Fake News No More

Zach Weier
Middlebury College
Middlebury, VT
zweier@middlebury.edu

Graham Booth
Middlebury College
Middlebury, VT
gbooth@middlebury.edu

ABSTRACT

Donald Trump popularized the term fake news in the historic 2016 presidential election. During the election, fake news circulated vigorously on social media platforms like Twitter and Facebook, misinforming the voters and the general public. Observing the influence and proliferation of fake news during the 2016 election, we sought to create a fake news classifier that could filter news articles with over 90% accuracy.

We implemented a neural network using Python and the machine learning package, Keras [1]. We built our neural network to consist of 1 input layer, 4 hidden layers, and one output layer. Utilizing more recent machine learning techniques, we implemented ReLU and binary cross entropy as our activation and loss functions, respectively, and we added dropout to our neural net to combat overfitting, enabling the network to generalize with high accuracy. After building and adding features regarding diction, writing style, grammar, the author, and the title of the article, our neural net classified news articles with 96.346% accuracy. What's more, our neural net classified articles with 94% accuracy using only writing style, diction, and grammar as features.

We trained and tested on two separate data sets from Kaggle consisting of 20,000 and 5,000 news articles, respectively [2]. These data sets accompanied a Kaggle competition to identify which user could classify articles with the highest accuracy. Although our project began after the submission deadline, Kaggle allowed users to simulate their score, and our accuracy of 96.346% would have place us 6th on the competition's public leader board.

Keywords

Neural Net; Keras; ReLU; Sigmoid; Tanh; Dropout; Overfitting; Classification

I Introduction

The 2016 United States presidential election seemed to be a mix of reality television, data leaks, playground name-calling, and occasionally, constructive political debates. On the forefront of the historic election was fake news. Fake news shaped the tactics employed by both parties during the 2016 election. Republican candidate, Donald Trump, repeatedly referenced articles with misinformation, and when challenged in a debate, Trump would commonly refute his opposition's argument by simply calling it "Fake News." Propaganda from both the far right and far left, frequently disguised as legitimate publications, circulated online like wild fire. The proliferation of misinformation and validation of false stories on social media outlets like Facebook created an unprecedentedly polarized nation [3]. After reading some of the ridiculous stories during the 2016 election, which many people believed as fact at the time, we decided to make a fake news classifier that could filter-out illegitimate articles, forming a more well-informed public.

We built our classifier by implementing a neural network with a data set of 20,000 fake and real news articles from a competition on Kaggle.com [2]. Using multiple approaches and methods ranging from linguistic and grammatical analysis to fundamental machine learning strategies, we were able to build a classifier with 96% accuracy. The remainder of this paper will be structured as follows: In section II, we will present in detail the problem we are solving and discuss any relevant background information. Section III will highlight literature related to classifiers and fake news detection. Section IV will note the methods and strategies we employed to create our classifier. In section V, we will outline our results. Lastly, in section VI, this paper will conclude with a brief discussion of our results.

II Problem Statement

Our goal for this project was classify “fake” and “real” news articles with an accuracy above 90%, using the Scikit learn package in Python [4]. To classify the articles, implemented a neural network. While it is certainly far from a perfect imitation, neural networks are designed to model the actions of the human brain by connecting “nodes” that mimic the functions of connected neurons in the brain. Neural networks, shown in Figure 1, contain an input layer, one or multiple hidden layers, and an output layer. The input layer receives information from the data set in the form of a feature vector. The feature vector is then passed through the neural net using a feed-forward algorithm. Once the feature vector reaches the output layer, the neural net compares the result it calculated to the actual result, seeking to minimize the loss function. Then, using a back-propagation algorithm, the program propagates from the output layer to the input layer, re-weighting the connections between nodes based on the loss function. This process is done for each feature vector in the data set.

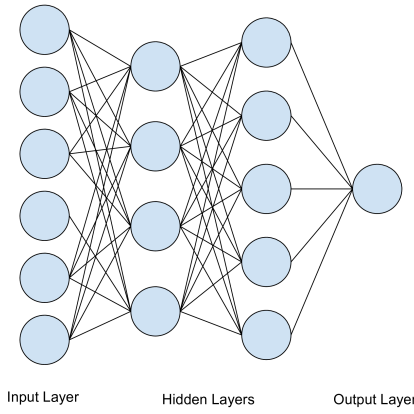


Figure 1. Neural Network Example

Neural Networks rely on activation and loss functions to learn properly and quickly. The activation function in a neural net manipulates the inputs received by a node and transforms them into the node’s output. Depending on the activation function, small changes to a node’s input could result in large fluctuations in the node’s output. The loss function proves integral to the neural net as well. The loss function is used to analyze the output of the neural net when learning. As the neural net learns, it predicts values, and based on those values, it generates a ”cost” or ”loss” from the loss function. Throughout learning, the neural net seeks to minimize this loss function

Particular characteristics of the neural network can be manipulated to increase the accuracy of the program. These characteristics are often referred to as ”tun-able parameters”, and include: the number of layers, the number of nodes in each layer, batch size, and alpha or the learning rate, which determines how quickly the network descends down the gradient.

III Related Work

Researchers have implemented a multitude of detection approaches ranging from linguistic to network analysis. Most closely related to our proposed project would be the linguistic approach utilizing classifiers. SVM’s, Bayesian models, and Neural Nets utilize linguistic approaches including detecting over-emotional or aggressive syntax that is commonly used in fake news. [3,5] Tabloid detection exploits grammatical and headline analysis by discovering headlines and subject matter that is more likely to attract clicks as well as detecting grammar commonly used in fake news like exclamation points. [6] We anticipated these tabloid and linguistic strategies to prove beneficial as features in our neural net.

A challenge when approaching fake news detection form a linguistic and grammatical strategy is separating satirical articles from misinformed articles. Rubin, Conroy, Chen, and Cornwell note the language and grammar used in satire closely resembles fake news. [7] Authors often use aggressive and provocative language to ”poke fun” at particular subjects or people. Rubin presents approaches to dealing with satire including, word-level features like n-grams, and semantic validity.

Additionally, papers have implemented non-textual approaches including image analysis and user behavior to discern fake news. Sebastian Tschiatschek leverages the power of the crowd in his research. Utilizing flags provided by users on social network platforms like Facebook, Tschiatschek develops an algorithm to block the spread of fake news. [8] Kai Shu analyzes fake news detection linguistically and visually. Shu offers an approach to analyzing images an videos embedded or attached to articles, using features and metrics such as: clarity score, coherence score, similarity distribution histogram, diversity score, and clustering score, statistical features include count, image ratio, multi-image ratio, hot image ratio, and long image ratio.

IV Methods

We constructed the neural network using the Python package Keras [1]. This package is essentially a simplified wrapper for a variety of more complex and heavy duty machine learning platforms. Keras supports three separate backends: Tensorflow, Theano and CNTK [1]. We used Tensorflow when working on a Mac.

We built our neural network with 4 hidden layers, which were structured as 4 Keras dense layers. Each hidden layer consists of anywhere from 200 to 1,000 nodes. In addition to Keras dense layers, we implemented 2 Keras dropout layers that dropout 10% of the nodes, and our output layer has one node that outputs a probability between 0 and 1, 0 being real and 1 being fake. The proceeding sub-sections will address how we chose our final architecture and built our features.

IV. (i) Hidden Layers and Respective Nodes

Throughout the project, we changed the number of hidden layers and number of nodes in each layer to obtain the highest accuracy possible. We started the project with 18 hidden layers containing anywhere from 200 to 1,000 nodes in each layer, but as we added more features, we needed to address overfitting. We started tweaking the number of nodes and hidden layers in our architecture and found that our optimal structure was 4 hidden layers containing anywhere between 200 and 1,000 nodes. Our final architecture allowed us to maintain the same accuracy while reducing the neural net's tendency to overfit.

IV. (ii) Dropout

To further prevent overfitting, we introduced dropout using Keras dropout layers [1]. Dropout prevents overfitting by discarding a fraction of the nodes randomly on each iteration (our neural net drops out 10%). If the network were weighting a node connection too heavily, downstream nodes would be influenced by the biased weighting. The network and downstream nodes would eventually heavily rely on that node connection to predict classifications. This would prevent the network from generalizing to different data sets with high accuracy. Randomly discarding nodes, or "dropping out" nodes, prevents the network from weighting one particular node connection too heavily, allowing the neural net to better generalize to other data sets.

IV. (iii) Output Layer

Our output layer consists of one node that holds a float value. This float value can be interpreted as the confidence the program has that a particular article is fake. Thus, we made 0.5 the threshold at which we consider an article fake or real. If the confidence is greater than 0.5, we classify it as fake, and if the confidence is less than or equal to 0.5, we classify it as real. We use the ReLu activation function, which will be discussed in more detail in the proceeding subsection, for our output layer [9]. However, our the binary cross entropy loss function reduces the effective maximum for the output layer, allowing us to set the threshold at 0.5.

IV. (iv) Activation Function

Throughout our project, we tested multiple activation functions including Sigmoid (Eq. 1), ReLU (Eq. 2), and Tanh (Eq. 3) [1, 9]. The equations for each are as follows, Sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{1}$$

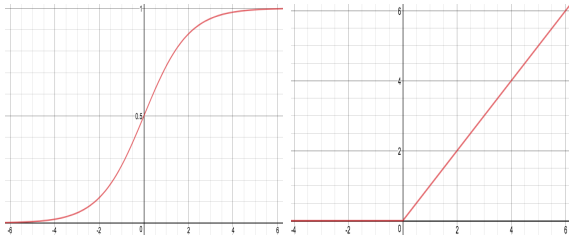
ReLU:

$$f(x) = \max(0, x), \tag{2}$$

and Tanh:

$$f(x) = \frac{1 - e^{-2x}}{2e^{-x}}. \tag{3}$$

As expected, ReLU performed the best for the hidden layer activation functions. With the ReLU activation function, we avoid the problem of vanishing gradients. Shown in Fig. 2, as the Sigmoid function approaches 0 and 1, the gradient becomes extremely small. This causes the neural net to either learn very slowly or not at all. ReLU, avoids the vanishing gradient problem by not capping the maximum value. Thus, the neural net learns much faster than it would with the Sigmoid activation function [9].



(a) Sigmoid (b) ReLU

Figure 2. Activation Functions

IV. (v) Loss Function

We used binary cross entropy for our loss function, shown below. Binary cross entropy is useful because it applies

essentially infinite cost to a wrong classification according to the equation

$$\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i}), \tag{4}$$

where, in the binary classification case, $N = 2$, but can be extended to multi-class classification. The extremely high penalty for incorrect classification greatly discourages the neural net from predicting incorrect classifications when learning. This makes it an ideal loss function for a neural network classifier.

IV. (vi) Data

We used data from the predictive modeling platform Kaggle. The data set consists of roughly 20,000 fake and real news articles. Each article has a unique identification number, the author of the article, the original text of the article, the title of the article, and a binary label to represent either fake or real. Kaggle also provided an additional testing data set structured identically to the aforementioned data set, but it contained only 5,000 articles.

IV. (vii) Feature Building

We built multiple features to increase the accuracy of our neural network. Below, we have provided descriptions of the features and the strategies employed to build them.

IV. (vii).1 Common Fake News Subjects

We thought a promising start to our features would be to manually create a list of subjects commonly mentioned in fake news articles. This list includes names like “Putin”, “Trump”, “Russia”, “Comey”, and other public figures associated with many fake news articles. We checked the titles and text of articles against the list of fake news figures, counting the number of times fake news figures were referenced. Dividing the number of references by the total number of words in the title and text body, we calculated the fraction of the article that mentioned fake news figures.

IV. (vii).2 Most Common Words

Expanding on our hardcoded list of fake news figures, we created a feature that extracts the most common words in the fake news articles. Our first approach to this feature looped through the titles and the text body of the articles. Using a dictionary containing the word as the key and the frequency as the value, we counted the number of times every word appeared in fake news articles. However, this feature did not increase our accuracy as much as we had anticipated. It proved difficult to ignore words like prepositions and conjunctions that indicated little about the content of the article but frequently appeared.

To discover more indicative words, we utilized the Scikit learn package in Python [4]. Scikit learn features a Count-Vectorizer that automatically tokenizes the training set text into a matrix of word counts. This allows users to easily discover the most frequently used words while ignoring extraneous words like prepositions and conjunctions. Additionally, it allows for Ngrams, which are combinations of n number of words. Consequently, we were able to find the most frequently used words and combinations of words in fake news articles, creating a more robust set of features. Using Ngrams from 1 to 5 words as features, we were able to greatly increase the accuracy of our program.

IV. (vii).3 Grammar

Existing literature on Fake News detection mentions the propensity of fake news authors to use provocative punctuation like question marks and exclamation points in article titles. Thus, we built a feature that counts the number of provocative punctuation marks in the title of an article. We noticed a small increase in accuracy with the implementation of this feature.

IV. (vii).4 Author Data

An intuitive feature to implement is the author of the article. Particular authors are more inclined to write fake news than others. It’s logical to think that it would be an effective identification characteristic of the article. Consequently, we included the author as a feature in our feature vector.

V Results

The greatest gain in accuracy we observed occurred after implementing the ”Ngrams” features from the Scikit learn package [4]. When manually looping through our data set, achieved roughly 85% classification accuracy. However, after more precisely identifying indicative words and combinations of words, we increased our accuracy to 94%. This is understandable. Having the ability to analyze the frequency of phrases including ”Crooked Hilary” or ”Barack Obama’s fake birth certificate”, greatly increases a human’s ability to detect if an article is fake compared to just analyzing the frequency ”Barack” or ”Hilary” ap-

pears. Thus, it is reasonable to expect a neural net to greatly improve its classification accuracy as well, when given improved analysis.

With the addition of author data, we further elevated our accuracy to 96.346%. We achieved our final test classification accuracy after training on the Kaggle data set consisting of 20,000 articles, and testing on the additional Kaggle data set that contained 5,000 articles [2]. The respective data sets were added to Kaggle as part of a fake news detection competition. Our final accuracy of 96.346% would place us 6th on the leader board in the public competition. We started our project after the submission deadline, but Kaggle provided the same testing data to allow users to simulate a submission score.

VI Discussion

We were very pleased with the 96.346% accuracy we achieved, especially considering how little training data we possessed. However, an accuracy this high begs the question: is the program overfitting? Overfitting for our neural net is very doubtful. We use many techniques throughout the project to minimize the chances that the program is overfitting. We use dropout, a completely separate testing data set, and observed the testing and training accuracy move in tandem at each epoch. Considering the preventative measures taken, there is minimal overfitting occurring.

Although an overall accuracy of 96.346% is notable, we think a more impressive metric is the 94% accuracy obtained with no author data included. Based on simply diction, writing style, and grammar, our program classifies fake news with 94% accuracy. Analyzing articles based on writing style generalizes better to other data sets, and separates our program from human ability even more. Humans can easily recognize authors who frequently publish fake news and subsequently scrutinize their articles more closely than others. However, if the author is unknown or disguised, humans struggle much more to identify fake articles – a task our classifier can accomplish with 94% accuracy.

While were are satisfied with our final accuracy and results, further work on our neural net that could increase accuracy certainly exists. Strategies we would like to explore more include spelling and grammar detection and user behavior analysis. We explored the use of provocative punctuation during our project, but further steps could be taken to analyze in-text grammar. For example, perhaps fake news articles frequently misuse commas, or have a tendency to misspell words. This is a strategy that could further improve our project. Additionally, analyzing user behavior is another technique that could improve our neural net's classification accuracy. Perhaps users spend less time on fake news articles, or interact with links embedded in the article more frequently. This was outside the scope of our project, but would be an excellent field to investigate. Finally, we were limited by the size of our training data. Creating a larger training data set will certainly improve the accuracy of our neural net.

VII References

[1] F. Chollet *et al.*, "Keras." https://github.com/keras-team/keras, 2015.

[2] U. M. L. Club, "Fake news." https://www.kaggle.com/c/fake-news, 2018.

[3] P. Davis, "Fake news, real consequences: Recruiting neural networks for the fight against fake news," (Palo Alto, CA, USA), Stanford, 2015.

[4] 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*, vol. 12, pp. 2825–2830, 2011.

[5] N. J. Conroy, V. L. Rubin, and Y. Chen, "Automatic deception detection: Methods for finding fake news," *Proceedings of the Association for Information Science and Technology*, vol. 52, no. 1, pp. 1–4, 2015.

[6] Y. Chen, N. J. Conroy, and V. L. Rubin, "Misleading online content: Recognizing clickbait as "false news"," in *Proceedings of the 2015 ACM on Workshop on Multimodal Deception Detection*, WMDD '15, (New York, NY, USA), pp. 15–19, ACM, 2015.

[7] V. L. Rubin, N. J. Conroy, Y. Chen, and S. Cornwell, "Fake news or truth? using satirical cues to detect potentially misleading news," in *Language and Information Technology Research Lab (LIT.RL)*, University of Western Ontario, 2016.

[8] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake news detection on social media: A data mining perspective," *CoRR*, vol. abs/1708.01967, 2017.

[9] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.