Investigating Propaganda Detection

Candidate: 260037

Advanced Natural Language Processing (968G5)
University of Sussex

May 11, 2024

Abstract

This report investigates different approaches to text classification in the context of two propaganda detection tasks. The investigation will be carried out through experimentation and analysis of two different machine learning approaches for each task, with performance analysed with reference to wider empirical considerations.

Owing to their prominence in contemporary natural language applications [5, 3] and their use in many of the best performing models in the paper from which the datasets in this assignment are derived [6], both approaches will use vectorised word-embeddings. The first approach will use pretrained static word embeddings trained as described by Mikolov et al. and often referred to as word2vec embeddings [7]. The second will use contextualised, 'dynamic' word embeddings encoded by the BERT transformer model as described by Devlin et al. [2]. In both approaches the embeddings will form the inputs to a multi-layer perceptron (MLP) classifier model.

The fundamental differences between the embeddings, model performance on the different tasks and comparison between them shall be used as a basis for discussion of the approaches as well as representation and classification in natural language processing more generally.

Words: 4094

1 Introduction

Being able to identify text as propaganda is an increasingly important and challenging machine learning task. It is important because identifying whether information is neutral and purely informative, or if it has an intention to influence or persuade, can help people make more informed decisions, and draw more objective conclusions rather than being swayed unknowingly by the intention of the person or organisation presenting that information.

It is a particularly difficult task though, because unfortunately "[propaganda] is most successful when it goes unnoticed by the reader" [6, p1393]. It can be genuinely difficult to identify propaganda as those deploying it can often deliberately try to obscure their intent.

The variety and sophistication of propaganda techniques is reflected in the fact that in this task a number of different 'types' of propaganda are identified. Da San Martino et al. [6] cite 14 distinct varieties from 'flag waving' to 'thought terminating cliche', of which 8 are represented in the dataset used here. The commonalities between them are well summarised when they say that all employ "rhetorical and psychological techniques"[6, p1378]. Training a model to spot such varied, nebulous and sometimes sophisticated communication techniques, which are often skilfully deployed to evade detection is a significant challenge.

2 Problem Outline

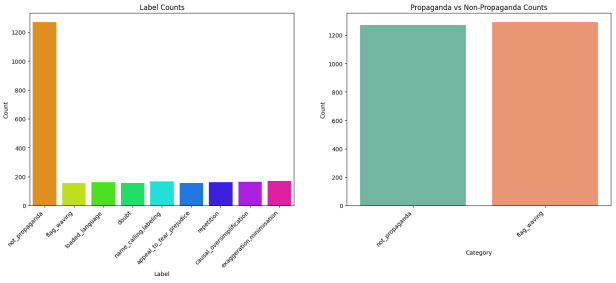
The broad aim of the assignment is to build and evaluate two approaches to two different classification tasks. The first task (T1) is to build a binary classifier which can classify a passage of text as containing propaganda or not. The second task (T2) is a multi-class classification problem where all non-propaganda instances are removed, and the classifier must determine the correct specific propaganda-type label for the given propaganda snippet removed from its context.

label	tagged in context
loaded_language	This is why it is so <BOS> heinous <EOS>.
not_propaganda	No, <BOS> he <EOS> will not be confirmed.
flag_waving	Where is that authorized in <BOS> our Constitution <EOS>?

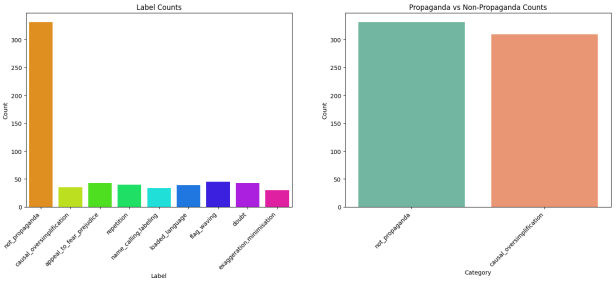
Figure 1: Original Form of Data

The data provided was in the form of a table of label and text pairs, where the label either described a type of propaganda found in the text (for example 'flag-waving' or 'loaded language') or identified the text as being 'not propaganda'. Each text item was a passage of text which contained segment start and end tags ($< BOS >$ and $< EOS >$). The label referred to the particular type of propaganda contained in the segment as seen in Fig. 1.

The data is a subset of a larger dataset created as part of the Propaganda Techniques Corpus (the 'PTC-SemEval20 corpus') [6] and was created from "a sample of news articles from .. mid-2017...to early 2019 [from] 13 propaganda and 36 non-propaganda news outlets". Labels were applied manually by 'professional annotators" who were responsible for "both spotting a propaganda snippet and ...labelling it with a specific propaganda technique" [6] (pp1381) It was provided in the form of training and validation datasets in *.tsv* file format. both were structurally identical with just two columns; **'label'** and **'tagged in context'**. The training dataset had 2560 entries and the validation dataset has 640 entries so



(a) Training data distributions (T2 and T1)



(b) Validation data distributions (T2 and T1)

Figure 2: Data distributions

the validation set is exactly 1/4 the size of the training data. Although there were 14 possible propaganda-type labels there were only 8 represented here. Those type were Flag Waving, Exaggeration/Minimisation, Doubt, Loaded Language and Appeal to Fear. Examples of these can be seen in Fig 17.

In a task such as this, one would typically reserve a test dataset in order to get a 'true' measure of performance on data that has been completely withheld during training. The assignment brief makes clear, however, that (absolute) performance is not the desired focus, and so given the helpful balance between training and validation data provided, it was decided to use these two sets only for investigating and understanding the proposed methods.

As can be seen in Fig 2, both datasets are well balanced with an almost even split between propaganda and not-propaganda labels for T1, as well as a balanced split between the individual propaganda labels for T2. There were no Null or Nan values in the dataset - it was clean and complete.

3 Method

3.1 Preprocessing

Very little preprocessing was required owing to the straightforward and complete nature of the data provided. Approach-specific preprocessing such as tokenisation is described the relevant sections below, though there were some basic transformations applied to the extract the appropriate data for each task.

For T1 a new binary target variable column was added to the data with a 0 value for any text item with a label of 'non propaganda', and 1 for any other columns. A new feature column for this task was also created with the original full context text with the snippet tags removed. See the left hand side of flow charts in Figs. 3 and 5 for examples.

For T2 any rows with the 'not propaganda' label were removed. The remaining labels were given numerical categorical labels in accordance with a class/label dictionary that remained consistent across all runs (i.e. labels 0,1,2,3 etc). For T2 the classifier is required to classify the propaganda based on the snippet only, so for this task a new feature column was created with just the text from inside the $< BOS >$ and $< EOS >$ tags resulting in a transformation viewable on the right hands side of flow charts in Figs. 3 and 5.

The final datasets passed forward for approach-specific processing were as follows: T1 had 2560 de-tagged training passages of text and 640 validation passages, all with binary labels indicating propaganda or not propaganda. T2 has 1291 de-tagged training snippets of text 309 validation snippets, all with numerical class labels between 0 and 7. As shall be discussed below, further preprocessing was required for each of the methods, and all datasets were converted into Pytorch datasets and Dataloaders before the MLP models were trained.

3.2 Approach 1 - Word2Vec embeddings and MLP (W2V)

3.2.1 Overview

"Word2Vec" generally refers to algorithms introduced by Tomas Mikolov et al. in their paper 'Efficient Estimation of Word Representations in Vector Space' [7]. This paper introduces a pair of methods for "computing continuous vector representations of words from very large data sets"[7, p1].

The process by which they do this (in both the continuous-bag-of-words (CBOW) and skip-gram approaches they describe) is not to directly try to learn or output the embeddings that they are trying to develop. Rather, they show that by training a model on a different language task on an enormous corpus (predicting a target word based on its surrounding context words for CBOW, or predicting the surrounding context words given a target word for skip-gram) the model learns weights for it's vocabulary words which can be extracted used in downstream natural language processing tasks, leveraging the captured semantic content of the embeddings.

It is important to emphasise that whilst the embeddings learnt via this process are *learned* during prediction tasks with many different contexts for each word, the final representation extracted is a 'static' representation of that word. It is a fixed representation that, in its 300 dimensions represents the many different contextual meanings that that word can have (or was seen in).

3.2.2 Implementation

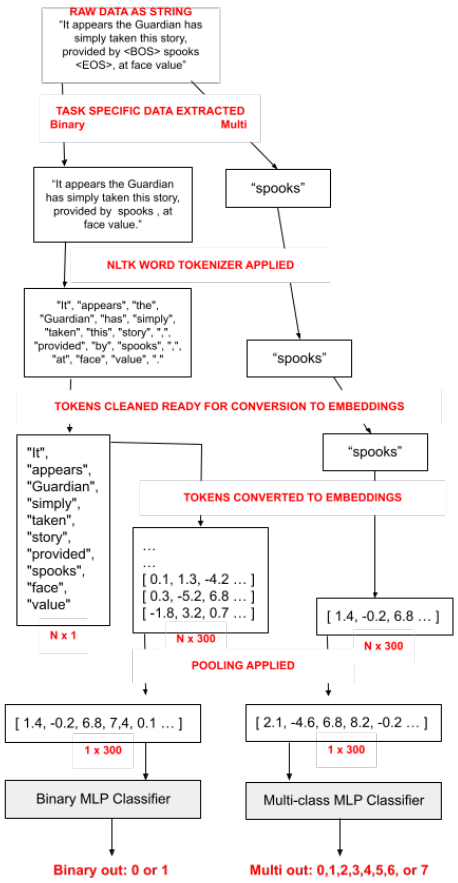


Figure 3: W2V flow chart

As shown in Fig 3, each sentence in string form was tokenised using the popular nltk word-tokenizer [10]. Short and common tokens were removed along with punctuation as as a preprocessing step before obtaining the word2vec embeddings. Removing these items ensured that only words that contribute significantly to the semantic content of the text strings are left and ultimately this means that the pooled representation are likely to be more reflective of the semantic content of the text.

After the raw text is transformed into a list of cleaned tokens of length n containing only the most semantically important tokens, each token is then transformed into its corresponding 300 dimensional word2vec embedding vector.

During this process if a token was not present in the word2vec vocabulary (i.e. it did not have a representation for that specific token) then if stemming (a hyperparameter) was set to be false then the token would be skipped, but if stemming was set to be true, then a packaged nltk stemmer (both porter stemmer and snowball stemmer were used [9]) was applied and the stemmed version of the token was tried against the word2vec vocab. If the stem was present the stemmed version was used. If stemming was false, then any words would just be skipped and so not vectorised.

As the MLP takes a fixed dimensional input and the input text token lists were of variable length, pooling was applied (either max or mean pooling - another hyperparameter) to transform the n by 300 dimensional representation into a 1 by 300 dimensional one.

The choice of pooling and how out of vocabulary words were handled are discussed further in section 3.5 below.

These 1 by 300 dimensional pooled representations of the text were then passed to the MLP classifier shown in Fig. 4 .

This classifier takes the pooled embedding as input and depending on the task has 2 or 8 output neurons which are transformed to a softmax probability distribution across these outputs, with the highest probability effectively being the chosen class by the model.

The MLP was trained as outlined in section 3.4.2 below.

```python
class Custom_MLP_W2V(nn.Module):
    def __init__(self, dropout=0.5, num_classes=2): # num_classes = 8 for multiclass
        super().__init__()
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(dropout)
        self.linear_1 = nn.Linear(300, 128)
        self.bn1 = nn.BatchNorm1d(128)
        self.linear_2 = nn.Linear(128, 128)
        self.bn2 = nn.BatchNorm1d(128)
        self.linear_3 = nn.Linear(128, num_classes)

    def forward(self, inputs):
        x = self.linear_1(inputs)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.linear_2(x)
        x = self.bn2(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.linear_3(x)
        return x
```

Figure 4: W2V MLP structure

3.3 Approach 2 - BERT embeddings and MLP (BERT)

3.3.1 Overview

Bidirectional Encoder Representations from Transformers (BERT) is a state-of-the-art pre-trained language model developed by Google AI Language and introduced by Devlin et al.[2]. Contrary to word2vec, BERT takes a text input and produces dense vector representations for each word that are specific to that word-use instance. That is, the 768 dimension representation produced for each token depends on the context in which that token appeared. For this reason these encoded embeddings are often described as 'dynamic' or 'contextualised'.

As with word2vec these embeddings are generally used for downstream NLP tasks. An extremely

helpful aspect of the BERT model is that as part of the tokenisation processes (BERT has it's own tokeniser which is discussed below) a [CLS] token is added to the beginning of any tokenised input.

This token plays a unique role in BERT. As Jerensky puts it: "The output vector in the final layer of the model for the [CLS] input represents the entire input sequence and serves as the input to a classifier head, a logistic regression or neural network classifier that makes the relevant decision."[5, p255] which is precisely as it is used here.

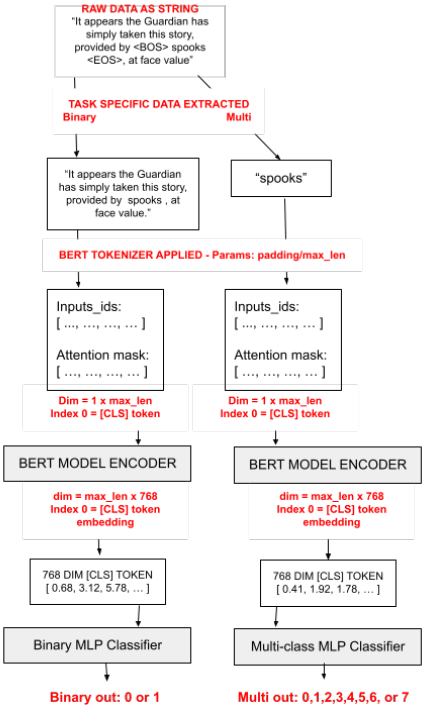


Figure 5: BERT flow chart

The BERT model is considered an enormous step forward in NLP as it utilises the power of transformers and self attention, enabling the model to process input sequences in a bidirectional manner, creating richer, more accurate representations of the contextual meaning of words.

The pretrained BERT model used here [4] was trained on an enormous corpus of unlabelled text, including the BooksCorpus (800M words) and English Wikipedia (2,500M words). It was trained on a number of objectives to ensure it is able to create both rich word embeddings and sequence level embeddings (such as in the [CLS] embedding). This ability is what will be utilised here.

3.3.2 Implementation

The process in which BERT was used can be seen in Fig 5. The BERT model does not take in raw text, so to obtain the BERT-encoded [CLS] representation the text is first tokenized using the BERT tokenizer. This tokenizer actually carries out a number of different steps [4, 5]. Firstly, it carries out tokenisation using a vocabulary of sub-word units learned as part of the pre-training process. These sub-word units, or "word pieces" [2] allow the tokenizer to handle any out-of-vocabulary words by breaking them down into smaller, more common parts which it will have in its vocabulary. It then adds model-specific tokens to the sequence, such as the already [CLS] token at the beginning.

It then converts these tokens into their corresponding integer IDs (the ID that the sub-word has in the BERT model vocabulary), as the BERT model operates on these IDs. Finally, it applies padding or truncation in accordance to user settings, to ensure that the sequences have a consistent length suitable for batch processing.

The resulting output from this process is a sequence of integer IDs representing the tokenised input text, ready to be fed into the BERT model for further processing into the output embeddings. As can be seen in flow chart 5 the tokenisation process actually outputs another vector that is used by the model - an attention mask vector which simply handles the variable length of inputs ensuring that the attention mechanism only pays attention to actual tokens and ignores any padding tokens.

```
class BertClassifier(nn.Module):
    def __init__(self,dropout=0.5,num_classes=2): # num_classes = 8 for multiclass
        super().__init__()
        self.bert=BertModel.from_pretrained('bert-base-uncased')
        self.dropout=nn.Dropout(dropout)
        self.linear=nn.Linear(768,num_classes)
        self.relu=nn.ReLU()

    def forward(self,input_id,mask):
        _, x = self.bert(input_ids=input_id,attention_mask=mask,return_dict=False)
        x=self.dropout(x)
        x=self.linear(x)
        x=self.relu(x)
        return x
```

Figure 6: BERT MLP structure

These vectors are fed into the pretrained BERT model which outputs as it's final hidden layer the contextualised word embeddings for each word and also the embedding for the [CLS] token which then passed forward through the other layers of the MLP.

As can be seen in Fig. 6, the BERT model actually forms the first layer in the MLP, and so the input the network is actually the input IDs and attention mask generated by the BERT tokeniser for each sentence. This process is handled within a custom Pytorch Dataset class.

As part of the MLP the BERT model layer takes in this input and outputs its final layer from which only the [CLS] token representation is passed forward to a linear layer which depending on the task us connected to a 2 or 8 neuron output layer, the outputs of which which are transformed into a softmax probability distribution, with the highest probability effectively being the class chosen by the model.

As it forms part of the neural network it should be noted that the BERT model parameters were trained along with all other parameters of this network. This means the new classifying layers were tuned to the tasks, but so were the weights in BERT, although it is likely owing to its size the most significant updates will have been in these last layers. This may account for the extreme over-fitting seen in the results section.

3.4 MLP

3.4.1 Overview

A Multi-Layer Perceptron (MLP) Classifier is a simple feed-forward artificial neural network that consists of an input layer, one or more hidden layers, and an output layer. In the approaches above, the MLP Classifier takes a vector representations a passage of text (or two in the case of the BERT approach - an input ID vector and an attention mask vector), and it learns to classify that input by having an output layer equal to the number of classes in the task.

The outputs of the final hidden layer are fed into the c dimensional output layer, which outputs logits which are converted to a probability distribution across the c dimensions by a softmax activation function.

During training, the MLP Classifier is optimised using the built in categorical cross-entropy loss function in Pytorch which converts the raw logits of the output layer into a softmax distribution, which it then compares with a one-hot encoded representation of the true class. It effectively measures the dissimilarity between the predicted probability distribution of the model (the softmax distribution) and the true class distribution (the one-hot encoded true class label). The goal of training is to minimise this loss by adjusting the weights of the network through backpropagation and gradient descent.

The MLP Classifier is a simple yet effective model for text classification tasks. The idea behind it is that the model through its weights can learn the relationship between the semantic information of the text as represented in the word2vec or BERT embeddings and the classes they belong to.

The flexibility of the MLP architecture allows for the addition of multiple hidden layers and the adjustment of the output dimension based on the specific requirements of the task, as well as some adjustments such as adding dropout and batch normalisation to try and combat overfitting.

3.4.2 Training

The training process for the W2V and BERT-based classifier models across both tasks were the same and specific implementation details can be seen in the code supplied as an appendix to this assignment.

Using the PyTorch deep learning framework, training and validation data were loaded, preprocessed, and converted into PyTorch DataLoader objects. The model were then initialised with the desired hyper parameters and trained using the CrossEntropyLoss criterion as the loss function and the Adam optimiser for parameter optimisation and trained using mini-batch gradient descent with the batch size being a hyper parameter that was experimented with.

For each epoch a forward pass was performed, the loss was computed, and the gradients were calculated using backward propagation. The optimiser updated the model parameters based on the gradients. After each training epoch, the models were set to evaluation mode, and the validation data were used to assess the models' performance. The validation accuracy and loss were accumulated, and the best-performing model state was saved along with the true and predicted labels.

The training results, including accuracy's, losses, hyper parameters, and evaluation metrics, were visualised using matplotlib and saved in a dictionary format as a JSON file. The training process was repeated for different hyper parameter configurations to find the best-performing models. The ultimate goal was to optimise the models' performance by minimising the cross-entropy loss and maximising the accuracy on the validation set. The best-performing models were selected based on their validation accuracy and the results are presented below.

3.5 Hyperparameter Exploration

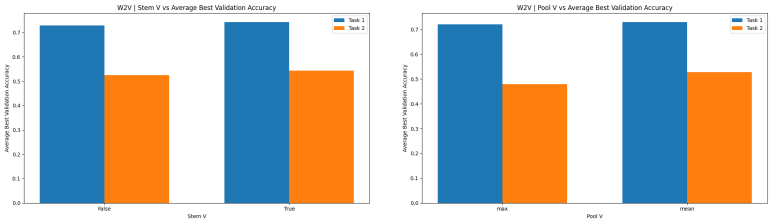
Various hyper parameters were explored during numerous runs across the two tasks. unfortunately it is not possible to display all the results here, however the run data is viewable in the plotting.ipynb file in the appendix.

For the MLP a variety of batch-sizes, learning rates and dropout rates were experimented with. It was found that the optimal dropout rate was 0.5 with little difference made around that value, with significant drop in performance as dropout approached 1, and approached 0. This is understandable as a dropout rate close to 1 sees most of the connections removed from a layer, whereas a dropout rate close to 0 removes any regularisation benefits almost completely, which in the case of overfitting is not desirable.

Batch size was found to make a negligible difference whereas the learning rate appeared to be crucial, especially for the BERT model where a sweet spot between the model failing to learn at all and then overfitting too quickly took some time to establish.

As the tokenisation of the BERT model was handled by the BERT tokeniser utilising the word piece algorithm there was no issue with out of vocabulary words, but it was possible to experiment with the max length setting, but again the initial setting tried (which was the max length of the text snippet in tokens) was the best performing, which made sense as it meant that all the information was captured for all the examples, and there is no detriment as the attention mask masks out any padding so they don't dilute the representation in any way.

The main choices for hyper parameters in the W2V approach were whether to use stemming or not, and which type of pooling to use. As can be seen in the Fig. 7 there was no dramatic impact from either of these measures, although using mean pooling slightly improved average performance in T1, and significantly improved average performance in T2, stemming only showed a very slight improvement. Interestingly the best performing W2V models did not use stemming, although they did both use mean pooling.



(a) Stemming (b) Pooling-Type

Figure 7: Stemming and pooling-type effects on performance of W2V on T1 and T2

It makes sense that mean pooling would be more effective than max pooling as it is possible that it captures the overall semantic content of the sentence better than max pooling. Where max pooling selects the maximum value along each dimension of the word embeddings, mean pooling calculates the average value for each dimension In max pooling it may be that certain extreme key words are emphasised - or the more extreme properties of the sentence which might seem good for propaganda detection. But there are many cases where propaganda techniques are more subtle or involve the use of language that is not explicitly extreme or loaded. By capturing the average meaning of the sentence, mean pooling enables the model to better identify any underlying patterns and characteristics rather than just extracting the extremities.

A number of additional model types and an additional pooling technique were tried, but as they offered little significant improvement and there is little space to describe their nuances. However it may be interesting to note that the exact same approach as W2V but using pretrained Glove Embeddings [8] got very similar results to W2V. The pooling technique smooth inverse frequency (or 'sif') which has been shown to be one of the best performing and yet relatively straightforward approaches to deriving sequence representation from word embeddings [1] also disappointingly did not show significant improvement.

Although many models were trained only the best performing from each approach for each task will be analysed below.

4 Results

Final performance on the two tasks can be seen in Figs. 8 and 9 below. Results are presented in this section with analysis following.

It is clear that overall the BERT approach performed significantly better in both tasks, the details of which shall be explored below. However it should be noted that it took significantly longer to train (up to 50 times longer), and considering the representational limitations and the difficulty of the task the performance of W2V is still impressive and shows the power of even mean pooled static vectors to represent semantic information.

model	best accuracy	best epoch	stemming	pooling	learning rate	batch size
BERT	**0.839063**	17	n/a	n/a	$5e^{-6}$	50
W2V	0.767188	54	False	mean pooling	$5e^{-5}$	100

Figure 8: T2 best of class results

model	best accuracy	best epoch	stemming	pooling	learning rate	batch size
BERT	**0.669903**	34	n/a	n/a	$1e^{-5}$	25
W2V	0.559871	91	False	mean pooling	$5e^{-4}$	128

Figure 9: T2 best of class results

4.1 TASK 1

Inspecting the performance graphs we can see that both models over fit to the training data, and indeed it was a theme across the numerous experiments that reducing overfitting was key to improving performance. Some interesting observations for further analysis are that are that even though it had a significantly lower learning rate, BERT still peaked and began to over fit more quickly than the W2V model. It should also be observed that the W2V model on did not actually reach 100% accuracy on the training set, which the BERT model was certainly approaching. There were many instances where the W2V classifier did achieve near 100% accuracy on the training set, showing that the model capacity was not an issue. A number of architectures were tried in order to reduce capacity in an effort to minimise overfitting and encourage generalisation, and the final one presented in section 3.2.2 was the one that balanced validation performance with training performance best.

Figure 10: W2V best T1 performance

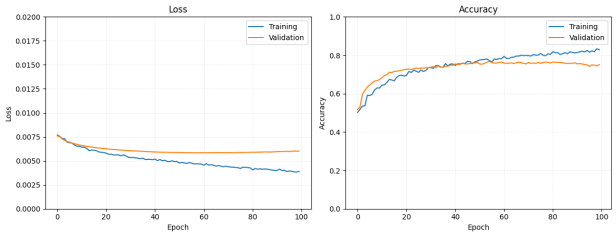
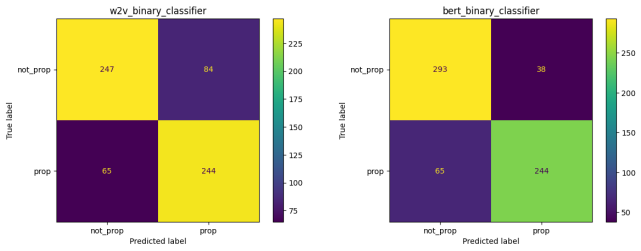


Figure 11: BERT best T1 performance

Figure 12: Confusion Matrices T1 (W2V — BERT)



From inspecting the confusion matrix, we can see that the models perform identically on the true propaganda instances – classifying the exact same amount correctly and incorrectly. Where BERT is significantly better is in correctly identifying instanced of 'not propaganda' with 0.84 recall for not propaganda compared to 0.75 for W2V, meaning the vec model over classifies text as being propaganda.

4.2 Task 2

This is a much harder task owing to the sparsity and variety of the data in many cases (single words and pairs of words are common, especially in the W2V cleaned, tokenised data, see 17 for examples).

Overfitting again was common and again quicker and more extreme in the case of BERT. The BERT model outperformed the W2V model in many areas. Looking the confusion matrices in Fig. 15 and at the the classification report in Fig. 16 we can see the most dramatic improvements in precision in the classes of casual oversimplification, doubt, and appeal to fear/prejudice showing that BERT was able to be more specific in its identification of these compared to W2V. In terms of recall the most dramatic improvements were in repetition, casual oversimplification, name calling/labelling and appeal to fear/prejudice.

Both models struggled with the exaggeration class and loaded language class the most. Loaded language was the one category which W2V actually performed comparatively well, with a precision exceeding that of BERT, although this was balanced by a poorer recall.

Figure 13: W2V best T2 performance

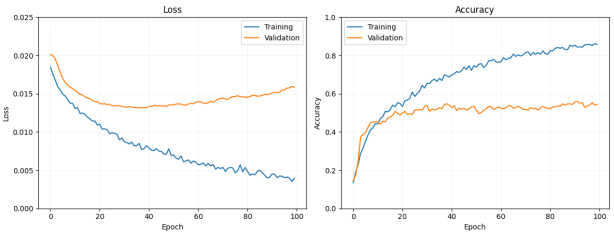


Figure 14: BERT best T2 performance

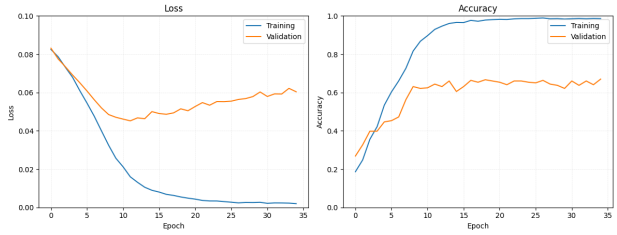
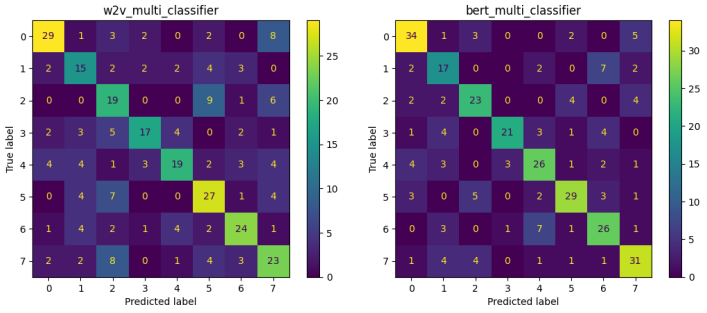


Figure 15: Confusion Matrices T2 (W2V — BERT)

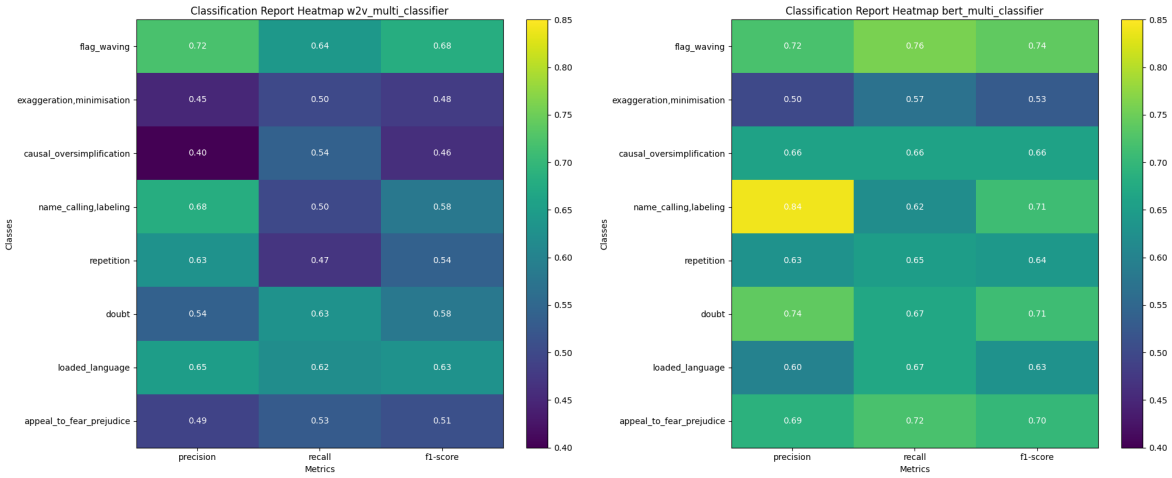


Overall BERT performed substantially better than W2V on both tasks. Some of the possible reasons for this shall be explored in the analysis below.

5 Analysis

To analyse and understand the performance of these two different approaches it is helpful to look at examples of the text they were trying to represent and classify. Fig. 17 shows examples from different classes. These examples were taken from the validation dataset and so were part of those used to evaluate the accuracy of the models.

As I hope can be seen by looking at these examples, there is a degree of arbitrariness to the labelling, as some exaggeration could be considered loaded, and some of the non propaganda texts have elements that could be mistaken for propaganda when taken as isolated pieces of text as they are. Furthermore there are a number of single-word snippets which are ambiguous - it is hard to say which category 'onslaught' would go in if one saw it out of context.



(c) W2V T2 (d) BERT T2

Figure 16: Classification reports

Label	**Text**
0 - Flag Waving	Iran's long rap sheet of aggression against America // Our // Efforts to protect the nation // Latino Voters Show Trump What It Means to Be American
1-Exaggeration/Minimisation	You have to whip them up into a frenzy in order for them to vote// He should never, ever been allowed to// Small // Flimsy and meritless accusations
4 - Repetition	Continent; not a country // Signed by Donald Trump himself // That will never happen // Corruption
5 - Doubt	Sociology. You really don't need to know anything // Accidentally // And just what kind of policies are we talking about // The Catholic Church has a structural problem when it comes to dealing with cover up accusations
6 - Loaded Language	Stop feeding the alligator in the hopes that you will be eaten last // Onslaught // A striking blow against freedom // Smear
7 - Appeal to Fear	Extermination // In the library, silence is golden. In the world of human rights, silence equals mass murder // It is on all of us to debunk them and to educate the public about their scheme
Non prop	Completing this poll grants you access to Freedom Outpost updates free of charge. // Those efforts didn't work // It is not useful to say Mass for the dead? // The Best of Jacob G. Hornberger

Figure 17: Labels with example snippets taken from the validation dataset

It is very impressive, then, that these models were able to perform as well as they did, especially on the multi class task where often single words had to be classified as one of 8 ambiguous, overlapping categories.

In the W2V approach the representation the model received is a 300-dimensional vector that has been averaged across each dimension from n vectors where n is the number of words that have been successfully tokenised. Each one of those vectors is itself a dense vector representation of many possible meanings of the word. These static word embeddings then are simply stacked on top of each other and then averaged in an attempt to get an overall 'average' meaning of the sentence they comprise, albeit with less meaningful words removed.

There are some significant issues with this approach, as key determinants of the meaning of a sentence or even the meanings of the words in a sentence are left out. Word order, negation and other important relationships *between* words are missing.

Representationally, word2vec vectors capture every 'sense' of a word in one static representation. As such, they may capture many important properties of the meaning of a word, but it is the context of a word which often refines or clarifies its specific meaning, and static embeddings used as they are here are unable to capture the refining quality of context. It is also an issue that when averaging across the n vectors of the sentence, as n gets bigger, the averaged vector will tend towards the centroid of the vocabulary's set of embedding vectors - not a good property of a representational system.

Given this, it is perhaps surprising that W2V performed so well, especially on the longer texts in

T1. But it is important to note that a 300 dimensional representation is still rich with information, and in the case of the sort of language that may be present in propaganda, it may be that there are properties of it (strength of meaning, harshness, declarative in nature for example) that are well represented within that abstract 300 dimensional vector space, and it might be this factor that leads to W2V performing well in T1, and relatively well for the loaded language and flag classes. One would perhaps expect it to perform better in exaggeration or minimisation in this case, which it does not. This may be because there are often adverbs that make adjectives more exaggerated or minimal (very bad, incredibly small etc) and these sort of contextual factors will not be captured by W2V.

The main benefit of BERT, on the other hand, is precisely that it DOES capture these contextual factors. The [CLS] token encodes the entire sequence into a single dense vector that represents the overall semantic meaning of the input including relationships and dependencies between words rather than just their individual meanings. This is because BERT's attention mechanism enables it to weigh the importance of different words in a sentence based on their context, enabling the representation to capture nuances and subtleties of language, such as sarcasm, irony, and figurative expressions.

For example, in the sentence"The result was hardly their finest hour," BERT would likely assign more weight to the word "hardly" than to "finest," correctly identifying the overall negative sentiment despite the presence of a typically positive word. W2V in this situation might average out the meaning to a representation that is broadly neutral, or even possibly positive. This ability to represent more complex or sophisticated properties of input texts and over entire sequences meaning can possibly be seen in BERTs significantly improved performance in the more tricky classes where the propaganda techniques may be more subtle such as appeal to fear, exaggeration and minimisation and doubt.

In terms of the general problems with performance – given that both models were able to fit so well to the training data, but were unable to generalise as well to the validations – a key thing to mention is the relatively small data size for training and validation – especially for the multi class classification task. It is possible that with significantly more data these models could both have learned to perform even better. However, for the reasons above I would expect BERT to always outperform W2V, and even with more data there would still remain issues as to how well performance on this particular dataset would translate to detecting propaganda in the real world.

6 Conclusion

BERT has been shown to perform significantly better than W2V in both tasks, and for the reasons given in the analysis above this makes sense.

The deeper problems in propaganda detection are mitigated in this assignment because the data has been pre-labeled and the hard problem of deciding what is and isn't propaganda, and what type of propaganda it is has been handled by those creating the dataset. However, it should be noted that although the text has been labelled as propaganda of a particular type, it does not necessarily mean there is real relationship between the class categories and the text labelled with that class - especially in more subjective cases where humans may struggle to consistently apply those labels. Furthermore, performance in this classification task does not necessarily imply the ability to identify propaganda in the real world.

Interesting further studies may look at unsupervised or semi-supervised learning approaches to propaganda classification. An interesting avenue might be to investigate the intent of a piece of text, and to distinguish that which seems to be intended to inform with that which seems to be trying to pursuade.

References

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *International conference on learning representations*, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Luis Gutiérrez and Brian Keith. A systematic literature review on word embeddings. In *Trends and Applications in Software Engineering: Proceedings of the 7th International Conference on Software Process Improvement (CIMPS 2018) 7*, pages 132–141. Springer, 2019.

[4] HuggingFace. Bert model documentation.

[5] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall, Upper Saddle River, N.J., 2009.

[6] Giovanni Da San Martino, Alberto Barrón-Cedeño, Henning Wachsmuth, Rostislav Petrov, and Preslav Nakov. Semeval-2020 task 11: Detection of propaganda techniques in news articles. *CoRR*, abs/2009.02696, 2020.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[8] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[9] NTLK project. nltk sample useage for stem documentation, 2023.

[10] NTLK project. nltk.tokenize.word$_t$*okenizedocumentation*, 2023.