Fake news: an algorithmic perspective on fact-checking

Martijn B.J. Schouten
11295562

Bachelor thesis
Credits: 12 EC

Bachelor's degree Information Science

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. M. J. Marx

ILPS, IvI
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

2019-06-21

Abstract

Add an abstract

Contents

1 Introduction

The ability to broadcast information on a large scale has been in the hands of large publishing organizations in the pre-Internet era, but nowadays everyone can share news via social media [11]. This introduces risks on validity and authenticity of news, as social media and digital platforms can speed up the spread of falsehoods without much effort from the author [7].

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 [21]. 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. 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 [37]. 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"* [7]. 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 [37]. 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 spreading 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 [29][10]. Although fake news classification has been attempted before [40][14], performance has been rather low. However, these new pre-trained textual embeddings have not yet been used in the fight against disinformation.

This thesis 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?

Add an overview of thesis section

2 Related Work

2.1 Automatic fake news detection

There have been several attempts in the past to create classifiers for automatic detection of lies and fake news. Wang used both linear and 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%[40].

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% [14].

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. It can automatically factcheck the accuracy of statistical claims, for example [4]. For detecting factual claims from texts, the organization uses InferSent, which is a way of transfer learning that has been proved to perform well for the use case of Full Fact [16].

Various tools with regards to fake news detection 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 [1]. Hoaxy, on the other hand, allows for the visualization of unverified claims through Twitter networks [35].

2.2 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 [17].

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 classifier 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 [20].

In 2017, Vaswani et al. proposed a novel architecture for embedding text data called the Transformer. With the main aim originally being translation from one sentence in some language to another sentence in another language, Transformers are based on an encoder-decoder model. These models take the

sequence of input words, convert it to an intermediate representation, after which the 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 the surrounding text [39].

Inspired by the Transformer architecture, numerous new text embedding techniques have been developed which give the possibility to classify texts by making use of the intermediate representation of the Transformer model. 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 text embeddings [10].

2.3 Pooling

Linear classifiers need data in a two-dimensional shape to be able to perform calculations. In the case of raw text data, sentences in the dataset have variable word lengths, resulting in a different vector length when turning the text into a vector representation. 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 of the vectors (*pooling*).

In computer vision, feature pooling is 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 [5]. In the case of transforming the shape of the 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 [34].

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 [36]. 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 [17][12][42]. In the case of text classification, max pooling strategies seem to be the most popular.

2.4 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 [13]. To fill the sequences, a value of zero is often used. Hu et al. also use zero values for padding their sequences [12].

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 [38]. Wen et al. apply padding to convolutional network models to prevent dimension loss [41].

2.5 Neural text classifiers

Wang has shown that neural networks perform slightly better on classifying fake news than linear 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) [40].

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 this model able to keep track of the context in a more intelligent way when compared to a standard non-neural classification algorithm [22].

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 [15].

3 Methodology

3.1 Description of the data

For classifying fake news, Wang's Liar dataset will be used [40]. The Liar dataset contains 12.791 short statements from Politifact.com, which are labeled manually by a Politifact.com editor on truthfulness. 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 original 6 labels can be seen in table 2.

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 [37]. This means author information and metadata will not readily be available in real world circumstances.

Check whether this number is still correct

Check whether this number is still correct

Check whether this number is still correct

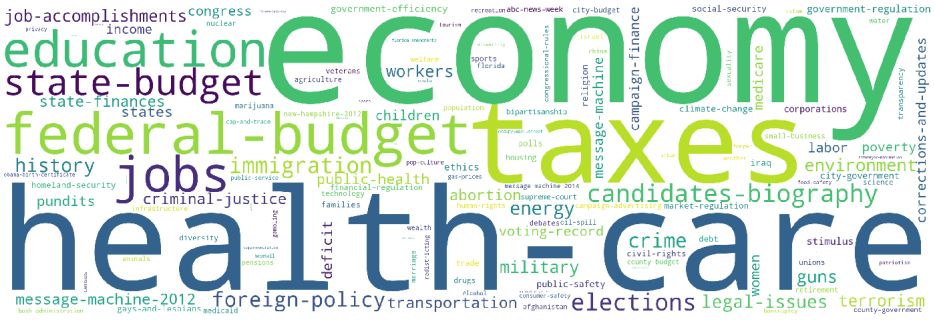

Figure 1: An overview of all statement topics in the Liar dataset.

id	11044.json
label	pants-fire
statement	The Mexican government forces many bad people into our country.
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 original dataset has been split beforehand into a test, train and validation set. The train set contains 80% of the total amount of statements, while the test and validation set both contain approximately 10% of the statements.

3.2 Data preprocessing and cleaning

3.2.1 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.

6 labels	3 labels	2 labels
true (16.1%)	true (35,3%)	true (55,8%)
mostly-true (19.2%)		
half-true (20.5%)	half-true (20.5%)	
barely-true (16.4%)	false (44,19%)	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.

3.2.2 Reducing labels

Wang's main objective was to classify fake news into a fine-grained category of fakeness [40]. 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 the original 6 labels, Khurana's division into three labels [14], and a binary classification. The division from the original 6 labels into the lesser amounts of labels can be seen in table 2. This way, we can better compare performance of pre-trained embeddings to existing research on this dataset.

3.3 Methods

3.3.1 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 [33][27]. Using Flair, we have access to the following 5 state-of-the-art pre-trained embedding techniques:

- ELMo (Embeddings from Language Models) [25];
- BERT [10];
- Generative Pre-Training (GPT) [28];
- Transformer-XL [8];
- Flair [2].

To apply these embeddings, the Flair framework first requires a sentence object to be created [32]. For dividing the statements into sentences, the

`sent_tokenize` function from the `nltk` package will be used [26]. After applying this split, Flair's sentence object takes care of tokenization and applying the selected embedding technique.

3.3.1.1 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[25].

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

3.3.1.2 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 [39]. 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 [28].

Each created word vector has a length of 1536.

3.3.1.3 Flair

Just like the ELMo embeddings proposed by Peters et al. [25], 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* [31].

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 potentially created word vectors that contain greater latent word-level semantics by concatenating embedding vectors trained on a character level and on a word level [3].

For our use, the recommended configuration of stacked embeddings by the Flair repository are used [31]: this is a combination of GloVe word embeddings [24] 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.

3.3.1.4 Bidirectional Encoder Representations from Transformer (BERT)

Devlin et al. propose a 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 [10].

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 (2.500 million words). Each created word vector has a length of 3072.

3.3.1.5 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 [8].

The model is pre-trained on the WikiText-103 dataset, which contains over 100 million tokens and is particularly focussed on long-term dependencies [19]. Each created word vector has a length of 1024.

3.3.2 Pooling and padding

For our classification tasks, we need to transform the shape of our data to a uniform style.

3.3.2.1 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 accuracy on a classifier to test what pooling technique works best on each embebdding technique. For applying these transformations, the NumPy package will be used [9]. NumPy contains tools for easily working with multi-dimensional arrays, and allows us to easily manipulate data.

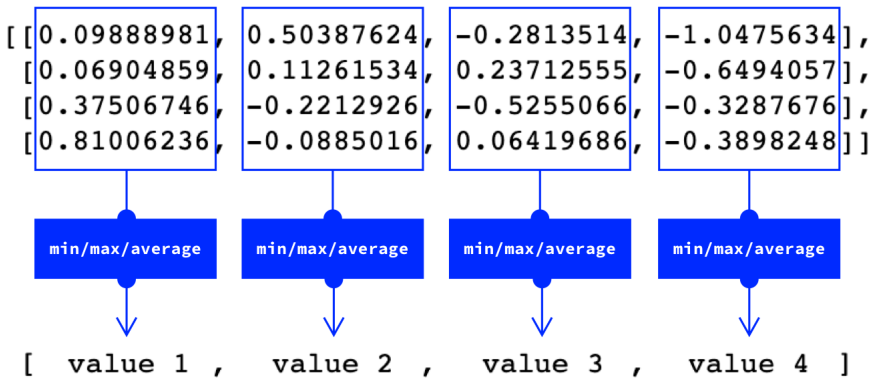

Figure 2: Applying pooling to reduce a matrix to a single vector.

As can be seen in figure 2, applying a pooling operation is rather straightforward: 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.

Check whether this number is still correct

3.3.2.2 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 [13]. This function allows us to specify a maximum sequence length to transform all our statements into.

3.3.2.3 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 [18].

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 [30].

3.3.3 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 [40][14]:

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. As reported by Wang, because the Liar dataset is balanced, f-measures and accuracies are expected to be similar [40]. Because of this, performance of classifications will be reported using accuracy over the test set as an evaluation metric. Each classification will be run 5 times, and the reported score will be the average of those 5 scores, with the aim of reducing as much random effects as possible.

For the first three non-neural classification algorithms, scikit-learn will be used for implementation [23]. 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 [6]. Both neural classifiers will use a dropout keep probability of 0.8 without a L_2 penalty. A batch size of 64 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.

4 Evaluation

4.1 RQ1

Which way of pooling vectors to a fixed length works best for classifying fake news?
The answer to this question will be given by comparing accuracy of a logistic regression with different pooling strategies (max, min, average).

4.2 RQ2

At what padding sequence length do neural networks hold the highest accuracy when classifying fake news?
The answer to this question will be given by comparing accuracy of a bidirectional LSTM architecture with variable padding lengths.

By comparing all embedding techniques, I plan on getting a peak padding length with the best overall performance shared by all embedding methods. At this moment, the optimal length averages at around a length of 22.

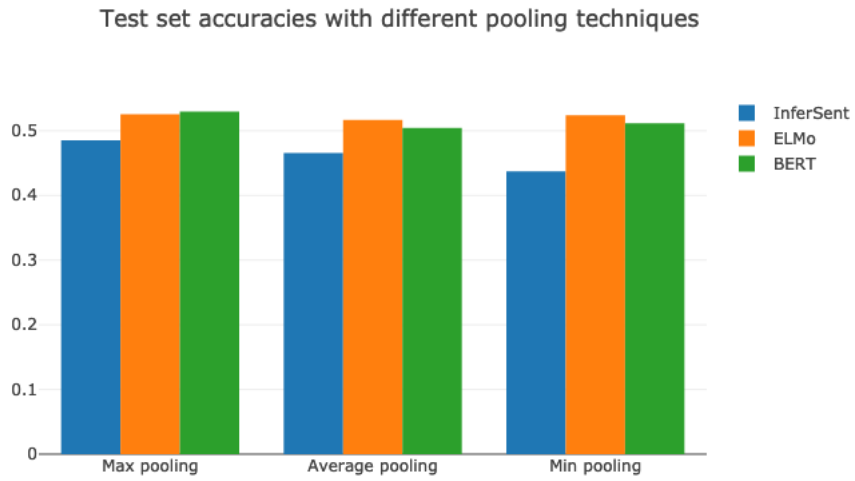

Figure 3: Comparing different pooling strategies.

4.3 RQ3

How well do neural network classification architectures classify fake news compared to non-neural classification algorithms?
The answer of this question will be given by comparing two linear classifiers with two neural classifiers.

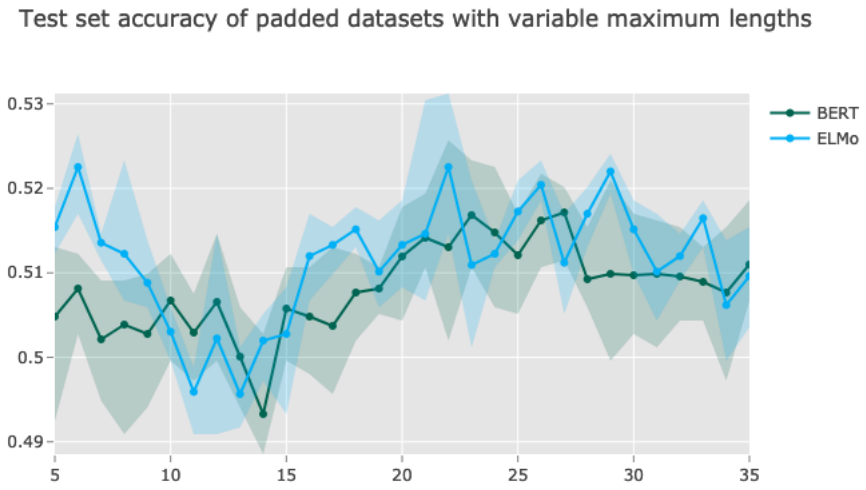

Figure 4: Comparing different maximum padding lengths.

5 Conclusions

5.1 Acknowledgements

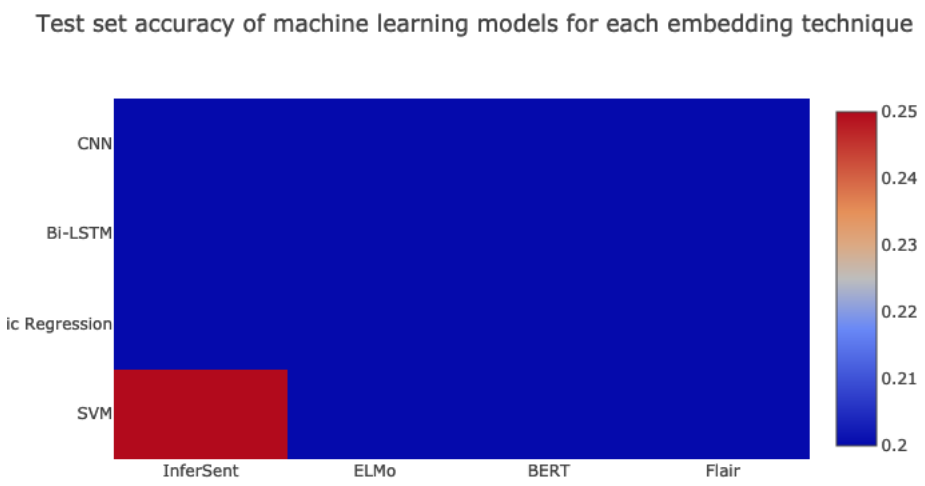

Figure 5: Comparing linear classifiers with neural classifiers.

References

[1] About fakerfact. `https://www.fakerfact.org/about`. Retrieved on 29th of May 2019.

[2] Alan Akbik, Tanja Bergmann, and Roland Vollgraf. Pooled contextualized embeddings for named entity recognition. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, page to appear, 2019.

[3] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[4] Mevan Babakar and Will Moy. The state of automated factchecking. Technical report, Full Fact, 2016.

[5] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[6] François Chollet et al. Keras. `https://keras.io`, 2015.

[7] European Commission. Fake news and online disinformation. `https://ec.europa.eu/digital-single-market/en/fake-news-disinformation`, 2018. Retrieved on 8th of April, 2019.

[8] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

[9] NumPy developers. Numpy. `https://www.numpy.org/`. Retrieved on 17th of June 2019.

[10] 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.

[11] Lee Howell et al. Digital wildfires in a hyperconnected world. *WEF Report*, 3:15–94, 2013.

[12] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050, 2014.

[13] Keras. Sequence preprocessing. `https://keras.io/preprocessing/sequence/`.

[14] Urja Khurana. The linguistic features of fake news headlines and statements, 2017.

[15] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[16] Lev Konstantinovskiy. Sentence embeddings for automated factchecking - lev konstantinovskiy. `https://www.youtube.com/watch?v=ddf0lgPCoSo`. Retrieved on 29th of May 2019.

[17] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[18] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[19] Stephen Merity. The wikitext long term dependency language modeling dataset. `https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/`, 2017. Retrieved on 17th of June 2019.

[20] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. *CoRR*, abs/1712.09405, 2017.

[21] Amy Mitchell and Hannah Klein. Americans still prefer watching to reading the news – and mostly still through television. 2018.

[22] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, August 2015.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[25] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[26] NLTK Project. nltk.tokenize package. `https://www.nltk.org/api/nltk.tokenize.html`. Retrieved on 10th of June 2019.

[27] PyTorch. From research to production. `https://pytorch.org/`. Retrieved on 10th of June 2019.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf*, 2018.

[29] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[30] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[31] Zalando Research. Bert, elmo, and flair embeddings. `https://github.com/zalandoresearch/flair/blob/master/resources/docs/TUTORIAL_4_ELMO_BERT_FLAIR_EMBEDDING.md`, 2018. Retrieved on 17th of June 2019.

[32] Zalando Research. Nlp base types. `https://github.com/zalandoresearch/flair/blob/master/resources/docs/TUTORIAL_1_BASICS.md`, 2018.

[33] Zalando Research. Flair. `https://github.com/zalandoresearch/flair`, 2019.

[34] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[35] Chengcheng Shao, Giovanni Luca Ciampaglia, Alessandro Flammini, and Filippo Menczer. Hoaxy: A platform for tracking online misinformation. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 745–750, Republic and Canton

of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[36] Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. *CoRR*, abs/1805.09843, 2018.

[37] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *CoRR*, abs/1708.01967, 2017.

[38] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[40] William Yang Wang. "liar, liar pants on fire": A new benchmark dataset for fake news detection. *CoRR*, abs/1705.00648, 2017.

[41] L. Wen, X. Li, L. Gao, and Y. Zhang. A new convolutional neural network-based data-driven fault diagnosis method. *IEEE Transactions on Industrial Electronics*, 65(7):5990–5998, July 2018.

[42] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.