Fake News: Approaching Automated Lie Detection Using Pre-trained Word Embeddings

Abstract

76% of young adults in the United States consume their news via the Internet. As a result, society is more prone to the spread of falsehoods compared to the pre-Internet era, where broadcasting information on a large scale was only in the hands of large publishing organizations. Due to weaknesses in human perception, an automated approach to recognizing fake news is needed. This paper explores the usage of state-of-the-art natural language techniques to classify fake news. To create a text classifier, techniques for creating a uniform shape are explored. By combining a pre-trained word embedding technique with a logistic regression, an accuracy can be reached that surpasses current research using traditional linguistic features.

Introduction

In the pre-Internet era, the ability to broadcast information on a large scale was in the hands of large publishing organizations. Nowadays, everyone can share news and information via social media with the possibility of reaching a large, global audience (**?**). This introduces risks on validity and authenticity of news, as social media and digital platforms can speed up the spread of falsehoods without requiring much effort from the author (**?**).

As a matter of fact, 63% of adults in the United States prefer to read their news on the Internet. Young adults take the lead: 76% of adults between the ages 18 and 49 get their primary news consumption via the web, compared to just 43% for adults of 50 years and older (**?**). As time passes by, social media is slowly becoming the primary source of news for more and more people.

The main danger of this development is that human perception is often skewed with regards to objectivity of facts. Psychological human traits such as naïve realism let consumers of news belief that their perception is right, while other's perceptions are uninformed. Furthermore, confirmation bias results in consumers preferring information that confirms beliefs they already have (**?**). This makes consumers vulnerable for the spread of misinformation or fake news.

According to the European Commission, *"disinformation - or fake news - consists of verifiably false or misleading information that is created, presented and disseminated for economic gain or to intentionally deceive the public, and may cause public harm"* (**?**). The answer to the problem of fake news as of recently has been to manually fact-check statements on validity, but, as Shu et al. underlines, one of the downsides to this approach is that fake news typically relates to newly emerging, time-critical events. This means the real news may not be fully verified by proper knowledge bases due to a lack of contradicting claims (**?**). An automated approach would both help in solving the problem of human subjectivity and the speed at which false information is spread in the current news consumption landscape. Furthermore, such an approach can help human fact-checking by targetting statements that are most likely to be false.

Natural language processing has been in rapid development over the past years. With the releases of OpenAI's GPT-2 model in February of this year and Google's BERT in the autumn of 2018, state-of-the-art pre-trained textual embedding techniques have shown promising results on various classification tasks (**?**)(**?**). Although fake news classification has been attempted before (**?**)(**?**), performance on these classifiers has been rather low. However, these new pre-trained textual embeddings have not yet been used in the fight against disinformation.

This paper is focussed on the following research question: *what is the performance of combinations of pre-trained embedding techniques with machine learning algorithms when classifying fake news?* This main question will be answered through the results of the following subquestions:

RQ1 Which way of pooling vectors to a fixed length works best for classifying fake news?

RQ2 At what maximum sequence length do neural networks hold the highest accuracy when classifying fake news?

RQ3 How well do neural network classification architectures classify fake news compared to non-neural classification algorithms?

The structure of this paper will be as follows: first, related

work regarding fake news detection, pre-trained word embeddings and preparing text for classification will be discussed. Then, the methodology will be laid out, including throughout explanation of our pre-trained embedding methods. To conclude, these embedding techniques will be applied to the Liar dataset and classification results will be shared.

Related Work

Automatic fake news detection

In past research, there have been several attempts to create classifiers for automatic detection of lies and fake news. Wang used both neural and non-neural classifiers to classify statements from the Liar dataset into 6 possible gradations of truthfulness. Furthermore, he added speaker metadata to improve the result of his classifications. Both with and without introducing speaker metadata, the best performing architecture was found to be a convolutional neural network. With an accuracy of 27% without, and 27,4% with metadata on the test set, Wang was able to perform 6,2% and 6,6% better than the majority baseline of 20,8%(**?**).

From the same dataset, Khurana extracted linguistic features such as n-grams, sentiment, number of capital letters and POS tags to classify the data into 3 labels instead of the original 6 labels. For classification, she used a set of non-neural classifiers. Her best performing classifier, using gradient boosting, obtained an accuracy of 49,03%, which performed around 5% better than the majority baseline of 44,28% (**?**).

The British factchecking organization Full Fact has developed an architecture that is able to monitor and factcheck statements from the British Parliament and major media outlets in the United Kingdom. One of its uses is automatically factchecking the accuracy of statistical claims made by members of the Parliament (**?**). For detecting factual claims from texts, the organization uses InferSent, which is a way of transfer learning on sentence level that has been proved to perform well for the use case of Full Fact (**?**).

Various tools with regards to fake news detection and research are also available. Faker Fact is a tool which can classify texts into a set of categories ranging from satire to agenda-driven, the former identifying humorous intent, the latter identifying manipulation (**?**). For tracking online misinformation, the Observatory on Social Media created Hoaxy. Hoaxy allows for the visualization of the spread of unverified claims through Twitter networks (**?**).

Pre-trained textual embeddings

Traditionally, feature representation for text classification is often based on the bag-of-words model, containing linguistic patterns as features, such as unigrams, bigrams or n-grams. However, these approaches completely ignore contextual information or word order in texts, and are unable to capture semantics of words. As a result, classifiers may be unable to correctly identify patterns, affecting the classification accuracy (**?**).

As an answer to these problems, pre-trained text embeddings have been rising in popularity, both in use and in research. Before classification is possible, text data needs to be transformed into numbers to be able to be interpreted by classification algorithms. Fundamentally, text embeddings are vector representations of linguistic structures, allowing for usage of text in classifiers. The process of turning words into these embeddings is typically powered by statistics gathered from large unlabeled corpora of text data (**?**).

In 2017, Vaswani et al. proposed a novel architecture for embedding raw textual data called the Transformer. With the main aim originally being translating one sentence from one language to another, Transformers are based on an encoder-decoder model. These models take the sequence of input words and convert it to an intermediate representation, after which a decoder creates an output sequence.

The main strength of the Transformer architecture is its focus on attention to create the intermediate representation. The encoder receives a sequence of inputs, and reads it at once, as opposed to sequentially (either from left to right or from right to left, as humans do). This allows the encoder to learn the context of a word based on all of its surrounding text (**?**), as opposed to traditional vector representation techniques that only allow a single context-independent representation for each word (**?**). As shown by the Bidirectional Encoder Representations from Transformer (BERT) model by Devlin et al., these techniques have beaten existing benchmarks in natural language processing, underlining the importance of context in textual data (**?**).

Pooling

Most non-neural classifiers need data in a two-dimensional uniform shape to be able to perform calculations. In the case of raw text data, sentences in datasets often have variable word lengths, resulting in a different vector length when turning the texts into a vector representation. Furthermore, when dealing with vector representations of words, the shape of a statement is turned three-dimensional instead of the needed two-dimensionality for non-neural classifications. To turn the vector representations into a uniform length, we can either cut off the vectors at a fixed length (*padding*), or we can perform calculations to reduce the length and dimensions of the vectors (*pooling*).

In computer vision, feature pooling is often used to reduce noise in data. The goal of this step is to transform joint feature representations into a new, more usable one that preserves important information while discarding irrelevant details. Pooling techniques such as max pooling and average pooling perform mathematical operations to reduce several numbers into one (**?**). In the case of transforming the shape of data, we can reduce vectors to the smallest vector in the dataset to create a uniform shape.

Scherer et al. compared performance of two pooling operations on a convolutional neural network architecture. The first pooling method extracted maximums and the second one was primarily based on working with averages. They have shown that a max pooling operation is vastly superior for capturing invariances in image-like data (**?**).

Shen et al. noted that in text classification, only a small number of key words contribute to the final prediction. As a result, simple pooling operations are surprisingly effective for representing documents (**?**). Lai et al., Hu et al. and Zhang et al. use a max pooling layer in a (recurrent) convolutional neural network for identifying key features in text classification (**?**)(**?**)(**?**). For text classification, max pooling strategies seem to be the most popular.

Padding

When padding a sequence, a list of sequences is transformed to a specific length. Sequences longer than the desired length will be truncated to fit the requirement, while sequences shorter than the desired length will be padded by a specified value (**?**). To fill the sequences, a value of zero is often used. Hu et al. also use zero values for padding their sequences (**?**).

Apart from controlling the size of the feature dimension, padding has other uses as well. Simard et al. make use of sequence padding for convolutional neural networks to center feature units, and concluded it did not impact the performance of the classifier significantly (**?**). Wen et al. apply padding to convolutional network models to prevent dimension loss (**?**).

Neural text classifiers

Wang has shown that neural networks perform slightly better on classifying fake news than non-neural classifiers. In his research, he compared accuracies on support vector machines, logistic regressions, bidirectional LSTMs and convolutional neural networks with each other. With his 6 label classification, his support vector machine implementation was the best performing linear classifier, but the performance was slightly worse than the best performing neural network (25,8% for the former, and 26% for the latter) (**?**).

Wang used two neural network architectures both well known for their robustness and performance when it comes to text classification. The first model, the bidirectional Long Short Term Memory (LSTM) network, is specifically tailored at keeping track of information for a long period of time. This makes the model able to keep track of the context in a more intelligent way when compared to a standard non-neural classification algorithm (**?**).

The second architecture, the convolutional neural network, applies a set of convolving filters in its layers that are applied to local features. These models are shown to be effective in numerous natural language processing applications, such as semantic parsing, search query retrieval, sentence modeling and other traditional NLP tasks (**?**).

Methodology

Description of the data

For classifying fake news, Wang's Liar dataset will be used (**?**). The Liar dataset contains 12.791 short statements from Politifact.com, which are labeled manually by a Politifact.com editor on truthfulness. The speakers in the dataset include both representatives from the two major political parties in the United States, as well as a large amount of social media posts.

id	11044.json
label	pants-fire
statement	The Mexican government forces many bad people
subjects	foreign-policy,immigration
speaker	donald-trump
speaker_job	President-Elect
state	New York
party	republican
context	an interview with NBC's Katy Tur
mostly_true_count	37
half_true_count	51
barely_true_count	63
false_count	114
pants_on_fire_count	61

Table 1: An example entry in the Liar dataset.

The statements are an average of 18 tokens long, and the topics vary from different political subjects, as can be seen in figure 1. Truthfulness is evaluated by assigning one of 6 labels, ranging from *pants-on-fire* to *true*. The distribution of statements across the 6 labels can be seen in table 2.

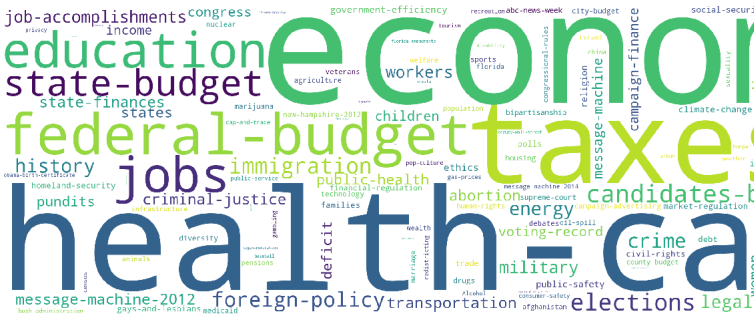

Figure 1: An overview of all statement topics in the Liar dataset.

For each statement, the dataset contains an id, a label, a subject, a speaker, the function title of the speaker, the affiliated state and political affiliation, the context of the statement and a vector with a truthfulness history. An example of such a data entry can be seen in table 1. Wang introduced this truthfulness history to boost the prediction scores, as speakers with a track record of lying are expected to have a lower chance of speaking the truth when classifying new statements. However, for our application we are only interested in the statement itself and its corresponding label. Due to cheapness and spreadability, a large amount of fake news is spread over social media (**?**). This means author information and metadata will not readily be available in real world circumstances.

The original dataset has been split beforehand into a test, train and validation set. The train set contains 80% of the

6 labels	3 labels
true (16.1%)	2*true (35,3%)
mostly-true (19.2%)	
half-true (20.5%)	half-true (20.5%)
barely-true (16.4%)	3*false (44,19%)
false (19.6%)	
pants-fire (8.19%)	

Table 2: Distribution of labels from the original label distribution when reducing the amount of labels.

total amount of statements, while the test and validation set both contain approximately 10% of the statements.

Data preprocessing and cleaning

Filtering statements The original dataset contained statements ranging from 1 sentence to 19. On closer inspection of statements with the high amounts of sentences, it was found that not all statements were processed from source files into dataframes correctly. As a result, records of some different statements were joined together, forming a single string. To combat this, the following regular expression was used to filter those statements out:

```
\\.json\\t(mostly-true|true|half-true|false|barely-true|pants-fire|\
```

After applying this regular expression, the total amount of sentences in the statements were reduced from a maximum of 19 to a maximum of 11. This reduced the original reported size of the dataset by 52 statements to a total of 12784.

Reducing labels Wang's main objective was to classify fake news into a fine-grained category of fakeness (**?**). For our main research question, we aim to predict whether the statements are fake news or not. This means the statements do not necessarily need to be distinguished into these fine-grained categories. Because of this, the classifiers used to predict fake news in this research will be trained on Khurana's division into 3 labels instead of the original 6 (**?**). The division from the original 6 labels into the 3 labels can be seen in table 2. This way, we can compare performance of pre-trained embeddings to Khurana's purely traditional, linguistic approach to classification of this dataset.

Methods

Applying embedding techniques As our main research question is focussed on pre-trained word embeddings, the first step in the classification process is to turn the statements of the Liar dataset into vectors. For this purpose, the Flair framework will be used. Flair contains interfaces for turning words into embeddings, built on the PyTorch platform (**?**)(**?**). Using Flair, we have access to the following 5 state-of-the-art pre-trained embedding techniques:

- ELMo (Embeddings from Language Models) (**?**);
- BERT (**?**);
- Generative Pre-Training (GPT) (**?**);
- Transformer-XL (**?**);
- Flair (**?**);
- GPT-2 (**?**);
- XLNet Add references;
- XLM Add references.

For each of our embedding techniques, we will generate a new dataset with statements from the Liar dataset in their embedded form. These datasets will then be used for classification, without the datasets being specifically adjusted or finetuned for specific models. This allows us to independently compare classification models for each of our embedding techniques.

To apply these embeddings, the Flair framework first requires a sentence object to be created (**?**). For dividing the statements into sentences, the `sent_tokenize` function from the `nltk` package will be used (**?**). After applying this split, Flair's sentence object takes care of tokenization and applying the selected embedding technique. This way, all statements of the Liar dataset will be turned into other representations one by one and saved for future use.

Embeddings from Language Models (ELMo) The first word embedding technique, ELMo, is based on a bidirectional language model. ELMo embeddings are different from regular word embeddings, because each token is a function of the entire input sentence. The underlying neural network architecture consists of a bidirectional LSTM trained on a large corpus. This corpus contained approximately 5.5 billion tokens, crawled from Wikipedia and WMT 2008-2012.

The representation is formed from combining internal states of the LSTM. The higher level LSTM states capture context-dependent aspects of word meaning, while the lower level states model aspect of syntax. Combining these internal states allows for rich representations that capture both complex characteristics of word use (syntax and semantics) and how word uses vary across linguistic contexts (**?**).

The model for our use will be the original ELMo model, containing 93.6 million parameters. Each created word vector has a length of 3072.

Generative Pre-Training (GPT) OpenAI's GPT model is among the first text embedding methods to use the Transformer architecture by Vaswani et al. (**?**), as described in section 2.2. Radford et al. chose specifically for this architecture, as these types of models allow for a more structured memory for handling long-term dependencies in text. This results in a robust transfer performance across different tasks.

GPT's training procedure consists of two stages: learning a language model on a large corpus of text, and a fine-tuning stage, where the model is adapted to a task with labeled data. For learning the language model, the BooksCorpus dataset was used, which contains over 7000 books, and is of comparable size to the datasets used to pre-train ELMo embeddings (**?**).

Each created word vector has a length of 1536.

Flair Just like the ELMo embeddings proposed by Peters et al. (**?**), Akbik et al. proposed Flair embeddings, which are contextualized word embeddings on a character level. According to these authors, *contextual string embeddings are powerful embeddings that capture latent syntactic-semantic information that goes beyond standard word embeddings. Key differences are: (1) they are trained without any explicit notion of words and thus fundamentally model words as sequences of characters. And (2) they are contextualized by their surrounding text, meaning that the same word will have different embeddings depending on its contextual use* (**?**).

Flair embeddings make use of a bidirectional LSTM architecture for language modelling. For pre-training, a sequence of characters is passed to this architecture, which at each point in the sequence is trained to predict the next character. Then, the hidden states of the neural network are used to create contextualized word embeddings.

Akbik et al. also proposed the concept of stacking embeddings, allowing for traditional word embeddings to be combined with Flair embeddings. This has the potential to create word vectors that contain greater latent word-level semantics by concatenating embedding vectors trained on a character level and on a word level (**?**).

For our use, the recommended configuration of stacked embeddings by the Flair repository are used (**?**): this is a combination of GloVe word embeddings (**?**) and both forward and backward Flair embeddings. Each created word vector from these stacked embeddings has a length of 4196, which is the longest vector length of all embedding techniques listed in section 3.3.1.

Bidirectional Encoder Representations from Transformer (BERT) Devlin et al. propose an embedding technique which is also based on the Transformer architecture described in section 2.2, but differs from the GPT Transformer, as it is trained using bi-directional self-attention instead of GPT's constrained left-only self-attention. Pre-training is conducted on 2 unsupervised learning tasks. The first utilizes a masked language model (MLM), which randomly masks some of the tokens from the input with the objective to predict the masked word, based only on its context. This MLM fuses the context to the left and to the right, allowing pre-training of a deep bi-directional Transformer architecture. In the second task, the model takes a sentence, and predicts the next sentence in the sequence. The first task is aimed on word level, while this second learning task aims at the model's sentence level understanding (**?**).

The model used for embedding our statements will be the original BERT Uncased model, which contains 110 million parameters and 12 layers. Pre-training for this model was conducted using a combination of the BooksCorpus (800 million words) and Wikipedia (2500 million words). Each created word vector has a length of 3072.

Transformer-XL Dai et al. build further on the original Transformer architecture with the Transformer-XL model, but add recurrence to allow for the model to conserve hidden states across batches. This means that for each new batch, the model uses previous batches to predict the first few symbols of the current batch. By using this concept of recurrence, the Transformer-XL architecture allows for creating long-term dependencies (**?**).

The model is pre-trained on the WikiText-103 dataset, which contains over 100 million tokens and is particularly focussed on long-term dependencies (**?**). Each created word vector has a length of 1024.

GPT-2

XLNet

XLM

Pooling and padding For our classification tasks, we need to transform the shape of our data to a uniform style. As described in section 2.3 and 2.4, we will make use of pooling and padding techniques to achieve this.

Applying pooling In section 2.3, we have identified three simple mathematical operations to reduce our vector length to a fixed length. We will apply average pooling, max pooling and min pooling and test their accuracies on a classifier to test what pooling technique works best for each embedding technique. For applying these transformations, the NumPy package will be used (**?**). NumPy contains tools for working with multi-dimensional arrays, and allows us to easily manipulate data.

As can be seen in figure 2, applying a pooling operation is rather straight-forward: for every word vector, the floating point number at every nth index is gathered. After that, a mathematical operation is performed to reduce those numbers to a single number, which will become the number in the nth index of the final full statement vector.

Applying padding For neural architectures, three-dimensionally shaped data can be used for classification. However, the problem still remains that each text sequence can have a different length of words. Padding the sequences, as described in section 2.4, will be used to combat this.

For applying padding, we will use the `pad_sequences` function from the Keras library (**?**). This function allows us to specify a maximum sequence length to transform all our statements into.

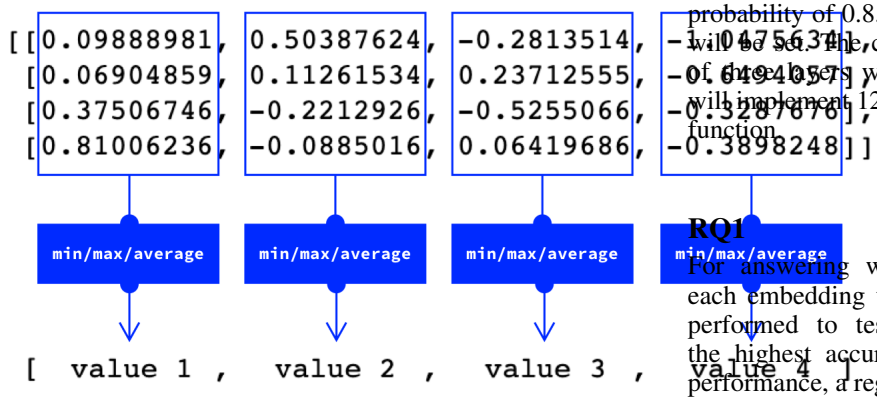

Figure 2: Applying pooling to reduce a matrix to a single vector.

doc2vec Of course, both pooling and padding result in some loss of data. To compare these data densing techniques to a situation in which all words of full sequences are taken into account, results using a doc2vec embedding will also be taken into account. The doc2vec model creates paragraph vectors instead of word vectors, allowing all text in each statement to condense back into one single vector, without specifiying a cut off point or a maximum length (**?**).

Together with the majority vote and the best score from previous research, the doc2vec score will function as a baseline for comparing performance of our classifiers. For gathering doc2vec scores, the doc2vec interface in the Gensim library will be used (**?**).

Classification To make the results of our embedding techniques comparable to existing research, four classifiers of Wang and the top performing model of Khurana will be used to classify our statements (**?**)(**?**):

1. Logistic regression;
2. SVM;
3. Gradient Boosting;
4. Bidirectional LSTM;
5. Convolutional neural network.

For all classifiers, hyperparameters will be tuned on the validation set to keep the experimental settings as close to the original setup by Wang as possible. Performance of the classifications will be reported using accuracy over the test set as an evaluation metric, similar to the research conducted by Khurana and Wang (**?**)(**?**). To reduce as much random effects as possible on the neural classifications, each neural classification will be run 5 times, and the reported score will be the average of those 5 scores.

For the first three non-neural classification algorithms, scikit-learn will be used for implementation (**?**). To tune the regularization strength for the logistic regression and SVM implementation and to tune the learning rate of the gradient boosting, a grid search will be used.

The Keras library will be used for the neural architectures (**?**). Both neural classifiers will use a dropout keep probability of 0.8. A batch size of 32 and 5 training epochs will be set. The convolutional neural network will consist of three layers with a filter size of (2, 3, 4). Each layer will implement 128 filters and will utilize a ReLu activation function.

Evaluation

RQ1

For answering what pooling technique pairs best with each embedding technique, a logistic regression has been performed to test which pooling technique results in the highest accuracy. However, before we can compare performance, a regularization type needs to be specified.

Regularization L_1 and L_2 regularizations have been tested on our max pooled text embedding techniques, and the results are shown in figure 3. For all but one of the embeddings, the L_2 regularization had the upper hand. That embedding technique, OpenAI's GPT, showed a small advantage when using a L_1 regularization, however, the difference was very small. We can conclude that L_2 regularization works best on most of our models.

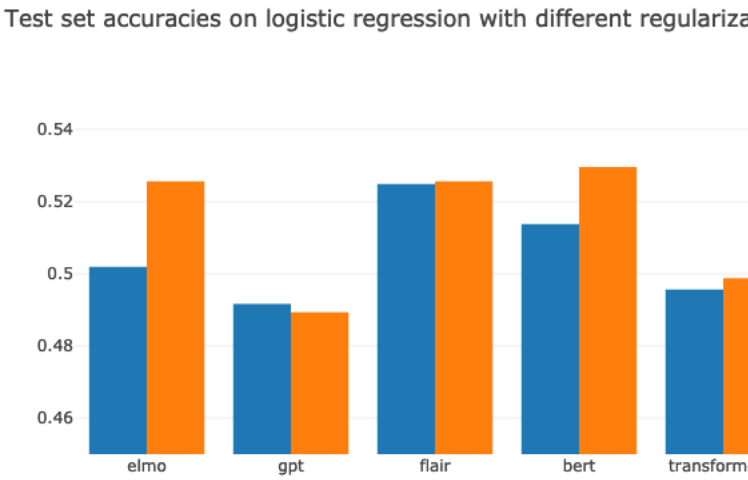

Figure 3: Comparing performance between a L_1 regularization and a L_2 regularization on max pooled datasets.

Comparing performance on a logistic regression model Now it is clear that the L_2 regularization performs best on our model, we can compare performance between different pooling strategies. The results of the classifications are shown in figure 4. Between the 5 embedding techniques, there was no shared top performing pooling operation. For 3 of our embeddings, the max pooling operation resulted in a higher accuracy, while OpenAI's GPT and Flair embeddings gained a higher accuracy with min pooling and average pooling, respectively.

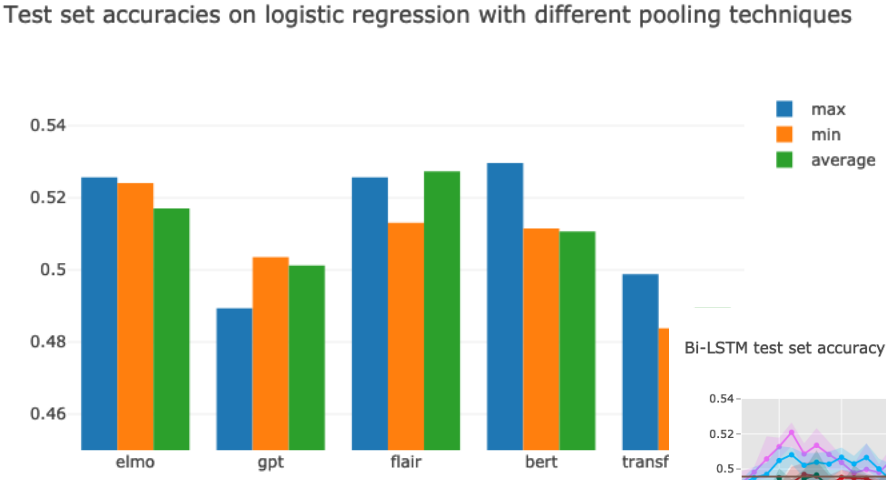

Figure 4: Comparing different pooling strategies.

RQ2

To find out which sequence length is optimal for neural classification, data with variable amounts of padding has been fed into bidirectional LSTMs and convolutional neural networks. The lengths between two standard deviations from the median of the total word lengths per statement have been used as a cutoff point, as is illustrated in figure 5. This allows the highest cutoff point ($median + 2 \times \sigma \approx 36$) to still contain approximately 80% of the tokens of the longest statement.

In figure 6, the accuracies with different amounts of

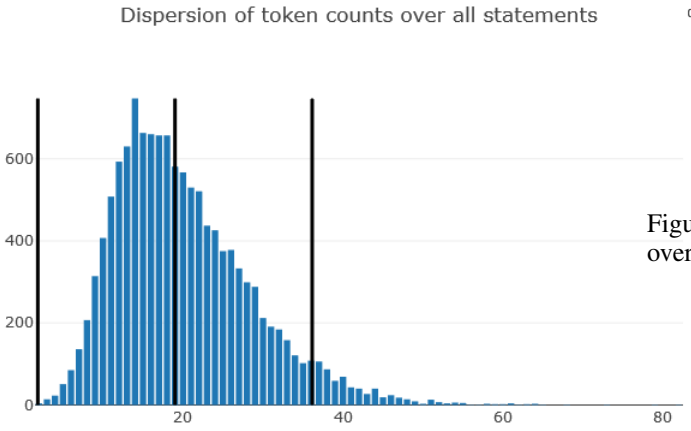

Figure 5: Total token counts per statement on the Liar dataset.

[b]1

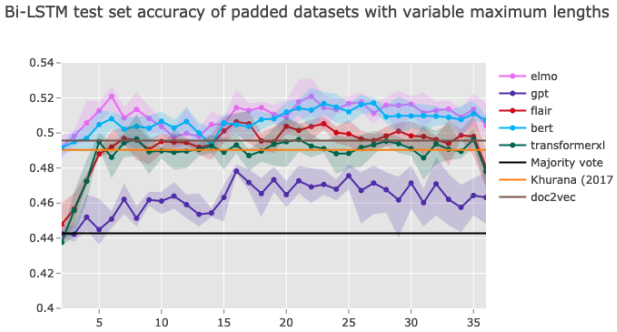

Figure 6: Bidirectional LSTM accuracies.

[b]1

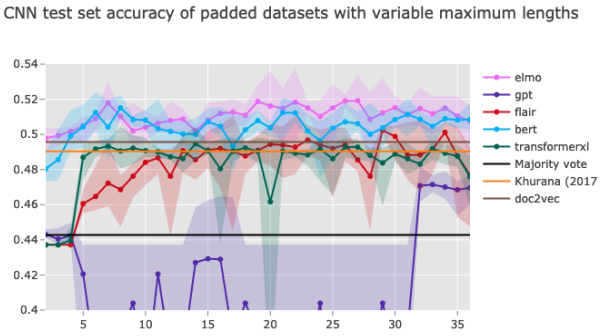

Figure 7: Convolutional neural network accuracies.

Figure 8: Test set accuracies with variable maximum lengths over neural network architectures.

padding can be seen compared to our 3 baselines: Khurana's best score, the majority baseline and our own best score using the doc2vec language model. In both architectures, performance did not significantly deteriorate or increase, but remained somewhat linear, regardless of how much padding had been inserted. From this, we can conclude that most of our word embeddings do indeed carry sentence-level context with them. During training, the whole sentence is taken into account when forming word embeddings, and traces of the whole context are embedded into single word embeddings. This allows for a stable performance, regardless of how many word embeddings are included in the classification.

The best maximum length seems to lay a litte bit above our average word length of 19, with the top performing model, the ELMo embedding, having an accuracy of 52,09% on the bi-LSTM architecture with a maximum length of 22. On the CNN, ELMo embeddings also perform the best, with the best accuracy being 51,92% when the word embeddings are cutoff at a maximum of 27.

RQ3

To conclude our search to the best combination of text embeddings with classifiers, performance differences between neural and non-neural classification algorithms have been evaluated. The results are shown in figure 7.

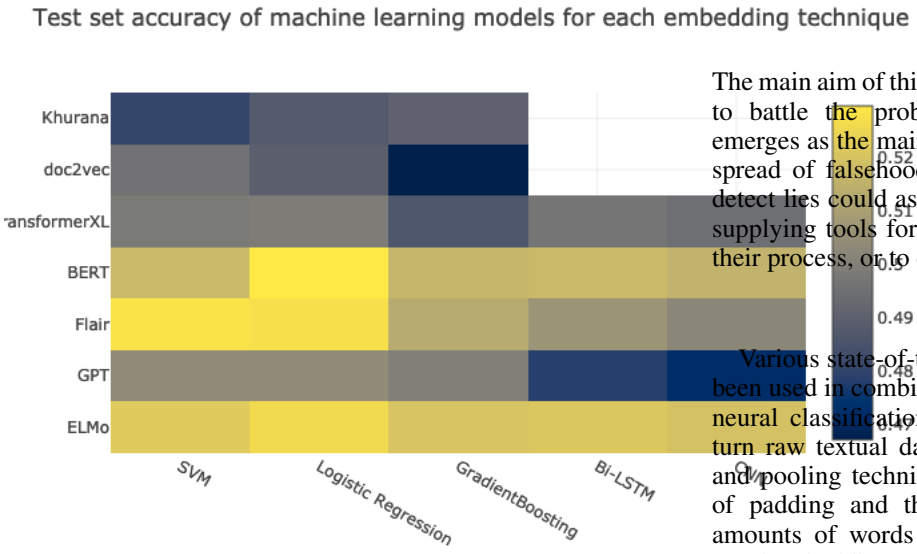

Figure 9: Comparing different non-neural classifiers with neural classifiers.

Clearly, neural architectures either performed worse, or were equal to non-neural classifiers. We can conclude that a combination of pre-trained (neural) embedding techniques and regular, non-neural classifiers allow for the best accuracy, as opposed to both a neural embedding and a neural classification architecture. Of all our tested combinations, a combination of a logistic regression with BERT embeddings performed the best. This combination resulted in an accuracy of 52,96%, almost 4% better than research conducted by Khurana, 3,4% better than the best doc2vec combination, and 8,86% better than the majority baseline.

When applying this combination on the original dataset with 6 labels, an accuracy of 27,51% has been achieved. Research by Wang reached 27% accuracy without introducing speaker metadata (**?**), making the combination of BERT and a logistic regression the best performing classification method at the time of writing.

Conclusions

The main aim of this thesis has been to apply new techniques to battle the problem of fake news. As social media emerges as the main platform for consumption of news, the spread of falsehoods is increasing. An automated way to detect lies could assist to combat this development by both supplying tools for factchecking organizations to speed up their process, or to counter subjective human intuition.

Various state-of-the-art word embedding techniques have been used in combination with a set of both neural and non-neural classification algorithms to classify fake news. To turn raw textual data into a uniform shape, both padding and pooling techniques have been compared. The amount of padding and the robust performance even on lower amounts of words allowed us to believe that pre-trained word embeddings above all contain contextual data, making an increase of word vectors unnecessary, as a small amount of word vectors already contain a large chunk of data concerning the context of the whole sentence.

To conclude, the highest achieved accuracy was 52,96% with a combination of BERT, which is a Transformer-based embedding technique, and a logistic regression. This combination performed almost 4% better than recent research to this same dataset, but using traditional linguistic features.