than the leave campaign. In a referendum that resulted in 51.89% of votes going to leave, and 48.11% going to remain, there is potential for even minor differences in campaigning and media coverage being enough to tip the result in one direction or the other.

But this still leaves one question open: can we further determine this political bias programmatically rather than via human opinion? If we are able to utilise statistical methods to predict leanings of articles, we may be able to confirm supposed bias in certain outlets, or we may be able to predict bias in outlets that do not explicitly state a bias. Finally, we may be able to give a political bias (as a percentage) for outlets that supposedly aim to completely unbiased (such as the BBC).

2.2 Background of Corpus Classification

2.2.1 Naive Bayes

Perhaps the most simple of machine learning algorithm is the Naive Bayes. A Naive Bayes classifier is based on the Bayes Theorem, which is as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.1}$$

Intuitively, the Bayes Theorem states that we can find the probability of A happening given the occurrence of B. To put this into the perspective of a text classification problem, we can find the probability of a sentence $X = (x_0, x_1, \ldots, x_n)$, containing individual words x_i, having a pro-remain bias (i.e a label y), given a similar article B which is known to have a similar bias. With this context, we can rewrite Bayes Theorem as follows:

$$P(y|X) = \frac{P(X|y)P(X)}{P(y)}. \tag{2.2}$$

Since X is a vector of words, we can expand these, which allows us to calculate the probability of their appearance in a sentence, given the input dataset.

$$P(y|X) = \frac{P(y|x_0)P(y|x_1)\dots P(y|x_n)P(y)}{P(x_0)P(x_1)\dots P(x_n)}. \tag{2.3}$$

This allows us to generate the likelihood of a certain output class (in our case, either pro-leave or pro-remain) given a certain set of input data. However, the Naive Bayes assumes that each feature is independent of each other; in our context this would imply that a word x_0 does not influence any other word in the input vector, X. Intuitively, we know that each word in natural language is dependent on and influences the context in which it is used. Regardless, the Naive Bayes can be thought of as a base of comparison for more complex methods in which to classify textual data.

2.2.2 Multi-layer Perceptron

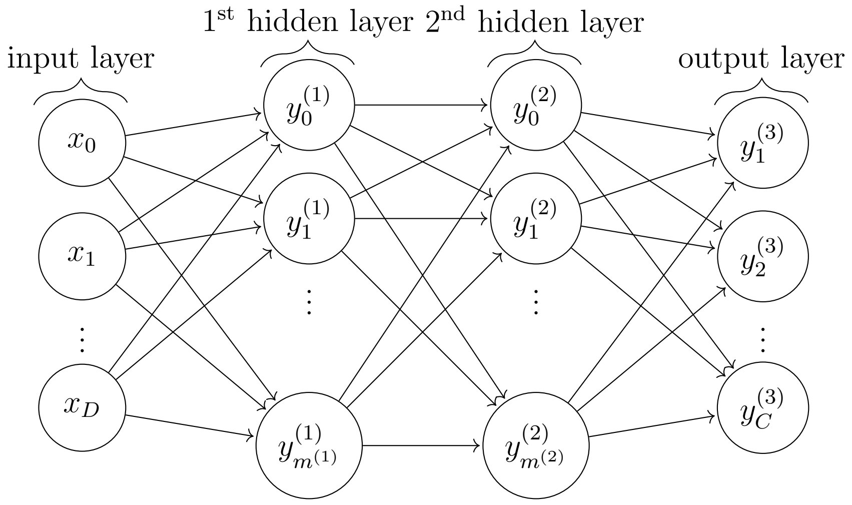


Figure 2.5: Network graph of a 3-layer perceptron with D input units and C output units. Each hidden layer l contains $m^{(l)}$ hidden units.

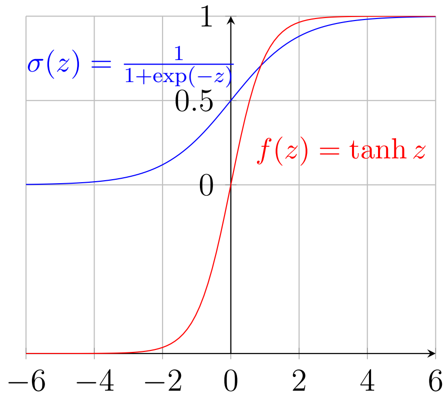
Of the deep learning algorithms that we see today, many of these stem from the multi-layer perceptron. The goal of the the multi-layer perceptron is to estimate some function, f^*. If we are to have a classifier $y = f^*(\boldsymbol{x})$, we are mapping our input, $\boldsymbol{x}$ to our label, $\boldsymbol{y}$.

In a typical feed forward network, we feed our input data $\boldsymbol{x}$ in a single direction through the network (as shown in Figure 2.5). Here, each node is a perceptron that has a classification rule as follows:

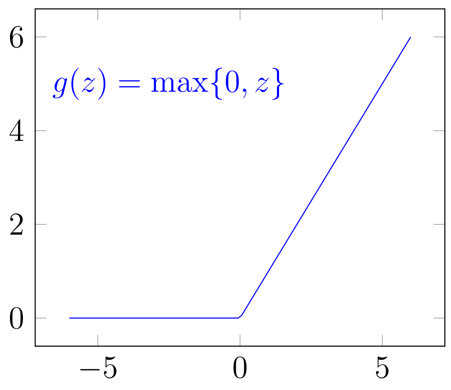
$$f(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

However, one caveat of using a single perceptron is that it is unable to capture complex nonlinear relationships between variables. To get around this, we can introduce more linear classifiers to be stacked on top of the current with certain activation functions (shown in Subsection 2.2.3) that allow us to more accurate capture these nonlinear relationships.

2.2.3 Activation Functions



(a) The logistic sigmoid (in blue) and tanh (in red) activation units.



(b) The Rectified Linear Unit

Figure 2.6

If we are to model a multi-layer artificial neural network with only linear layers, it can be shown that the whole function remains a linear function of its input. For example, if we had two functions $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^\top \boldsymbol{x}$ and $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^\top \boldsymbol{w}$, then we can say that $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{W}^\top \boldsymbol{x}$. This can be simplified down to $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w'}$, if we say that $\boldsymbol{w'} = \boldsymbol{W}\boldsymbol{w}$. Therefore, we want to use nonlinear functions to correctly capture the complex nonlinear relationships within our input features.

One such activation function is the logistical sigmoid. Since the logistical sigmoid is a nonlinear function, it is able to model the nonlinear mappings between inputs and outputs more accurately. Its equation is shown below:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{2.5}$$

However, one drawback with the sigmoid function is that it pulls values to each end of the Y axis. That is, there is a lot of variance in output y values when given a z value that is close to zero. This means that if we have very high or very low values of z, this will result in very little change in the output of the sigmoid function. This is known as vanishing gradients. This is especially the case when dealing with long term dependencies (a common case with RNNs, described in Section 2.2.5)

One alternative to the logistical sigmoid function, to help us with the vanishing gradients problem is the Rectified Linear Unit (ReLU). As visible in Figure 2.6b, the increase in y values is linear with respect to any positive input values z. Since ReLU is almost a linear function, this makes it easier to optimise using gradient based methods. One additional benefit is its capability to output a true zero value, reducing the potential amount of computations needed. However, due to its non-negative linearity, this puts it at risk of exploding gradients. This means that it is not ideal for recurrent neural networks, as these introduce more long term dependencies where gradients can continually spiral.

$$g(z) = \max\{0, z\}. \tag{2.6}$$

2.2.4 Training

2.2.4.1 Loss functions

When our model has made a series of predictions, we need a way to be able to gauge the quality of our predictions, which we can then use to tune the learnable weights of our model. For many binary classification problems, we would use cross entropy with a number of two possible classes (using a mean reduction over losses for each prediction).

$$\ell(x, y) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)). \tag{2.7}$$

In this equation, we take the log probability of a positive score ($y = 1$), and add $\log(p(y))$ to the loss for this prediction. Then we add the probably of the negative outcome ($y = 0$), $\log(1 - p(y))$, and add this to the loss for this prediction. Then, we take the mean across all predictions and this gives us out total loss value.

Since we are working with predictions that are in the range $[0, 1]$ (i.e a Bernoulli output distribution), we will often use a sigmoid activation (shown in equation 2.5), which will allow us to "squash" our outputs from our neural network into this range.

2.2.4.2 Optimisation

When we are training our model, we need to be able to work calculate the loss and tune our models learned parameters based on this. One of the more simple and popular approaches to solving this problem is Stochastic Gradient Descent (known also as SGD).

SGD descent is an extension of the basic gradient descent computation, shown as follows:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla L(x^{(i)}, y^{(i)}, \theta). \tag{2.8}$$

In the above equation, we are calculating the gradient of the objective function $J(\theta)$ for each input vector x_i in our training set of size m. We can see here that the gradient of the objective function is simply the average of the gradient of the loss function, for our parameters $\theta \in \mathbb{R}$. We then apply this in our update function with a static learning rate η as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta). \tag{2.9}$$

One of the issues with calculating gradient in this way is that its computational cost is $O(m)$. This is fine for smaller training sets but can quickly becomes more troublesome as our training set size, m, increases. For this reason, we can calculate the gradient based on a mini-batch of samples (of size n) that are drawn uniformly from the training set. We often keep the number of mini-batches used, m', constant in gradient calculations, which allows us to keep our gradient calculation cost at a minimum when working over larger training sets.

$$\nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) = \frac{1}{m'}\nabla\theta \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \theta). \tag{2.10}$$

In the same way as standard gradient descent, we apply the learning rate to the estimate of the gradient in order to take a step for our learned parameters.

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}). \tag{2.11}$$

One drawback of updating the weights in this way is that our learning rate η is static throughout the entire training process of the model, and can have a significant impact on the model performance (if we have too high of a learning rate, we are less likely to converge on an optimum, if we have a low learning rate, we might not reach our optimum parameters at all). One way of solving this issue is having adaptive learning rates throughout the course of learning. We can do this by introducing momentum into our update calculations. For SGD, instead of applying the learning rate directly to the gradient estimate, we use it to calculate a velocity value v (parameterised by a momentum value γ) as follows:

$$\begin{aligned} v_t &\leftarrow \gamma v_{t-1} - \eta \nabla_\theta J(\theta) \\ \theta &\leftarrow \theta - v_t. \end{aligned} \tag{2.12}$$

However, one drawback to this velocity based approach is a lack of foresight. Our velocity value is calculated entirely based on the previous step γv_{t-1}. But, we can utilise the **Nesterov Accelerated Gradient** [32] to apply our velocity with momentum to our parameters, to give us an estimate of the next position of the parameters, giving us an estimate of the next parameters:

$$\begin{aligned} v_t &\leftarrow \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &\leftarrow \theta - v_t. \end{aligned} \tag{2.13}$$

One adaptive learning algorithm that utilises this approach is **Adagrad** [17]. However it goes one step further by adapting the updates for each individual parameter, based on its importance. For parameters with frequently occurring features, learning rates are decreased. Conversely, parameters with infrequent features are given higher learning rates. Here we show the gradient at time step t for each parameter θ_i and its corresponding update step:

$$\begin{aligned} g_{t,i} &\leftarrow \nabla_\theta J(\theta_{t,i}) \\ \theta_{t+1,i} &\leftarrow \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i,i} + \epsilon}} \cdot g_{t,i}. \end{aligned} \tag{2.14}$$

In the update rule, we can see that the learning rate η is modified to be for each parameter, based on the past gradients that have been computed for each specific parameter θ_i. In each update step we divide the learning step by G_t, which is a diagonal matrix, where each element i, i is the sum of the squares of gradients, with respect to the parameter θ_i at the given time step t. We then apply a smoothing term ϵ to avoid any divisions by zero. However, the usage of this squared gradients is also its main flaw: the sum will always increase throughout training (since each added value is positive), therefore the learning rate will continue to shrink and become smaller, reducing the effectiveness of the algorithm as time progresses.

Adadelta [47] is an extension on Adagrad that reduces this rapid reduction in learning rate. Adadelta instead stores that previous squared gradients as a decaying average. We then store the running average $E[g^2]_t$ at time step t which is calculated only on the previous average, and the current gradient value. This allows us to replace the diagonal matrix G_t from Adagrad with our running average:

$$\begin{aligned} E[g^2]_t &\leftarrow \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \\ \Delta\theta_t &\leftarrow -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t. \end{aligned} \tag{2.15}$$

However, when using a gradient (rather than parametric) set of units in algorithms such as SGD or Adagrad, these do not align with the type of units that we are outputting with the above equations:

$$\texttt{units of } \Delta x \propto \texttt{units of } g \propto \frac{\delta f}{\delta x} \propto \frac{1}{\texttt{units of } x}. \tag{2.16}$$

To get around this, we replace the learning rate with another exponentially decaying average, based upon the squared parameter updates.

$$E[\Delta\theta^2]_t \leftarrow \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\theta_t^2. \tag{2.17}$$

We then take the root mean squared error of these parameter updates, and slot this into our update rule:

$$\begin{aligned} \Delta\theta_t &\leftarrow -\frac{\sqrt{E[\Delta\theta^2]_t + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} g_t \\ \theta_{t+1} &\leftarrow \theta_t + \Delta\theta_t. \end{aligned} \tag{2.18}$$

One popular adaptive learning algorithm that employs a similar approach is known as **Adaptive Moment Estimation** [26] (**Adam**). Adam takes a similar approach to Adadelta, by taking an exponentially decaying average of past squares gradient. But in addition to this, it also takes a decaying average of past gradients, which is similar to momentum. Both of these are computed as follows. Both of these are estimates of the first and second moments of the gradients respectively.

$$\begin{aligned} m_t &\leftarrow \beta_1 m_{t-1} + (1-\beta_1)g_t \\ v_t &\leftarrow \beta_2 v_{t-1} + (1-\beta_2)g_t^2. \end{aligned} \tag{2.19}$$

One issue with these vectors m_t and v_t is due to their zero-initialisation values, they are biased towards zero. To get around this, a bias is applied to each estimate:

$$\begin{aligned}\hat{m}_t &\leftarrow \frac{m_t}{1-\beta_1^t}\\ \hat{v}_t &\leftarrow \frac{v_t}{1-\beta_2^t}.\end{aligned} \tag{2.20}$$

Similar to Adagrad, we take the square root of the second moment matrix estimate, and use this as a denominator to divide the learning rate η by, finally multiplying with the first moment matrix estimate to calculate the update step quantity:

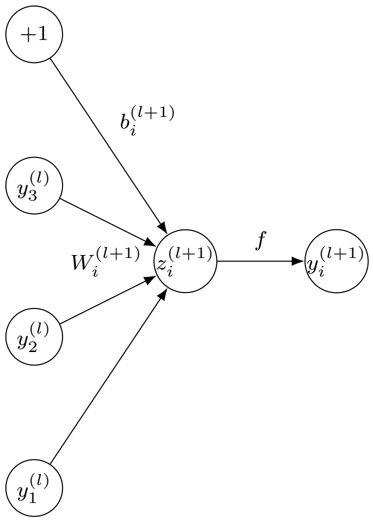
$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + e}}\hat{m}_t. \tag{2.21}$$

Another adaptive learning algorithm that utilises adaptive learning rates is **RMSProp** [22]. This is a modification on the existing AdaGrad algorithm, but we use an exponentially decaying average, which allows us to remove history from the past. This prevents us from setting a learning rate that is too small, when using a non-convex function to train the neural network.

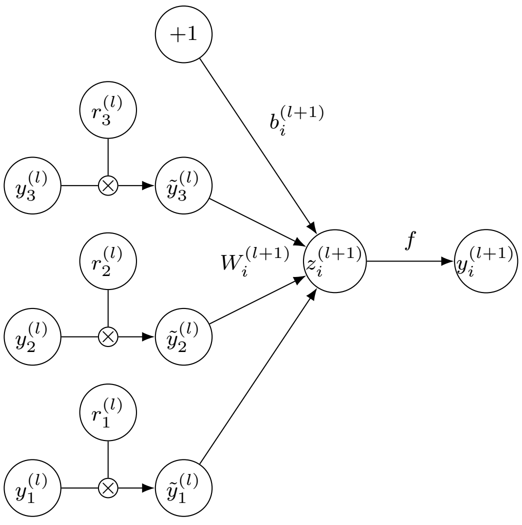
$$\begin{aligned}v_{t+1} &\leftarrow \alpha v_t + (1-\alpha)(\Delta_\theta J(\theta_t))^2\\ \theta_{t+1} &\leftarrow \theta_t + \epsilon\frac{\Delta J(\theta_t)}{\sqrt{v_t}+\lambda}.\end{aligned} \tag{2.22}$$

2.2.4.3 Regularisation: Dropout

Dropout [41] is a computationally inexpensive method in which to prevent overfitting in large-scale neural networks. It works by training an ensemble of sub-networks that are possible permutations of the original network, but with a certain percentage of non-output layers removed. In modern networks, we can 'remove' a unit from the network by multiplying its output value by zero (which is also utilised in forget gates in gated recurrent neural networks, as shown in Section 2.2.5.3). Dropout works alongside minibatch based learning, with functions such as SGD. Each time we load a minibatch, we take a random sample (of a certain size) a binary mask to apply to all of the input and hidden units in the network. Each binary mask for each unit is sampled independently from all other mask vectors. For any of these layers, the mask is a variable r^t that has a probability p of being 1. We then perform an



(a) Standard network



(b) Dropout network

Figure 2.7: Two diagrams showcasing a standard network, versus one with a dropout layer at layer l. The vector of random Bernoulli variables $r_i^{(l)}$ is a set probability p that the value will be 1. We then perform an element-wise multiplication in order to drop certain connections in the network, producing thinned outputs $\tilde{y}_1^{(t)}$, which can then progress through the network as normal.

element-wise multiplication with the outputs of that layer y, in order to create a thinned output $\tilde{y}$. These thinned outputs are then fed into the next layer (where the dropout process repeats).

2.2.5 Recurrent Neural Networks

When we are looking to work with sequential data, and with sequences with variable length, Recurrent Neural Networks [36] are a natural fit due to the way that they are able to reuse the same parts of the model recurrently for each element in a sequence. This means that a single recurrent unit can be reapplied to each element in a sequence of arbitrary length. After each time step, a hidden state $\boldsymbol{h}$ can be updated based on the previous hidden state, and the new input $\boldsymbol{x}^t$ at time t.

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^t; \boldsymbol{\theta}). \tag{2.23}$$

This means that we can ‘unfold’ a recurrent neural network into a series of the same function calls, with differing sequential inputs x_t at each time step t being reused on the transition function f. If we are to introduce an activation function and learned parameters, the update function of recurrent hidden state can be shown as in Figure 2.8.

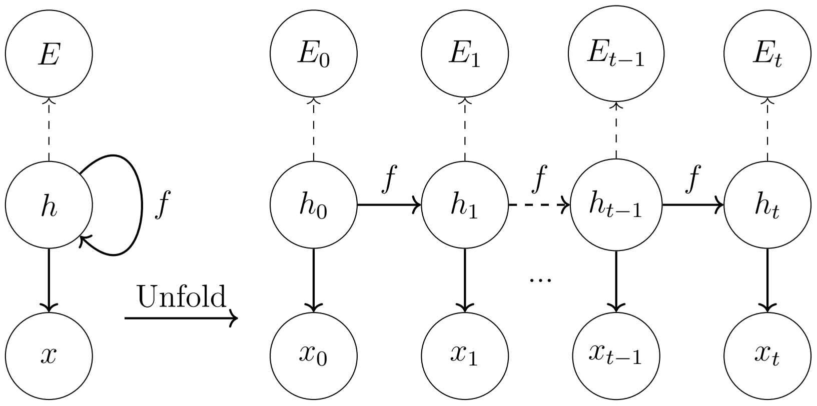


Figure 2.8: A basic recurrent network architecture, without any activation layers or outputs. Hidden states are calculated from left to right using the same objective function, taking in new parameters x at each time step t

However, an issue that arises with this approach is maintaining gradients calculated many iterations ago. Many of these gradients will tend to vanish or explode (This is covered more in depth in Section 2.2.3). When we are working with weights that become exponentially smaller the further through the network we progress, it becomes a challenge to maintain the relevance of past weights further down the network (i.e as time increases).

2.2.5.1 Backpropagation through time

When calculating gradient for a recurrent neural network, we can apply the same general backpropagation algorithm utilised for feedforward neural networks, however we apply it instead to the unfolded computational graph.

As shown in Figure 2.8, for each time step t taken in our recurrent neural network, we have a new error value E. This error is used to calculate the gradients needed for our optimiser function (such as gradient descent or Adam). We calculate these gradients in much the same way as a traditional feed-forward neural network would, using backpropagation.

The basis of backpropagation in a recurrent neural network is to calculate the sum of the partial derivatives of our learned parameters, with respect to our error E at each time step t. For example, to calculate the gradient of the error with respect to a weight vector W across our RNN we would calculate:

$$\frac{\delta E}{\delta W} = \sum_t \frac{\delta E_t}{\delta W}. \tag{2.24}$$

To calculate the partial derivative of our weight vector with respect to our error E, we can utilise the chain rule to be able to take the partial derivative of a weight at a certain time step:

$$\frac{\delta E_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta E_t}{\delta \hat{y}_t} \frac{\delta \hat{y}_t}{\delta h_t} \frac{\delta h_t}{\delta h_k} \frac{\delta h_k}{\delta W}. \tag{2.25}$$

In the above equation we assume that we output some prediction $\hat{y}$ at a time-step t, and this prediction is calculated using our hidden state W which is parameterised by our learned weight vector W, that is directly dependent on the previous state h_{t-1}. This means that each of our hidden states is directly dependent on every other hidden state k that comes before it.

2.2.5.2 The Vanishing & Exploding Gradients Problem

However, calculating gradients in this way over a long sequence lends itself to a problem known as the vanishing gradients problem, first showcased by Hochreiter [23]. If we are performing our backwards calculations on over activation functions such as the sigmoid function (shown in Figure 2.6a), if some of our activation functions result in low gradients (such is the case at either end of the sigmoid graph), these gradients shrink exponentially fast throughout the backpropagation calculations through the network, which can cause them to effectively vanish completely. Conversely, if we have high gradient values, then these can propagate through the network and cause the opposite, for our gradients to explode in value. In the worst case in a practical scenario, this can cause our training process to crash due to the presence of a `NaN` gradient value.

The vanishing & exploding gradients problems are not exclusive to RNNs, however. They are simply more prevalent in RNNs due to the often high sequence length, which is (somewhat) equivalent to a very deep feedfoward network, which would also suffer from the same issue. With this in mind, we can easily see many practical issues with traditional recurrent neural networks, especially when wanting to preserve the semantic value of long-term dependencies. This is when gates that regulate the flow of data through a RNN come in handy, as explained in the next section.

2.2.5.3 GRU

The Gated Recurrent Unit, proposed by Cho at al. [14] aims to resolve the issue of long term dependencies within RNNs by using series of sophisticated activation functions that consist of affine transformations, followed by an ele-

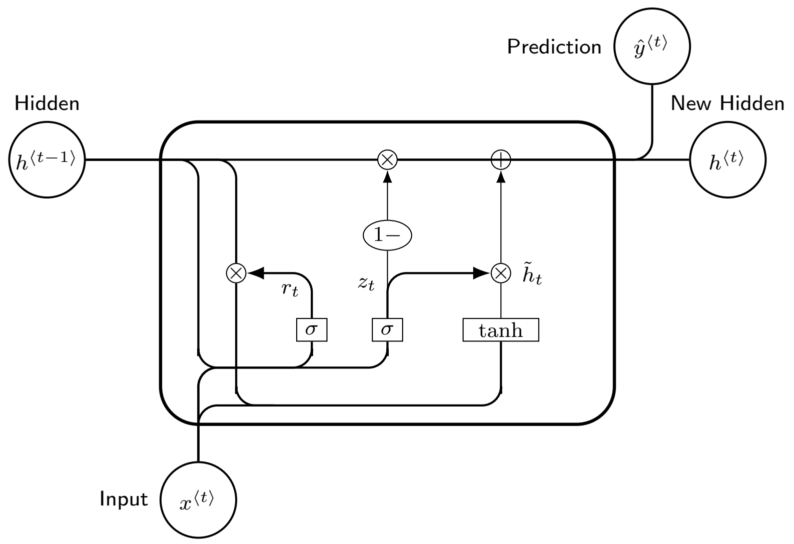


Figure 2.9: A diagram showing a single GRU cell. Each line represents a vector, carrying output from one function to the input of another. Circles represent point-wise operations, and boxes represent learnable network layers. When lines are joined, this represents a vector concatenation, and when paths are split, this is done so via creating a copy.

ment nonlinearity function (known as a gated unit). The activation function of a GRU is broken down as follows:

$$\begin{aligned} r_t &= \sigma(W_r \cdot [h_{(t-1)}, x_t]) \\ z_t &= \sigma(W_z \cdot [h_{(t-1)}, x_t]) \\ \tilde{h}_t &= \tanh(Wx_t \cdot [r_t * h_{(t-1)}, x_t]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t. \end{aligned}$$

Figure 2.10: The calculations used in a single GRU cell. h_t denotes the hidden state, z_t denotes the update gate, $\tilde{h}_t$ denotes the candidate activation and r_t denotes the reset gate, all at time step t

The activation function h_t is a linear interpolation between the candidate activation, $\tilde{h}_t$ and the previous activation h_{t-1}. The candidate activation $\tilde{h}_t$ is created in a similar fashion to a typical recurrent unit, performing an affine transformation on the input vector x_t, but is different in that it does a multiplication of the previous hidden state with a reset value, r_t. The reset gate acts in a similar way to the update gate, but due to the multiplication in the candidate layer, it can cause the network to 'forget' the previously computed state if the computed r_t is close to 0.

Due to the learned parameters of the reset and update gates allows us to account for longer term dependencies (by setting z_t to a higher value), whilst also reducing the risk of the vanishing gradient problem (by learning to remove unimportant information at each timestep, whilst preserving values that carry the most relevance). Similar to the GRU, the LSTM contains a forget gate, which dictates the flow of information to the internal cell state. If our forget layer is able to learn when past information in the sequence is redundant, then it can be discarded via this gate, f_t. We then want to choose what information will be stored inside the cell state, we do this with a sigmoid layer, multiplied by a tanh layer output vector $\tilde{C}_t$. We can then add this to the forget layer output vector f_t multiplied by the old candidate state $C_{(t-1)}$ in order to get the new cell state C_t.

2.2.5.4 LSTM

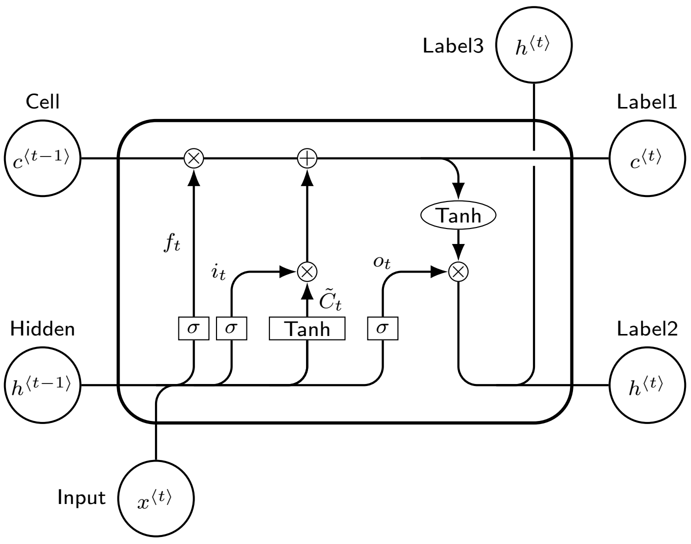


Figure 2.11: A diagram showing a single LSTM cell. Please refer to the corresponding GRU Figure 2.9 for information on diagram semantics.

The Long short-term memory model, proposed by Hochreiter and Schmidhuber [24] predates the gated recurrent unit, and shares a similar cell-based architectural philosophy.

Similar to the GRU, the LSTM contains a forget gate, which dictates the flow of information to the internal cell state. If our forget layer is able to learn when past information in the sequence is redundant, then it can be discarded via this gate, f_t. We then want to choose what information will be stored inside the cell state, we do this with a sigmoid layer, multiplied by a tanh layer output vector $\tilde{C_t}$. We can then add this to the forget layer output vector f_t multiplied by the old candidate state $C_{(t-1)}$ in order to get the new cell state C_t.

However, this cell state C_t cannot be used for predictions, so we push the values through a tanh layer in order to squash them into a Bernoulli distribution, and finally multiply this by the output of another sigmoid gate o_t, ensuring that we only output the parts of the cell state that we want to be included. This then gives us out hidden layer h_t which can be used for making predictions from our model.

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{(t-1)}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{(t-1)}, x_t] + b_i) \\
\tilde{C_t} &= \tanh(W_C \cdot [h_{(t-1)}, x_t] + b_C) \\
C_t &= f_t * C_{(t-1)} + i_t * \tilde{C_t} \\
o_t &= \sigma(W_o[h_{(t-1)}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t).
\end{aligned}
$$

Figure 2.12: The calculations used in a single LSTM cell. h_t denotes the hidden state, i_t denotes the input gate layer (equivalent to the update gate in the GRU), $\tilde{C_t}$ denotes the candidate cell state, f_t denotes the forget/reset gate, and o_t denotes the sigmoid output layer, all at time step t

Both the LSTM and GRU work in a similar fashion, by utilising forget and update gates in order to regulate flow of information throughout the network. However, whilst the LSTM uses an internal cell state in order to regulate the flow of information onto the next layer, the GRU instead uses only reset and update gates, and the reset gate is applied in the candidate activation calculations, rather than applied afterwards. Due to the smaller quantity of layers in the GRU, this will often take less time to train than a corresponding LSTM, but the LSTM has the capability to store a richer set of interactions with its weights and biases.

However, in practice, the LSTM and GRU are often used interchangeably, and it often falls down to the problem space (such as political bias detection) and the input data structure that gives rise to one architecture producing higher accuracy than the other.

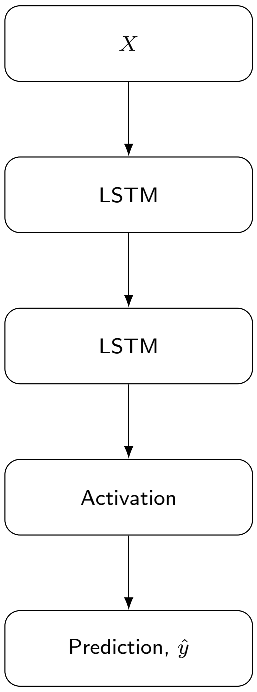


Figure 2.13: A figure to showcase the architecture of a basic 2-layered stacked LSTM, with activation producing an output prediction $\hat{y}$, over all time steps (i.e covering an entire input vector X)

2.2.5.5 Stacked Recurrent Neural Networks

One approach to improve the performance of LSTM and GRU networks is to introduce additional cells and "stack" them on top of one another. This approach has been shown to produce state-of-the-art results in speech recognition [19]. Using a stacked RNN brings benefits to increasing the hidden size of a single RNN:we introduce a higher quantity of learned parameters, and since we separate our RNN cells in to a number of layers, we allow for a higher level of flexibility to capture long-range context.

2.2.5.6 Bidirectional Recurrent Neural Networks

One flaw with the RNN architectures mentioned above is that the hidden state for each element only contains semantic information based on the hidden state at the time steps prior to the input x at its corresponding time-step t. This means that if an input sequence contains highly important information towards the end, the hidden layers before this point are not able to learn with this information being taken into account. A solution to this is the to use bidirectional RNNs.

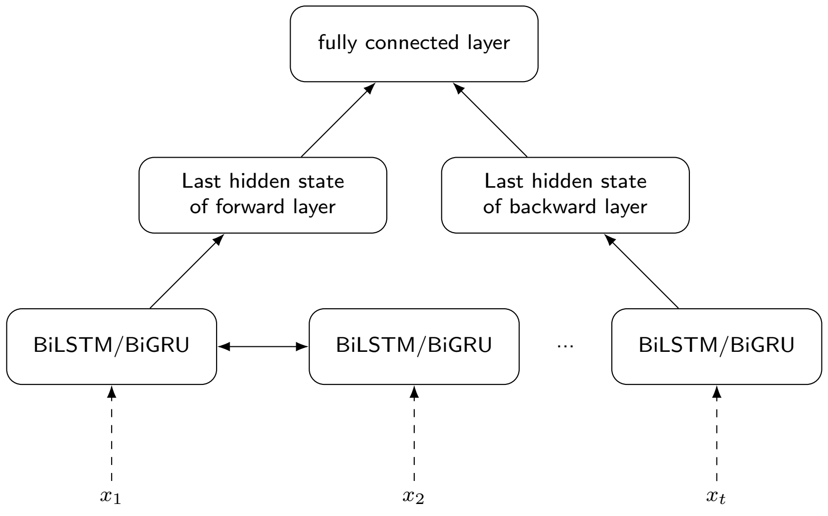


Figure 2.14: A figure to showcase how a how tail concatenation of BiLSTMs or BiGRUs can be used to create a representation of an entire input vector $\boldsymbol{X}$ with equal weighting given to each end, x_1 and x_t.

Bidirectional RNNs [39] work by training a recurrent neural network simultaneously in positive and negative time direction, using two sub-RNNs $h^{(t))}$ and $g^{(t)}$. This then allows us to generate output units that contain information on both the past and the future, but still remains most sensitive to information within that is within close proximity to the input x at time t.

For a classification task that operates over an entire sequence, there isn't much value to be gained from a bidirectional LSTM alone. Since we can just take the final hidden output h_t as a representation of the entire sequence, which should take into account the long term dependencies through the entire sequence. However, it has been shown [49] [40] that we can better represent the entire sequence using BiRNNs when combined with pooling layers. Pooling layers help our model outputs be more invariant to small changes in the input. These methods work by treating the bidirectional output as a 2-dimensional 'image'. When considered in such a way, we can use traditional convolutional neural network methods such as 2-dimensional maximum pooling in order to create a fixed-length summary of the bidirectional output.

Another approach used (with success) by Shen [40] is to use a 'Tail BiLSTM Model' (shown in Figure 2.14), wherein the last hidden state of the backward layer, is combined with the last hidden state of the forward layout from the BiLSTM in order to create a fully-connected layer where the beginning and end of the sequence are treated with equal importance.

2.3 Word Embeddings

A word embedding can be defined as a way of representing words in a particular vector space. Particularly what makes this relevant is the capability to be able to group words in such a way that their cosine similarity, or the 'distance' between words that hold a similar semantic meaning, is minimised. This can be utilised in political bias classification by being able to determine similar sentences and understand when the sentiments are similar. For example, lets look at two politically charged sentences:

- "The EU is a positive force to the UK"
- "The EU should not force its policies on the UK"

We can see here using our human natural language understanding, that these two sentences hold entirely different meanings. However, there are many words in these sentences that are shared between the two. If we can create a word embedding such that the inference can be derived accurately, then this makes it easier for any further classifications to be able to make divisions between two seemingly similar, but very different sentences. Let's look at another example:

- "Boris Johnson has become the face of the Leave campaign by putting a number on a bus"

Here there are some common phrases that will almost always be used together. The words "Boris" and "Johnson" will very often occur in that order, and always as a pair. For us to be able to classify sentences containing many phrases such as these, additional value is gained when the model is able to more easily infer a singular meaning spanning over multiple individual words.

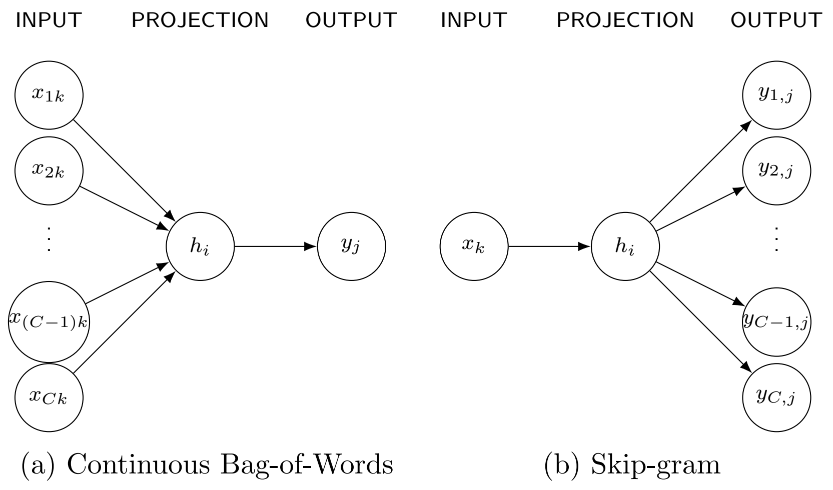
One metric of determining the similarity between two words, represented as vectors is via the cosine similarity, shown as follows:

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{(i=1)}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}. \tag{2.26}$$

The cosine similarity between two vectors $\vec{v}$ and $\vec{w}$ is a popular method in which to able to measure similarity between two vectors. We calculate it by taking the dot product of the two vectors $\vec{v}$ and $\vec{w}$ and then normalise this by the absolute values of each of the two vectors, to make the calculated output invariant to the lengths of each input vector.

There are many ways in which to convert words into a vector space, but one of the more recent and influential developments has been Word2Vec [30], an "efficient method for learning high-quality distributed vector representations thatx capture a large number of precise and semantic word relationships". Word2Vec is actually two different approaches in which to achieve the goal of creating high-quality word embeddings (with some additional hyperparameters that can be tuned for higher accuracy). These two approaches are called Continuous Bag-of-Words (CBOW), and Skip-Gram. Word2Vec creates vector representations that are short (in our studies ranging from 10 to 300 values) and dense (the majority of values are nonzero). This means that we have less weights needed in order to train our model, which decreases the training time of our model, and can often lead to better results.

The intuition behind Word2Vec as a whole is that instead of counting how many times a word occurs in a document (as used in TF-IDF), we instead train a simple logistic regression classifier that is able to give a prediction of how likely a word x is to show up close to a given word w. The word embeddings that are created are actually a bi-product of this process (these are the learned weights of the classifier). CBOW and Skip-Gram both tackle this in slightly different ways, but both achieve the same goal.



(a) Continuous Bag-of-Words (b) Skip-gram

Figure 2.15: Model architectures showcasing the intuition between Skip-gram and CBOW methods of Word2Vec. CBOW aims to predict the current word based on the surrounding context, and Skip-gram aims to predict the surrounding words given the current word.

2.3.1 Word2Vec: Skip-Gram

The Skip-Gram model [30] trains a classifier that aims to predict the surrounding words, given an input word. It does this by maximising the average log probability of words w_{t+j} appearing in a training context of size c surrounding a given word w_t.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-C\leq j\leq C, j\neq 0}\log p(w_{t+j}|w_t). \tag{2.27}$$

Let's represent our input word as a one-hot encoded vector called $\boldsymbol{x}$, which maps to $y_1, \ldots, y_c$. Our learned weight matrix $\boldsymbol{W}$ contains our vector mappings for each word in our vocabulary. The matrix has dimensions $V \times N$, where V is the size of our vocabulary, and N is the size of our hidden layer. We can say that the i^{th} row of the weight matrix corresponds to the i^{th} word in the vocabulary. Our output is a $C \times V$ matrix, where C corresponds to the size of our training context size.

Since we know that the our input vector is one-hot encoded (i.e only a single value in the vector is 1, the rest are 0), we can say that the only element where the weights will be used in the hidden layer computations is the positive element in our input vector. So we can say that if our input vector $\boldsymbol{x}$ has $x_k = 1$, then the following holds:

$$\boldsymbol{h} = \boldsymbol{x}^\top \boldsymbol{W} = \boldsymbol{W}_{(k,.)} := \boldsymbol{v}_{W_I}. \tag{2.28}$$

This says that the outputs of our hidden layer will be equivalent to the k^{th} row of our weight vector W, since this is the only one that will get forwarded through our hidden layer.

Therefore, we can say that our input to the j^{th} node of the c^{th} output word is:

$$u_{c,j} = \boldsymbol{v}'^{\top}_{w_j}\boldsymbol{h}. \tag{2.29}$$

Here, $\boldsymbol{v}'^{\top}_{w_j}$ represents the j^{th} column of our output weight matrix $\boldsymbol{W}'$. We can then plug $u_{c,j}$ into a soft-max function in order to get the probability of the output word w_O, given input word w_I:

$$p(w_{c,j} = w_O|w_I) = \frac{\exp{(u_{c,j})}}{\sum_{j'=1}^{V}\exp{(u'_{j'})}}. \tag{2.30}$$

This tells us the probability that the j^{th} element of the c^{th} output word is equal to the actual value of the j^{th} index of the c^{th} output vector.

However, one issue with calculating the probability in such a manner is efficiency, since the number of calculations needed scales linearly with our vocabulary size V, which can often be very large if we are working over a large corpus. One approach to solve this issues is the use of Noise Contrastive Estimation (NCE), introduced by Gutmann and Hyvärinen [20]. We can simplify this whilst retaining vector representation quality to create negative sampling, which can be shown by:

$$\log \sigma \left(\boldsymbol{v}'^{\top}_{w_O} \boldsymbol{v}_{w_I}\right) + \sum_{i=1}^{k} \mathbb{E}_{w_i} \sim P_n(w) [\log \sigma \big(-\boldsymbol{v}'^{\top}_{w_i} \boldsymbol{v}_{w_I}\big)]. \tag{2.31}$$

Here, the aim is to distinguish the target word w_O from draws from a random noise distribution $P_n(w)$ using logistic regression, where there are k negative samples pulled for each piece of sample data. The intuition is that instead of updating the trained weights for all values in the vocabulary for each input output pair, we instead take a sample of negative words that have their weights updated. This not only rapidly decreases the amount of time needed in which to train the network, but it has the possibility to actually improve the quality of the learned weights in the process.

2.3.2 Word2Vec: Continuous Bag-of-Words

The continuous bag-of-words model follows and almost completely inverted approach to the skip-gram version. Here, we receive a series of one-hot encoded context words $x_1, ...x_C$ with a context window of size C. We output a V (size of vocabulary) dimensional one-hot output vector which corresponds to the word that is most probable, given the input word vectors.

We calculate our hidden layer by taking the average of the input vectors, and taking the dot product of this and our weight matrix $\boldsymbol{W}$:

$$h = \frac{1}{C} \boldsymbol{W} \cdot (\sum_{i=1}^{C} w_i). \tag{2.32}$$

Next, we are able to calculate the input to each element in our output layer as follows:

$$u_j = \boldsymbol{v}'^{\top}_{w_j} \cdot \boldsymbol{h}. \tag{2.33}$$

Here, we can say that $\boldsymbol{v}'^{\top}_{w_j}$ is the j^{th} column of our output weight matrix $\boldsymbol{W}'$. Then, similarly to the skip-gram model, we are able to calculate our probabilities by feeding this through a soft-max function:

$$y_j = p(w_{y_j}|w_1, \ldots, w_C) = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u'_j)}. \tag{2.34}$$

This then gives us our output y_j, which tells us intuitively the probability of a word appearing given a window of size C of input words. Similarly to the skip-gram model, these functions amount to two learned parameters for the continuous bag-of-words model, W (when mapping from input to our hidden layer) and W' (when mapping from our hidden layer to our outputs). In fact, the only major difference between the operations done by the skip-gram and continuous bag-of-words models is that CBOW performs a weighted average step, in order to project to a single N dimensional hidden state. Skip-gram doesn't perform this step, but instead uses the output projection weight matrix W' to map to multiple output words $u_{c,j}$ for the context size C rather than a single output word u_j for CBOW.

Chapter 3

Methods

Political bias detection is a well-explored research area, especially where highly opinionated and short-form texts are within reach [48]. However, there is lack of labelled data for news sites, as many traditional outlets consider themselves to be politically neutral. The UK is exceptional in this matter, and by using the explicit endorsements that were given to either the pro-leave or pro-remain campaigns during the 2016 referendum vote, we can use these as a ground truth in order to classify further data on its political bias, within the domain of pro-leave or pro-remain philosophies. It is possible to use crowd-sourcing methods in order to determine a more fine grained approach on political bias, but this effectively introduces a level of subjectivity into those who are responsible for the labelling, as determining political bias is inherently a subjective task. For this reason, whilst using an endorsement as a representation of all articles within a certain topic area can be construed as an aggressive labelling approach, it removes the biggest potential risk in a study such as this — subjectivity and opinion.

3.1 Word Embeddings

As shown by Mikolov et al. [29], it is possible to model word relationships in a very precise manner by utilising both variations of the Word2Vec algorithm (that is, skip-gram and continuous bag-of-words). However, it is completely possible that our embeddings could be randomly instantiated (as long as we preserve some means of converting an index that represents a word, to it's corresponding embedded vector). It has been shown by Ivyer et al. [25] that initialising with word embeddings created using the Word2Vec algorithm (the skip-gram variant) for political ideology classification yields much higher accuracy than random initialisation methods. This is because we are able to

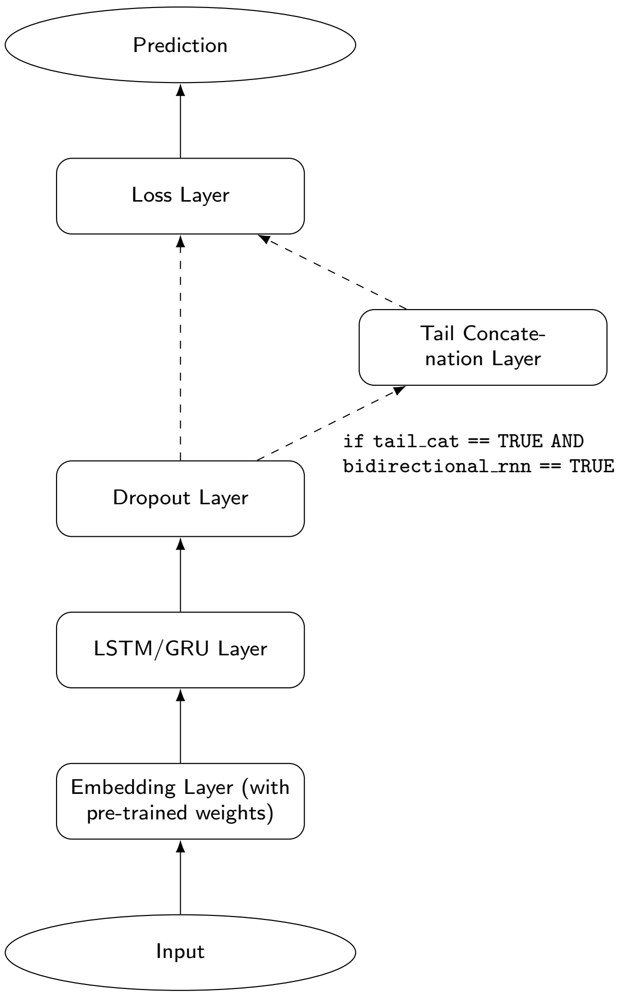

Prediction
Loss Layer
Tail Concate-nation Layer
if tail_cat == TRUE AND bidirectional_rnn == TRUE
Dropout Layer
LSTM/GRU Layer
Embedding Layer (with pre-trained weights)
Input

Figure 3.1: A chart depicting the flow of data through the model. This assumes that the trained word embedding weights are passed in at run-time.

pick semantic features from the datasets more accurately, and doesn't require us to manually define the features of the inputs, as others have done in order to boost accuracy without using more intelligent methods of embedding [12].

With this in mind, we will focus on the Word2Vec family of algorithms for our embedding training. We will create a series of different embedding weights and compare and contrast the effectiveness these in our RNN models. We will tweak a series of hyperparameters in the learning process as follows:

- **Window Size:**
  If we decide to use a larger window, the learned vector representing the vectorised representation of a word is based on a larger context of each

words usage (as shown graphically in Figure 2.15). This means that we might be able to capture important ways in which terms are linked that may not be captured with a smaller window size (i.e regularly used N-Grams where N is a higher value). However, if we set this to be too high, our model may struggle to learn important relationships between smaller quantities of terms (i.e Bi-Grams and N-Grams where N is a smaller value).

- **Minimum Quantity of Word:**
  By decreasing the minimum quantity of term appearances required to be given a term vector representation, we increase the post-embedding step length of our input vector. For example, if we have an input sentence "BoJo has ruined the UK", but the slang term "BoJo" does not appear in our embedding weights, then after embedding out input vector will be consisting of the input sequence "has ruined the UK" (assuming all other terms have embedded representations). This takes away an arguably important part of our input vector, which makes it trickier for our RNN models to be able to learn accurately. But, if we set our minimum quantity to be too low, then we may give our models a lower quality of embeddings that introduces words that have little relevance within our problem domain, and so increasing the length of sequences fed into our RNN layer (which in turn, also increases the potential of overfitting).

- **Quantity of Negative Samples Used:**
  When we are using the skip-gram variant of the Word2Vec algorithm, we have the possibility to improve the quality of our embeddings by feeding in random "noise" samples for each word (alongside the terms in the window) as a showcase of negative samples (this is shown in Equation 2.31). We could increase this (i.e showing more negative samples for each term), but this increases the time taken in which to train our model, and may yield little benefit after a certain quantity of negative samples used.

- **Embedding Vector Size:**
  By increasing the size of the dimensionality of the word embedding vectors (i.e the size of our logistic regression classifier used in the Word2Vec training process), we increase the possibility of modelling more complex relationships between words. However, if we increase this too much, we could make our model slower to train (due to a higher dimensionality of input vector into our RNN layer), and increase the risk of overfitting in out model. If we make this too low, we run the risk of not modelling

the relationships between terms enough, and so resulting in a lower overall quality of embeddings generated.

In addition to manual training over word2vec over our own news dataset, we propose to compare this to a pretrained model supplied by Google [11], which has been trained over the Google News dataset (consisting of 3 million words and phrases). This model is a skip-gram model with negative sampling, with embedding vector size of 300. Since this model was trained over a far larger dataset than our own self-trained variant, it should represent term relationships much more effectively. However, since it is an older dataset (compiled in 2013), it may miss many term specific phrases that are important within the domain of the 2016 EU referendum vote (such as "David Cameron" and "Brexit" are unlikely to have much data in the Google News dataset, but will be very prevalent in our own).

3.1.1 Evaluating Embedding Quality

When creating our word embeddings, we need to formulate a series of criteria that can be used to evaluate the "quality" of the generated embeddings. One such popular method is to utilise the SimLex-999 [21] method of similarity estimation, to be able to gauge a general level of semantic understanding of a model. SimLex-999 is a table of 999 word pairs with a corresponding semantic similarity value, used as a "gold standard" of gauging semantic understanding. The SimLex-999 dataset was collected from 500 native English speakers, when given pairs of adjectives, nouns, and verbs, and asked to rate their similarity. By operating on similarity only, the dataset aims to operate solely on word semantics, rather than association. For example, terms that are highly similar (and also associated), such as "happy" and "dull" receive a high score, of 9.55. However, terms that can be highly associated, but not semantically similar will receive a low score, such as the terms "sharp" and "dull" receiving a score of 0.6. In addition to training loss, we can use similarity measurements between our trained embeddings and the SimLex-999 dataset to formulate the following metrics of evaluating embedding quality:

- **Training Loss:**
  When creating our Word2Vec embeddings, as our model is a shallow neural network, we will be given a loss value for our generated embeddings. We can utilise this loss to give us a rough understanding of the effectiveness of the model, which is operating over the same dataset that will also be used for training our RNN classifier. However, there are caveats to using the training accuracy as a sole metric of embedding quality, such as risk of overfitting (our model may have a record

high accuracy / low loss). This risk of overfitting is further amplified when the small size of the data set is taken into account. For these reasons, we look to additional general metrics to validate the quality the generated embeddings.

- **Pearson Correlation Coefficient:**
  The Pearson correlation coefficient [13] is a value in the range $[-1, 1]$ that dictates the extent in which two variables are (linearly) related. In the context of word embeddings, we take the correlation coefficient between the similarity scores given by the SimLex-999 dataset, compared with the similarity scores given by our own embedding models. The Pearson correlation coefficient can be defined as follows:

$$\begin{aligned} r_{x,y} &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \\ r_{x,y} &= \frac{\texttt{cov}(X, Y)}{\sigma_X \sigma_Y}. \end{aligned} \tag{3.1}$$

  Intuitively, the following equation tells us to divide the covariance between the two samples, by the product of the standard deviation. n represents the sample size (for us, this is the number of word pairs in the SimLex-999 dataset that also appear our trained embedding vectors), x_i and y_i represent the similarity value for each word pairing in each dataset, respectively.

- **Spearman Correlation Coefficient:**
  The Spearman correlation coefficient is a measure of correlation comparison between rank variables. Each raw score x_i and y_i is converted into a rank value $\texttt{rg}x_i$ and $\texttt{rg}y_i$. The Spearman score is then defined in the same way as the Pearson correlation over these rankings;

$$r_{\texttt{rg}_X, \texttt{rg}_Y} = \frac{\texttt{cov}(\texttt{rg}_X, \texttt{rg}_Y)}{\sigma_{\texttt{rg}_X} \sigma_{\texttt{rg}_Y}}. \tag{3.2}$$

  By using the rank rather than the raw variable, the Spearman correlation is able to give a perfect correlation between two samples of data, even if the relationship between them is nonlinear. By taking the rank, we abstract away the relevance of the data values themselves (making the resulting correlation value more resistant to noise in the data), and instead put emphasis on their position relative to other data points.

For this task, we place a higher importance on the Pearson correlation, rather than the Spearman correlation, due to the importance of the difference in actual output values that our model gives us for word-pair similarities. Our relationship is a linear one, for any change in word, we expect that to have a proportional change (of some kind) to another word. However, the Spearman correlation is still a relevant metric to monitor, as it gives us a slightly more noise-resistant method of gauging the relationship between our embedding models and the SimLex-999 dataset.

3.2 Proposed Classification Model Architecture

In our methodology, we propose to slice an entire article into individual sentences, which can each be associated with a label dictating whether it is from a pro-leave or pro-remain outlet (discussed more in depth in Subsection 2.1.3). While it is possible to use only statistical methods to automatically categorise news texts into a political ideology by the properties of the language used [48], this still relies on some labelled dataset with a similar language style (i.e speeches from politicians, political manifestos), in the case of this study we are able to use the EU referendum endorsements as a more stable ground truth, as we are working entirely over the dataset that we wish to analyse. Using the endorsements as a ground truth is also referenced in our research questions, in Section 1.2 as RQ1. Using the endorsements in such way allows us to focus more deeply on the *framing bias* used between newspapers.

Due to the nature of the data that we are looking to analyse, recurrent neural networks appear to be the most natural fit. This is backed up by related studies in political bias classification, most of which also resort to using variants of recurrent neural networks [25] [35]. Due to the potentially large variance in length of input vectors (as we are not looking to clip or transform the input vectors to a certain dimension), recurrent neural networks allow us to easily account for this due to their use of a recurrent hidden state, which is passed through the network for each input x of an entire input vector X. It has been shown that support vector machines are also a viable option of this task due to the faster training time whilst maintaining a high accuracy [12], however this is less ideal for our case due to the high variance in input sequence lengths. It could also be argued that taking a non-neural network based approach would be more suitable for this task, due to the inherently finite amount of articles published in the year leading up to the EU referendum vote (due to neural network architectures traditionally needing high amounts of labelled data to work most effectively). With this in mind, comparing the performance would be an ideal point of research beyond what has been done in this paper, as it would contribute to finding out which model architecture yields the highest accuracy in classifying the political bias of news articles (RQ3 as shown in Section 1.2).

Ultimately, creating a predictor in this way will allow us to feed in arbitrary unlabelled sentences for testing. We could split up an article (or a selection of articles) reporting on the EU referendum campaigns and our model will give us a series of predictions on their stances. We can then aggregate over these to create an average bias weighting for an article, or outlet. This will allow us to answer our research question 2 as defined in Section 1.2, as we will be able to feed in supposedly unbiased outlets, such as the BBC, and we will be given an estimate of their political bias with relation to the EU referendum vote.

3.2.1 GRU & LSTM

Due to our input data being sentences plucked from news articles (i.e each period acts as a delimiter in our data), we have potentially very long input sequences to be fed into our neural network. This puts us at a high risk of suffering from vanishing gradients (as described in Section 2.2.5.2). For us to be able to mitigate this, utilising gates to maintain the relevance of long term dependencies is an important area to compare and contrast. To this end, we propose to use the Gated Recurrent Unit, and the Long Short-Term Memory Unit as focal points for our RNN architecture.

Within this subdomain, we propose to test varying hyperparameter configurations, in an attempt to see which approach yields the highest accuracy, and the shortest period of time in which to train (that is, the shortest time it takes for the model to converge). These tested parameters will be:

- **Batch size:**
  Increasing the batch size would decrease the time taken in which to cover a single epoch over the dataset. However, by increasing the batch size, we have less samples to be used for backpropagation calculations, which can lead to higher overall training times, due to a slower rate of convergence.

- **RNN hidden size:**
  Increasing the hidden size of the recurrent layer increases the potential for modelling complex relationships within the data. However, this also increases the risk of overfitting, which has the potential to become more of a problem as the input sequence length increases.

- **Dropout percentage:**
  Introducing dropout allows us to regulate the flow of information through our network (as shown in section 2.2.4.3). This allows us to decrease the risk of overfitting, but if we set this to be too high, we may begin to drop important parts of our learned model, and so decreasing the overall accuracy of the learned weights.

- **Number of RNN layers:**
  By "stacking" RNN layers, we increase the quantity of learned weights by effectively feeding the outputs from one RNN layer into another RNN layer. This offers a way in which to increase the learning potential without increasing the overall width of each layer. Similar to increasing the hidden size, this also introduces a risk of overfitting in our model. Stacked recurrent neural network architectures are explained more in depth in Section 2.2.5.5.

In addition to these quantity-based metrics of tuning hyperparameters, we will also place a focus on utilising bidirectional RNNs (described in Section 2.2.5.6), to see if this yields better results is representing long term dependencies in our input data. This will also be tested alongside a tail concatenation approach (showcased in Figure 2.14) which serves as a way of taking the most important features of the learned weights at each end of the bidirectional RNN. This, in theory, should be an approach that allows us to even more effectively model long-term dependencies, without needing to increase the hidden size or stacking many different RNN models together. But, this runs the risk of not being an effective approach (due to the traditional gated RNN architectures being sufficient in preserving these long term features).

Next, we will compare varying similar optimisation functions described in Section 2.2.4.2. The analysed algorithms will be SGD, SGD with momentum, Adagrad, Adadelta, RMSProp and Adam. All of these algorithms will be calculating the gradients on minibatches to improve training times, but SGD takes a noticeably more simple approach with a predefined unchanging learning rate, η, whilst the other algorithms are slight variations on ways to adaptively update the learning rate based on the direction and momentum of the loss function.

Finally, we will also compare and contrast the strengths and weaknesses of using the GRU versus the LSTM in this domain. We expect to see similar results due to the similarities in architecture (both described and shown in depth in Section 2.2.5). Due to the slightly simpler architecture of the GRU, we may see that it takes less time to converge than the LSTM. However, the

LSTM may be able to capture long term dependencies more effectively due to it's use of a designated cell state, C_t.

3.2.2 Loss Calculations

Due to the nature of our problem being a binary classification task, we propose to use a binary cross entropy loss, following a sigmoid activation in our network in order to calculate the error for our model (this is shown in Section 2.2.4.1).

3.3 Performance Metrics

Finally, we want to be able to measure the effectiveness of our trained model, we will do this via a number of criteria:

- **Precision/Recall:**
  The precision/recall matrix builds upon the accuracy, but instead gauges the quality of a model based on how often it classifies each case correctly. We can define precision as follows:

  $$\texttt{Precision} = \frac{TP}{TP + FP}. \tag{3.3}$$

  The precision tells us: "out of how many of our positive classifications, what percentage of those were true positives?". This lets us know how often our model is correctly classifying our positive cases.

  We can define the recall as follows:

  $$\texttt{Recall} = \frac{TP}{TP + FN}. \tag{3.4}$$

  The recall tells us: "Out of our positive labels, how many of these were correctly classified by our model?". This tells us how effective our model is at covering the entire search space. We want to find a model that is able to maximise both precision and recall — we can do this by plotting the precision/recall values for each of our proposed models and use this to help us determine the most effective one.

- **Accuracy:**
  When training our model, we will partition our data in to training, validation and test sets (this will be split into 70:20:10 partitions by

ratio). We will extend the precision/recall values calculated and use these to calculate the F1 score for the given model. This is defined as follows:

$$\texttt{F1 Score} = 2 \cdot \frac{\texttt{Recall} \cdot \texttt{Precision}}{\texttt{Recall} + \texttt{Precision}}. \tag{3.5}$$

This is effectively a weighted average of our precision and recall values. This gives us a value which takes into account both false positives and false negatives, giving a more accurate summary of the quality of our model.

- **Time to converge:**
  We will also consider our model to be performant based on the number of epochs needed before our model converges to within a certain threshold value. We care about this because whilst we are currently working over a small dataset (so more epochs does not present much of an issue), we would like this approach to be scalable to much larger sets of data, and so a faster time to converge is a desirable feature.

3.3.1 Comparison to Naive Bayes

In this study, we will also employ the use of a Naive Bayes (summarised in Section 2.2.1) to serve as a point of comparison between a (potentially deep) recurrent neural network and a more simple method of classification. Here we will observe the time taken to train the model, which is expected to be much lower for a more simple method, but we expect the performance of the Naive Bayes to trail that of the more complex model architectures. This is due to the fact that Bayes Theorem (Equation 2.1) makes the assumption that all input features are mutually exclusive. That is, no input feature x_i has influence over input feature x_j. When the input vector X is a sentence of individual words x_i, we know that this assumption does not hold.

Chapter 4

Environment & Implementation

4.1 Data collection

For our training data, we retrieved articles relating to the 2016 UK referendum on its EU membership in a one-year time period leading up to the vote itself (i.e all articles within the dates 2015-06-23 to 2016-06-23). Since the referendum itself was brought into law by the British parliament after the Conservative victory in the 2015 General Election with the introduction of the European Union Referendum Act of 2015 [34], we can assume that all articles pulled from this period of time are not speculative about a vote, but speaking about a concrete referendum vote that will be taking place on a given date.

4.1.1 Tools Used

To collect the articles, a software-as-a-service platform called EventRegistry was used [6]. This service offers a News API that allows for querying of articles from over 30,000 publishers. Within this API, we were able to query for all news articles in English, that were relating to the 2016 UK EU referendum vote (this was based on a "Concept URI" which links to the Wikipedia page for this topic [43]). We also place a filter on our search such that all results are quantified as "news". Next, we are able to specify the sources that articles are pulled from. For this, we associated each media outlet with its equivalent news website (for example, The Daily Mail would be associated with the website `dailymail.co.uk`).

Using these methods, a total of 5232 articles were pulled. Each of these articles was then given a label of 0 or 1 (each denoting pro remain, or pro leave respectively according to the newspapers endorsements given, showcased in

Section 2.1.3). At this point we then pass our data along to be processed and used for training our model.

4.1.2 Flaws & Assumptions Surrounding Data

For this data collection, an amount of trust is placed in our source to supply fair and truthful data. It could be possible that some sources are missed due to how the articles are collected by EventRegistry. Furthermore, we exercise trust in the filters that are used in querying serve their purpose accurately, such that the "news" filter does indeed ensure that all results qualify as "news" and not opinion pieces etc. In the same vein, we are trusting that all articles categorised as ones reporting on the 2016 EU referendum vote in the UK are in-fact reporting on that topic. If such articles were returned from our query that were not on this topic, it could pose a risk to the quality of model that was created.

Another possible issue with respect to our data is a potential lack of it. It could be possible that the utilising neural networks with a dataset of 105024 sentences does not allow for our model to accurately learn the semantic relationships that form framing bias in news articles in this domain, resulting in a low accuracy. An approach to solving this would be to use a different method of machine learning that does not rely as heavily on high amounts of labelled data, such as support vector machines. This could be used as a further point of study (as described in Section 6).

4.2 Data processing

Before the articles can be fed into the neural network, We would need to create the Word2Vec word embeddings so that the network can vectorise each term, to then be fed into the recurrent neural network layer. These embeddings were created using the Gensim utility library for Python [7]. For each of the articles, we would take it's body (i.e the headline would not be used) and we would preprocess it. This preprocessing would remove all accents from letters, lowercase all words, and return an vector of tokens, which can be used for embedding.

Once the corpus is tokenised and embeddings have been created, we can then feed this into the network. The model is created using PyTorch [9] and has been trained locally on a Linux workstation.

Chapter 5

Evaluation & Discussion

5.1 Results

For our dataset, we queried the EventRegistry [6] to retrieve all articles relating to the EU referendum vote, within a one-year timeframe leading up to the referendum date. Exact parameters for the querying can be found in Figure 5.1. We repeated this query for each of our sources (`theguardian.com`, `ft.com`, `mirror.co.uk`, `inews.co.uk`, `independent.co.uk`, `thetimes.co.uk`, `dailymail.co.uk`, `thesun.co.uk`, `express.co.uk`, `telegraph.co.uk`).

After running the following expression for each source, and collating data into a single CSV file, we are left with 5232 articles in total. After each of the articles has been split down into individual sentences and given a class label (either pro-brexit or pro-remain), we are left with a total of 101415 sentences with a label of pro-leave, and 73430 sentences with a pro-remain label. Due to the imbalance of class label distribution in our data, we apply a weight vector to our loss function (explained in Section 2.2.4.1) for positive classifications.

Due to the finite nature of the dataset (as it is not possible to pull more data from further backwards in time, as the referendum vote hadn't been announced at that point) we are left with a lack in data points to be fed into the network. This introduces potential issues such as higher risk of overfitting when training the classifier and lower quality embeddings due to less frequent appearances of certain terms.

5.1.1 Word Embeddings

For each of the models, we trialled a range of hyperparameters from predefined sets, and measured the loss on each combination of possible options. Our parameter ranges are showcased in Table 5.1.

```
q = QueryArticlesIter(
    conceptUri=(
        ‘http://en.wikipedia.org/wiki/’
        ‘United_Kingdom_European_Union_membership_referendum,_2016’
    ),
    sourceUri=sourceUri,
    categoryUri=‘news/Politics’,
    dataType=‘news’,
    dateStart=‘2015-06-23’,
    dateEnd=‘2016-06-08’,
    lang=‘eng’,
    isDuplicateFilter=‘skipDuplicates’,
)
```

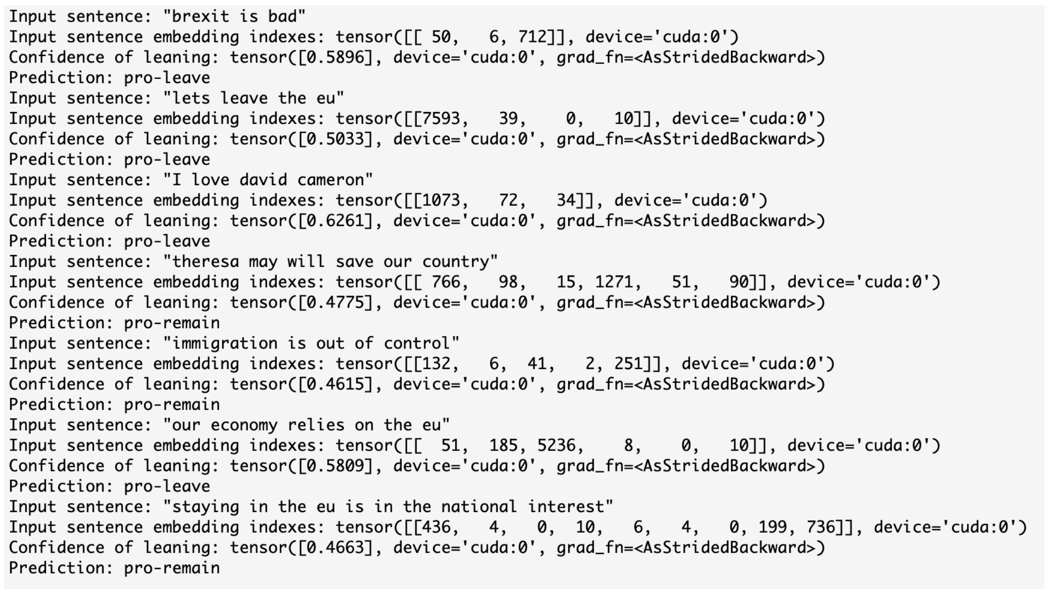
Figure 5.1: The Python expression used for building a query to retrieve news article data on the European Union referendum, within the selected timeframe. The `sourceUri` variable is changed to be for each data source domain. For example, to collect article data from the Daily Mail, we would set `sourceUri` to be `www.dailymail.co.uk`

Each possible permutation was run for ten iterations with a starting learning rate of 0.025, decreasing to a minimum of 0.0001, and the resulting model was saved if it produced a new “record” loss value.

In Table 5.1, we can see that there is a large difference in optimum loss values for each implementation. This is due to the differing loss functions for each variant. When calculating skip-gram embeddings, we calculate loss as the likelihood of a series of zzwords w_{t+j} appearing in a training context of size c surrounding a given word w_t. This is inverted for continuous bag-of-words approach, where we are gauging the quality of the model by its ability to guess a most probable term y given a series of one-hot encoded context words $x_1, \ldots, x_C$.

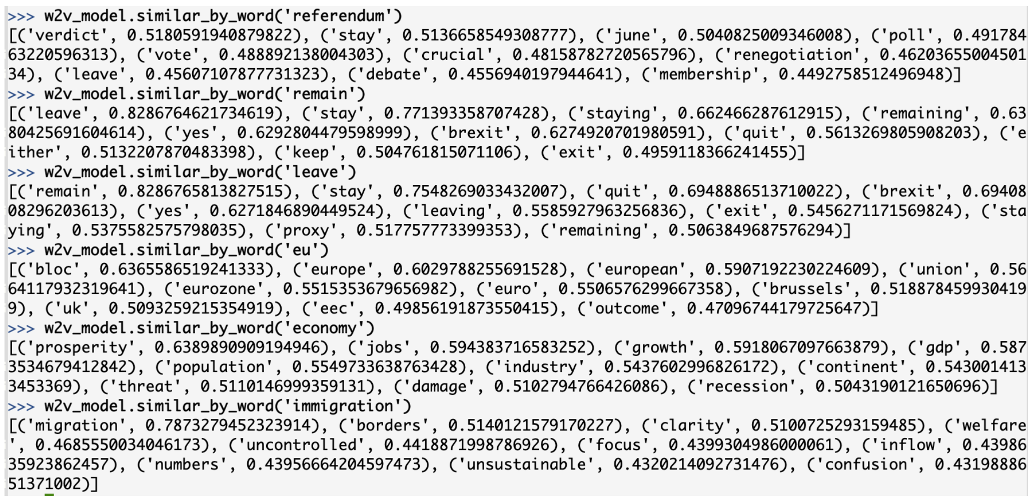
We can also see in Table 5.1 that a similar set of hyperparameters have been selected as optimal by both implementations. Both approaches choose a context window of 3 words to be ideal. This could be due to that the probability of a given word appearing with a chosen series of context words is easier to maximise on a smaller set of context words, given the smaller size of the input dataset.

In Figures 5.4 and 5.5 we are able to see that the best performing models (in terms of training loss) are also the best performing when taking the Pearson coefficient against the SimLex-100 [21] semantic similarity dataset (with



```
Input sentence: "brexit is bad"
Input sentence embedding indexes: tensor([[ 50,   6, 712]], device='cuda:0')
Confidence of leaning: tensor([0.5896], device='cuda:0', grad_fn=<AsStridedBackward>)
Prediction: pro-leave
Input sentence: "lets leave the eu"
Input sentence embedding indexes: tensor([[7593,   39,    0,   10]], device='cuda:0')
Confidence of leaning: tensor([0.5033], device='cuda:0', grad_fn=<AsStridedBackward>)
Prediction: pro-leave
Input sentence: "I love david cameron"
Input sentence embedding indexes: tensor([[1073,   72,   34]], device='cuda:0')
Confidence of leaning: tensor([0.6261], device='cuda:0', grad_fn=<AsStridedBackward>)
Prediction: pro-leave
Input sentence: "theresa may will save our country"
Input sentence embedding indexes: tensor([[ 766,   98,   15, 1271,   51,   90]], device='cuda:0')
Confidence of leaning: tensor([0.4775], device='cuda:0', grad_fn=<AsStridedBackward>)
Prediction: pro-remain
Input sentence: "immigration is out of control"
Input sentence embedding indexes: tensor([[132,   6,  41,   2, 251]], device='cuda:0')
Confidence of leaning: tensor([0.4615], device='cuda:0', grad_fn=<AsStridedBackward>)
Prediction: pro-remain
Input sentence: "our economy relies on the eu"
Input sentence embedding indexes: tensor([[  51,  185, 5236,    8,    0,   10]], device='cuda:0')
Confidence of leaning: tensor([0.5809], device='cuda:0', grad_fn=<AsStridedBackward>)
Prediction: pro-leave
Input sentence: "staying in the eu is in the national interest"
Input sentence embedding indexes: tensor([[436,   4,   0,  10,   6,   4,   0, 199, 736]], device='cuda:0')
Confidence of leaning: tensor([0.4663], device='cuda:0', grad_fn=<AsStridedBackward>)
Prediction: pro-remain
```

Figure 5.2: A showcase of individual test input sentences fed into a trained classifier. Shown is the corresponding indexes for the input sentences, which are used to convert the terms to embedding vectors which are fed into the RNN layer. Then, the generated output is shown as a confidence value (i.e the direct output of a sigmoid activation on the output of the hidden to output linear layer), and a rounded "guess" of this confidence scalar. The following image is taken using the CBOW model with lowest loss, shown in Table 5.1.



```
>>> w2v_model.similar_by_word('referendum')
[('verdict', 0.5180591940879822), ('stay', 0.5136658549308777), ('june', 0.5040825009346008), ('poll', 0.491784
63220596313), ('vote', 0.488892138004303), ('crucial', 0.48158782720565796), ('renegotiation', 0.46203655004501
34), ('leave', 0.45607107877731323), ('debate', 0.4556940197944641), ('membership', 0.4492758512496948)]
>>> w2v_model.similar_by_word('remain')
[('leave', 0.8286764621734619), ('stay', 0.771393358707428), ('staying', 0.662466287612915), ('remaining', 0.63
80425691604614), ('yes', 0.6292804479598999), ('brexit', 0.6274920701980591), ('quit', 0.5613269805908203), ('e
ither', 0.5132207870483398), ('keep', 0.504761815071106), ('exit', 0.4959118366241455)]
>>> w2v_model.similar_by_word('leave')
[('remain', 0.8286765813827515), ('stay', 0.7548269033432007), ('quit', 0.6948886513710022), ('brexit', 0.69408
08296203613), ('yes', 0.6271846890449524), ('leaving', 0.5585927963256836), ('exit', 0.5456271171569824), ('sta
ying', 0.5375582575798035), ('proxy', 0.517757773399353), ('remaining', 0.5063849687576294)]
>>> w2v_model.similar_by_word('eu')
[('bloc', 0.6365586519241333), ('europe', 0.6029788255691528), ('european', 0.5907192230224609), ('union', 0.56
64117932319641), ('eurozone', 0.5515353679656982), ('euro', 0.5506576299667358), ('brussels', 0.51887845993041
9), ('uk', 0.5093259215354919), ('eec', 0.49856191873550415), ('outcome', 0.47096744179725647)]
>>> w2v_model.similar_by_word('economy')
[('prosperity', 0.6389890909194946), ('jobs', 0.594383716583252), ('growth', 0.5918067097663879), ('gdp', 0.587
3534679412842), ('population', 0.5549733638763428), ('industry', 0.5437602996826172), ('continent', 0.543001413
3453369), ('threat', 0.5110146999359131), ('damage', 0.5102794766426086), ('recession', 0.5043190121650696)]
>>> w2v_model.similar_by_word('immigration')
[('migration', 0.7873279452323914), ('borders', 0.5140121579170227), ('clarity', 0.510072529315948), ('welfare
', 0.468555003404617), ('uncontrolled', 0.4418871998786926), ('focus', 0.4399304986000061), ('inflow', 0.43986
35923862457), ('numbers', 0.43956664204597473), ('unsustainable', 0.432021409273147), ('confusion', 0.43198886
51371002)]
```

Figure 5.3: A showcase of some common topic-related individual input words, with an output of the top 10 terms that have the highest cosine similarity (shown in Equation 2.26) with the input. The following image is taken using the CBOW model with lowest loss, shown in Table 5.1.

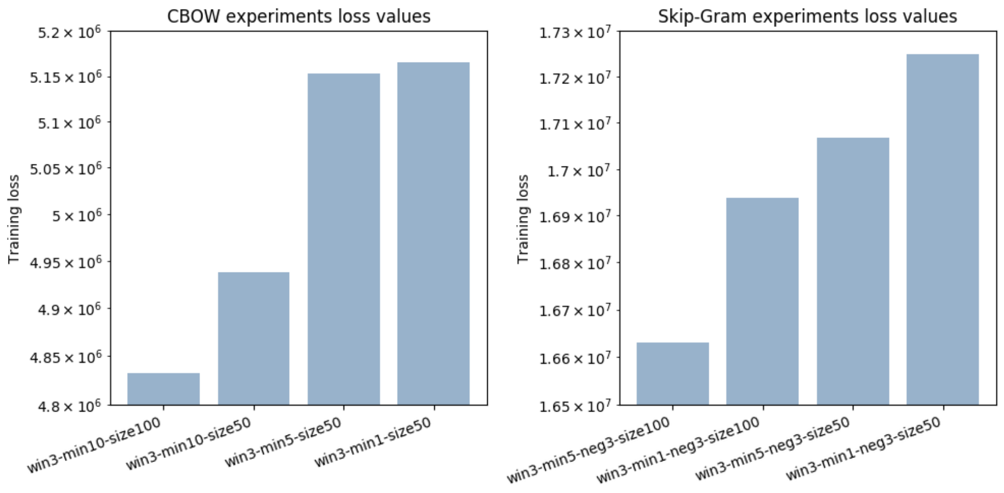


Figure 5.4: Two bar charts representing the final loss value from the four best performing combinations of hyperparameters (that is, the combination of hyperparameters resulting in the lowest loss value) for CBOW and Skip-Gram effectively. The hyperparameters are shown as labels on the x axis, where ‘win’ corresponds to window/context size, ‘min’ corresponds to minimum word count required to be included in embedding, ‘size’ corresponds to embedding vector size, and ‘neg’ corresponds to number of negative samples taken when using negative sampling for Skip-Gram.

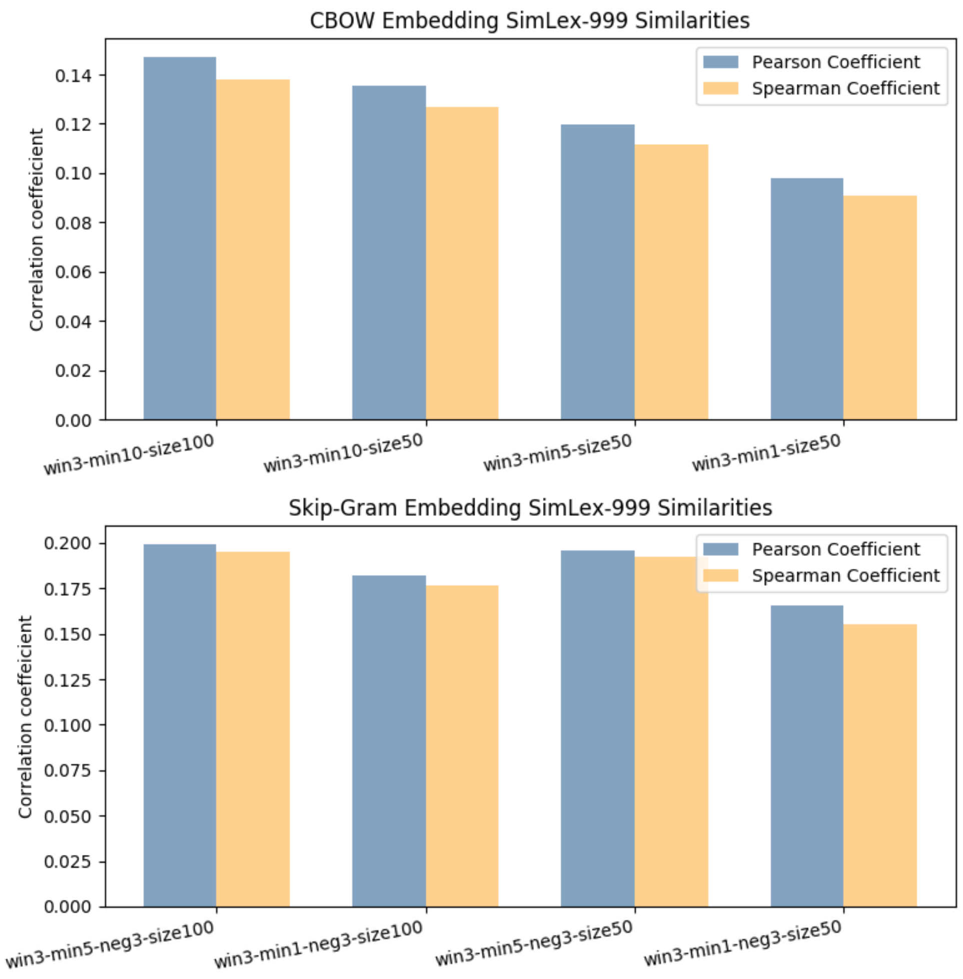
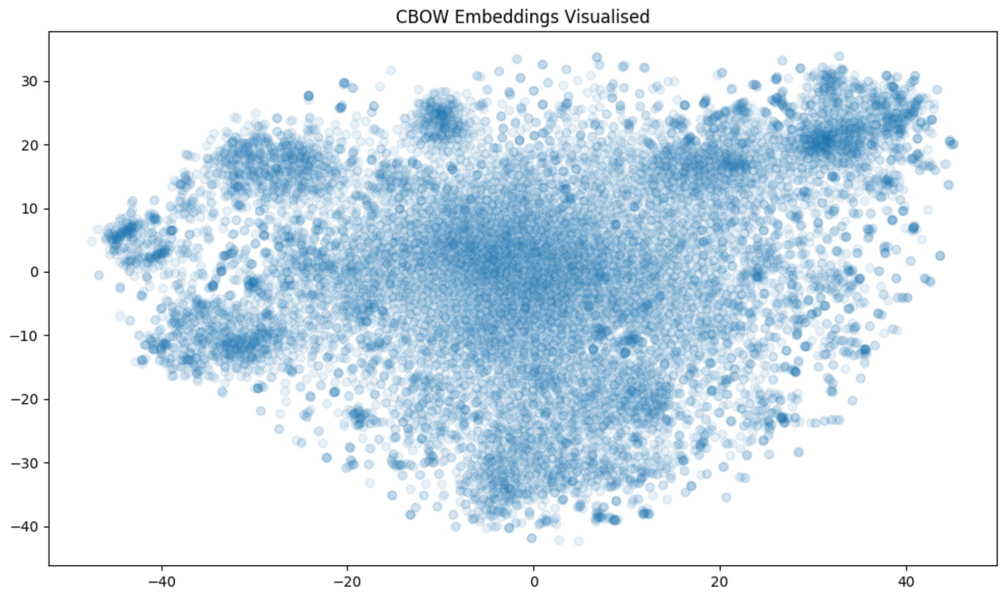


Figure 5.5: Two bar charts representing the SimLex-999 similarities with the four best performing combinations of hyperparameters for CBOW and Skip-Gram over the news article dataset. Similarities are scored by both the Pearson and Spearman Coefficients.



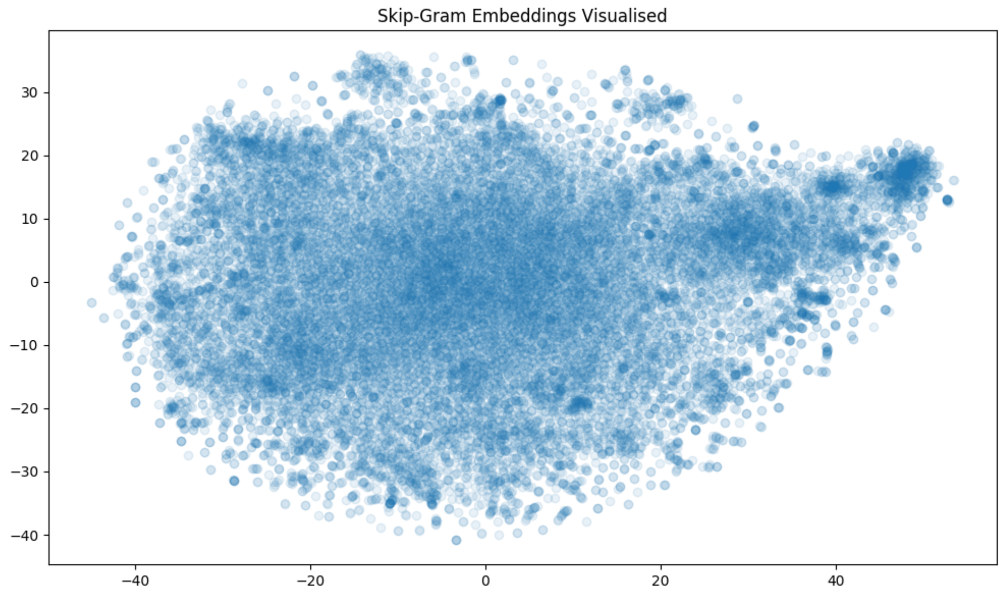


Figure 5.6: Two scatter graphs showing the optimal embeddings for CBOW and Skip-Gram respectively, mapped in a two dimensional space using t-distributed stochastic neighbour embeddings for dimensionality reduction.

Parameters	**Tested Values**
Training Model Type	Skip-Gram, CBOW
Context Window Sizes	3, 5, 7
Minimum Word Counts	1, 5, 10
Embedding Dimension Sizes	50, 100, 300
Quantity of Negative Samples (if Skip-Gram)	3, 5, 7

Table 5.1: All of the possible values that were used in the training process of the embedding weights. Each of the parameters is explained more in depth in Section 3.1.

Parameters	**Parameter Value**	
Training Model Type	Skip-Gram	CBOW
Context Window Sizes	3	3
Minimum Word Counts	5	10
Embedding Dimension Sizes	100	100
Quantity of Negative Samples	3	—
Loss	16629545	4832437

Table 5.2: Combinations of hyperparameters that resulted in the lowest training loss value for skip-gram and continuous bag-of-words, from experiments conducted over differing permutations of parameters shown in Table 5.1

the exception of the `win3-min1-neg3-size100` skip-gram embedding performing worse than the `win3-min5-neg3-size50` embedding, despite having a lower training loss). Due to this linear relationship between training loss and correlation coefficients, we can assume that the embeddings for both skip-gram and CBOW are indeed optimal for this specific domain. Furthermore, compared to the 0.282 Pearson correlation for a Word2Vec model over 150m total tokens using the RCV1 Reuters news corpora [28], an optimal skip-gram result of 0.199 indicates that our model can be considered an acceptable approximation of semantic relationship between general (i.e not specific to the political domain) terms.

In Figure 5.6, we take the optimal embeddings for each Word2Vec variant (shown in Table 5.1) by utilising t-distributed stochastic neighbour embeddings (known as t-SNE), a method of dimensionality reduction, used particularly when attempting to visualise high dimensional data. t-SNE performs nonlinear transformations on different areas of the data, with the use of a perplexity parameter. It does these nonlinear transformations to aim to preserve local structure within the data, meaning that it gives more precedence to relative distance between data points, rather than the overall structure. The perplexity parameter allows for the a level of weighting with respect to local vs. global aspects shown in the data. In the scatter graph for the CBOW embeddings, we are clearly able to see more explicit clustering of similar terms. This tells us that the CBOW approach is more aggressive than Skip-Gram in terms of putting similar words in our corpus into similar embedding vector ranges. This could yield a benefit when using these embeddings for textual classification, as more "opinionated" embeddings may grant additional exposure to certain politically bias heavy features in our training data. However, this may come at the sacrifice of higher quality embeddings (which is reinforced by our SimLex Pearson similarity coefficients across models). The Skip-Gram model shows a much more even distribution of terms across the value spectrum, with only small vague clusters visible in the graph. However, this may be emphasised by the fact that there is more raw data included in the Skip-Gram visualisation, due to the minimum count being only 5 for the Skip-Gram model, as opposed to 10 for the CBOW embeddings. This visualisation shows there is more potential to find complex relationships between terms, but this may be made more difficult be a less obvious term-based clustering of data, which will be used to represent our terms as vectors in our RNN layer.

In summary, in our analysis of creating vector representations of terms in our corpora using Word2Vec has given rise to two sets of hyperparameter configurations that result in the highest quality embeddings (both shown in Table 5.1). Whilst the CBOW embeddings resulted in much lower loss-value at training times, this can be attributed to the different loss functions between each implementation. When checking our best performing embeddings against a "gold standard" method of semantic representation quality, SimLex-999, the Skip-Gram models showed a more accurate understanding of general word relationships than the CBOW trained embeddings. Due to these factors, we choose to take the Skip-Gram model forward as our primary embeddings when training the classification model.

5.1.2 Classification Model

5.1.2.1 Experimentation process

For the experimentation process, a Pytorch [9] model was created that allowed for convenient tuning of hyperparameters. The choices of possible values of hyperparameters was done so in a less structured way than for the training of the Word2Vec embeddings (shown in Table 5.1), due to the sheer number of potential permutations of hyperparameters. Rather, a manual process was employed where a small number of hyperparameters would be changed each time, when training a new model (as shown in Figure 5.7).

```
def train_with_defaults():
    hyperparameters = {
        'epochs': 15,
        'learning-rate': 0.0001,
        'hidden-size': 64,
        'dropout': 0.8,
        'batch-size': 64,
        'num-layers': 2,
        'bidirectional': False,
        'embed-file':
        'model-to-be-tested.model',
        'data-file': 'data/all-articles.csv',
        'optimiser': 'adagrad',
        'rnn': 'lstm',
    }
    model, w2v_model, results = train(*hyperparameters.values())

    # should write to text file hyperparameters
    out_file =
        f'models/{time.strftime("%Y-%m-%d--%H-%M-%S")}--{results["f1"]}'
    torch.save(model.state_dict(), out_file + '.pt')
    logging.info(f'Model trained and saved to {out_file}.pt')
    params_file = open(out_file + '.json', 'w')
    json.dump({**hyperparameters, **results}, params_file, indent=4)
    logging.info(f'Hyperparameters saved to {out_file}.json')

    return model, w2v_model, results

train_with_defaults()
```

Figure 5.7: A sample of code showcasing the invocation of the RNN subroutine with differing hyperparameters. After each training, the model would be saved to a file with a corresponding JSON file containing the hyperparameters and the accuracies over the training, validation and test data for the given model.

Due to this manual hyperparameter tuning process, it is unlikely that an optimal configuration of hyperparameters was achieved. This could perhaps have been improved by employing the use of a program to automatically tune hyperparameters.

For each model that was trained, the F1 score (As outlined in Section 3.3) was used as the primary metric for discerning higher performing models from lower performing ones. However, precision and recall values were also taken into account due to their ability to expose potential flaws in certain models. Finally, while time to convergence was not formally taken into account, hyperparameter values that resulted in lower training times were preferred, such as keeping the number of layers to less than five, and favouring single directional RNNs rather than their bidirectional counterparts. The number of epochs was kept static throughout the training process, as 15 epochs seemed to provide each model enough time to converge around a given set of model weights.

After selecting the four best performing models based on F1 score, these models were further scrutinised by runing predictions over individual news outlets. In this process, we used The Daily Mail and The Guardian as test data, as these each represent (intuitively) pro-leave and pro-remain sentiment, respectively. The BBC was also supplied as an intuitively neutral outlet. For The Daily Mail and The Guardian, the model was given to predict the data from each news source that was also used in training. Since this data was used in training, the model would be expected to quite accurately predict the political leaning of each sentence (if we are to assume that each newspaper produces biased content at a sentence level, that is differentiable). We also then supply a new sentence vector for the model predict containing similar sentences, but over a more recent time period than was used in training. Since this would then be, in effect, "true" test data, this would serve as a point of comparison as to whether the models bias predictions are consistent within newspapers, but across periods of time.

Furthermore each of these models would be compared against a more simple approach to sentence classification. In this case, a Naive Bayes model was created over the same data, to serve as a point of comparison between the RNN models created and a more simple approach. This allows for methodical analysis of the usage of RNNs as a whole for the use case of sentence classification for newspaper political bias.

5.1.2.2 Results

After experimenting with different possible hyperparameter configurations, four RNN models were settled upon as being the best performing based on

F1 score alone. Figure 5.8 shows the results of our core performance metrics for determining the quality of a model across our RNNs, along with results from a Naive Bayes model. Here we are able to see that the model `lstm-rmsprop-h64-l2` produced the highest F1 score across all tested hyperparameter permutations. The hyperparameter configurations for this model (and the other best performing RNNs) can be found in Table 5.3.

It is clear from the results that the high F1 score generated from the `lstm-rmsprop-h64-l2` model is due to the disproportionately high precision (explained more in detail in Section 3.3), as the recall and general accuracy over the test set were the lowest out of our best performing models.

We can also see from the data shown in Figure 5.8 that a Naive Bayes approach is capable of generating similar quality results to a more complicated neural network architecture. These results do not seem to be disproportionately weighted (as seems to be the case with `lstm-rmsprop-h64-l2`). This indicates that the naive bayes approach also creates consistent results when making predictions over test data.

Model Name	**RNN Type**	**Optimiser**	**Batch Size**
lstm-rmsprop-h64-l2	LSTM	RMSProp	64
lstm-adam-h64-l2	LSTM	ADAM	64
lstm-adam-h128-l2	LSTM	ADAM	64
lstm-adam-h64-l1	LSTM	ADAM	64
lstm-adam-h128-l2-drop	LSTM	ADAM	64

Model Name	**Learning Rate**	**Bidirectional**	**Epochs**
lstm-rmsprop-h64-l2	0.0001	False	15
lstm-adam-h64-l2	0.0001	False	15
lstm-adam-h128-l2	0.0001	False	15
lstm-adam-h64-l1	0.0001	False	15
lstm-adam-h128-l2-drop	0.002	False	10

Model Name	**Hidden Size**	**Number of Layers**	**Dropout**
lstm-rmsprop-h64-l2	64	2	0.8
lstm-adam-h64-l2	64	2	0.8
lstm-adam-h128-l2	128	2	0.9
lstm-adam-h64-l1	128	1	0.9
lstm-adam-h128-l2-drop	128	2	0.4

Table 5.3: Above showcases some of the best performing RNNs and their parameter configurations. Each parameter is explained more in depth in Section 2.2. Each model used the Skip-Gram embeddings with the lowest loss value, as shown in Table 5.1

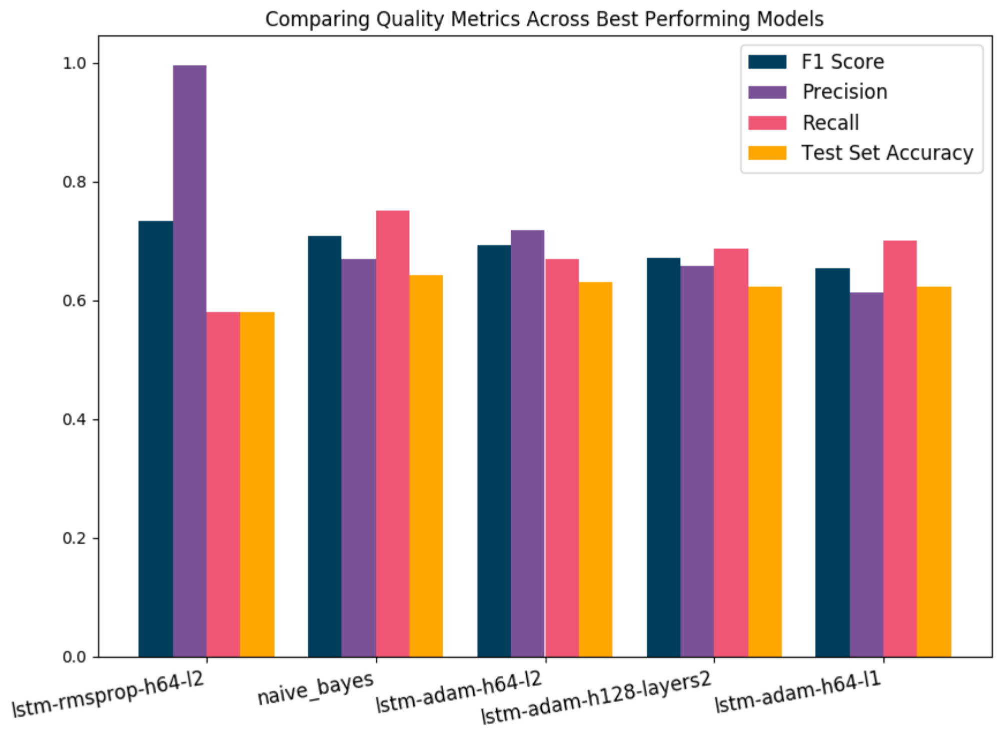


Figure 5.8: A showcase of some common topic-related individual input words, with an output of the top 10 terms that have the highest cosine similarity (shown in Equation 2.26) with the input. The following image is taken using the CBOW model with lowest loss, shown in Table 5.1

Table 5.3 gives the full list of hyperparameters and their given values. We can see that all of the best performing models favoured more simple architectures; The highest quantity of layers in a model across the best performing set is two, and three out of the four best performing models favoured a hidden layer size of 64 (hidden layer sizes up to 512 were tested). However, contrary to this, all of the best performing RNN models favoured the slightly more complicated LSTM over the more simple GRU RNN architecture. While this is counter to the belief that all of the best models were found using the most simple permutations of hyperparameters, it does expose the fact that the best performing models are very similar in architecture, with minimal changes between each. This highlights that results were consistent on the best performing groups of hyperparameter configurations. Furthermore, none of the highest performing models were bi-directional, even after utilising mean-pooling (described in Section 2.2.5.6) in order to more effectively manage long term dependencies in the network.

Model Name	**Guardian**	**Guardian 2019**
lstm-adam-h64-l1	0.4438	0.4658
lstm-adam-h128-l2	0.4487	0.4743
lstm-adam-h64-l2	0.4742	0.4964
lstm-rmsprop-h64-l2	0.5171	0.5175
naive-bayes	0.4138	0.5289

Model Name	**Daily Mail**	**Daily Mail 2019**	**BBC**
lstm-adam-h64-l1	0.5827	0.5831	0.5263
lstm-adam-h128-l2	0.6072	0.6073	0.5457
lstm-adam-h64-l2	0.6054	0.6029	0.5525
lstm-rmsprop-h64-l2	0.5166	0.5171	0.5168
naive-bayes	0.8336	0.7475	0.6581

Table 5.4: The above figure shows the predicted biases for differing media outlets, using the same data used in training, and more recent data, collected in the same way outlined in Section 4.1, but over a one month period in September 2019. A result of 0 would correspond to a unanimous series of pro-remain predictions, while a result of 1 would correspond to a unanimous series of pro-leave predictions.

In Table 5.4 we are able to see the predicted biases of the best performing models. Here we are able to expose more clearly the flaws with the `lstm-rmsprop-h64-l2` model: based on the predictions shown in Table 5.4 it has converged on predicting very similar biases for all input data. This could be due to a low number of positive classifications when training, but out of these positive classifications, many were correct. This would result in a high precision, but would leave the possibility for a low recall (as is the case, shown in Figure 5.8). To this end, we can say that while `lstm-rmsprop-h64-l2` has a high F1 score, it seems to have converged on a local maxima that renders it unusable on anything outside of the test data. In addition to this, this allows us to assume (within the our own trained models) that the ADAM optimiser yields the best results in this case. The usage of adaptive learning rates allows for our training process to more slow down learning rates when approaching a global accuracy maxima. This is reinforced by the fact that the only other optimiser that is present in our best performing models also utilises adaptive learning rates (RMSProp), albeit in a more simplified way than with ADAM which utilises adaptive moment matrices for velocity calculations.

Based on this we can see that `lstm-adam-h128-l2` yielded the best results when giving concrete predictions over real world data (in addition to F1 score alone). This model was accurately able to provide similar predictions of political bias for the same newspapers across time periods. Since this model was the only one out of our top performing models with a hidden layer size of 128 (and with two stacked layers), we could potentially say that the added ability to classify political bias can be down to the create weight vector size, giving the model a greater ability to represent more complex relationships in the training data.

Next, we can see that the Naive Bayes model (summarised in Section 2.2.1) gives strong predictions for the data that was used in training, giving the Daily Mail data used in training a 0.8336 in likelihood of possessing a pro-leave bias. While this is to be expected, the consistent results also hold up on subsequent data that was not used in training. A 0.7475 average prediction for pro-leave bias in 2019 Daily Mail data shows not only that the Naive Bayes model is able to accurately predict biases in data not used in training, but it also showcases that there are bias links that persist across time. This reinforces the idea that we are able to classify news outlets by political leaning, when utilising the newspapers' own explicit political endorsements as a ground truth. However, it was not expected that the Naive Bayes approach would result in a higher F1 score and more consistent results than many of the highest performing RNN models. This is due to the fact that Bayes Theorem 2.1 makes the assumption that all of the input features are independent of each other, which is not the case when working over sentences containing words that each influence each other.

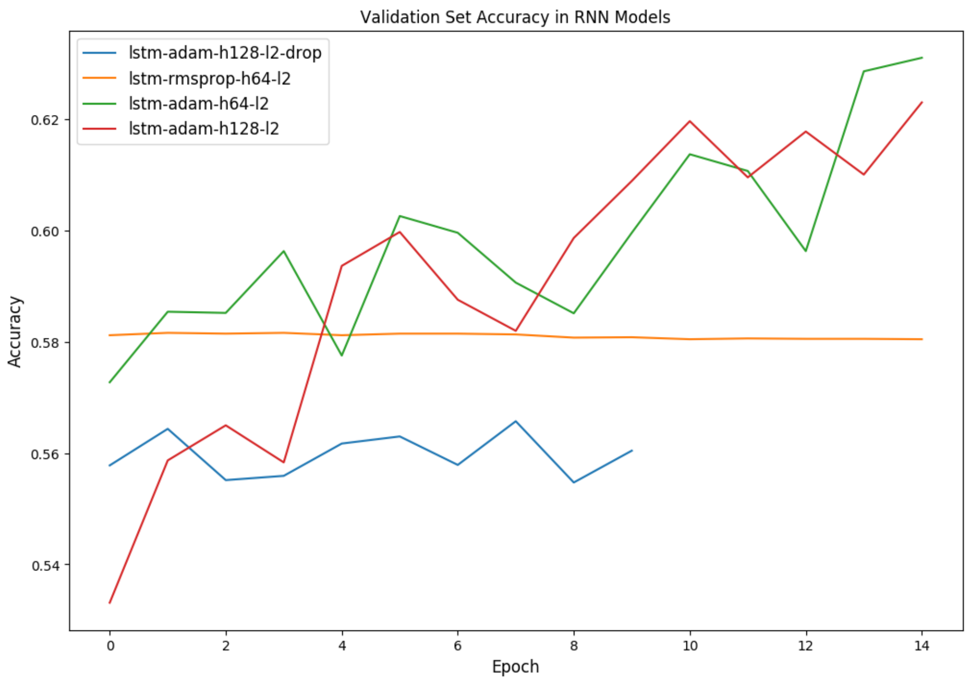


Figure 5.9: The progression of validation set accuracy in a selection of trained models.

5.2 Analysis

5.2.1 Simpler RNN Architectures Are Favoured

In Table 5.3 we show a model named `lstm-adam-h128-l2-drop` which is similar to other highest performing models, but instead has a much lower rate of dropout (at 0.4). In Figure 5.9 We can see the model `lstm-adam-h128-l2-drop` struggle to improve on loss over the test data. However, the training set accuracy for this model increased from 0.5352 to 0.5810 over the course of the training process. This disparity between the training and validation set accuracies show that there was a problem with this model during training. It is likely that this model was prone to overfitting, and so was in need of higher amounts of regularisation.

We can also see in Figure 5.9 that increasing the size of the hidden layer does not necessarily result in a lower loss value over the validation set when training. This could possibly be due to limitations in the data; the combination of a relatively small amount of data (with only 5232 articles used in the training data) and the ground truths labelling applied over large sets of data without pconsidering writing style from the outset gives a higher likelihood of a concrete ceiling in terms of model accuracy. For this reason, a simple model (with a lower number of layers, higher dropout and lower hidden size) could be just as likely to achieve good results, while also being less prone to overfitting. This theory is backed up in Table 5.4 where the models that gave the best predictions over new data were the most simple ones.

5.2.2 Potential Missed Opportunities

Each of the best performing models were manually tested against individual news outlets to assess the quality of their bias predictions. This was expanded by also testing the models against data that was not included in the original training set. This allows us to more confidently extrapolate the findings of a certain model. For example, if a model is able to correctly predict the Daily Mail has a pro-leave bias in datasets from 2016 and 2019, then this gives more credibility to the model. However, the testing of the unbiased news source, the BBC, was only against the data that was included in the training data. For this reason, it gives us less confidence to state concretely that the BBC possesses a bias that is more sympathetic to pro-leave policies and philosophies.

Next the usage of automatic hyperparamter tuning via an external service would have allowed for more accurate optimisation of hyperparamters. For example, the learning rate of 0.0001 remained static throughout much of the training process, due to this being the default learning rate for the ADAM optimiser in the PyTorch library. However, tuning this paramater (amongst others) might have given the opportunity to yield better results. Due to the high number of tunable hyperparameters, this would not be possible manually (or would take a long period of time), but automation of this task would be possible.

5.2.3 Flaws in Results

For one of the best performing models, `lstm-adam-h64-l1`, a dropout of 0.9 is used. However, due to the fact that dropout requires a network with more than one layer to have any effect (Dropout regularisation is described in Section 2.2.4.3) this dropout value is redundant.

5.3 Revisiting Research Questions

In Section 1.2 we posed three questions that we would aim to answer throughout the course of this study. Here we will revisit each RQ and concretely draw conclusions based on the collected data:

5.3.1 Predicting Political Bias in Print Media

All of the highest performing models generated were able to correctly identify the political leaning of a given newspaper, even when given data that is from outside of the time period in which the model was trained. This means that not only are the generated models able to determine political bias in various news outlets, this model can also be used over slightly different sets of data, while still yielding similar results.

Conversely, for The Daily Mail (widely considered to be one of the most explicitly Pro-Leave media outlets in the UK), the best performing RNN model was only able to predict a Pro-Leave bias with a score of 0.6072, where a score of 1 is entirely Pro-Leave, and score of 0 is entirely Pro-Remain. This shows that there are many potential gains to be made with a model such as this. The potential for improvements is also highlighted by the higher prediction scores for the same dataset when using a Naive Bayes model, which gave a score of 0.8336.

Furthermore, it is not possible for us to conclude that since we are able to place newspapers into a binary classification, that this is due to the political bias that is present in each. It is plausible that the differentiating factor between each newspaper is simply down to the different writing styles in each outlet. This philosophy is reinforced by the high scores generated by the Naive Bayes model, which favours a more simple approach (which is not considering long term dependencies and complex relations within sentence structure).

To summarise, we are able to differentiate newspaper articles into a binary classification that corresponds with the explicit endorsements given by each newspaper. This means that it is possible that our models are able to classify sentences based on political bias, but it is also possible that they are differentiating news outlets by writing style and sentence structure, which also aligns with the EU referendum endorsements provided by each outlet.

5.3.2 Predicting Political Bias in Unbiased Media

From the models that were created, each model gave subtle, yet consistent results on the political biases in supposedly unbiased media (in this case, the BBC). The Naive Bayes model gave the highest prediction score of 0.6581 likelihood of possessing a pro-leave bias, and the lowest prediction score of 0.5168 from the `lstm-rmsprop-h64-l2` also indicates a slightly pro-leave bias.

This would imply that from our study, we can say that it is possible that the BBC gave opinions that were more inline with a pro-leave philosophy, than a pro-remain one over the course of the year leading up to the EU referendum vote in 2016. However, the potential issues from the previous RQ remain: it is plausible that the writing style of the BBC is more in line with newspapers such as The Daily Mail, which does not explicitly correspond to a pro-leave philosophy.

Due to the above points, we are able to conclude that while our models are able to indicate a leaning of some kind from the BBC that aligns with a pro-leave philosophy, it is difficult to make the conclusion that this is explicitly due to a political motive that is present in the BBC's print media.

5.3.3 The Best Performing Model for Determining Political Bias

Based on the Figure 5.8 we can see that the highest performing model based on F1 score alone is `lstm-rmsprop-h64-l2` (hyperparameter configuration can be found in Table 5.3). However, as described in Section 5.1.2.2, due to the low precision and test set accuracy, as well as the low concrete performance when predicting political biases over news outlets with explicit political endorsements (Shown in Table 5.4), we can see that this model does not perform as expected in a real-world scenario.

Due to this, the next best performing model (based on our performance criteria) is the Naive Bayes model. The Naive Bayes model has been able to generate a high F1 score, and has a high prediction accuracy compared to the vastly more complex RNN models. This could be due to a lack of training data, which makes the generated RNN models more prone to overfitting, and so they rely on low numbers of hidden layers and high amounts of dropout, which reduces the amount of complexity that can be learned by the networks. Furthermore, the Naive Bayes is able to train in a much shorter period of time due to its simplicity (outlined in Section 2.2.1).

Outside of the Naive Bayes approach, the best performing RNN model is the `lstm-adam-h64-l2` model, which achieved a comparable F1 score to the Naive Bayes model, and was able to correctly predict the explicit political biases of certain newspapers, but was much more expensive to compute than the Naive Bayes, and it didn't give as confident predictions of newspaper biases as the Naive Bayes model (from the data shown in Table 5.4).

Chapter 6

Conclusions

In this thesis we highlighted the need for a way to determine political bias in the traditional British print media. This has been reinforced by studies that showcase the lack of trust [2] and the perceived political bias [8] in the British newspapers. It has also been shown that the print media has potential to have a large impact on the opinions of their readership [42], which highlights that the political bias prevalent in the British newspapers has potentially a very tangible impact on the voting patterns at general elections and referendums.

In any form of print media, some levels of bias are almost certainly unavoidable. This can range from whether the story gets covered by the media outlet at all (known as *selection bias*), to the explicit opinions of the person who wrote the article (known as *statement bias*). In this thesis, we decided to focus on *framing bias*, which is the way in which facts are conveyed to the reader. The words that are chosen, the sentence structure, and the intonation of the sentences themselves have the potential to have a large impact on opinions formed by the reader. We decided to focus on this type of bias as this is not something that is somewhat subconciously processed by the audience. There is an element of inherent trust between the reader and the media outlet in that the facts that are relayed to the user are done so in a trustworthy manner, without being framed in a certain way so as to push a political agenda. Some outlets, such as the BBC, go as far as to avoid potential biases by adding it to their code of conduct [1], pledging to the reader that any information conveyed is done so in a way that will not push them towards certain ideologies or political parties. However, due to the subtle nature of framing biases, it is possible that this is not actually the case. In this paper, we set out to determine if these supposedly unbiased outlets do in fact convey small amounts of bias in the way that their articles are framed. Furthermore, in the openly political biased outlets, can we confirm this bias in the way that their news articles are conveyed to the reader?

Within this domain, we decided to focus on the 2016 referendum of the membership of the UK in the EU as a focal point for determining this political bias, due to the explicit endorsements that newspapers provided, urging their readership to vote in a certain way [27]. This provides us with a ground truth that can be used to confirm or determine political bias in other articles that also reported on this topic. The bias that we set out to detect was done so as part of a binary classification. We would partition all of the articles covered into pro-leave or pro-remain, and use these to perform a classification of either pro-leave or pro-remain on subsequent articles.

To achieve this goal, we turned to recurrent neural networks and variations upon them. It was opted to go in this direction due to the ease of processing sequences of undefined lengths (which would be common in a classification task that works over articles of varying sizes). This decision was also reinforced by their usage in related studies [35] [48] [25]. Furthermore, we decided to focus on the gated recurrent units and long short-term memory units, as these have a proven track record of being able to pick up the major pitfalls of traditional RNN models, more specifically management of long-term dependencies.

We also looked to compare different methods of creating vector representations of terms to be used in our classification tasks. For this we turned to the popular Word2Vec [30] algorithm as a basis for this task. Within this, we sought to compare the effectiveness of the two variants of Word2Vec, skip-gram and continuous bag-of-words. For this we aimed to find a combination of hyperparameters that resulted in the highest accuracy over our corpus of roughly 101415 sentences from EU referendum articles, published in the year leading up to the referendum vote (on the 23rd June 2016).

As a result of our experiments we were able to correctly classify articles as being pro-leave or pro-remain with a highest F1 score of 0.70826. Furthermore, we were able to confirm perceived political bias in certain outlets, such as The Daily Mail having a pro-leave preference, or The Guardian having a pro-remain bias.

Perhaps more controversially, all out of the 5 best performing models predicted the BBC to have a slight bias towards a pro-leave philosophy, with the Naive Bayes model going as far as to a predict a 0.6581 likelihood that the EU referendum-relating articles from the neutral, publicly owned media corporation had a pro-leave bias. However, many of the more complex RNN based models gave less confident predictions, with most giving a bias score in the range of 0.51 to 0.56.

Whilst all of the RNN-based models were able to correctly determine political bias in many contexts, the training process of these models required large amounts of regularisation. Many of the best performing models had high dropout rates of 0.9 to minimise the risk of overfitting. The lack of input data being prone to overfitting also meant that many of the best performing RNN-based models were very simple, favouring few layers, and small hidden layer sizes.

This study sought to utilise more recent developments in machine learning, to pursue whether this would be applicable in the domain of political bias in long-form media text, where few studies have been done in this area previously. However, this study proved that working over a smaller dataset (only 101415 sentences), more simple methods such as the Naive Bayes still are an effective means to perform document classification. The Naive Bayes model created to serve as a comparison point to the more complex RNN architectures actually ended up getting a higher F1 score than most RNN models, and also provided the most consistent predicting of sentence bias, compared to the generated RNN models. In addition to this, the Naive Bayes model took only a fraction of the time taken to train, compared to the RNN model training times. Due to these reasons, it would be advised to continue pursuing more simple methods of determining political bias in long form media, due to the lack of labelled data.

6.1 Future Work

For this thesis we primarily focused on specific architectures, for both creating word embeddings and the classification tasks. For each of these tasks there are multiple alternative approaches to solving these problems.

More recently, there have been developments in creating word representations based on subsets of words, rather than tokenising entire words directly. One such example is ELMo [5] which aims to model complex characteristics of word use, by utilising bidirectional language models. Another example is BERT [16], which also uses bidirectional transformers to be able to create deep representations of terms. Both of these come with pretrained models which could be used in this context, to compare further how effectively transfer learning can be utilised within this problem space.

Furthermore, we make an acceptance in this paper that a certain level of generalising is done in this study, with respect to how we work out our ground truths. Whilst the endorsements that the newspapers provide to a certain philosophy surrounding evens such as the EU referendum vote are explicit, it can be argued that these endorsements are not represented in all articles. It may be that certain journalists at each newspaper inadvertently endorse a certain philosophy in the way that they frame the facts in their articles. We could improve upon these flaws by utilising a more semi-supervised approach to calculating our ground truths, via clustering of articles that are worded in a similar fashion and attributing a ground truth to these. In a similar vein, it would could also be possible to give articles are score of *how* much they lean towards a certain political philosophy (as a probability value on a Bernoulli distribution). This way we would be able to more accurately give confidence scores of a certain political bias, rather than approaching the topic as a binary classification with a type of "all or nothing" result.

Finally, a convenient point of continuation for this project would be to expand the research questions into that of a multi-class classification task. A good example of this would be determining political bias over news articles leading up to general elections in the UK. In this space, there are often a number of different parties that receive endorsements from the traditional newspapers. Since this is often a major point of news for the entire election campaign, many newspaper outlets will provide their endorsements early on in the election campaign process, giving more validity to applying the ground truths in this way.

6.2 Closing Words

In this thesis we were successfully able to create a model that is able to reliably give a correct political bias, when fed arbitrary sentences that are relating to the 2016 UK-EU referendum vote. However, this is not without flaws — it is possible that our model has instead learned the writing styles of various outlets and is able to make predictions based on this, rather than the reported content itself. We also showed that using simple methods such as a Naive Bayes can produce similar results to a more complicated RNN, while taking a fraction of the time to train. Furthermore, we were able to successfully confirm that certain outlets with a perceived political bias in a certain direction by the British public, do indeed write articles that are classified as leaning in that direction by our predictor. We were also able to show that supposedly unbiased outlets such as the BBC, do in fact posess a slight bias towards a pro-leave philosophy, be it in writing style or perhaps in genuine political motive.

In our study, we were able to reach the conclusion that many models studied achieved similar levels of accuracy, with the main difference being in small nuances, such as minor variations in number of layers, or size of hidden layers. Taking all of these performance metrics into account, we were able to show that the LSTM was able to perform best for this task. This is due to the fact that the added complexity of the LSTM allows the model to learn more complex relationships in data, while still being able to manage weights such that overfitting doesn't ruin the resulting model. However, it would be advised for the situation where this task is expanded upon, and used in a more “real-world” context, simpler methods such as the Naive Bayes are used as a base and expanded upon, rather than using complex deep RNN archictectures from the outset.

Bibliography

[1] British broadcasting corporation section 4: Impartiality, 2019. [Online; accessed 9-August-2019].

[2] Britons least likely of 22 nations to trust information on social media, 2019. [Online; accessed 14-August-2019].

[3] Circulation of newspapers in the united kingdom (uk) as of april 2019 (in 1,000 copies), 2019. [Online; accessed 9-August-2019].

[4] Edrm duke law: Coverage bias. `https://www.edrm.net/glossary/coverage-bias/`, 2019. [Online; accessed 3-September-2019].

[5] Elmo – deep contextualized word representations. `https://allennlp.org/elmo`, 2019. [Online; accessed 2-September-2019].

[6] Eventregistry. `http://eventregistry.org`, 2019. [Online; data collected in 2019].

[7] Gensim. `https://radimrehurek.com/gensim/`, 2019. [Online; data collected in 2019].

[8] How left or right-wing are the uk's newspapers?, 2019. [Online; accessed 15-August-2019].

[9] Pytorch. `https://pytorch.org/`, 2019. [Online; data collected in 2019].

[10] Vote leave: The eu immigration system is immoral and unfair. `http://www.voteleavetakecontrol.org/briefing_immigration.html`, 2019. [Online; accessed 3-September-2019].

[11] Word2vec, 2019. [Online; accessed 9-August-2019].

[12] Agarwal, A., Xie, B., Vovsha, I., Rambow, O., and Passonneau, R. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Language in Social Media (LSM 2011)* (2011), pp. 30–38.

[13] BENESTY, J., CHEN, J., HUANG, Y., AND COHEN, I. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[14] CHUNG, J., GULCEHRE, C., CHO, K., AND BENGIO, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[15] DEACON, D., WRING, D., HARMER, E., DOWNEY, J., AND STANYER, J. Hard evidence: analysis shows extent of press bias towards brexit.

[16] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[17] DUCHI, J., HAZAN, E., AND SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research 12*, Jul (2011), 2121–2159.

[18] ENTMAN, R. M. Framing bias: Media in the distribution of power. *Journal of communication 57*, 1 (2007), 163–173.

[19] GRAVES, A., MOHAMED, A., AND HINTON, G. E. Speech recognition with deep recurrent neural networks. *CoRR abs/1303.5778* (2013).

[20] GUTMANN, M. U., AND HYVÄRINEN, A. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research 13*, Feb (2012), 307–361.

[21] HILL, F., REICHART, R., AND KORHONEN, A. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics 41*, 4 (2015), 665–695.

[22] HINTON, G., SRIVASTAVA, N., AND SWERSKY, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on 14*, 8 (2012).

[23] HOCHREITER, S., BENGIO, Y., FRASCONI, P., SCHMIDHUBER, J., ET AL. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[24] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[25] Iyyer, M., Enns, P., Boyd-Graber, J., and Resnik, P. Political ideology detection using recursive neural networks. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2014), pp. 1113–1122.

[26] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[27] Levy, D. A., Aslan, B., and Bironzo, D. Uk press coverage of the eu referendum.

[28] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research 5*, Apr (2004), 361–397.

[29] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[30] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119.

[31] Moore, M., and Ramsay, G. *UK media coverage of the 2016 EU Referendum campaign.* King's College London, 2017.

[32] Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR* (1983), vol. 269, pp. 543–547.

[33] Nunan D, Bankhead C, A. J. Catalogue of bias collaboration: Coverage bias. `http://www.catalogofbias.org/biases/selection-bias/`, 2019. [Online; accessed 3-September-2019].

[34] Parliament, B. European union referendum act 2015. `https://www.legislation.gov.uk/ukpga/2015/36/contents/enacted`, 2015. [Online; accessed 24-August-2019].

[35] Rao, A., and Spasojevic, N. Actionable and political text classification using word embeddings and lstm. *arXiv preprint arXiv:1607.02501* (2016).

[36] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. Learning representations by back-propagating errors. *Cognitive modeling 5*, 3 (1988), 1.

[37] Saez-Trumper, D., Castillo, C., and Lalmas, M. Social media news communities: gatekeeping, coverage, and statement bias. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management* (2013), ACM, pp. 1679–1684.

[38] Scammell, M., and Harrop, M. The press: Still for labour despite blair. *The British general election of* (2005), 119–145.

[39] Schuster, M., and Paliwal, K. K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing 45*, 11 (1997), 2673–2681.

[40] Shen, L., and Zhang, J. Empirical evaluation of rnn architectures on sentence classification task. *arXiv preprint arXiv:1609.09171* (2016).

[41] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research 15*, 1 (2014), 1929–1958.

[42] Wanta, W., Golan, G., and Lee, C. Agenda setting and international news: Media influence on public perceptions of foreign nations. *Journalism & Mass Communication Quarterly 81*, 2 (2004), 364–377.

[43] Wikipedia contributors. 2016 united kingdom european union membership referendum — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=2016_United_Kingdom_European_Union_membership_referendum&oldid=912249913`, 2019. [Online; accessed 24-August-2019].

[44] Wikipedia contributors. Endorsements in the 2010 united kingdom general election — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Endorsements_in_the_2010_United_Kingdom_general_election&oldid=909888211`, 2019. [Online; accessed 14-August-2019].

[45] Wikipedia contributors. Endorsements in the 2015 united kingdom general election — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Endorsements_in_`

the_2015_United_Kingdom_general_election&oldid=909888618, 2019. [Online; accessed 14-August-2019].

[46] Wikipedia contributors. Endorsements in the 2017 united kingdom general election — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Endorsements_in_the_2017_United_Kingdom_general_election&oldid=909889554, 2019. [Online; accessed 14-August-2019].

[47] Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[48] Zhitomirsky-Geffet, M., David, E., Koppel, M., and Uzan, H. Utilizing overtly political texts for fully automatic evaluation of political leaning of online news websites. *Online Information Review 40*, 3 (2016), 362–379.

[49] Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., and Xu, B. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639* (2016).