NLP based fake news detection
A social media approach

Pablo González, Jeffrey Lee
Georgia Institute of Technology
pablogv@gatech.edu, jlee3703@gatech.edu

Abstract

Fake News has become a relevant and challenging problem due to the boom of digital content in modern society. Many misinformed articles are intentionally written to manipulate the public's view and achieve a certain goal. Due to the fluid and dynamic nature of textual content, there is always ongoing research and currently no consensus on the best modeling architecture to detect fake-news. The purpose of this paper is to investigate a new approach for fake-news detection, where we use Doc2Vec to create fixed size embeddings for each news article (from ISOT Fake News Dataset), and feed this into a fully connected network. To evaluate how well this performs, we compare this to approaches commonly used in this space, specifically using Word2Vec with a variety of different neural network architectures. The Doc2Vec approach achieved an accuracy of 99.03%, compared to 94.48% from Word2Vec approach, confirming that the new approach has superior performance.

1. Introduction/Background/Motivation

Fake News has become a relevant and challenging problem due to advancement in technology enabling the rapid sharing of digital information. This has widespread implications in political, social and economic aspects. For example, in the 2016 US presidential election, it's found that 25% of the tweets from Twitter [1] contain either fake or extremely biased news, which may have directly or indirectly affected the supporters and swing voters of the presidential candidates. A more recent example would be the global coronavirus pandemic, where all sorts of new articles made contradictory claims on the effectivenes of various measures, drugs and vaccines. For these reasons, it is increasingly important that there are ways to detect fake news.

This is not a new problem, and many organisation and researchers have tried to use various natural language modeling (NLP) and deep learning techniques to tackle this issue. NLP is considered as one of the most challenging topics in machine learning. Unlike image processing, whereby the image features can be encoded by sophisticated convolutional neural networks, it's extremely hard to vectorize the text for which the meaning is affected by many factors including the communicator, the context of the sentence etc. In the NLP field, online fake news detection is becoming one of the key challenges due to the boom of digital media, including social media and self-media.

There have been a variety of approaches used in literature around fake news detection. Federico Monti et.al. [2] propose three different approaches; content based - which relies on differences in the text content, context based - which looks at other information such as user, location etc. and finally an innovative propagation-based approach - looking at how fake news spread, and postulates that fake news and true news spread differently, much like an epidemic. Despite these innovative approaches however, the most commonly used approach that has seen success is a content-based approach, which involves encoding text to numeric vectors using embedding algorithms and feeding this into a deep learning model. This has been shown by benchmark comparisons [3] to outperform other more traditional approaches significantly. The most commonly used embedding approach is using GloVe, although Word2vec and other algorithms have also been used. The state-of-the-art deep learning architecture from various papers [4][5] use a hybrid combination of CNN and RNN/LTSM, which take as inputs the embedded text. This has been shown to outperform a single type of deep learning architecture.

In this paper, we will be exploring different text embedding algorithms and deep learning architectures to understand how effective they are at detecting fake news article. The aim is to establish which methods and techniques are effective for the specific task of fake news detection, and direct future research based on what shows promise. The dataset that will be used is the ISOT Fake News Dataset. This is a compilation of several thousands fake news and truthful articles, obtained from different legitimate news sites and sites flagged as unreliable by Politifact.com. This dataset contains the text for each news article and the subject. We will only be using the text field, since we will not always have a subject defined for each article, thus including this would limit the applicability of the fake news detection model.

We will be exploring two different approaches to creating embeddings. The first is similar to existing research, where we will be using Word2Vec to create embeddings for each word. We will then convert each article to a sequence of tokenised words, and use a bi-directional Long Short-term Memory (LSTM) vs Convolutional neural network (CNN) to classify news articles as fake or true. Our second approach will be an approach that has not been commonly used in research, involving converting each article into a single embedded vector using Doc2Vec. Unlike the previous approach, the entire document is embedded at once, and all articles will have a fixed dimension. As a result, a standard multi-layer perceptron will be used as the neural network, as there is no sequence and thus no advantage in using complex architectures such as CNN and LSTM. The results of these two experiments will show which embedding method shows more promise in the field of fake news detection.

2. Approach

The goal is to build a classifier to predict which news articles are fake. The key steps involved are to first clean the data to ensure it is suitable for modeling. The articles then need to be converted to numeric representation using text embeddings to ensure that they can be fed into downstream models as inputs. Once these are embedded, we will then use various neural network architectures to classify each article as Fake or True. Each of these steps will be described in detail in the sections to follow.

We split the new articles into three separate datasets, namely the train, validation and test dataset. 80% of the data is used in training, and 10% will be used for both the validation and test sets. We will only use the train data set for building the Doc2Vec model for text embedding, as well as training the neural network. The validation set will also be used to tune the hyper-parameters of the neural network. Finally, the test set will be used for final evaluation of the models. The test set will not be used for any other purpose, to ensure that we can accurately evaluate the model.

2.1. Data Preparation

The ISOT Fake News Datasets contains two separate files, one for articles labelled True and another for those labelled as Fake. Both datasets contain the following columns:

- **title**: Title of the news article
- **text**: Body of text for the news article
- **subject**: Subject of the news article
- **date**: Date the article is written

From these columns, we will only be using the 'text' field. This is because we do not want to restrict the application of this model to only those article that come with a title. For similar reasons, we do not use subject as these may not be well defined or readily available in real life. The date will not be meaningful to include, since during inference this will be dates in the future which have not been learnt by the model.

In total (across both datasets), there were 44898 news articles. Of these news articles, it was found that 5793 of these were duplicated, which leaves 39105 after deduplication. The label was defined to be 1 for Fake news, and 0 for True news, making this a classification problem. The label is very balanced, with 52% of the articles being True, and 48% marked as False, which means there is no need to solve an imbalanced label problem.

Within the 'text' field in the dataset, it was also noticed that certain words referencing news sources were included in the articles. After examining the proportion of fake news under each source, it was clear that news that contained the word 'Reuters' were nearly always 'True' news. On the other hand, many other news source such as huffington were primarily 'Fake' news. Given this, it was decided that these words should be removed from all the articles, both so that the model does not learn to just determine the label based on the source, as well as the fact that we do not want to limit the model's application to specific sources of news.

Table 1. Proportion of Fake News by keyword

Referenced Agency	Proportion (%)
CNN	73%
Reuters	1%
Bloomberg	59%
ABC	69%
Fox	77%
Buzzfeed	85%
CBS	70%
Huffington	93%
Yahoo	77%

2.2. Text preprocessing and embedding

There are two main considerations for creating numeric vectors for each article. These are:

- **Preprocessing**: Includes removing stop words, converting to lower case, stemming and lemmatization, among other related actions.
- **Embedding**: Conversion of the cleaned article into a numeric representation.

The ultimate goal is to produce meaningful numeric representations which allows the downstream models to differentiate and extract meaning from the news articles.

2.2.1 Preprocessing

Current research indicates that there is no gold standard for the best way to preprocess text. This is because the downstream use-case and algorithm used for embedding will affect how the text should be preprocessed. We convert all the text to lower case, since generally it doesn't make sense to treat words with upper and lower case differently. Similarly, we also decide to apply stemming and lemmatization to all the words to ensure that those words retain the same meaning. However we will experiment with both removing stop words and keeping them in. This is particularly important for Doc2Vec, where stop words could potentially be important for understanding surrounding context. Given that there is no clear consensus in current research on the preferred approach, we will try both methods.

2.2.2 Embedding

For embedding, we will consider two different approaches. In the first approach, we will use Word2Vec to obtain an embedding for each word. We will also tokenize each word, thus each article will be converted to a sequence of tokenized words, with each word having numeric representation from the weights of the Word2Vec model. We capped the input sequence to 700 words since nearly all the articles fell within this word limit with the exception of a few outliers, and zero padded those with fewer than 700. Word2Vec has two approaches which both work well in practice:

- **Skip-gram**: Given a word, we attempt to predict its surrounding words
- **Continuous bag of words (CBOW)**: Given a set of surrounding words, we attempt to predict the center word.

In both cases, intuitively we are saying that the context of the word (i.e. words surrounding the given word) should be similar if two words are similar. This is formalised as a semi-supervised problem, where we have our inputs and outputs (as defined above), and we train a neural network. Ultimately, the weights of the hidden layer will be our numeric encoding for each word in the case of Word2Vec. The window size (number of words on either side of the center word) used in the experiments will be 3 words.

In our second approach, we embed each news article as a numeric vector, using the Doc2Vec algorithm. This is similar to Word2Vec, however one additional change is to also have a paragraph id as an input when training, which represents the encoding for the context of the document. We have treated each news article as a document, thus each article will have its own paragraph id (during training only, not for inferring vectors). This paragraph id will then also be an input (along with the words in the news article).

Similar to Word2Vec, Doc2Vec also has two approaches:

- **Paragraph Vector - Distributed Memory (PV-DM)**: This is similar to skip-gram, but it randomly samples from the paragraph consecutive words as context and we predict the center word
- **Distributed Bag-of-Words (DBOW)**: This is similar to CBOW, however the input is not the center word but a paragraph id, and the target is randomly sampled words from the document

Once we have trained a Doc2Vec model using this approach, we then re-infer vectors for the training, validation and test sets. We note that only the training set was used to train the Doc2Vec model, to ensure no leakage. The end result of this is a numeric vector for each news article. Note that unlike the Word2Vec approach, we directly embed each news article as a fixed length numeric vector.

The key decisions required to be made for the embeddings are whether to use the skip-gram/CBOW approach for Word2Vec and PV-DM/DBOW for Doc2Vec, and what the output dimensions of the vector should be. We will experiment with these options and evaluate them based on its impact on the final downstream model for detecting fake news. This extrinsic method of evaluation is preferred over an intrinsic evaluation of the numeric vectors, since it reflects the end goal of why we are embedding the articles, and also provides a more rigorous and objective measure for evaluation.

2.3. Neural network architecture

A neural network will be used to classify the news articles into Fake and True articles. The inputs to the neural networks will be the numeric vectors outputted from the Word2Vec and Doc2Vec models. With these numeric inputs, the modeling problem becomes a simple binary classification problem.

2.3.1 Candidate Neural network architectures

For the first approach where we used Word2Vec, the inputs are a sequence of tokenized words. There are a few challenges with this kind of problem. Firstly, the size of the inputs will vary, since each article has different length. Standard feed-forward neural networks required fixed size inputs, thus they will not work directly with these inputs. To tackle this problem, suitable architectures include RNN and LSTM, which are able to handle varying length input. Even CNN has seen frequent usage in NLP problems. Both RNN and LSTM are considered to be recursive network structures, and can be visualized as a series of feedforward networks (or cells) unrolled across time. Key to this design is that each cell has the same shared weights, thus the model when trained will learn what weights to use for all the cells. This design allows it to handle varying input sequence length, since the same weights are being shared (otherwise if this was not the case, it would not generalize to unseen sequence lengths). We note here that we have padded our inputs, however this is not a theoretical requirement, but a practical consideration for encoding sequence data into contiguous batches. The model will recognise the 0 as padding and will ignore it - known as masking.

We focus specifically on LSTM and CNN. LSTM was selected as a candidate due to the use of three gates (forget, input and output gate) to avoid the short term memory problem. For both RNN and LSTM, the state of the previous cell is passed to the next cell. However, with RNN a common issue is the vanishing gradient problem, where because the gradient is backpropagated over a very deep network (once unrolled), the gradient gets smaller and smaller to the point it 'vanishes'. A LSTM's forget gate will determine what information should be passed on to the next cell and what to forgot, and is better able to handle this issue of vanishing gradients. Finally, because text often derives meaning from words before and after it, we decided to use a bi-directional LSTM (BiLSTM). This effectively trains an LSTM in both directions, such that at each state, it contains information from both the past and future.

We also experiment with using a CNN. Whilst CNN is typically used for image problems, they have also been used effectively in NLP. In this case, our 'image' would consists of a matrix where each row is an embedded word with weights from Word2Vec. Thus an article would consists of many rows, depending on the number of words in the article. For example, with an embedding size of dimensions 100, and 500 words in the article, the input would be a 500x100 size matrix. In a similar fashion to image detection, we will slide a moving kernel across the input. Whilst the standard concepts of local invariance and compositionality that are key to CNN are less intuitive for NLP compared to images, CNN has still performed well in practice for NLP and thus will also be considered in our approach.

For the second approach, our Doc2Vec embeddings will result in a fixed size vector representing each news article. The key difference here is that we have the same size input for each article, and we are not passing in a sequence of inputs but just a single numeric vector. This means more complex architectures such as CNN, RNN and LSTM have no additional value since there is no state that is being passed along. Thus we will use a standard feed-forward multi-layered perceptron as our classifier for the Doc2Vec approach.

For all these networks, the loss function used is a binary cross entropy loss (BCELoss). This measures the binary cross entropy between the target and the output, and is suitable for this task as it is the preferred loss function under the inference framework of maximum likelihood for binary classification. We note that the target is balanced, thus there was no need to account for this imbalance (e.g. using Focal Loss). The Adam optimizer has been used for all the networks. This adaptive learning rate algorithm is suitable, as it combines estimations of first and second moments of gradient to adapt the learning rate for weights in the network. We will keep these consistent so as to make the experiments a fair comparison, and there is no reason to suggest they are not suitable for either approach. Finally for each network, we ensured that just enough epochs were run such that the validation loss started plateauing, to avoid over-fitting.

2.3.2 Detailed breakdown of selected networks

For approach 1, we considered a Bi-LSTM and CNN.

The layers of the Bi-LSTM are as follows:

Table 2. Bidirectional LSTM Network

Layer No.	Description
Layer 1	Embedding layer using pretrained weights fromWord2Vec
Layer 2-3	2 Stacked Bidirectional LSTM layers
Layer 4	Fully connected layer with leaky ReLU activation function ($\alpha = 0.2$)
Layer 5	Fully connected layer with softmax activation function

In terms of hyper-parameter tuning, the employed network used a hidden layer size of 128 and 64 for the Bidirectional LSTM and fully connected layers respectively. Furthermore, 10 epochs were run, the learning rate was set to 5e-4 and beta coefficients to (0.999, 0.9999) for Adam Optimizer.

Key to this architecture is the Bi-LSTM layer. Multiple fully connected layers with leaky ReLU activation were used as it demonstrated improvement in performance. Leaky ReLU was used to avoid the dying ReLU problem, and prevent the vanishing of gradients. Layer normalisation was included to enables smoother gradients, faster training, and better generalization accuracy. Likewise the drop-out layer was used to ensure the network does not overfit and generalises better. Finally a softmax output layer is used to ensure the outputs are probabilities, as we are solving a binary classification problem.

Additionally, the architecture of our Convolutional Network can be described as follows:

In terms of hyper-parameter tuning, the employed unidimensional convolutional layers (Conv1d) used a kernel size

Table 3. Convolutional Neural Network

Layer No.	Description
Layer 1	Embedding layer with pretrained weights from Word2Vec
Layers 2-4	3 Stacked unidemsional convolutional layers with Leaky ReLU activation functions
Layer 5	Max pooling layer
Layers 6-8	3 Stacked unidemsional convolutional layers with Leaky ReLU activation functions
Layer 9	Max pooling layer
Layer 10	Fully connected layer with leaky ReLU activation function
Layer 11	Drop-out layer (50%)
Layer 12	Fully connected layer with a softmax activation function

of 8, padding of 3 and stride of 1. The initial layer outputted 512 channels and each subsequent Conv1D layer outputted half the amount of channels of the previous one. Furthermore, 10 epochs were run, the learning rate was set to 5e-5 and beta coefficients to (0.999, 0.999999) for the Adam Optimizer.

In this Network, the first layer is an embedding layer to convert tokenized inputs using trained weights from Word2Vec. Additionally, multiple unidimensional convolutional layers with a kernel size of 8, padding of 3 and leaky ReLU activation functions are used (similar to our Bi-LSTM networks) and Max Pooling layers are also used to prevent overfitting, reduce variance and reduce computational operations required. Finally, a dropout layer is also used to avoid overfitting and a softmax activation function is employed.

For approach 2, using Doc2Vec embeddings, a multilayer perceptron is used, with the following layers:

Table 4. Multilayer perceptron (MLP)

Layer No.	Description
Layer 1	Fully connected layer with leaky ReLU activation function
Layer 2	Batch Normalization layer
Layer 3	Fully connected layer with leaky ReLU activation function
Layer 4	Batch Normalization layer
Layer 5	Fully connected layer with leaky ReLU activation function
Layer 6	Batch Normalization layer
Layer 7	Drop out layer (50%)
Layer 8-14	Repeat layers 1-7
Layer 15	Sigmoid output layer

Since it is a MLP, it simply consists of many fully connected layers with leaky ReLU activation, for the same reasons as per the LSTM and CNN. Many batch normlization and drop out layers are used to prevent overfitting. We note that having both dropout and normalization is not always ideal, however in this particular case it was found to be effective. The final layer is a sigmoid, which is equivalent to a softmax for binary classification.

2.4. Experimental setup

We used Python to develop the data preparation and modeling pipeline. Both the embeddings for Word2Vec and Doc2Vec used gensim package, and the neural networks were all modelled using Pytorch. GPU's were used to efficiently train the neural network models and their performance was measured in terms of accuracy on the previously defined test dataset.

3. Experiments and Results

3.1. Modeling and evaluation design

To evaluate how well the classifier performs, we will be using accuracy. This is suitable because the target variable is rather balanced. As discussed in the approach, a variety of different approaches will be examined using different embeddings, hyper-parameters and architectures. To ensure no leakage in evaluation of the models, we will use only the train dataset to build models for the embeddings. In the case of Doc2Vec, we will also use the trained Doc2Vec to infer vectors for the train, validation and test set (noting that we reinfer vectors for training dataset because the vectors from training the Doc2Vec differs from those inferred after the Doc2Vec model is trained). The downstream neural network will also be trained only using the train dataset, and the validation dataset is reserved for tuning the hyper-parameters of each architecture. Finally, we will evaluate the final models selected on the test set once, ensuring that this is the only time the test set is seen.

3.2. Experimental evaluation

The various experiments and results are presented below. All the reported accuracies are on the test set (which has not been used anywhere else to avoid leakage).

3.2.1 Approach 1: Word2vec

Table 5. Results with preprocessed data

Embedding	Neural Network	Accuracy
Skip-gram, dim=100	CNN	87.91%
CBOW, dim=100	CNN	90.41%
Skip-gram, dim=100	Bi-LSTM	92.20%
CBOW, dim=100	Bi-LSTM	94.48%

For approach number one, we use Word2vec to create the embeddings. For the neural network, we considered both a Bi-LSTM and a CNN. It is clear that a Bi-LSTM outperforms as CNN, which is not suprising given that a Bi-LSTM is conceptually more aligned to solving NLP problems. We also saw in both cases that CBOW performs better than skip-gram.

Table 6. Results with raw data

Embedding	Neural Network	Accuracy
Skip-gram, dim=100	CNN	87.01%
CBOW, dim=100	CNN	88.04%
Skip-gram, dim=100	Bi-LSTM	91.13%
CBOW, dim=100	Bi-LSTM	93.89%

Additionally, we also compared the same results when stop words were not removed, and no text preprocessing was applied. We can see there is a slight drop in performance, which suggests that preprocessing the text does slightly improve the results. From approach 1, the best embedding are from Word2Vec with CBOW, and the best network is a Bi-LSTM, with an accuracy of 94.48% on the test set.

3.2.2 Approach 2: Doc2vec

Table 7. Results with Doc2vec embeddings

StopWords	Embedding	Neural Network	Accuracy
Removed	PV-DM, dim=50	MLP	87.90%
Removed	DBOW, dim=50	MLP	99.03%
Removed	DBOW, dim=100	MLP	98.97%
Retained	DBOW, dim=50	MLP	98.95%

For approach two, we use Doc2Vec to convert each news article to a single numeric vector and feed this into a feed-forward MLP. In the first experiment, we removed stop words and used the PV-DM approach, which achieved an accuracy of 87.90%. In the second experiment, we set the approach to DBOW, keeping everything else the same to evaluate the impact of the Doc2Vec approach. The results were significantly better at 99.03%, making it clear that for this use case the preferred approach is DBOW. Experiment 3 and 4 were to evaluate the impact of changing the vector dimension and stop words removal respectively. Both these changes had little impact on the results, and in fact performed slightly worse.

The best model from this approach is to remove stop words, using DBOW and having an embedding dimension of size 50.

3.3. Conclusion

Comparing the two approaches, approach number two with Doc2Vec produced better results, with 99.03% accuracy compared to 94.48% from Word2Vec with Bi-LSTM. This is somewhat surprising, given that using Doc2Vec to convert an entire news article to a single vector is not a common approach seen in existing research.

4. Challenges and future work

Throughout the experiments, we were able to successfully compare different approaches to fake-news detection, and come to the conclusion that using Doc2Vec to create embeddings for news articles shows promise. However, there are still many areas / approaches that can be further investigated, and many other challenges when it comes to applying these to real-life situations.

Our experiments have also been limited to testing different architectures and embeddings, thus no pretrained models were used in the comparison. The results may differ if pretrained models are utilised, which can take advantage of huge training samples. We refer to the Appendix Table 8, where we build a classifier using a pretrained Roberta and Distilbert model which also showed how pretrained models show promise. An interesting avenue for further research would be to compare Word2Vec and Doc2Vec approaches using pretrained models instead of training from scratch.

There are also other approaches not covered in these experiments. Besides Word2Vec and Doc2Vec, there are many other approaches which have shown to perform well in similar tasks, such as GloVe. We have focused on Bi-LSTM and CNN for approach 1, however there are many other state of the art models which could also produce good results, such as XLNet.

One of the limitations of this work is that the underlying data set is heavily biased to political news. Whilst this is certainly one of the most important areas where fake-news is a real issue, fake news is not limited to just politics. To better understand which approaches would apply to more general sources of news, we note that collecting more examples would be important. This can include web-scraping from popular news websites, and training models using more recent data. We further note that commonly, digital content and ’fake news’ are not from articles, but from tweets and social media posts. Textual data could be extended to other types of social media content such as ”tweets”, ”Facebook posts” or even subtitles from video content. Other potential things to consider would be news in other languages, which may require different treatment and preprocessing due to vastly different grammatical structures.

5. References

1. Bovet, A., & Makse, H. A. (2019). Influence of fake news in Twitter during the 2016 US presidential election. Nature communications, 10(1), 1-14.

2. Monti, F., Frasca, F., Eynard, D., Mannion, D., & Bronstein, M. M. (2019). Fake news detection on social media using geometric deep learning. arXiv preprint arXiv:1902.06673.

3. Khan, J. Y., Khondaker, M., Islam, T., Iqbal, A., & Afroz, S. (2019). A benchmark study on machine learning methods for fake news detection. arXiv preprint arXiv:1905.04749.

4. Jamal Nasir, J. A., Khan, O. S., & Varlamis, I. (2021). Fake news detection: A hybrid CNN-RNN based deep learning approach. International Journal of Information Management Data Insights, 1(1), 100007.

5. Kumar, S., Asthana, R., Upadhyay, S., Upreti, N., & Akbar, M. (2020). Fake news detection using deep learning models: A novel approach. Transactions on Emerging Telecommunications Technologies, 31(2), e3767.

6. Appendix

6.1. Results from pretrained models

Additional to the two approaches that were compared, we also applied some pre-trained transformers such as Roberta and Distilbert from Huggingface. For this purpose, both pre-trained tokenizers and models were employed and followed by two fully connected networks with a LeakyReLU and Softmax activation functions. The following table shows their accuracy after 5 epochs.

Table 8. Results with pre-trained transformers

Embedding	Neural Network	Accuracy
Byte-level BPE	Roberta	99.87%
WordPiece	Distilbert	99.87%

Both models obtained the same level of accuracy on the testing set, although Distilbert had notably shorter training time. This shows that pre-trained models can achieve better performance, as well as that their performance can be considered as a soft upper-bound.