Training with Adversarial Data for Fake News Classification

Sameerah Helal shelal@ucdavis.edu
Melissa Liu maliu@ucdavis.edu
Sanskruti More smore@ucdavis.edu

1 Introduction

Fake or misleading news has the ability to sway elections, lead social movements, and spread misinformation. For this reason, identifying misleading or false information has become a priority for many social networking platforms to prevent the rapid spread of potentially harmful rhetoric or "truths". Across several popular platforms, machine learning has been effectively leveraged to aid human-driven content moderation, and as a result, platform users have begun to actively use language in attempts to gain or attack these algorithms. To examine an aspect of this issue, our research looks at different ways to introduce adversarial training data into a fake news classifier. We propose several ways to create adversarial data at scale and compare the effectiveness of their additions to training a neural network.

1.1 Problem Definition

We aim to answer the following questions.

1. Can training a neural network model on adversarial data increase its accuracy on unseen data?
2. Can training a neural network model on adversarial data allow it to resist attacks of the same and other adversarial methods?
3. What kinds of adversarial attacks can be deployed against natural language machine learning models?

1.2 Data Description

WELFake is a dataset of 72,134 news articles comprising 35,028 real and 37,106 fake news articles. Authors merged four popular news datasets (Kaggle, McIntire, Reuters, and Buzzfeed Political) to prevent model overfitting and enable better training (1). Each instance includes article title, article text, and labels real (1) and fake (0). Based on the dimensionality limit of the BERT model (2), we only consider article titles for the purposes of training our model.

2 Methods

We intend to build a robust classifier by training it on adversarial data. To do this, we build a BERT based neural network model, generate adversarial data on which we train the model, and perform adversarial attacks against the model. In the following sections we explain the model architecture and the different adversarial datasets we generate.

2.1 Model Architecture

The model architecture can be described in two parts. The first part takes the titles as input to a pretrained smaller BERT model. This smaller BERT model is chosen to optimize for reduced

computational resources. Using 4 hidden layers (i.e., Transformer blocks), a hidden size of 512, and 8 attention heads, this model outputs sentence embeddings of 512 dimensions. The second part is a single layer neural network of 128 dimensions which takes the sentence embeddings as features in the model. We use the ReLU (Rectified Linear Unit) activation function on the dense layer and apply the sigmoid activation function on the output layer to classify the data. Finally, we use the binary cross entropy to evaluate loss. We train the network using the Adam optimizer with a learning rate of 0.001 for 10 epochs.

2.2 Adversarial Data Generation

2.2.1 Background

Formally introduced in 2013 by Szegedy (3), adversarial training was introduced in computer vision problems. Using the MNIST dataset, Szegedy showed that images could be altered with intentional high-loss inducing perturbations that were indistinguishable to the human eye, but resulted with models incorrectly labeling the perturbed data with high confidence. These adversarial attacks revealed a significant weakness of neural networks that was fundamentally different from human learning patterns. Following this revelation, training with adversarial data or noise has been leveraged in academia and industry as a way to increase model robustness and generalization capability.

The notion of adversarial attacks in natural language came directly from practice, notably with evasion techniques targeting email spam filters. Linear classifiers could easily be subverted with the misspelling of "bad" words and inclusion of "good" words. These small perturbations are not as intuitive in natural language, and instead have been largely focused on the word, sentence, and character levels. Examples of altering natural language include using synonyms (4), making common spelling mistakes (5), or rearranging sentences (6).

To narrow our scope, we focus our perturbations at the word level and create adversarial attacks by mimicking spam-evasion techniques. We use the original WELFake dataset to generate three different adversarial samples to train the BERT model: advDataRandom, advDataKey, and advDataAppend.

2.2.2 Adversarial Datasets

Keyword detection

We use scikit-learn's word vectorization and logistic regression to identify keywords for the corpus of all titles: words that strongly indicate that a title is fake or true. With the snowball English stemming algorithm from Natural Language Toolkit (nltk), we reduce words to their stems (e.g. "running" becomes "run"), thus limiting the dimensions of the tokens in the tokenizing step. We then tokenize the stems with CountVectorizer from scikit-learn, which represents the titles by the frequency within the sentence of each word in the corpus. Feeding the tokenized titles into a logistic regression classifier gives us a coefficient matrix from which we can identify the features with the most positive and negative coefficients; i.e. the most true- and fake- indicating words. We compute the inflection points of the coefficient graph to decide where to cut off the number of keywords.

Datasets

The keywords obtained from the previous analysis are used in generating advDataKey and advDataAppend

- **advDataRandom**
  This dataset is generated by randomly perturbing the starting character of three randomly selected words of every article title in the WELFake dataset. Only words with a length of over four characters are perturbed to focus the noise on nouns and verbs, versus prepositions, which tend to be shorter in length and not as relevant to the meaning or class of the sentence. For each title, we randomly select a number of words; in our case, three. We choose a random lowercase character of the alphabet for each word and swap it for the first character of the word.
- **advDataKey**
  This dataset is generated by randomly perturbing the class-indicating keywords for each label. Words that are strong indicators for real news are perturbed for all real article titles, and indicators for fake news are similarly perturbed for all fake article titles. Article titles

that have no keywords strongly indicating their class are not perturbed and are instead dropped from the dataset.

- **advDataAppend**
  This dataset is generated by adding a keyword strongly indicating the opposite class to each fake-labeled title; all true-labeled news articles are dropped. This dataset most closely mimics rudimentary spam-evading techniques by adding "good" words to "bad" inputs in order to fool the model into misclassifying them as "good".

2.3 Training and Testing

Our datasets are split such that 75% is allocated to training and 25% to testing. We train a total of four different models. The baseline model is trained exclusively on WELFake data, with no generated adversarial data included. Three models are trained on the WELFake dataset combined with an adversarial dataset. Each of these training datasets maintain the same amount of WELFake data as the baseline model, but adversarial data from one of the three generated datasets is added such that it makes up 20% of the dataset.

After training, each model is tested on four different test datasets: the baseline WELFake dataset with no adversarial inputs and the three test datasets generated by adding one type of adversarial data to the baseline dataset. For these latter three datasets, the split between original and perturbed data is also 80/20.

3 Results

3.1 Experiment Results

We refer to the baseline model trained on unaltered data as the "original model" and a model trained on advDataX as the "advDataX" model.

From table 1 below we observe that the original model performs around the same as or slightly better than the adversarially trained models as measured by AUC scores and testing on the unaltered data. In fact, 1a shows that the original model's baseline for training accuracy is surpassed by the model trained on advDataRandom. This result is reinforced by the losses plotted in 1b

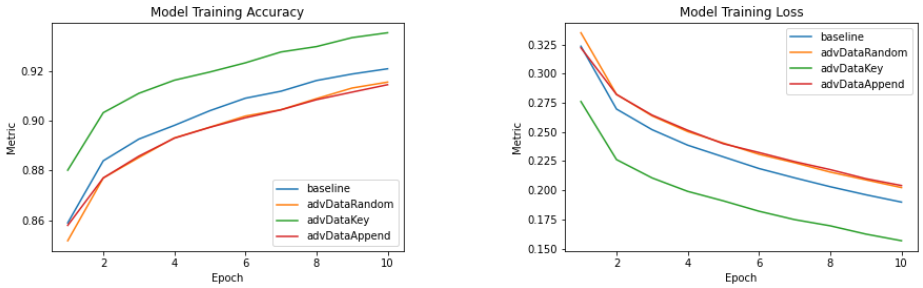
To evaluate the models' robustness against adversarial attacks, we test each of the models on each of the adversarial datasets. From 2, we see that the original model has the lowest accuracy when tested on advDataKey, and that advDataKey in general causes lower accuracies when used as a test dataset. The advDataAppend model has the highest average accuracy among all models when tested on the different adversarial datasets.

Model	Train Accuracy	Test Accuracy	AUC-PR
Original	0.9014	0.9076	0.9717
AdvDataRandom	0.8940	0.9020	0.9708
advDataKey	0.9174	0.8958	0.9675
advDataAppend	0.8936	0.8916	0.9701

Table 1: Metrics on Non-Adversarial Test Set

Model	Random Test	Key Test	Append Test	Average
Original	0.8899	0.8525	0.9021	0.8815
AdvDataRandom	0.8993	0.8777	0.9011	0.8927
advDataKey	0.8801	0.9216	0.8741	0.8919
advDataAppend	0.9008	0.9020	0.8988	0.9005
Average	0.8925	0.8885	0.8940	

Table 2: Test Accuracy on Adversarial Attack Test Set



(a) Training Accuracy (b) Training Loss

Figure 1: Training Accuracy and Loss for 4 Models

4 Discussion

The results show that the original model still produces the highest level of classification accuracy on the original WELFake dataset, which can be a sign of overfitting. On test datasets including adversarial data, however, the adversarial models did not necessarily perform the best on their respective test dataset (e.g. advDataAppend model produced a higher accuracy on the Random Test data than the advDataRandom model did). This indicates that adding perturbations at the word level increases the model's ability to generalize.

Amongst all models, advDataAppend had marginally the highest average test accuracy. Given the adversarial data's spam-evasion technique, this dataset likely either forced the model to consider better keywords or place greater emphasis on sentence tokens besides just vocabulary, resulting in higher test accuracy.

Conversely, the model trained on advDataKey yielded low accuracies across the board when used as a testing set. Considering that advDataKey had the highest training accuracies and lowest loss, this suggests that the attack method used in creating advDataKey is the most effective adversarial attack among our set.

As observed from the training accuracy plots, we see that the training has not plateaued at our stopping point, which means we could obtain better performance if we trained the models for more epochs. However, due to GPU resource limitations, we made the decision to conduct the experiment with fewer epochs. In future iterations of this experiment, we could add more epochs and experiment more layers in the model as well.

5 Conclusions

The problem of identifying fake news is challenging even for humans, especially with reduced attention spans and increased consumption of short form content. Teaching a model to do this is similarly extremely difficult. The "truth" of a fact requires validation of the source and research into the context, and cannot be purely extracted from semantics. And given the briefness of news cycles, the extreme variety in topics, and the large role that chronology plays, testing this model on today's news does not guarantee that the model would classify articles correctly. Thus, our model is limited by the input data to a relatively small vocabulary and a lack of additional context.

However, examples of natural language adversaries exist outside of news cycles. Our results show that general content moderation, such as in emails or on social media, can benefit from training with adversarial models. The best adversarial training data helps models generalize better, training them to rely less on features that may be unique to the sample and focus more on language and semantics that can be translated across topics.

A simple continuation of our research can be mixing all these adversarial examples together to better imitate the real distribution of adversarial attacks. As attacks and defenses against them develop, it is our hope that they can be used to both to further increase model efficacy and to better defend against newer and more sophisticated attacks.

6 Member Contributions

Model architecture and training were set up by Sanskruti More, and the methods to extract keywords and generate the adversarial data were written by Sameerah Helal and Melissa Liu. All members contributed to researching methods, training models, and writing the report.

References

[1] P. K. Verma et al. "WELFake: word embedding over linguistic features for fake news detection." IEEE Transactions on Computational Social Systems 8.4: 881-893. 2020.

[2] A. Tariq et al. "Adversarial Training for Fake News Classification." IEEE Access 10: 82706-82715. 2022.

[3] C. Szegedy et al. "Intriguing properties of neural networks." arXiv preprint, 2013. arXiv:1312.6199.

[4] D. Jin et al. "Is bert really robust? a strong baseline for natural language attack on text classification and entailment." Proceedings of the AAAI conference on artificial intelligence, 2020. Vol. 34. No. 05.

[5] B. Liang et al. "Deep text classification can be fooled." arXiv preprint, 2017. arXiv:1704.08006.

[6] S. Samanta and S. Mehta. "Towards crafting text adversarial samples." arXiv preprint, 2019. arXiv:1707.02812.

[7] S. Bird, E. Loper and E. Klein. Natural Language Processing with Python (NLTK). O'Reilly Media Inc, 2009

[8] F. Pedregosa et al. Scikit-learn: Machine Learning in Python. JMLR 12, 2019. pp. 2825-2830.

[9] K. Turc et al. "Well-Read Students Learn Better: On the Importance of Pre-training Compact Models." arXiv preprint, 2019. arXiv:1908.08962v2.

7 Appendix

7.1 Code Appendix

Our code and both the original and our adversarial data are available in our Google Drive folder.

Alternatively, the data can be downloaded directly from Kaggle. We provide a snippet in `Generating Adversarial Data.ipynb` to do this through the Kaggle API).

There are four notebooks containing our code. All `Training advDataX.ipynb` notebooks contain code to train and test the BERT text classifier on dataset advDataX, and `Generating Adversarial Data.ipynb` contains code to identify the most important keywords in the original dataset and use them to generate the adversarial data.

If running the code from the Google Colab notebooks in the aforementioned folder, all notebooks should run without any extra steps. If running outside the folder, download the data and run `Generating Adversarial Data.ipynb` before any of the `Training advDataX.ipynb` notebooks.