Department of Computer Science

Submitted in part fulfilment for the degree of BSc Computer Systems (with a year in industry).

Towards a More Accessible Style Transfer Mechanism

Pedro Henrique Machado Wigderowitz

30/04/2019

Number of pages (whole document): 56

Number of words (whole document): 14429

Number of pages (main body): 30

Number of words (main body): 10351

The main body includes everything from the start of the introduction until the end of the conclusion. Words were counted with the MS Word word counter.

Supervisor: Suresh Manandhar

Acknowledgements

Thank you to my supervisor for the assistance throughout the project.

Thanks to Google for offering free high-end computing resources through Google Colab[1].

Thanks to Jack Hessel from Cornell University for his time in helping set up his Python library locally.

[1] https://colab.research.google.com/

Statement of Ethics

We evaluate the ethical considerations for this project against departmental guidelines[2]. We consider whether the techniques and outcomes elaborated in this project could be used to cause harm, but there is no significant risk that would warrant taking action.

Further to the above, the project makes use of freely available datasets containing no personal or confidential data. No data is collected either, meaning that there is no concern about protection of privacy.

[2] https://www-users.cs.york.ac.uk/alistair/teaching/Project%20ethics/Ethics16-17%20slides.pptx

Table of Contents

Table of Figures

Table of Tables

0. Executive Summary

In this report, we tackle the recent phenomenon of style transfer in natural language processing [1]–[5]. As stated in [1], textual style transfer is "the task of rephrasing (…) text to contain specific stylistic properties without changing the intent or affect (sic) within the context".

The applications of this field of research are varied. Mainly, it allows us to build situation-aware applications. For example, in machine translation, the user could be offered the possibility of modulating the politeness of the output in order to suit their needs: if they want to know how to ask someone to be quiet in English, it is probably helpful to output polite-modulated and impolite-modulated alternatives, respectively "*Please be quiet.*" and "*Shut up!*", for instance. This kind of feature is obviously desirable as it assists in the production of naturalistic text.

Politeness modulation as a task has varied usefulness. In some cultures, expression of politeness is deeply ingrained, a fact which is reflected in local languages; such is the case of the Japanese language [6]. In Japanese, variation of lexical features between formal and informal speech is much higher than in the case of English, even warranting literature to be published on the subject [7]. Because of this, it is particularly important for a student of Japanese to be able to learn formality-related nuances so that fluency can be attained appropriately. Our goal in this paper is to elicit a method that allows us to perform effective style transfer with Japanese sentences, modulating the level of formality (or politeness) from informal to formal. This could power a tool that allows easier language acquirement by students.

We approach our objective by first analysing current style transfer techniques. A discussion on their effectiveness and the quality of their result takes place, and we eventually opt to reproduce the methods devised by Prabhumoye et al. in 2018 [1]. The idea is to first reproduce the original experiment, and then modify it to achieve our goal.

The implementation of the mechanism is carried out using the Keras high-level neural networks API for Python [8], on top of the well-established lower-level TensorFlow library [9]. The reason for choosing Keras is that the original implementation of the paper of choice is carried out under the OpenNMT system [10], so attempting a reimplementation with a different software would prove to be a richer learning experience rather than simply running already existing code.

At a certain point throughout the reimplementation process, we provide a discussion highlighting the difficulty in replicating the paper's

proposed mechanism, particularly when it comes to the prohibitiveness of hardware requirements that would allow the achievement of a reasonable result in a realistic time frame. Referring to other work in the area, we consider that there is a notable absence of style transfer methods that are more accessible, presumably due to the assumption that simpler, naïve methods would be less capable in fulfilling the task at hand. We challenge that assumption and shift our focus from reproducing the method from the chosen paper to eliciting our own, aiming to come up with an easy to reproduce, novel, and non-generative technique that can be fully reproduced under publicly available hardware, such as Google Colaboratory [11].

What we develop is a style transfer technique involving a fusion between the doc2vec document embedding model [12], [13] and some of the approaches from [1].

Surprisingly, we do in fact achieve respectable results. Although there is no such thing as a widely agreed upon evaluation process for style transfer quality [14], we follow the process proposed in [1].

We demonstrate that our proposed technique can achieve quality that is at least comparable to that from the work of Shen et al. from 2017 [2], which is the baseline paper that [1] compares itself to. Then, we evaluate the extensibility of our system by putting it up against our initial objective of transferring formality of Japanese sentences. We observe that our model fails at this task as it is very meaning-sensitive, whereas our approach is particularly lacking when it comes to literal meaning preservation.

In this paper, we conclude that simpler non-generative models can, in fact, be more powerful than initially thought when it comes to the task of style transfer. We thus fail in the task of transferring formality of Japanese sentences but succeed in providing a simple and accessible yet effective style transfer model, which we hypothesise can perform quite well under circumstances where literal meaning preservation is not essential. Given our accomplishment, we propose that it could be worthwhile to further investigate non-generative models for style transfer in natural language processing, as there is potential for good performance.

We complete this report with a statement on legal, social, ethical or professional issues, highlighting that there is no reasonable possibility that our work could be modified to cause harm to others.

1. Introduction

In the Japanese language, expression of politeness is much more varied and grammatically established than in most other languages, to the point where we see entire textbooks published on the subject [15]. There are phrasal structures that semi-objectively define the level of politeness of a sentence into particular categories [16], a characteristic that is not found in English. For example, if we take a very simple sentence in English:

The train is coming.

There would be multiple ways of translating it to Japanese according to the desired expression of politeness:

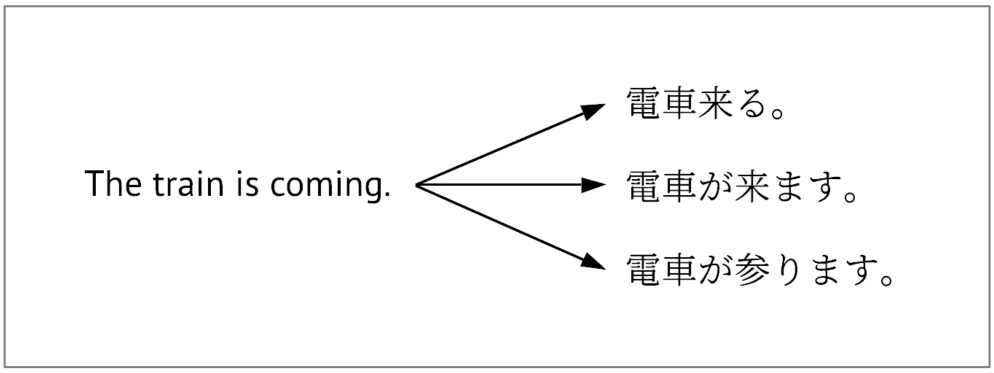


Figure 1.1: A demonstration of a variety of ways in which a phrase in English can be translated to Japanese (manual translation).

In the above figure, one can evidently notice, even without any understanding of Japanese, that the three translations are different. The version on top would be considered plain (*futsūtai*), the middle one would be considered polite (*keitai*), and the latter formal (*keigo*).

The three aforementioned forms do not comprise the entire formality classification system of the language, but in the interest of maintaining the discussion appropriate to the scope of this report, we only consider said three forms. Regardless of the classification system, the important point to note is that a student of Japanese must master the expression of all levels of formality so that fluency can be achieved. If one is speaking to a closer friend, they would probably want to stick to the plain form. In the case of an acquaintance or someone they don't know, the polite form is more appropriate. Most importantly, in a business setting, it is vital to be able to apply the formal form effectively. By not making use of the appropriate level of politeness in a conversation, the naturalness of speech is jeopardised.

In terms of language acquisition, one of the issues with Japanese is that acquiring the ability to communicate naturally can be quite arduous. Most popular textbooks focus on teaching about the polite form since it forms the base of the language [16]. More advanced students may choose to look for literature targeting formal speech, which is not difficult to find [15]. However, content conveying plain spoken Japanese is not popular. One hypothesis that justifies this being the case is that being able to communicate in a formal fashion tends to be essential in most if not all business settings in Japan while being able to communicate in plain form is not strictly necessary when it comes to effectively getting points across; it is rather a necessity when it comes to speaking naturally. Achieving natural speech is important nonetheless and made even further difficult since written and spoken Japanese are frequently so distinct [17], meaning that one must often rely on immersion (which is usually prohibitive) or the consumption of popular audio-based media so that appropriate mastery of the plain form can be developed.

We now consider Google Translate, widely known to be one of the most popular free online translation services, with approximately 4.7 million daily visits [18]. A small selection sentences are translated from English to Japanese, and we manually classify them according to formality:

English sentence	Japanese translation	Formality
The train will come.	電車が来ます。	2
Where are you going?	どこに行くの?	1
What are you doing?	何してるの?	1
I am going to the shop.	私はその店に行きます。	2
Let's go.	行こう。	1
Where is the bus stop?	バス停はどこですか?	2

Table 1.1: A demonstration of Japanese output from Google Translate, given an English sentence as input. Formality is defined by three levels: 1 is plain, 2 is polite, and 3 is formal. Outputs were collected on 25/4/2019.

Besides some out-of-scope issues that compromise naturalness, one immediate observation is that outputs from Google Translate are inconsistent in terms of their formality. This is not necessarily a problem, but rather a waste of potential. If a translation tool with the capability of accurately providing formality-modulated outputs existed, it could serve as an interesting tool for learners of Japanese as well any other languages that suffer from high structural variation in accordance with the politeness setting.

This is the point when textual style transfer comes into play. Style transfer is technique that originated recently as a fusion between computer vision and machine learning, where the task is to take two input images, one with content drawn in artistic style A and another with content drawn in artistic style B, and outputting the former image with the same content, but modified so that the style is transferred to B (or vice-versa) [19]. Even more recently, mostly from the year of 2017, this task was ported over to the domain of natural language processing and has been gaining considerable traction [5]. Textual style transfer, according to Prabhumoye et al., is the task of rephrasing text so that it presents specific stylistic properties while maintaining the contextual intent or effect [1]. One of the classic applications of such is in modifying the sentiment of an input sentence from positive to negative (or vice-versa). We take some highlighted results from state-of-the-art research [20] as a demonstration (1 is positive, 0 is negative):

	Sentence	Style
Input	happy to find this hidden gem near my office. great food and best of all, fast delivery.	1
Output	the restaurant near my office was such a dump. late delivery and gross, cold food.	0

Table 1.2: An excerpt from the results of positive to negative style transfer in [20]. Style 1 is positive, and style 0 is negative.

	Sentence	Style
Input	the bread here is crummy, half baked and stale even when “fresh.” i won’t be back.	0
Output	the ice cream here is delicious, soft and fluffy with all the toppings you want. i highly recommend it.	1

Table 1.3: An excerpt from the results of negative to positive style transfer in [20]. Style 1 is positive, and style 0 is negative.

It is evident that current advancements can be quite capable of handling the modification of sentiment in sentences. As such, we explore the applicability of style transfer to our motivation of developing an English to Japanese translation system capable of politeness modulation.

Firstly, we lay out some cutting-edge research on the field of textual style transfer, briefly summarising and highlighting their methods and results. The selected papers are [1]–[3]. One very important common factor among these papers is that none of their methods depend on having style-specific parallel data. This point, highlighted in [2], [4], is critical since parallel data is scarce; building a model from style specific non-parallel data allows for a high potential of extensibility, which wouldn’t be possible otherwise.

Starting with [1], their work is largely based upon discriminated variational autoencoders (VAEs) as seen in [21], initially introduced in [22]. One novel factor from [1] is that data to be fed as inputs into the

VAE-based model is first back-translated into a pivotal language, with the hypothesis that some stylistic features are suppressed while meaning is preserved [23]. Overall, and in simpler terms, the architecture is structured as follows: a sentence is translated from some original language A into pivotal language B, and the new representation of the sentence is then fed into the VAE. The encoder part of this VAE is the same throughout the entire experiment. Decoders are trained individually and conditioned on a discriminator, always decoding sentences back into language A. The discriminator is simply a previously trained classifier which can determine the style of a sentence. Thus, during training phase, the loss of decoders is guided through the classification loss of outputs, according to the desired style. This allows for building decoders which can output sentences with transferred style.

Moving onto [2], this is an older piece of literature, and the baseline which [1] compares itself against. The method in this paper also bases itself on the basic VAE structure, eliciting a very complex model dubbed cross-aligned autoencoder. As the name implies, an autoencoder model that transfers sentence style from A to B is cross-aligned in training with one that transfers sentences from B to A. The hypothesis backing this approach is that the alignment of hidden states can enable the output to remain within the target domain, which seems to imply that there would be a higher preservation of meaning than otherwise.

Lastly, [3] proposes a very interesting approach, once again following the template VAE architecture, but replacing the classifier discriminator for a language model. This language model is a novelty that assigns high probability to sentences with desired effect, at a level that classifiers are not robust enough to discriminate. It is argued that the feedback that language models can provide to generators (decoders) during training is more stable and useful than that of classifiers.

All individual methods in [1]–[3] are novel and attain significant results in their own right. In terms of complexity, it is trivial to see that [1] is the simplest, [3] is in between, and [2] is the most elaborate. Regardless, we are mainly interested in determining which of the methods has the highest potential. While [3] seems to boast the best results, one issue is that only numerical metrics are provided. There are no tables providing insight on what the outputted sentences look like. This is somewhat suspicious as metrics can be deceiving, leaving the reader not knowing what to expect. On the other hand, [1] provides tables with multiple comparisons against [2], as well as numerical metrics with impressive figures. From these comparisons, we can observe that [1] generally outperforms [2]. Taking this into account, as well as the fact that [1] proposes the simplest method of the group, we elect to make use of it to achieve our previously outlined objectives.

2. Methods

2.1. Overview

Our initial intention is to reimplement the methods from [1] under Python, using the Keras API with the TensorFlow backend (dependencies). The reason why we choose Keras is that, apart from it being a popular and mature API, it exposes some underlying mathematical components to the user, thus enabling a rich learning experience while maintaining brevity and ease of prototyping. While the paper does provide its own implementation online [24], the code is based on the trendy OpenNMT API, which unfortunately is not very useful to us as the syntax is too distinct from that of Keras, added to the fact that documentation is poor. As for [21], from which [1] draws most of its methods from, the situation happens to be mostly the same: the implementation was carried out in TensorFlow [25], and documentation is poor as well.

Within [1], the main experiment is conducted on political slant transfer, and supporting experiments on transferring gender and sentiment also take place. For the sake of brevity, we attempt to reproduce solely the political slant task.

Below, figures are provided to aid in the visualisation of implementation phases:

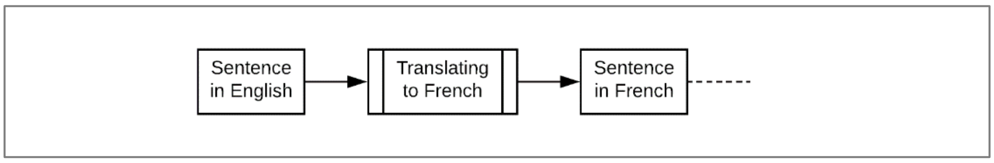
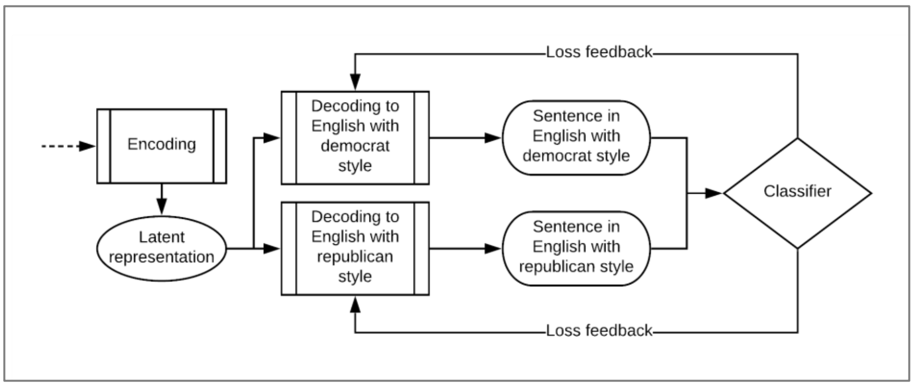


Figure 2.1: A visual representation of the first half of the method from [1]. We see a depiction of the back-translation of a sentence in English into the chosen pivotal language, in this case chosen to be French, and lastly the French sentence being directed over to the latter half (figure 2.2).



Loss feedback
Encoding
Latent representation
Decoding to English with democrat style
Decoding to English with republican style
Sentence in English with democrat style
Sentence in English with republican style
Classifier
Loss feedback

Figure 2.2: A visual representation of the second half of the method from [1]. We see a depiction of the back-translated sentence from the first half being fed as the input of a generative VAE, which is the component responsible for performing style transfer. Decoders are trained such that their outputs are passed through a pre-built classifier which provides loss feedback, allowing the decoder to adapt during training, eventually achieving the ability to output biased sentences that have thus suffered style transfer when compared to the input.

These two figures help us visualise individual components that, when joined together, represent the complete method from [1], as well as the order in which they fall into place.

The first component would be the classifier (figure 2.2), split into four sub-sections: corpora sanitisation, input augmentation through style-specific lexicon extraction [26], the actual training process, and round-up of results.

Next, we have the development of the machine translation system that performs back translation of the input sentence (figure 2.1). This is split into three sub-sections: corpora sanitisation, training process, and reflection. Under this reflection sub-section, we address the VAE and start discussing difficulties in reproducing the chosen paper's technique, consequently distancing ourselves from a faithful reproduction of it. We elaborate on a variety of purposes for our efforts to be shifted towards eliciting a simpler novel method for performing style transfer.

Then, a new section is included where we present a novel style transfer method based on doc2vec [12], [13]. This section is separated into five sub-sections: introduction, hypothesis, motivation, method, and experimentation.

Finally, in our last section, we apply our elicited novel method to the original motivation of modulating politeness in the Japanese language. This section is split into two sub-sections: data preparation, and method application.

2.2. Classifier

2.2.1. Corpora Sanitisation

As previously mentioned, we will take on the experiment of political slant transfer. The corpus used in this experiment was first seen in [27]. In total, it contains 597922 non-parallel sentences obtained from social media, with one half presenting democratic stylistic attributes, and the other half presenting republican stylistic attributes. There is no other sentence-level annotation or labelling apart from “democratic” or “republican”. The criteria for determining the style of a sentence throughout the process of corpus generation is thoroughly discussed in [27].

The data comes pre-split in a 268961:2000:28000 format for each class (training:validation:testing). While some research exists concerning the ideal split of data for training algorithms [28], there is no exact consensus on how to handle this [29]. One rule of thumb is to follow the well-known Pareto principle [30]. In our case, by opting for a 90%:5%:5% split, we end up with approximately 14948 validation and testing samples per class. Therefore, it is more than fair to say that 90%:5%:5% is a safe split in this scenario. The data is thusly re-split into the new proposed format using (file 9).

Then, we consider sanitising the corpus to remove some noise (file 10). Concerning observable noise, we can see that there are traces of HTML encoded characters and emojis, both of which we remove. We also remove punctuation and in-line sentence labels (because sentences of different classes are pre-separated by file).

While no in-depth analysis regarding differences between the original corpus and our final sanitised version is particularly necessary, we do note that total file size is reduced by approximately 16.4%.

2.2.2. Input Augmentation

In [1], it is stated that style-specific lexicons are extracted from the corpora, such that upon feeding inputs to the classifier, any vocabulary that is identified to belong to either the democrat or republican lexicon is duly augmented through the concatenation of a binary style indicator, the motivation being that doing so improves the accuracy of the classifier. It is implied that style-specific lexicons are obtainable through methods found in [26].

We analyse [26], and observe that its aim is to explore a variety of techniques for selecting words that indicate partisanship within some polarising topic. Out of all highlighted techniques, the conclusion is that evaluating z-scores of weighted log-odds-ratios in conjunction with an informative Dirichlet prior produces the best results, and that is, in fact, the approach taken in [1].

While in [26] there is no mention of lexicon extraction, we can see that a z-score cut-off of 1.96 is proposed as a selection mechanism. Thus, all vocabulary with an absolute z-score value measured to be higher than said cut-off could be used to construct a style-specific lexicon.

In accordance, we implement the weighted-log-odds-ratio scoring method with 1.96 as a threshold, in conjunction with both an uninformative and an informative Dirichlet prior for comparison (files 13 and 12 respectively).

At the time of writing, two Python libraries implementing the desired method can be found online [31], [32]. Both are implemented in Python 2, which is inconvenient. The former [31] is provided as a package while the latter [32] is loose code. For our first attempt, we opt to use the former, after converting the source code to Python 3 through a popular online tool [33]. Unfortunately, we run into quite a bit of difficulty when it comes to obtaining the expected output, and no help is obtained after contacting the author. For our second attempt, we try to use the latter implementation, again converting it to the most recent version of Python. After once again running into some difficulty getting the expected results, we contact the author, this time receiving a reply which does help us to achieve the right results. We then package the loose code (file 1) so that we can easily import the desired functions during development.

Running our experiment on top of the political corpus, we quickly find that the backend behind the implementation consumes memory at a very high spatial complexity, such that under a computer with 8 GB of RAM we can only input around 10000 samples per class (total of 20000) without the program crashing. This is an unacceptably small fraction of our data, and we consequently make use of Google Cloud Platform's [34] free incentive so that we may have access to a private server with a high amount of memory. At the time of writing, the server type with the highest amount of memory boasts a massive 624 GB, and with it, we can successfully run the code in under 5 minutes. In the end, we do not find out what is the minimum amount necessary to run our program with the entire 597922 sample size political dataset, but we do find that 256 GB is insufficient. In any case, the results of the uninformed Dirichlet prior version are as follows:

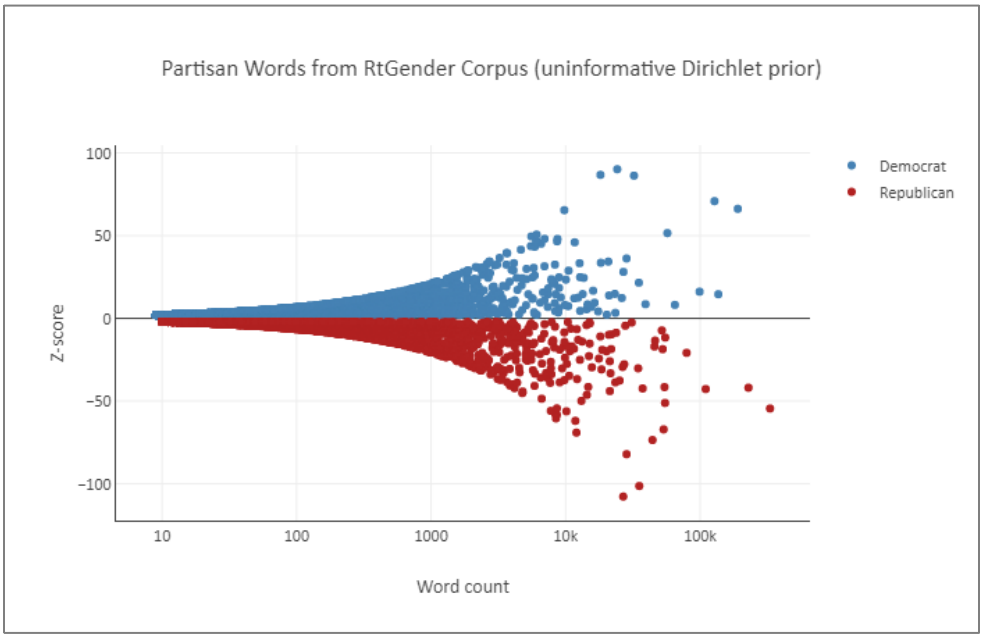


Figure 2.3: Shape of results from implementation of weighted-log-odds-ratio with an uninformative Dirichlet prior over the political dataset.

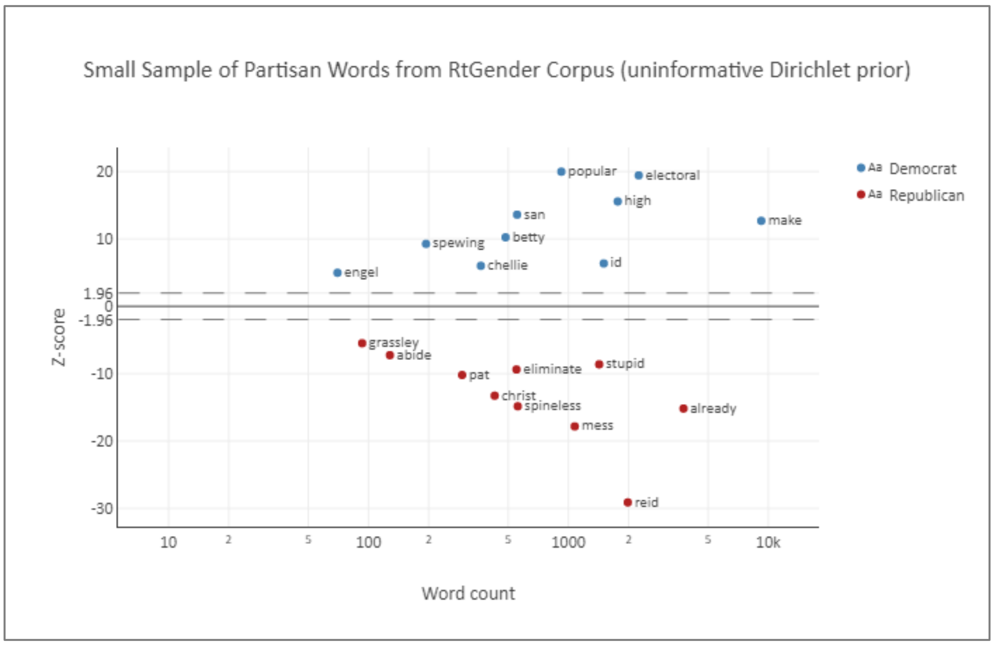


Figure 2.4: Close-up of results from implementation of weighted-log-odds-ratio with an uninformative Dirichlet prior over the political dataset.

It is observable in figure 2.3 that the shape of our results is the same as that of the original publication [26]. One evidently different characteristic is that some scores from our output are much higher than the highest in [26], which could be justified by our data's vocabulary being more partisan, or perhaps because our data is more plentiful.

In figure 2.4, a focused sample of the results is provided, such that the vocabulary represented by data points can be observed. It is important to note our cut-off of 1.96, as no vocabulary scored lower than that is considered to be partisan. We further present tables with the top 10 z-scoring words from each class:

Democrat	
Word	**Z-score**
thank	90
trump	87
senator	86
for	71
you	66
hillary	65
your	52
women	51
gun	50
big	48

Table 2.1: Top 10 z-scoring words that indicate democrat partisanship. Values have been rounded to 0 decimal places.

Republican	
Word	**Z-score**
obama	108
he	101
no	82
our	74
government	69
it	67
congress	62
tax	60
republican	58
constitution	57

Table 2.2: Top 10 z-scoring words that indicate republican partisanship. Values are absolute and have been rounded to 0 decimal places.

Scores from tables 2.1 and 2.2 were obtained with an uninformative Dirichlet prior of 0.05. One noticeable point is that some high-scoring partisan words are unexpectedly quite neutral-sounding. However, they still carry some information. For example, we see that “he” is the second highest scoring republican word. Since “obama” is the word with the highest score, it is fair to assume that phrases uttered by republicans would often make reference to former American president Barack Obama, which in turn means that the pronoun “he” would often make an appearance too, replacing “obama” as the subject.

Our partisan vocabulary is found to contain a total of 7640 words (republican and democrat added together). We also identify that the vocabulary size of our political corpus is of 89131 words. This means that approximately 8.6% of words used in political discourse between republicans and democrats could be used to draw a prediction on the partisanship of the speaker, which sounds like a realistic fraction.

Concerning the informed Dirichlet prior version of our program, we first handpick some common words that have little chance of providing any useful information on partisanship, such as "a", "is", "so", and "an". Then, their prior is set to 1 divided by the number of times they appear in text, such that their z-scores are shrunk during calculation. The prior of other words is left as 0.05. We find that the output contains exactly the same amount of words as in the uninformed example and that the partisan vocabulary is also the same at least until the 50th highest scoring word, in the case of both democrat and republican vocabulary.

We can reasonably conclude that augmenting our classifier’s inputs through z-scores calculated with either the informed or the uninformed method will provide very similar results. Thus, we choose to use uninformedly generated scores for augmentation.

2.2.3. Training

In order to cater for partisan words, we must pre-train our word embeddings so that we may concatenate binary style indicators as per [1] (file 11). The exact way how this is done in [1] is by generating 300-dimension word embeddings, and then increasing them by 2 extra dimensions. These 2 extra dimensions are binary and initialised as 0, where the former of the 2 may be set to 1 if the word in question is found in the democrat style lexicon, and the latter may be set to 1 if the word is found in the republican lexicon. If the word is in neither lexicon, both extra dimensions are left as is.

We train word embeddings with the word2vec function [35]–[37] of the popular *gensim* library for Python. Hyperparameter settings from

[1] are followed, such that our embeddings have a dimensionality of 300, plus the 2 extra bits for partisanship indication.

We are now ready to start building our classifier. Not a lot of information is given in [1] concerning the classifier, but we are told that it is a CNN with 100 filters of size 5 (with max pooling) and that it achieved the accuracy of 92%. Therefore, it seems wise to train multiple classifiers with different architectures in an iterative fashion [38].

There are 4 different architectures that we elicit, on top of which we build 4 individual models. We then assess the results of each model and select the one with the best results.

Our architectures are named “Baseline”(file 5), “Augmented” (file 2), “Kim” (file 3), and “Trainable” (file 4) for ease of reference. We build generic helper files shared amongst all our models such that we can ensure consistency when loading data and saving results (files 6 and 8).

The Baseline architecture is based on Yoon Kim’s findings in [39]. As this is our first attempt, we greatly simplify his proposed method such that we only have a single convolutional layer. Furthermore, as we wish to check whether our pre-trained word embeddings with style indicators do in fact improve the accuracy of our classifier, we do not use them and let Keras train new embeddings from scratch in this case.

The Augmented architecture is the same as the Baseline version, with the only difference being that we do provide our pre-trained embeddings with style indicators, and do not let Keras train them.

The Trainable architecture is identical to the Augmented architecture, with the only difference being that we do let Keras further train our pre-trained embeddings.

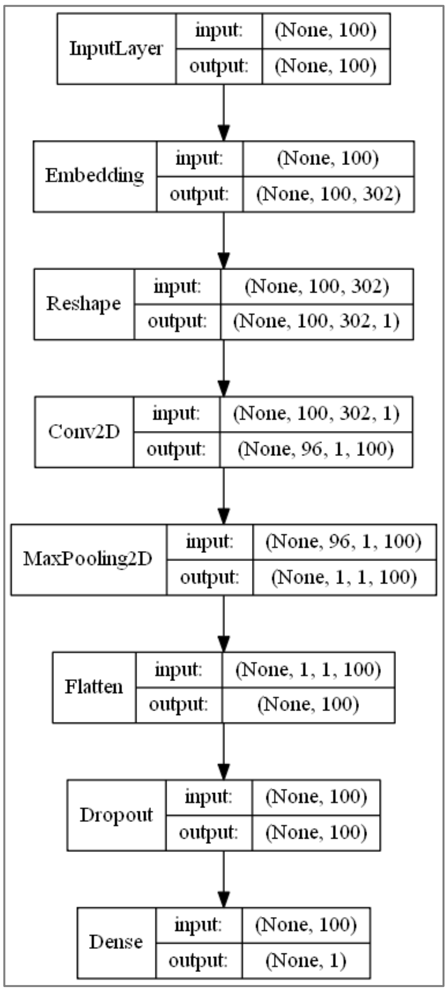


Figure 2.5: Our “Baseline”, “Augmented” and “Trainable” architectures: simplified versions of Yoon Kim’s proposed architecture.

As we can see in Keras' plot of our Baseline, Augmented, and Trainable architectures (figure 2.5), the model has many features which should help us achieve high accuracy, with a 2-dimensional convolutional layer, and good generalisation, with a dropout layer [40]. In terms of hyperparameters, we reproduce those from [1]: the convolutional layer boasts 100 filters and a kernel size of 5. Other hyperparameters are discussed later.

Concerning the Kim architecture, it is a more faithful reproduction of Yoon Kim's work [41] (figure B.1). Like in Augmented, we provide our pre-trained embeddings to it and do not let Keras train them any further. Since we follow the work of Yoon Kim more closely in this case, our architecture boasts 3 parallel convolutional layers, each with their own individual max pooling, which then have their outputs joined together by a concatenation layer. This time, all the convolutional layers' filters were set to 512. In terms of kernel size, one layer had it set to 5, another had it set to 4, and the last had it set to 3. This architecture is clearly more sophisticated than our previous ones, and it could yield better results.

As for so far unmentioned hyperparameters, all models obeyed the following settings:

- Inputs were 100-dimensional (as our longest sentence had about 95 words).
- Embeddings were always 302-dimensional, no matter whether they were pre-trained or not.
- Convolutional layers' weights were initialised with normal distribution, and the activation function was always set to ReLU [39], [41].
- Pooling size was set to 100 minus the pooled layer's kernel size plus 1, and strides were set to (1, 1) [39], [41].
- Dropout was set to 0.5 as that provides the most regularisation [39], [42].
- The output layer was a single dense node with sigmoid activation.
- Optimiser was set to Adam due to its general idealness [43].
- Loss was set to binary cross-entropy as per [1].
- Epochs were set to 10 arbitrarily.
- Batch size was set to 30 as per [41].

Any other unmentioned hyperparameters were left as Keras defaults.

Now, with everything in place, we are ready to start training our 4 models. All classifier training is performed under an NVIDIA GPU with compute capability of 6.1 [44] and 8 GB of system memory, taking no longer than 30 minutes to complete on any occasion.

2.2.4. Results

Initially, after training and testing our models, our findings showed that, generally, testing accuracy was low and validation accuracy was high: approximately, averages were 74% and 99% respectively. This would indicate overfitting, although our training graphs did not accuse such. Additionally, some literature indicates that an accuracy of 74% could be realistic for text classifiers [27], [45]. However, after much debugging, we come to realise that our testing rig is set up such that testing data is not tokenised in the same way our training data was before training our models. After amending said tokenisation, we see testing accuracies much closer to validation metrics. This highlights the importance of ensuring appropriate evaluation of results. See figures below for a summary of our iterative training process:

	Baseline		**Augmented**		**Trainable**		**Kim**	
	Acc.	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**	**Acc.**	**Loss**
Train.	99.83	0.011	98.81	0.039	99.76	0.018	99.59	0.013
Val.	99.51	0.039	99.14	0.028	99.49	0.050	99.33	0.023
Test.	99.51	n/a	99.22	n/a	99.49	n/a	99.28	n/a

Table 2.3: Summary of training results from all classifiers. Accuracies are reported as percentages and rounded to 2 decimal places. Losses are reported as is and rounded to 3 decimal places.

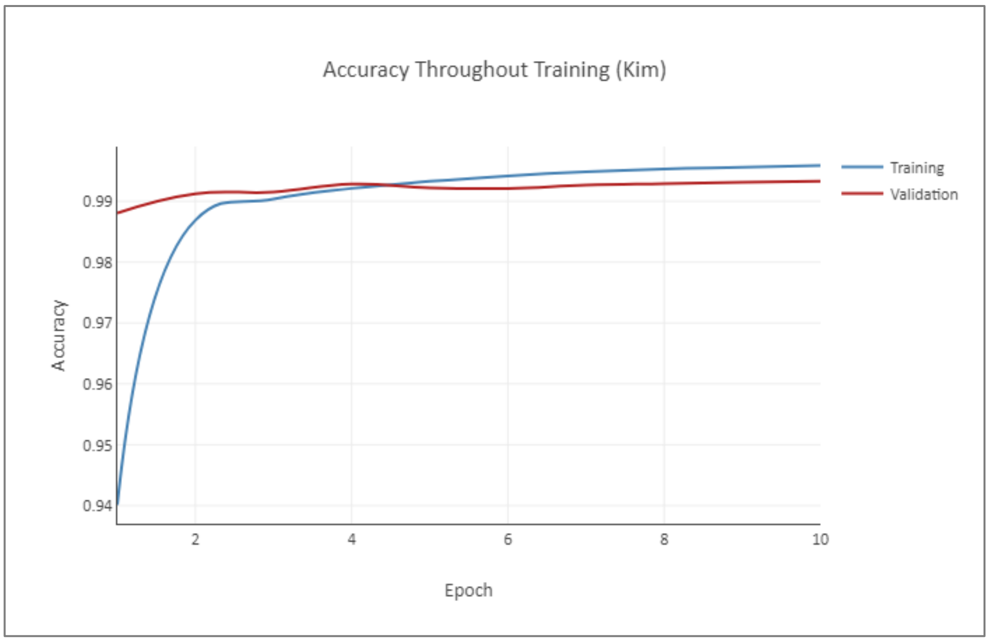


Figure 2.6: History of accuracy seen throughout the training of a model with the Kim architecture.

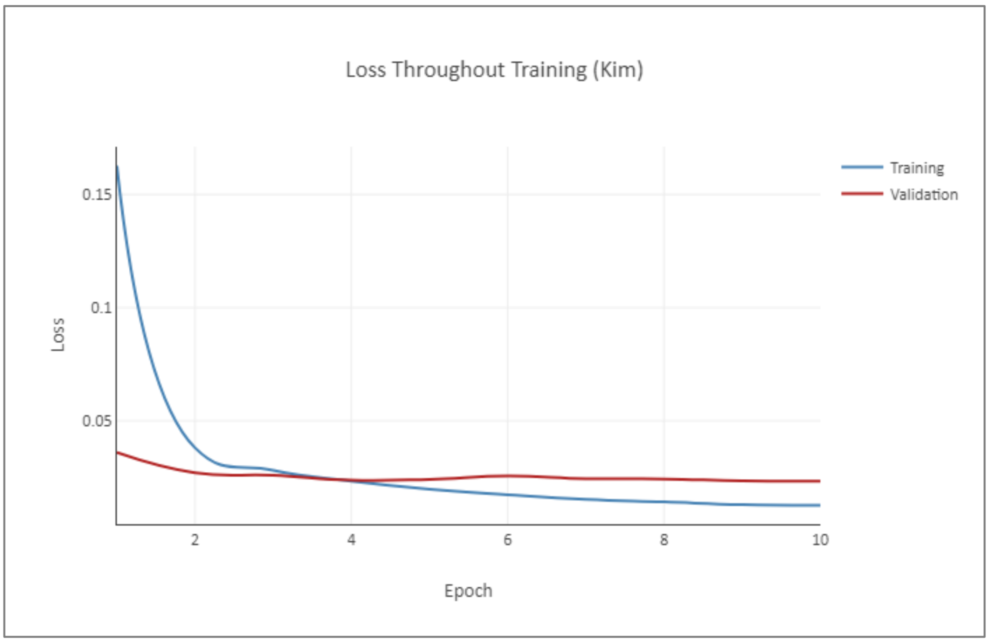


Figure 2.7: History of loss seen throughout the training of a model with the Kim architecture.

By observing table 2.3, it is evident that across the board, classifier accuracy is very high, and in fact, we obtain a substantial improvement over the 92% accuracy from [1].

Graphs displaying the accuracy and loss values of our Kim model throughout training are provided in figures 2.5 and 2.6. We see a typical shape, characteristic of dropout-regularised models [46], where performance figures calculated over the training set are worse than the validation counterparts in the initial iterations. Furthermore, we can conclude that the dropout layer does, in fact, work quite well in the case of the Kim model, as the validation loss is constantly dropping throughout the epochs, with no ups and downs.

Regarding other models, their training histories are provided in figures B.2 through B.7. One pattern that is evident after comparing the histories of the Baseline and Trainable models to those of Augmented and Kim is that pre-trained embeddings considerably support the regularisation of our models. According to literature, this is to be expected [47]. We observe that the Baseline and Trainable models do suffer some overfitting, but seeing as their loss is still remarkably low, any amount of overfitting in this case would not justify branding them as overfit.

The most straightforward choice of model would be Baseline, as it boasts the highest testing accuracy, which is a reliable metric in this case seeing as our class samples are perfectly balanced (since there is no feasible possibility the classifier could be obtaining high accuracy by just guessing). However, the reason for our simplest model outperforming even the most sophisticated one is left unexplained. One hypothesis is that while training and validation datasets used to train our models always individually contained the same data points, they are shuffled before training (file 6), such that our results are not deterministic. What this means is that a favourable data ordering could have been obtained by chance when training Baseline. Yet another hypothesis is that Baseline may simply fit our data better. One fact which supports this rationale is that during training, taking all epochs into account, Baseline presents the lowest loss across all models, with a value of 0.016 at epoch 2. Regardless, testing accuracy variation across the board is quite low, at around 0.3% when comparing the worst performing case to the best. It is fair to say that all models are high-performing and that any of them would likely suit our needs.

Lastly, reflecting on our work on input augmentation under section 2.2.2, our results show that such technique seems to have had little to no effect on performance. However, this is expected in our context, as even our simplest model achieves more than 99% accuracy, meaning that the impact of any further attempted improvement is less visible. In the case of [1], where the classifier accuracy of 92% was achieved by augmenting inputs, it is very possible that the same model would perform considerably worse with simple embeddings, although not much information on this is provided. The effectiveness of input augmentation as per [1] is thus found to be inconclusive.

Taking all discussed points into account, we elect the Baseline model as the ideal model considering our needs.

2.3. Translator

2.3.1. Corpora Sanitisation

According to [1], the machine translation system is built upon data from the Tenth Workshop on Statistical Machine Translation [48]. There are three different English-French parallel corpora which were joined together to form one big corpus: the Europarl v7 corpus, the News Commentary v10 corpus, and the Common Crawl corpus. Combined, these corpora boast more than 5.4M aligned sentence pairs, drawn from a variety of different sources.

Before training our translator, it is advisable that our data is sanitised beforehand, although no mention of such is seen in [1]. Recently, some amount of literature has been published on parallel corpora sanitisation [49]–[51], the common ground between most techniques being removal of pairs that are likely misaligned. This is different from the non-parallel corpora sanitisation discussed in section 2.2.1: in that case, the method is generally non-reductive, whereas in this case, methods are generally reductive. That is, sanitising parallel data will incur removal of some data points.

From the aforementioned literature, it is not completely clear whether the proposal from [50] is language agnostic, so we look for alternatives. In [51], a very robust sanitisation tool is suggested, with its implementation jointly published online [52]. While meritful, this tool is perhaps oversophisticated for our needs, as the setup time may outweigh the quality of the output. This leaves us with the work of Rikters [49], where straightforward yet seemingly effective language agnostic approaches are proposed. Unfortunately, since the contents of [49] were written with Baltic languages in mind, we are not able to use the official implementation [53] without modification, but by basing ourselves on it, we can build our own without much difficulty (file 24).

The sanitisation process is thus established as such: after joining our corpora (file 23), we establish a maximum sentence length and filter out pairs where at least one of the sentences exceeds the maximum (this is not part of the framework from [49]). In our case, that value is set to 50 words. Then, the following major steps are applied:

1. Sentence pairs where both sentences are identical are removed.
2. Duplicate sentence pairs are removed.
3. Sentence pairs where the source or target sentences are duplicates are removed.
4. Sentence pairs where the source or target sentences are not in the specified languages are removed. This is achieved with the assistance of the *langid.py* library [54]–[56].
5. All punctuation is removed, and everything is converted to lowercase.

Lastly, data is split such that 50K samples are reserved for testing, and the remainder is kept for training and validation.

Throughout this process, the largest bottleneck in terms of speed is found to occur because of language detection through *langid.py*: step 4 takes about 30 minutes to complete whereas others take no more than 2 minutes (under an i5-3470 CPU and 8 GB of system memory). One faster implementation of the chosen library does exist, namely *whatthelang* [57], which makes use of certain techniques that allow for faster text classification [58], [59]. However, the *cysignals* dependency of *whatthelang* is not compatible with Windows, which is obviously a considerable limitation. Accordingly, we maintain the *langid.py* library, as a total running sanitisation script runtime of around 40 minutes is definitely not unreasonable, particularly when the size of our data is in the order of millions of samples.

After completion, the sanitisation script leaves us with approximately 4.7M sentence pairs. Our sample size is reduced by roughly 13%, and total file size is reduced by about 22%, both sizeable fractions.

2.3.2. Training

In [1], it is stated that translation models are trained using a sequence-to-sequence framework, as seen in [60], [61]. Our translator is built based on Keras' official sequence-to-sequence learning tutorial [62], which is conveniently based upon the work from [60]. One of the key differences between our implementation and the tutorial is that we implement machine translation at word level, rather than at character level, since character-level machine translation is quite unusual [62], although recent publications have demonstrated how a character-level approach can be quite successful [63].

Following settings as per [1], our model is architected with LSTM-based layers, where encoders are bidirectional. The only specified hyperparameters are the input size of 300, and the hidden dimension of 500. We reduce the input size to 55 to improve training performance since that is more than our maximum sentence length (including start of sequence and end of sequence tokens) as discussed under section 2.3.1.

During initial prototyping, it is found that the total amount of vocabulary in our data is of more than 800K (for either English or French), much more than the vocabulary size of 100K from [1]. While methods for vocabulary compression do exist [64], no such thing is mentioned in [1]. Thus, we populate our word index with only the 100K most common words from our large vocabulary, which intuitively should completely cover the vast majority of sentences from our data. When encoding a word that is not found in our vocabulary, it is replaced with the “unk” token.

Before training, we must address the problem of our data size. Files are already large by themselves, and during training, this becomes a serious problem since numeric arrays of high dimensionality must be generated to form our batches. This means that we must batch our data on the fly, as it would be impossible to hold everything in memory. This problem can be solved through Keras' data generators [65].

Ultimately, we have multiple files in charge of training our model: one loads our data files and generates our vocabulary (file 28), one generates our batch data on the fly (file 27), and one file declares our architecture and trains our model (file 25). See figure B.8 for the final model architecture.

To speed up our training speed, we make use of Google Colab's [11] free GPU-enabled servers. At the time of writing, free access is given to servers boasting a high-end NVIDIA GPU and approximately 12 GB of RAM. While continuous free access is allowed, sessions are broken up into 12 hours each [66], which means that continuous computations that take longer than 12 hours to complete will be interrupted.

As we start to train our model, we can gauge how much data we can use to train it within 12 hours, and what is the maximum batch size we could use during training without exhausting resources. A high batch size of around 256 is desired, as it is suggested to be ideal by current literature [67].

Unfortunately, even when training under Google Colab's powerful servers, we encounter a plethora of issues that are hard to handle without unrestricted access to state-of-the-art hardware. Firstly, as our model is so complex, the amount of trainable parameters (with a vocabulary size of 100K) exceeds 200M. This makes it so that GPU memory gets largely occupied by intermediate operations, leaving less space for data batches. As such, we eventually find that the maximum usable batch size is 20, as we run out of GPU memory otherwise. This is far from the ideal value of 256 and leads to yet another complication: training speed. Reducing our batch size will slow down training, which is particularly undesirable as it is critical that our training completes in under 12 hours (although realistically we wouldn't want our training to take longer than that anyway). Through experimentation, and given our maximum batch size, we regrettably find that if we restrict ourselves to approximately 7% of our data, it is possible to train our model for only a single epoch within the time limit, which is obviously unsatisfactory.

2.3.3. Reflection

Following up on the previous discussion about the training of our machine translation model, we highlight complications behind the development of the custom VAE from [21], which would be the final step to complete the reimplementation of [1].

Firstly, while there are some resources on how to build text-generating VAEs with Keras [68], none of them employ the same approach as in [21]. It is unfortunate too that the official implementation of [21] is based on TensorFlow [25], which makes interpretation difficult. At the time of writing, no unofficial Keras implementation exists either. Although directly interpreting the complex methods from [21] could be quite difficult, we still make an attempt (files 26 and 22), but our approach fails as there is no way to evaluate predicted values in Keras in real-time with the TensorFlow backend [69], a feature which our approach depends on.

Further to the above, even if we did successfully build a VAE, we would once again run into difficulties with its training due to prohibitive hardware requirements. This is further illustrated by the fact that even running a toy example like Keras' sequence-to-sequence tutorial [62] with all available data (160K samples) is not viable under Google Colab, which provides high-performing servers.

Our experience leads us to reflect on whether it would be possible to perform textual style transfer through simpler means, in particular ones that wouldn't necessarily require cutting-edge hardware to achieve their full potential. This draws us to more naive methodologies that are nevertheless worthy of being explored.

Since success criteria in style transfer are not necessarily dependent on meaning preservation [1], we hypothesise that overlooked non-generative methods could obtain success in the task too. In the next section, we elaborate on this intuition and describe an accessible and novel non-generative style transfer technique, which we then apply to the political slant transfer task of [1].

2.4. doc2vec

2.4.1. Introduction

Paragraph Vector, also known as doc2vec, is "an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents" [12]. It is an extension of the very popular word2vec model [36], the only difference being that an additional document vector is trained in conjunction with word vectors. This is a simple but powerful modification, as the document vector can act as the topic of the vectorised document, or as a memory that remembers what is missing from the current context [12], which allows for more robust embeddings and the potential of applications that would not be possible with just word2vec.

We can see in the evaluation experiments conducted in [12] that the doc2vec approach produces state-of-the-art results when it comes to performance of information retrieval (finding the most similar document), as well as generating pre-trained embeddings that improve results of supervised learning tasks, among other scenarios.

2.4.2. Hypothesis

Our hypothesis is very simple. The size of available sentiment analysis data currently is already quite large, and taking into account the ongoing data growth phenomenon [70], it will tend to grow. Thus, having a corpus in hand, it is not unreasonable to assume that given some input sentence, it would be possible to find samples within the corpus with structure and meaning very similar to those of the input.

Referring to the task of political slant transfer, if we take the following democrat-slanted sentence as an input:

Vote Bernie 2016!

Then either of the two following republican-slanted sentences could be seen as similar to our input:

Vote Trump 2016!

Don’t vote Bernie!

These two examples are identified as similar because at least two out of three words in each of them are present in the original sentence. In being similarly structured, they have similar meaning, and thus these sentences can also be considered as style-transferred versions of the input.

Thus, the general idea behind this hypothesis is that if we wish to transfer the style of some text from style A to B, and we have a corpus in hand with sufficient style A and style B text samples, we can then take from our corpus a sample of style B that has similar structure to the input text and select it as our output.

With the doc2vec approach, the process of finding similar documents or sentences within a list of documents or sentences is quick and effective, enabling the application of our hypothesis.

2.4.3. Motivation

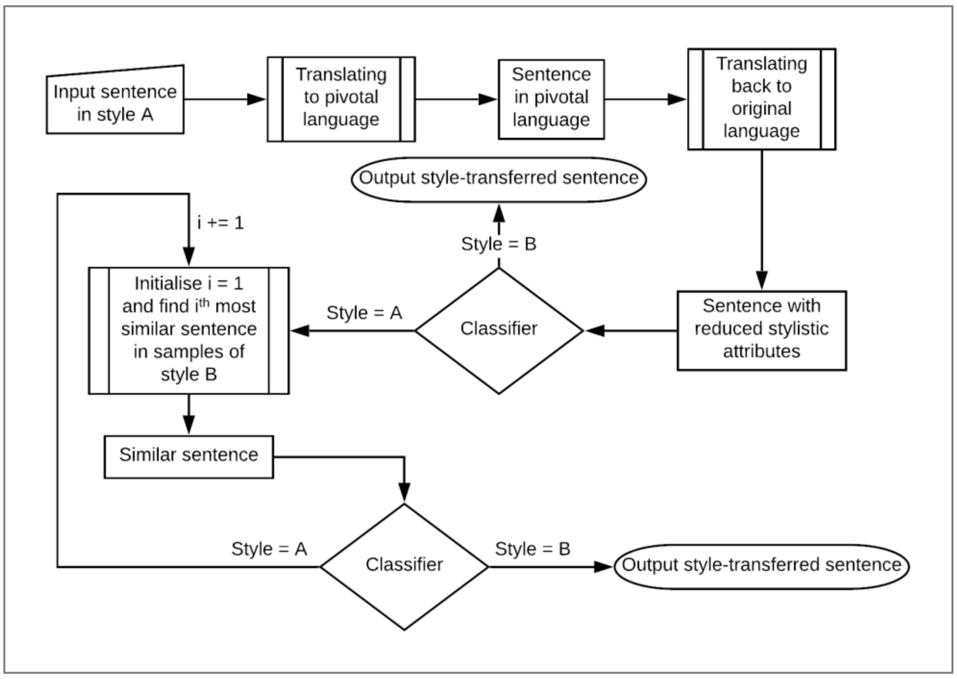
As discussed under section 2.3.3, the main motivation behind this approach is that the application of most if not all current style transfer techniques calls for prohibitive hardware requirements and long training times. We propose a new method in hopes that it would achieve reasonable performance with the advantage that no extensive training nor extremely powerful hardware are required, in order to provide an accessible alternative in a scenario where data is plentiful but only ordinary hardware or public platforms such as Google Colab can be accessed.

In addition, considering the result evaluation procedure from [1], success of style transfer is measured on three components: style transfer accuracy, sentence readability, and preservation of meaning. Our approach will maximise the performance for the first two components, which is quite beneficial. What remains to be found is whether the level of preservation of meaning will be up to standard (results are discussed in section 3).

It is worth noting too that the point of this proposal is not to suggest that non-generative methods have more potential than generative ones, but rather that they have untapped potential, such that good results could be still produced without having to worry about the inaccessibility often associated with the training of deep generative models, and that more research on non-generative models for style transfer could be warranted.

2.4.4. Method

Our method takes inspiration from techniques seen in [1], integrating them with doc2vec to form our own unique approach. First and foremost, we provide a flowchart depicting the proposed procedure:



Input sentence in style A
Translating to pivotal language
Sentence in pivotal language
Translating back to original language
Output style-transferred sentence
i += 1
Style = B
Initialise i = 1 and find i^th most similar sentence in samples of style B
Style = A
Classifier
Sentence with reduced stylistic attributes
Similar sentence
Style = A
Classifier
Style = B
Output style-transferred sentence

Figure 2.8: A flowchart illustrating our proposed textual style transfer method. Styles A and B be can be swapped according to the desired outcome.

As is observable in figure 2.8, the first step of our method is to back-translate the input sentence through a chosen pivotal language. Following the rationale of [1], the intent is to choose a pivotal language close to the original, as the quality of back-translated sentences would be worse if a very distinct pivotal language is chosen, like choosing Chinese as the pivotal language when the original language is English. To further ensure back-translation quality, we may also use well-established publicly accessible translation systems, such as Google Translate.

After back-translation, we classify the new sentence, which should now present reduced stylistic attributes [1]. If the outputted class is representative of the desired style, we take the new sentence and provide it as our output. This is, however, an unlikely outcome. In the more plausible case that the outputted class is representative of the style of the original sentence still, we then vectorise the back-translated sentence and find the sample most similar to it in the corpus of samples belonging to the desired style. Said sample is then taken and classified to ensure it does, in fact, exhibit the desired style. If it does, we provide it as our output. If it does not, we repeat the process where we look for a similar sample, this time finding the second most similar sample, repeating as necessary until the classifier determines that the sample exhibits the desired style.

Evidently, the success of this method depends on the breadth of our data, as it expects that the data contains a sample similar enough to the input.

It is worth noting too that while figure 2.8 provides support for a binary style transfer task, the model can nonetheless be easily extended to provide support for a multiclass scenario.

In the next section, we reproduce the political slant transfer experiment from [1] using the proposed approach instead. We also discuss our choice of hyperparameters as well as some other nuances.

2.4.5. Experimentation

Concerning our classifier, it is built and trained exactly like the Baseline model from section 2.2.3 (files 14 to 17). Political data is also sanitised as per section 2.2.1 (file 19). The only difference is that in this case, we reserve 100K samples per class for training and validation, leaving the remaining samples for testing and style transfer. That is, samples from training and validation sets are not considered as style-transferred candidates, the reasoning being that if they were, the chance of misclassification is considerably higher compared to sampling style-transferred candidates only from the testing set since the classifier will tend to be biased towards samples seen during training. Because of that, it is essential that we ensure that we have as many testing samples as possible. Therefore, our final split per class, in this case, is approximately 90K:10K:199K (file 18).

After training and evaluation, we find that our new classifier performs quite well, with performance comparable to the results presented in section 2.2.4. See table 2.4, where metrics have been calculated and presented in the exact same way as table 2.3 (figures B.9 and B.10 show training history). In addition, we calculate the average prediction confidence of this new classifier. Bearing in mind that our outputs are sigmoid-activated and that label 1 represents democrat style while label 0 represents republican style, our averaged outputs over democrat testing samples is 0.9908, and that of republican testing samples is 0.0059, meaning that our classifier has very high prediction confidence. We can use this to our advantage to ensure better sentences are selected during style transfer.

	Acc.	Loss
Train.	99.86	0.006
Val.	99.34	0.041
Test.	99.35	n/a

Table 2.4: Performance metrics of the new classifier.

With the classifier out of the way, we train our doc2vec models with the *gensim* library for Python [13] (file 20). An official tutorial is followed to ensure best practices are followed and ideal hyperparameters are chosen [71]. The models are trained individually on our test data on a class basis, such that one model is trained only with democrat-labelled data, and the other with only republican-labelled data. Training runs for 20 epochs, generating 300-dimensional vectors, and there are two caveats: the first one is that learning rate is initialised to 0.025 and manually controlled throughout training, such that it is linearly decreased until the minimum value of 0.0001 is reached on the final epoch. The second caveat is that our data is shuffled in between epochs. Both of these are suggestions from [71]. Training is configured to use 4 workers and completes in less than 10 minutes under an i5-3470 CPU and 8 GB of system memory.

Onto inference of style-transferred sentences, we basically follow the flowchart in figure 2.8, but again with some nuances (file 21). Firstly, as suggested previously, data is back-translated with Google Translate, through the *googletrans* Python library [72]. Back-translations are sanitised in the same fashion as our corpus was. Then, when we have them ready to undergo style transfer, we use the model and corpus of desired style to find the most similar sample. When inferring vectors of back-translations, we set the learning rate to 0.025, and infer over 20 epochs. This is very important because vector inference in doc2vec is not deterministic, so inferring over multiple epochs virtually guarantees consistent results [73]. Moreover, when selecting from the vector's most similar sentences, we ensure that two further criteria are met: the length of the selected sentence must be longer than half and shorter than double that of the back-translation (both word-wise and character-wise), and the classifier must have high confidence in the class of the selection (output of 0.95 or more when transferring to democrat style, and 0.05 or less in the case of republican style).

Following the aforementioned hyperparameters and customisations, we achieve results of reasonable quality (this is discussed in section 3). In any case, the method can be configured according to personal discretion, such that the desired level of leniency or restrictiveness can be achieved.

It is worth noting that, in general, we find that back-translated sentences do lose stylistic attributes sometimes as confirmed by classifier prediction confidences, satisfying the hypothesis from [23]. In any case, this is generally not enough for the classifier to recognise them as style-transferred, that is, for them to be classified as democrat when they were originally republican and vice-versa.

In the next section, we apply our doc2vec method to our initial motivation of politeness modulation in the Japanese language.

2.5. Politeness Modulation in Japanese

2.5.1. Data Preparation

Having the objective to modulate politeness in Japanese through style transfer, it is imperative that we have data containing spoken Japanese sentences with varying levels of formality (we cannot use written corpora because of the findings from [17]). While no corpora exist detailing sentence formality in Japanese, the JESC [74] and OpenSubs [75], [76] corpora provide us subtitle data from a variety of movies. Given the context of utterance of most dialogue in movies, it is expected that most sentences from these datasets would exhibit the plain level of formality (as discussed in section 1). However, eliciting an unsupervised technique to label Japanese sentences according to their level of formality is out of the scope of this project. We thus use the aforementioned corpora basing ourselves on some assumptions (which we bring up in the next section), mainly for the purpose of benchmarking our method's capabilities when it comes to achieving the objective of politeness modulation.

Preparation of our Japanese data is similar to that of any other language. One difference is that we must use specialised software for tokenisation. We use the *natto-py* library for Python [77], which integrates the popular *MeCab* tokenisation software for Japanese [78] into Python. With these in place, we join our two corpora, remove duplicates, tokenise samples, and remove punctuation (file 29). Our final combined corpus boasts approximately 2.8M samples.

2.5.2. Method Application

We may now train our doc2vec model with our prepared data. Given that we do not have information about different sentence classes such as “plain”, “polite”, and “formal”, we must adapt our method for the sake of this benchmark. As it is not possible to build a classifier, we assume that all of our samples exhibit plain level of politeness. Hence, after back-translation, when the most similar samples are drawn from the back-translated sentence, we must also assume that the picked sample would be classified as plain.

Our doc2vec model is trained in mostly the same way as described under section 2.4.5, with the exception that this time we only train one model (file 30). Additionally, training is performed under Google Colab servers so that execution completes quickly, in less than 30 minutes.

When it comes to inferring style-transferred sentences in this case (file 31), the input is provided as a sentence in English, which is translated into Japanese through the *googletrans* library. The single translation into Japanese should be sufficient to devoid the input from stylistic attributes. Then, the Japanese translation is tokenised and prepared just like our corpus was, and vectors are inferred following the strategy specified under section 2.4.5. Lastly, after drawing the most similar samples from our inferred vector, we simply select the top one, thus completing our adapted style transfer process. Due to Japanese grammatical structure, we are not concerned about sentence length like we were before.

In the next section, we present the results from the above experiment as well as those from section 2.4.5.

3. Results

Without further ado, we provide the tables below highlighting the results from our elicited doc2vec-based approach for political slant transfer:

Democrat to Republican Style Transfer

Input	CAE	BST	D2V
as a hoosier, i thank you, rep. visclosky.	a lot , i am proud of you <unk>.	as a hoosier, i’m praying for you sir.	thank for standing up for the constitution
thank you for standing up for justice and against bigotry--racism, homophobia, sexism, misogyny, religious and xenophobia.	do you for standing up for highly and in bigotry–racism, programming, cut, granddaughters, unprescedented and excludes.	thanks for standing up for the constitution and get rid of obamacare, homophobie, cut, and actuality.	thanks trey for keeping justice honest and not crony political justice
thank you for all you are doing for us, attorney general harris!	thank you for standing up for us and i am proud of us!	thanks lawmaker for all you do for us, senator scott!	thanks obama for the shut down

Table 3.1: Comparative table putting results of our elicited style transfer method against those of state-of-the-art literature. In particular, this table shows transfer of democrat to republican political slant. “BST” refers to [1], “CAE” refers to [2], and “D2V” refers to our own method.

While evaluation of style transfer method performance is still an open research problem [4], [14], we follow the approach described in [1], where there are three different evaluation perspectives. The first is style transfer accuracy, that is, the proportion of outputs that match the desired style. The second is fluency, where readability and naturalness of style-transferred sentences are considered. The last is preservation of meaning

For the first two points of analysis, our method outperforms [1] and [2]. The reason for the former being that our model guarantees that style-transferred sentences match the desired style, and the reason for the latter being that outputted sentences are human-written rather than generated.

What we must consider now is the last evaluation criteria: preservation of meaning. According to [1], the definition of meaning in

Republican to Democrat Style Transfer

Input	CAE	BST	D2V
i will continue praying for you and the decisions made by our government!	i am proud of you and your vote for us!	i will continue to fight for you and the rest of our democracy!	and i will continue to stand by you mr ellison
tom, i wish u would bring change.	i agree, senator warren and could be.	brian, i am proud to have you representing me.	thank you senator wyden for a voice of reason
all talk and no action-why dont you have some guts like breitbart	and then we will be praying for them and i am proud of this position and i am proud of	keep up and dont know, you have a lot of respect as breitbart	liz warren is 100 correct about trump

Table 3.2: Comparative table putting results of our elicited style transfer method against those of state-of-the-art literature. In particular, this table shows transfer of republican to democrat political slant. “BST” refers to [1], “CAE” refers to [2], and “D2V” refers to our own method.

style transfer is not trivial, as the literal meaning of a sentence tends to change after style transfer. Thus, the definition of meaning must be more liberal, such that scrutiny is placed on the “intent of the utterance within the context of the discourse” [1]. For example, as illustrated in [1], if the intent is to criticise a restaurant with an online review, changing a single word representing a food item (like “pizza” for “hamburger”) will not jeopardise the intent. However, if the intent is to order food, then such change would be quite detrimental.

Given our results as per tables 3.1, 3.2, C.1, and C.2, we observe that in the relatively meaning-insensitive context of political discussion, the proposed method can preserve the intent of inputs to a certain extent, producing outputs of respectable quality in many cases.

It is unfortunate that during this study it was not possible to recruit volunteers to run an investigation on human preference for meaning preservation, as in [1]. In any case, the expectation is that our results would not be preferred if compared against those of [1]. However, if compared to [2], our method could be deemed preferable due to the superior fluency of its outputs, an attribute that the results of [2] visibly lack.

Turning our attention to the original motivation of politeness modulation in Japanese, and considering the insensitivity of our model towards meaning, the expectation is that it would produce poor results, as the task of translation is quite meaning-sensitive. This is, in fact, the case, as table C.3 shows. Still, as we briefly elaborate under section 2.5.1, achieving this initial objective is difficult at the moment since there is no available corpus in Japanese that labels samples according to level of formality or politeness, and producing such data is in itself a non-trivial topic warranting a reasonable amount of research.

Moreover, one limitation of our approach, which is admittedly also the case for [1] and [2], is that longer inputs are very hard to handle; the longer a sentence is, the harder it is to find a sample reasonably similar to it within already existing data. This is, however, a challenge many generative models are currently attempting to overcome [20], consistently finding improvement upon previous iterations.

Taking into account what we have observed about the capabilities of our method, it is likely that the more plentiful the data and the less meaning-sensitive the domain, the potential of our model is progressively maximised. It is possible that for the task of sentiment transfer, such as the aforementioned restaurant review example, our approach could produce results of higher quality, even more so if our data sample size is in the order of units of millions.

In addition to the above, our doc2vec model trained under section 2.4.5 could probably be improved by further tuning of hyperparameters or finding the sweet spot of our training/validation sample size, such that the testing set size could be maximised, allowing us to draw more style-transferred candidates. Also, through experimentation, our findings indicate that the maximum sentence similarity measured throughout the political slant transfer experiment was of about 65%. This is not necessarily an unsatisfactory value, but it is quite possible that it still isn’t the highest we can obtain.

Lastly, comparing word2vec to doc2vec puts into perspective how a small change can add so much potential to a technique. With that in mind, we consider that it might be possible to slightly modify the doc2vec approach to better suit the task of information retrieval, although we must recognise that even at its current stage, the doc2vec model already offers ground-breaking performance.

4. Conclusion

In this study, we examine the state-of-the-art textual style transfer method proposed in [1]. Throughout our work, we detail the implementation of said method, reproducing it partially and highlighting the complications and general inaccessibility of the computations behind it. This culminates into the proposal of a novel non-generative textual style transfer method, with a focus on ease of reproducibility.

Through our efforts, the proposed method achieves reasonable style transfer quality, with varied potential according to available data and meaning sensitiveness of context.

Our results suggest that further research in non-generative models for textual style transfer could be warranted, as there is much improvement that can still be carried out upon the proposed technique.

Additionally, we do not achieve the initial motivation of Japanese politeness modulation through style transfer, concluding that at the time being, this is not an achievable task due to lack of specific data.

Lastly, taking into account all of the work carried out, it is evident that no concerns are raised on legal, social, ethical or professional issues (LSEPI). All of the research that took place is purely experimental, and none of the elaborated tools or techniques could be used to cause harm. Finally, no personal data is collected or used, which eliminates any risk of exposure.

It is hoped that the outcomes and products of this project may be useful to others, sparking the development of forthcoming innovations that may be valuable to the field of natural language processing.

Appendix A

Packaged under the root folder of this report file, we find all source code used throughout the implementation of this project, under a folder named “*code*”.

In this appendix, we lay out the structure of the source code, and for ease of reference throughout the writing of this document, we assign a number (between parentheses) to some of the items below. After the structural display, we provide download links for all relevant corpora. Lastly, we provide tables listing all dependencies required to run the experiment code.

```
code
│
├───fightin_words (1)
│   │   .gitignore
│   │   LICENSE
│   │   README.md
│   │   setup.py
│   │
│   └───fightin_words
│           __init__.py
│
└───shizen
    ├───first_stage
    │   ├───classifier
    │   │   ├───bst
    │   │   │   │   classifier_augmented_baseline.py (2)
    │   │   │   │   classifier_augmented_kim.py (3)
    │   │   │   │   classifier_augmented_trainable.py (4)
    │   │   │   │   classifier_baseline.py (5)
    │   │   │   │   classifier_load_data.py (6)
    │   │   │   │   classifier_plot_helper.py
    │   │   │   │   classifier_test.py (7)
    │   │   │   │   classifier_utils.py (8)
    │   │   │   │   corpora_resplit.py (9)
    │   │   │   │   corpora_sanitise.py (10)
    │   │   │   │   lexicon_information.py
    │   │   │   │   word_embeddings.py (11)
    │   │   │   │   z_scores_informed.py (12)
    │   │   │   │   z_scores_plot.py
    │   │   │   │   z_scores_uninformed.py (13)
    │   │   │   │
    │   │   │   ├───corpora
    │   │   │   │   ├───original
    │   │   │   │   └───resplit
    │   │   │   │       ├───sanitised
    │   │   │   │       └───unsanitised
    │   │   │   └───out
    │   │   └───d2v
    │   │       │   classifier_baseline.py (14)
    │   │       │   classifier_load_data.py (15)
    │   │       │   classifier_plot_helper.py
    │   │       │   classifier_test.py (16)
    │   │       │   classifier_utils.py (17)
    │   │       │   corpora_resplit.py (18)
    │   │       │   corpora_sanitise.py (19)
    │   │       │   doc2vec.py (20)
    │   │       │   style_transfer.py (21)
    │   │       │
    │   │       ├───corpora
    │   │       │   ├───original
    │   │       │   └───resplit
    │   │       │       ├───sanitised
    │   │       │       └───unsanitised
    │   │       └───out
    │   └───lstm
    │       │   classify.py (22)
    │       │   corpora_join.py (23)
    │       │   corpora_sanitise.py (24)
    │       │   lstm_en-fr_baseline.py (25)
    │       │   lstm_fr-en_democrat.py (26)
    │       │   lstm_generator.py (27)
    │       │   lstm_load_data.py (28)
    │       │   lstm_plot_helper.py
    │       │   lstm_utils.py
    │       │
    │       ├───corpora
    │       │   ├───custom
    │       │   │   ├───sanitised
    │       │   │   └───unsanitised
    │       │   └───original
    │       │       ├───commoncrawl
    │       │       ├───europarl-v7
    │       │       └───news-commentary
    │       └───out
    └───second_stage
        │   corpus_generator.py (29)
        │   doc2vec.py (30)
        │   style_transfer.py (31)
        │
        ├───corpora
        │   ├───original
        │   │   ├───jesc
        │   │   └───opensubs
        │   └───tokenised
        └───out
```

Political corpus: link.

EuroParl v7, Common Crawl, and News Commentary v10 corpora: link.

JESC corpus: link.

OpenSubs corpus: link.

The development of this project was carried out almost entirely with Python 3.6.8 under a Windows 10 64-bit environment, the exception being that at some points code was executed under the Google Colab managed server environment. Below, we provide tables of code and software dependencies:

Python package	Version
Keras	2.2.4
tensorflow-gpu	1.12.0
scikit-learn	0.20.3
scipy	1.2.1
numpy	1.16.2
plotly	3.7.1
ann_visualizer	2.5
graphviz	0.10.1
tqdm	4.31.1
gensim	3.7.2
googletrans	2.4.0
langid	1.1.6
natto-py	0.9.0
pydot	1.4.1

Table A.1: List of Python dependencies.

Software	Version
NVIDIA GPU drivers	418.81
CUDA Toolkit	9.0
cuDNN SDK	7.4.2
Visual Studio	2017
Graphviz	2.38
MeCab	0.996

Table A.2: List of external dependencies.

Appendix B

In this appendix, we include figures which are too large to fit in the main body of the report, or supporting figures that have some relevance, but not enough to occupy the main body.

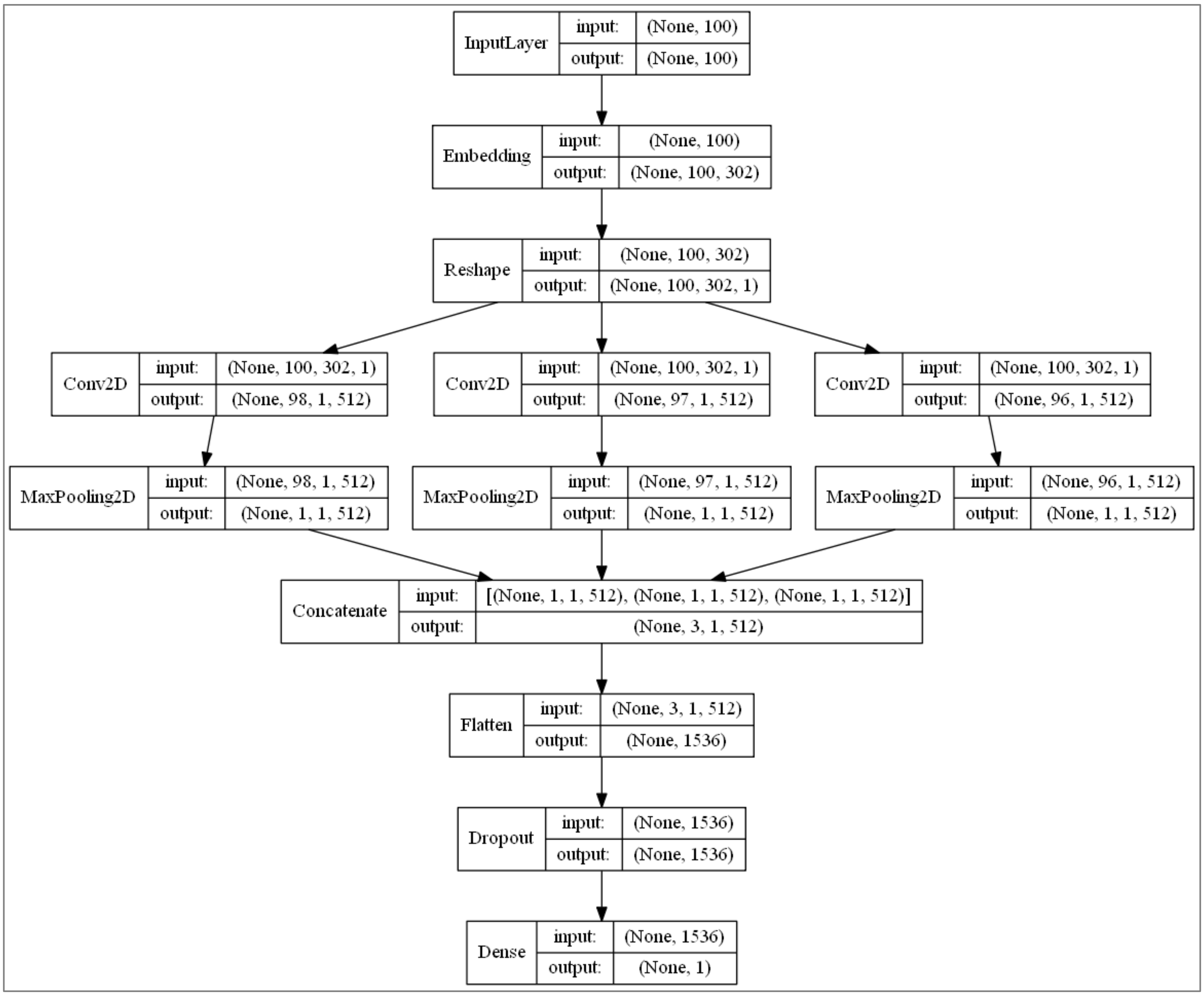

InputLayer | input: (None, 100) | output: (None, 100)
Embedding | input: (None, 100) | output: (None, 100, 302)
Reshape | input: (None, 100, 302) | output: (None, 100, 302, 1)
Conv2D | input: (None, 100, 302, 1) | output: (None, 98, 1, 512)
Conv2D | input: (None, 100, 302, 1) | output: (None, 97, 1, 512)
Conv2D | input: (None, 100, 302, 1) | output: (None, 96, 1, 512)
MaxPooling2D | input: (None, 98, 1, 512) | output: (None, 1, 1, 512)
MaxPooling2D | input: (None, 97, 1, 512) | output: (None, 1, 1, 512)
MaxPooling2D | input: (None, 96, 1, 512) | output: (None, 1, 1, 512)
Concatenate | input: [(None, 1, 1, 512), (None, 1, 1, 512), (None, 1, 1, 512)] | output: (None, 3, 1, 512)
Flatten | input: (None, 3, 1, 512) | output: (None, 1536)
Dropout | input: (None, 1536) | output: (None, 1536)
Dense | input: (None, 1536) | output: (None, 1)

Figure B.1: A more faithful reproduction of Yoon Kim's proposed architecture. This is the underlying architecture of our Kim classifier model (see section 2.2.3).

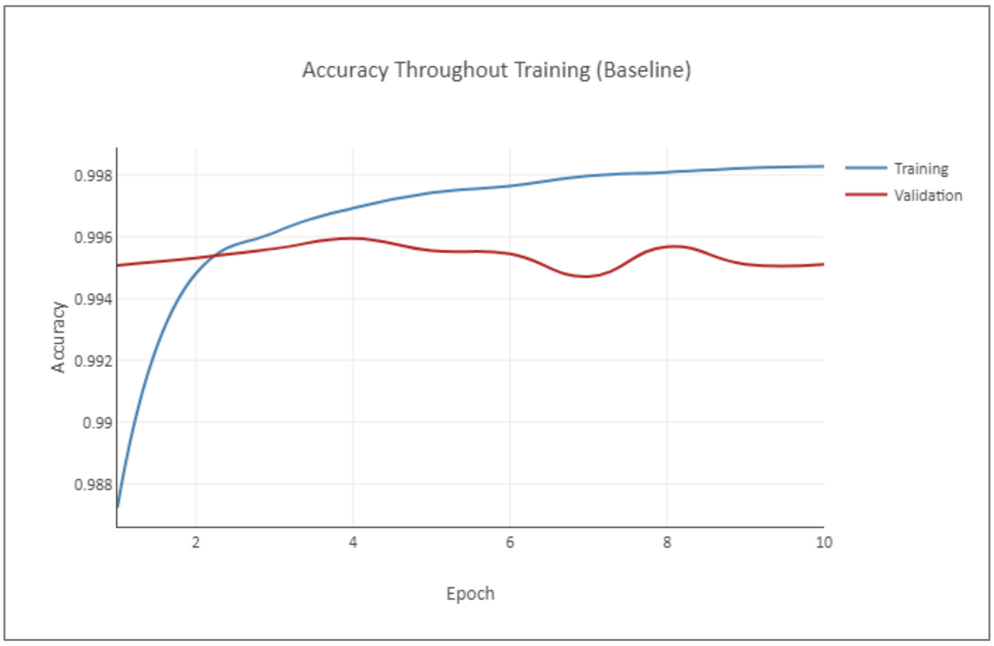


Figure B.2: History of accuracy seen throughout the training of a model with the Baseline architecture (see section 2.2.4).

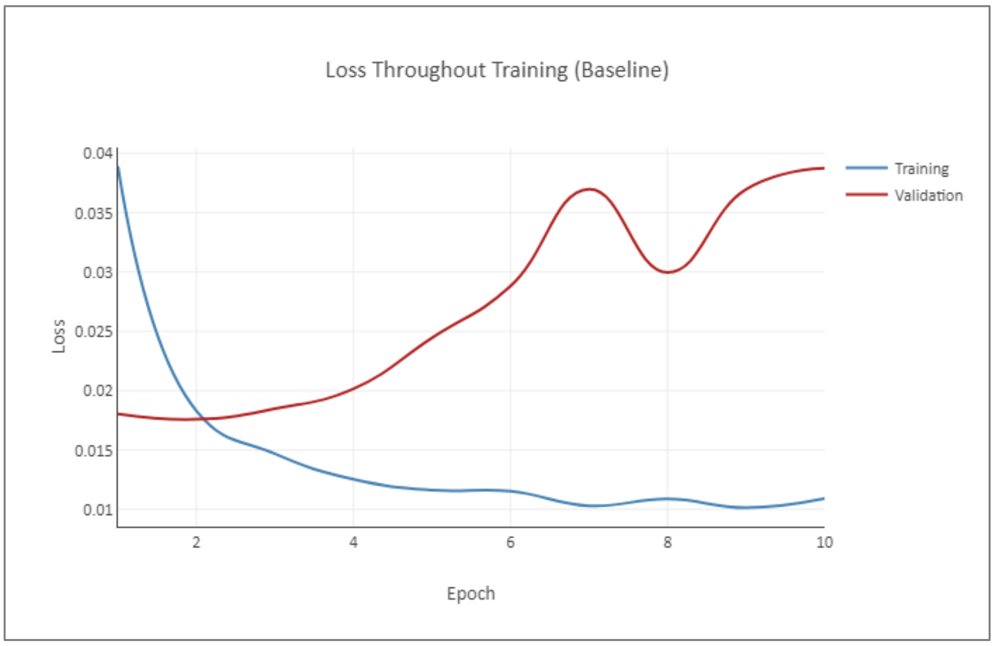


Figure B.3: History of loss seen throughout the training of a model with the Baseline architecture (see section 2.2.4).

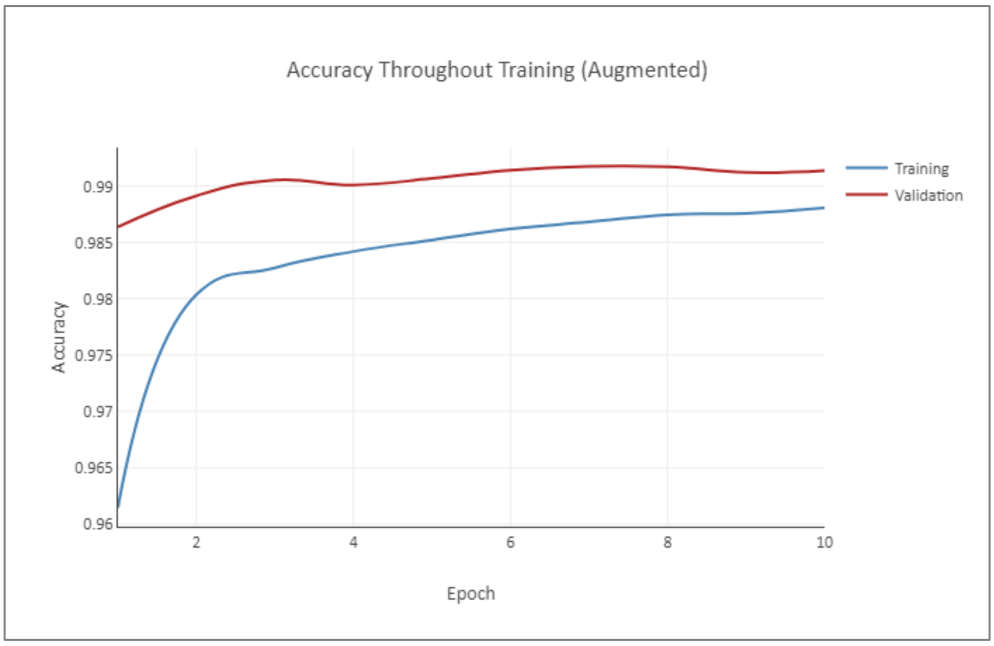


Figure B.4: History of accuracy seen throughout the training of a model with the Augmented architecture (see section 2.2.4).

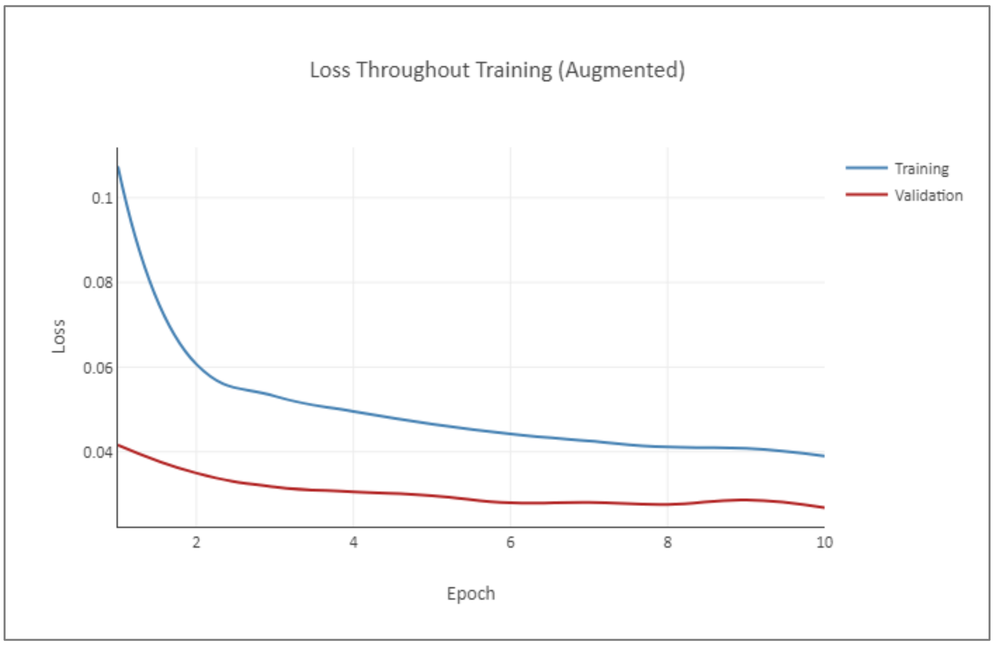


Figure B.5: History of loss seen throughout the training of a model with the Augmented architecture (see section 2.2.4).

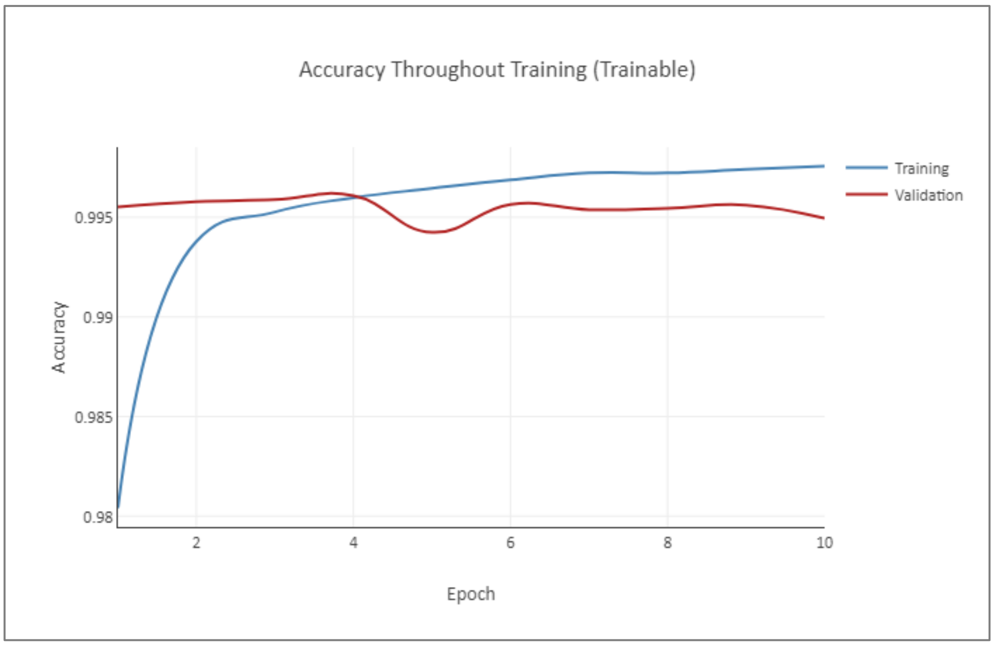


Figure B.6: History of accuracy seen throughout the training of a model with the Trainable architecture (see section 2.2.4).

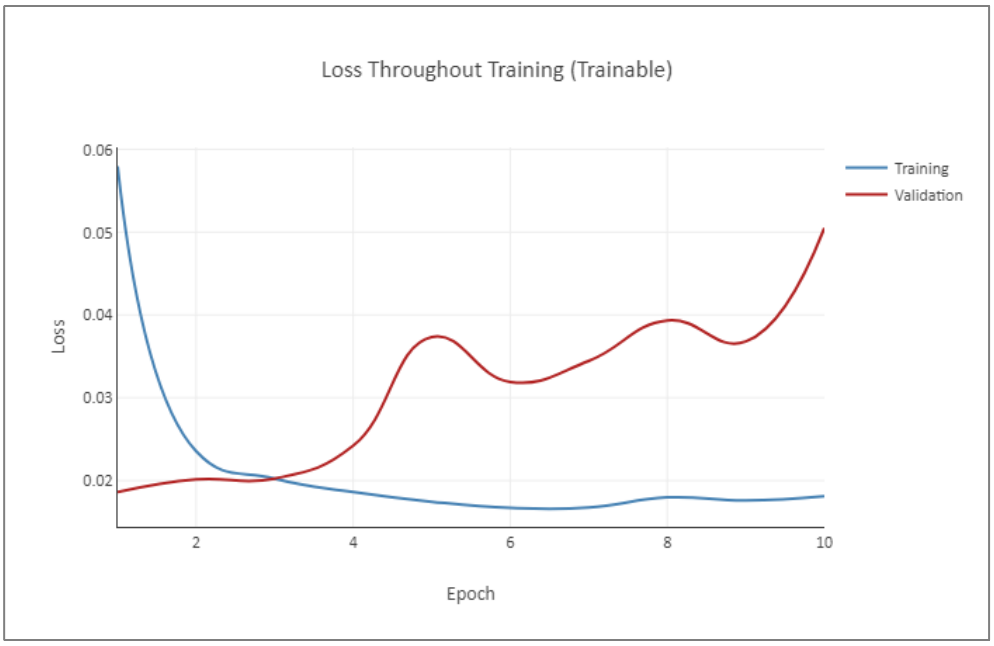


Figure B.7: History of loss seen throughout the training of a model with the Trainable architecture (see section 2.2.4).

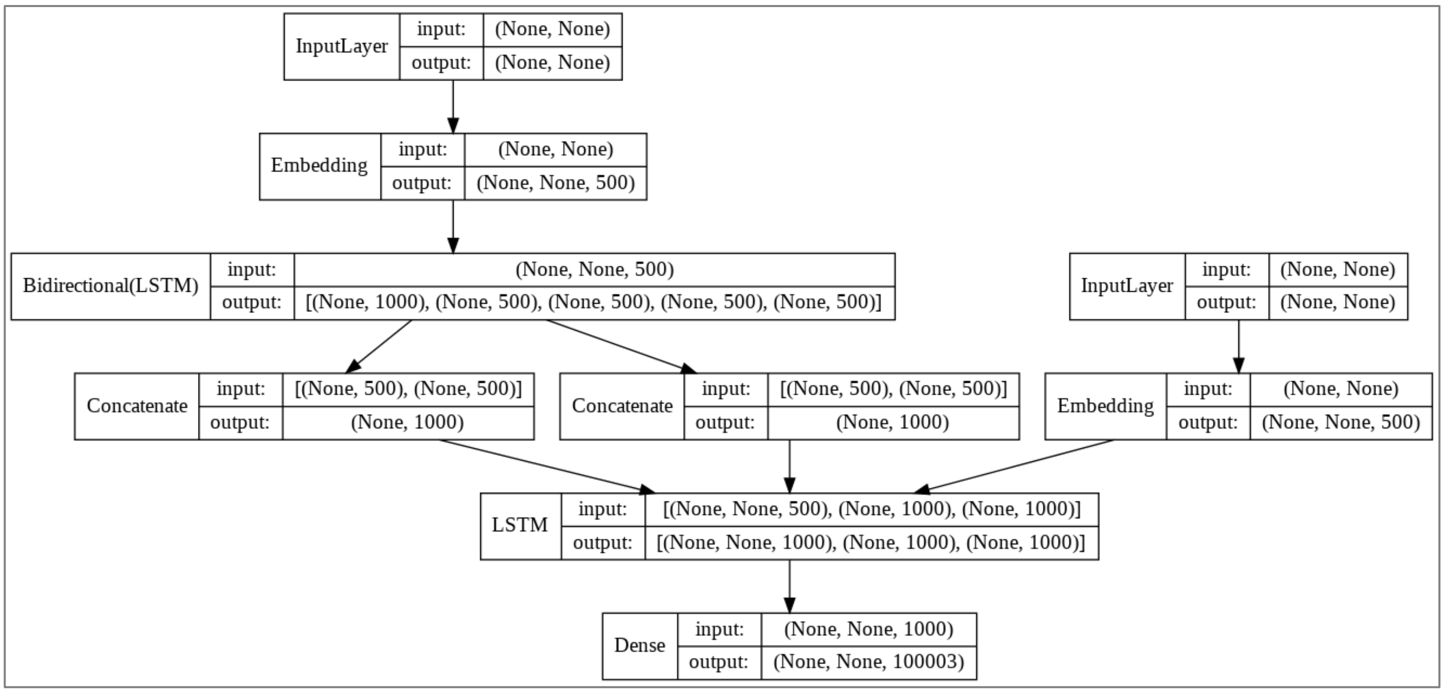

InputLayer
input: (None, None)
output: (None, None)
Embedding
input: (None, None)
output: (None, None, 500)
Bidirectional(LSTM)
input: (None, None, 500)
output: [(None, 1000), (None, 500), (None, 500), (None, 500), (None, 500)]
InputLayer
input: (None, None)
output: (None, None)
Concatenate
input: [(None, 500), (None, 500)]
output: (None, 1000)
Concatenate
input: [(None, 500), (None, 500)]
output: (None, 1000)
Embedding
input: (None, None)
output: (None, None, 500)
LSTM
input: [(None, None, 500), (None, 1000), (None, 1000)]
output: [(None, None, 1000), (None, 1000), (None, 1000)]
Dense
input: (None, None, 1000)
output: (None, None, 100003)

Figure B.8: Full architecture of the machine translation model (see section 2.3.2).

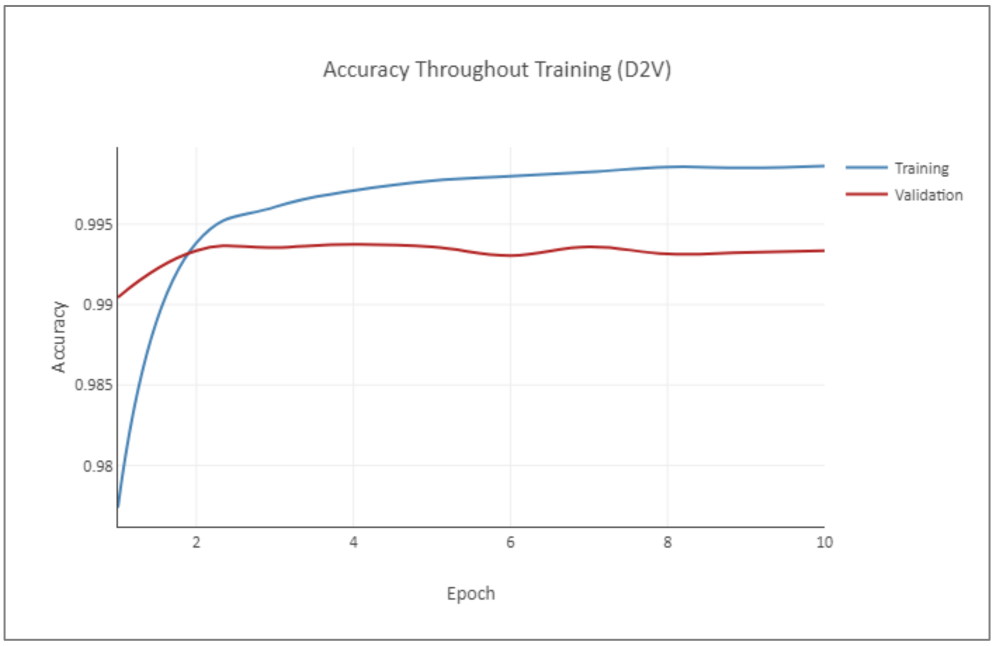


Figure B.9: History of accuracy seen throughout the training of a model with the Baseline architecture, used during experimentation of the doc2vec method (see section 2.4.5).

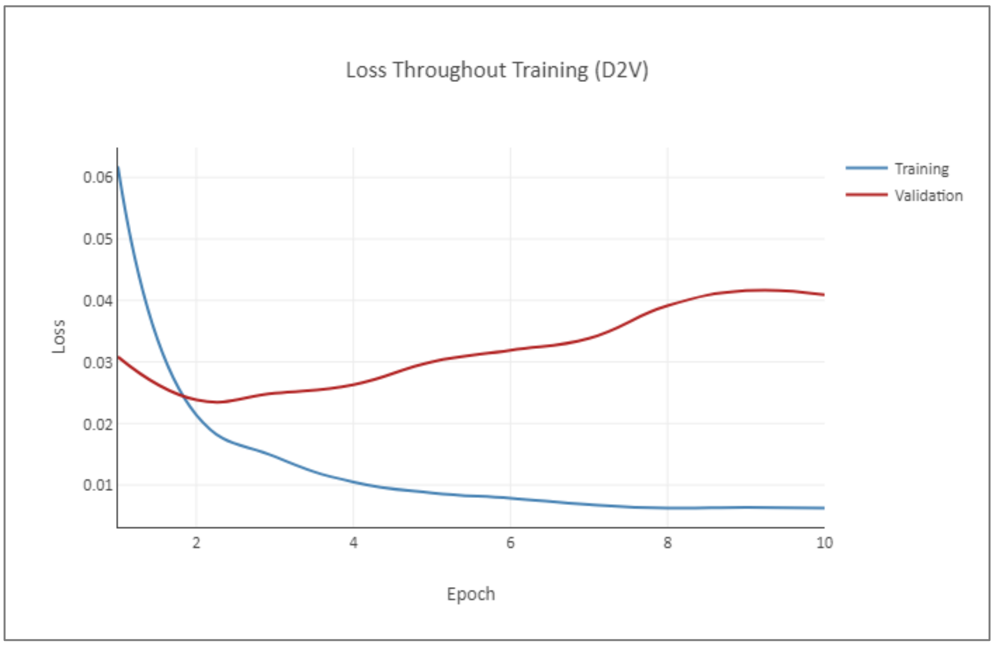


Figure B.10: History of loss seen throughout the training of a model with the Baseline architecture, used during experimentation of the doc2vec method (see section 2.4.5).

Appendix C

In this appendix, we include tables which are too large to fit in the main body of the report.

Democrat to Republican Style Transfer

Input	Style transfer through D2V
we need to keep democrats in the white house	may god bless you and keep you at it
just vote for bernie and forget the others	just vote to repeal the whole mass
i am a life long proud and vocal democrat	congressman mcclintock spoken like a true democrat
thank you for your vote sen klobuchar	beat duke vote senator trey grayson
please run for president in 2020 senator coreyhey chuck how about you stop casting yes votes for all his cabinet picks	mr awesome trey gowdy please stop stop and stop muslim immigrating to the united states
i dont see trump supporters rioting	not listening to your constituents we see
so sorry for your loss my deepest condolences to you and your family	youre wasting your time and our tax dollars
senator warren i respect you	south carolina loves you senator
get a grip elizabeth	trey gowdy take boehners job plezzzzzzz

Table C.1: Extra samples of style-transferred sentences using the doc2vec model. In this case, we transfer from democrat political slant to republican political slant.

Republican to Democrat Style Transfer

Input	Style transfer through D2V
stand for liberty and the constitution	recall senator wyden
too bad you were too spineless to back him	do you want hilary for a friend or do you want to win in november
youre a fraud paul	you know who is bernie sanders
sorry senator but you have done nothing to secure our borders	i am sure she is doing this for her massachusetts constituents
keep it up senator mccain	happy passover weekend dear congresswoman
the obama administration are just buying time	just please run in 2020
stick to your principles and support of the constitution mia	thank you elizabeth for your words of wisdom
oh little marco how does it feel to sell your soul to the devil	we need to send in a ranger team to kick some ass and deliver the girls back home to their families
you guys have lost touch with the american people	kamala harris you lost
we the people have had just about enough of him playing dictator of our country	have a great rally you got my support congresswoman jackson lee

Table C.2: Extra samples of style-transferred sentences using the doc2vec model. In this case, we transfer from republican political slant to democrat political slant.

Style Transfer of Japanese Sentences

	Input	Google	D2V	Desired
Japanese		電車が来ます。	電車に乗り継ぎしないと	ああ電車
English	The train will come.	The train will come.	I have to transfer to the train.	Oh, the train.
Japanese		私はその店に行きます。	その食堂で私はさまざまな段階を経て	俺はこの店に行ってみる
English	I am going to the shop.	I am going to the shop.	I went through different stages in the cafeteria.	I'll try going into this shop.
Japanese		お名前は何ですか?	名前は何ですかです (sentence is broken)	では君の名前は
English	What is your name?	What is your name?	What is your name?	And what's your name?

Table C.3: A display of the results of some of our model's attempts to transfer the style of Japanese sentences to the plain level of politeness. It is observable that the output quality is not very good. The "Input" column represents our input (only applicable in the case of English), the "Google" column represents the Google Translate translation of our input to Japanese, the "D2V" column represents our model's attempt to transfer the style of the translated sentence to plain, and the "Desired" column represents a plain style sentence similar to our input which was handpicked from our data and would have been a good candidate for a style-transferred version. The "English" row provides manual translations of Japanese sentences when appropriate for more ease of understanding.

Bibliography

[1] S. Prabhumoye et al., “Style Transfer Through Back-Translation,” in *Association for Computational Linguistics*, 2018, pp. 866–876.

[2] T. Shen et al., “Style Transfer from Non-Parallel Text by Cross-Alignment,” in *Neural Information Processing Systems*, 2017.

[3] Z. Yang et al., “Unsupervised Text Style Transfer using Language Models as Discriminators,” in *Neural Information Processing Systems*, 2018.

[4] Z. Fu et al., “Style Transfer in Text: Exploration and Evaluation,” in *AAAI Conference on Artificial Intelligence*, 2018, pp. 663–670.

[5] Z. Fu, “A Paper List for Style Transfer in Text,” 2019. [Online]. Available: https://github.com/fuzhenxin/Style-Transfer-in-Text. [Accessed: 23-Apr-2019].

[6] M. Haugh, “Revisiting the Conceptualisation of Politeness in English and Japanese,” *Multilingua*, vol. 23, pp. 85–109, 2004.

[7] P. J. Wetzel, *Keigo in Modern Japan: Polite Language from Meiji to the Present*, 1st ed. Hawaii: University of Hawaii Press, 2004.

[8] F. Chollet et al., “Keras.” GitHub, 2015.

[9] M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” in *Operating Systems Design and Implementation*, 2016, pp. 265–283.

[10] G. Klein et al., “OpenNMT: Open-Source Toolkit for Neural Machine Translation,” in *Association for Computational Linguistics*, 2017, pp. 67–72.

[11] Google, “Welcome To Colaboratory - Colaboratory.” [Online]. Available: https://colab.research.google.com/. [Accessed: 24-Apr-2019].

[12] Q. Le and T. Mikolov, “Distributed Representations of Sentences and Documents,” in *International Conference on Machine Learning*, 2014.

[13] R. Řehůřek, “models.doc2vec – Doc2vec paragraph embeddings,” 2019. [Online]. Available: https://radimrehurek.com/gensim/models/doc2vec.html. [Accessed: 24-Apr-2019].

[14] A. Tikhonov and I. P. Yamshchikov, “What is wrong with style

transfer for texts?," 2018.

[15] Japan Online School Corporation, "List of textbooks for business Japanese." [Online]. Available: http://j-os.com/japanesematerial/business-en.html. [Accessed: 25-Apr-2019].

[16] B. Barrett and L. Cook, "けいご【敬語】," 2001. [Online]. Available: http://www.jekai.org/entries/aa/00/no/aa00no31.htm. [Accessed: 25-Apr-2019].

[17] I. Fujimura et al., "Lexical and grammatical features of spoken and written Japanese in contrast: exploring a lexical profiling approach to comparing spoken and written corpora," in *GSCP*, 2012, pp. 393–398.

[18] SimilarWeb, "Translate.google.com Analytics - Market Share Stats & Traffic Ranking." [Online]. Available: https://www.similarweb.com/website/translate.google.com. [Accessed: 25-Apr-2019].

[19] L. A. Gatys et al., "A Neural Algorithm of Artistic Style," 2015.

[20] S. Subramanian et al., "Multiple-Attribute Text Style Transfer," 2018.

[21] Z. Hu et al., "Toward Controlled Generation of Text," in *International Conference on Machine Learning*, 2017.

[22] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," 2014.

[23] E. Rabinovich et al., "Personalized Machine Translation: Preserving Original Author Traits," in *European Chapter of the Association for Computational Linguistics*, 2017, vol. 1, pp. 1074–1084.

[24] S. Prabhumoye et al., "Style Transfer Through Back-Translation," *Style Transfer Through Back-Translation*, 2018. [Online]. Available: https://github.com/shrimai/Style-Transfer-Through-Back-Translation. [Accessed: 25-Apr-2019].

[25] Z. Hu et al., "Text Style Transfer," *Toward Controlled Generation of Text*, 2017. [Online]. Available: https://github.com/asyml/texar/tree/master/examples/text_style_transfer. [Accessed: 25-Apr-2019].

[26] B. L. Monroe et al., "Fightin' Words: Lexical Feature Selection and Evaluation for Identifying the Content of Political Conflic," *Polit. Anal.*, vol. 16, no. 4, pp. 372–403, 2008.

[27] R. Voigt et al., "RtGender: A Corpus for Studying Differential

Responses to Gender,” in *Language Resources and Evaluation Conference*, 2018, pp. 2814–2820.

[28] I. Guyon, “A scaling law for the validation-set training-set size ratio,” 1997.

[29] T. Hastie et al., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 12th ed. New York: Springer, 2009.

[30] N. Bunkley, “Joseph Juran, 103, Pioneer in Quality Control, Dies,” *New York Times*, p. B7, 2008.

[31] K. Lim, “fightin-words,” *GitHub*, 2019. [Online]. Available: https://github.com/kenlimmj/fightin-words. [Accessed: 26-Apr-2019].

[32] J. Hessel, “Fightin’ Words,” *GitHub*, 2016. [Online]. Available: https://github.com/jmhessel/FightingWords. [Accessed: 26-Apr-2019].

[33] J. C. García Sigüenza, “Python 2 to 3!” [Online]. Available: https://pythonconverter.com/. [Accessed: 26-Apr-2019].

[34] Google, “Google Cloud.” [Online]. Available: https://cloud.google.com/. [Accessed: 26-Apr-2019].

[35] R. Řehůřek, “gensim: models.word2vec – Word2vec embeddings,” 2019. [Online]. Available: https://radimrehurek.com/gensim/models/word2vec.html. [Accessed: 26-Apr-2019].

[36] T. Mikolov et al., “Efficient Estimation of Word Representations in Vector Space,” 2013.

[37] T. Mikolov et al., “Distributed Representations of Words and Phrases and their Compositionality,” 2014.

[38] S. O’Keefe, “Solving more complex problems,” *INNS*. University of York, 2018.

[39] Y. Kim, “Convolutional Neural Networks for Sentence Classification,” in *Empirical Methods in Natural Language Processing*, 2014, pp. 1746–1751.

[40] N. Srivastava et al., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[41] B. V. Oswal, “CNN-text-classification-keras,” *GitHub*, 2016. [Online]. Available: https://github.com/bhaveshoswal/CNN-text-classification-keras. [Accessed: 26-Apr-2019].

[42] P. Baldi and P. J. Sadowski, “Understanding Dropout,” in *Neural Information Processing Systems*, 2013, pp. 2814–2822.

[43] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *International Conference for Learning Representations*, 2014.

[44] NVIDIA Corporation, “CUDA GPUs | NVIDIA Developer,” *NVIDIA Developer*, 2019. [Online]. Available: https://developer.nvidia.com/cuda-gpus. [Accessed: 27-Apr-2019].

[45] A. Bakliwal et al., “Sentiment Analysis of Political Tweets: Towards an Accurate Classifier,” in *Workshop on Language in Social Media*, 2013, pp. 49–58.

[46] F. Chollet et al., “FAQ - Keras Documentation,” 2015. [Online]. Available: https://keras.io/getting-started/faq/#why-is-the-training-loss-much-higher-than-the-testing-loss. [Accessed: 27-Apr-2019].

[47] D. Erhan et al., “Why Does Unsupervised Pre-training Help Deep Learning?,” *J. Mach. Learn. Res.*, vol. 11, no. Feb, pp. 625–660, 2010.

[48] O. Bojar et al., “Findings of the 2015 Workshop on Statistical Machine Translation,” in *Workshop on Statistical Machine Translation*, 2015.

[49] M. Rikters, “Impact of Corpora Quality on Neural Machine Translation,” in *Baltic Human Language Technologies*, 2018.

[50] W. Krzysztof, “Noisy-parallel and comparable corpora filtering methodology for the extraction of bi-lingual equivalent data at sentence level,” *Comput. Sci.*, vol. 16, no. 2, pp. 169–184, 2015.

[51] V. M. Sánchez-Cartagena et al., “Prompsit’s submission to WMT 2018 Parallel Corpus Filtering shared task,” in *Machine Translation*, 2018, vol. 2, pp. 955–962.

[52] V. M. Sánchez-Cartagena et al., “bicleaner,” *GitHub*, 2018. [Online]. Available: https://github.com/bitextor/bicleaner. [Accessed: 28-Apr-2019].

[53] M. Rikters, “Corpora Cleaning Tools,” *2GitHub*, 2018. [Online]. Available: https://github.com/M4t1ss/parallel-corpora-tools. [Accessed: 28-Apr-2019].

[54] M. Lui, “langid.py,” *GitHub*, 2017. [Online]. Available: https://github.com/saffsd/langid.py. [Accessed: 28-Apr-2019].

[55] M. Lui and T. Baldwin, “Cross-domain Feature Selection for

Language Identification,” in *International Joint Conference on Natural Language Processing*, 2011, pp. 553–561.

[56] M. Lui and T. Baldwin, “langid.py: An Off-the-shelf Language Identification Tool,” in *Association for Computational Linguistics*, 2012, pp. 25–30.

[57] Indix, “whatthelang,” *GitHub*, 2017. [Online]. Available: https://github.com/indix/whatthelang. [Accessed: 28-Apr-2019].

[58] A. Joulin et al., “Bag of Tricks for Efficient Text Classification,” in *European Chapter of the Association for Computational Linguistics*, 2017, pp. 427–431.

[59] A. Joulin et al., “FastText.zip: Compressing text classification models,” 2016.

[60] I. Sutskever et al., “Sequence to Sequence Learning with Neural Networks,” 2014.

[61] D. Bahdanau et al., “Neural Machine Translation by Jointly Learning to Align and Translate,” 2015.

[62] F. Chollet, “A ten-minute introduction to sequence-to-sequence learning in Keras,” *Keras*, 2017. [Online]. Available: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html. [Accessed: 28-Apr-2019].

[63] J. Lee et al., “Fully Character-Level Neural Machine Translation without Explicit Segmentation,” *Trans. Assoc. Comput. Linguist.*, vol. 5, no. 1, pp. 365–378, 2017.

[64] N. Chirkova et al., “Bayesian Compression for Natural Language Processing,” in *Empirical Methods in Natural Language Processing*, 2018, pp. 2910–2915.

[65] A. Amidi and S. Amidi, “A detailed example of data generators with Keras,” 2017. [Online]. Available: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly. [Accessed: 28-Apr-2019].

[66] T. Afonja, “AISaturdayLagos: Leveraging on Google Colab,” *AISaturdayLagos*, 2018. [Online]. Available: https://medium.com/ai-saturdays/aisaturdaylagos-leveraging-on-google-colab-313bab053603. [Accessed: 28-Apr-2019].

[67] M. Neishi et al., “A Bag of Useful Tricks for Practical Neural Machine Translation: Embedding Layer Initialization and Large Batch Size,” in *Workshop on Asian Translation*, 2017, pp. 99–109.

[68] G. Nicola, “Text generation with a Variational Autoencoder,” 2018. [Online]. Available: https://nicgian.github.io/text-

generation-vae/. [Accessed: 28-Apr-2019].

[69] T. Liu, “Errors when evaluating a tensor in custom loss function?,” 2016. [Online]. Available: https://github.com/keras-team/keras/issues/4075. [Accessed: 27-Apr-2019].

[70] J. Paulsen, “Enormous Growth in Data is Coming — How to Prepare for It, and Prosper From It,” *Seagate Blog*. [Online]. Available: https://blog.seagate.com/business/enormous-growth-in-data-is-coming-how-to-prepare-for-it-and-prosper-from-it/. [Accessed: 28-Apr-2019].

[71] R. Řehůřek, “Doc2vec tutorial,” *RARE Technologies*, 2014. [Online]. Available: https://rare-technologies.com/doc2vec-tutorial/. [Accessed: 29-Apr-2019].

[72] S. Han, “Googletrans,” *GitHub*, 2015. [Online]. Available: https://github.com/ssut/py-googletrans. [Accessed: 29-Apr-2019].

[73] S. Zwicklbauer, “Doc2Vec infer_vector() could (as option?) offer deterministic result,” *GitHub*, 2015. [Online]. Available: https://github.com/RaRe-Technologies/gensim/issues/447. [Accessed: 29-Apr-2019].

[74] R. Pryzant et al., “JESC: Japanese-English Subtitle Corpus,” 2017.

[75] P. Lison and J. Tiedemann, “OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles,” in *Language Resources and Evaluation Conference*, 2016, pp. 923–929.

[76] P. Lison et al., “OpenSubtitles2018: Statistical Rescoring of Sentence Alignments in Large, Noisy Parallel Corpora,” in *Language Resources and Evaluation Conference*, 2018, pp. 1742–1748.

[77] B. M. Fujita, “natto-py,” *GitHub*, 2019. [Online]. Available: https://github.com/buruzaemon/natto-py. [Accessed: 29-Apr-2019].

[78] T. Kudou, “mecab,” *GitHub*, 2018. [Online]. Available: https://github.com/taku910/mecab. [Accessed: 29-Apr-2019].