Adversarial Learning for Robust Fake Tweet Detection

Upamanyu Dass-Vattam

September 2020

1 Motivation

Nowadays social media, especially Twitter, is one of the battlegrounds for political confrontations. Although a majority of people involved in these discussions are using Twitter to legitimately express their opinions and support their positions, there are also users who deliberately post fake news and spread misinformation to undermine political opposition. This phenomenon has reached a point where fake tweet detection tools are being developed to flag such posts.

Fake tweet detection tools are generally data-driven, and trained by showing positive and negative examples of fake tweets to a machine learning model that learns patterns in the data. In other domains like computer vision, it has been shown that such machine learning models are vulnerable to adversarial attacks and can be easily fooled by adding the right kind of perturbations to the examples that are presented to the model. Therefore, there is a need to investigate if machine learning models used for fake tweet detection are also vulnerable to such adversarial attacks.

2 Problem

This work investigates the vulnerability of machine learning models trained to classify an input tweet as "authentic" or "fake" to adversarial attacks. The class of models used in this investigation are *Transformer*-based deep learning models. The adversarial attacks are based on *TextAttack*, a Python framework for adversarial attacks, adversarial training, and data augmentation in Natural Language Processing (NLP).

The problem consists of three parts: (1) to quantify the performance of the trained model before and after the adversarial attack, (2) to improve the performance of the classifier with the help of adversarial training, (3) to develop a Generative Adversarial Network (GAN) that produces a classifier that is robust to adversarial attacks.

3 Data Collection

The data used in this work is compiled from three datasets: the LIAR dataset, PHEME dataset of rumours and non rumours, and the FakeNewsNet dataset. In this section, a description of each of the datasets will be provided, as well as an analysis of the overall data.

A. LIAR Dataset

LIAR is a benchmark dataset for fake news classification first presented in Yang 2017. It is a publicly available dataset scraped from POLITIFACT.COM, a Pulitzer-prize winning website for its work in fact checking. The dataset itself contains 12.8k short labeled statements collected from 2007-2017 and then labeled by their nonpartisan team of fact checkers as one of: pants-on-fire, false, barely-true, half-true, mostly-true, and true. The distribution is fairly well balanced: the counts for each class range from around 2000 to around 2600, with the exception of the "pants on fire" class, which only has 1050 samples.

B. PHEME Dataset of rumours and non-rumours

PHEME is a dataset presented in [8], and consists of Twitter rumours and non-rumours posted during breaking news events. This data is limited to five events: the shooting of Michael Brown in Ferguson, MO in 2014 and the resulting protests and unrest, the shooting in Ottawa's parliament hill in October 2014, the hostage crisis at a chocolate cart in Sydney in 2014, the assault on the satirical newspaper Charlie Hebdo in 2015, and the Germanwings plane crash in the French Alps in 2015. The tweets were categorized as rumours and non-rumours by a team of independent journalists.Overall, there are 5802 tweets, 1972 of which were labeled rumours, and the remaining 3830 were labeled as non-rumours.

C. FakeNewsNet dataset

FakeNewsNet is a dataset first presented by [7], later revised in March 2019, containing not only news itself but also social context and spatio-temporal information. The tweets are either labeled by the aforementioned POLITIFACT.COM, but also GossipCop.com, a website which fact-checks news related to celebrities and the entertainment industry. There were 23196 tweets in total, with 17441 labeled as real and the rest labeled as false. Although the dataset contains more than just the tweets, including the number of retweets and information about the user that wrote the tweet, everything except the actual tweets was excluded to match the other datasets.

The overall dataset has mismatched labels, as none of the three share labels. Although FakeNewsNet and PHEME both have binary labels, the LIAR dataset has categorical labels. For the purposes of classification, "rumours" and "false" tweets from PHEME and FakeNewsNet respectively were classified as fake and their counterparts were classified as "true." The "pants-on-fire", "false", and "barely-true" items from the LIAR dataset were classified as fake and the rest were classified as true. Although the "barely-true" samples contain a little true information, they are often misconstrued or taken out of context, so they were categorized as false.

There are some concerns with the data. First, and most importantly, all of

the tweets are manually labeled, and with different groups of people. Not only is there always room for human error in classification, there are multiple groups of people with multiple sets of criteria labeling these tweets. Additionally, all of the data is old, with the most recent data still being a year and a half old. The news topics mentioned in the data are most likely no longer relevant.

4 Victim Model

Text classification is one of the most common tasks in NLP. It is applied in a wide variety of applications, including sentiment analysis, spam filtering, news categorization, etc. Here, a classifier is developed which can detect tweets as "false" or "authentic" using deep learning models. The *Transformer* is the basic building block of most current state-of-the-art architectures of NLP. Its primary advantage is its multi-head attention mechanisms which allow for an increase in performance and significantly more parallelization than previous competing models such as recurrent neural networks. Pre-trained BERT was used here [2], one of the most popular transformer models, and fine-tuned on fake tweet detection task. An LSTM model was also tried, but BERT model was selected because of its superior performance.

Building the classifier consists of the following steps:

1. Get pre-trained models: Download and import BERTokenizer and BERTSequenceClassification to construct the tokenizer and classification model.

2. Preprocess and prepare the dataset: BERT expects the input in a particular format and in order to get accurate representations of the words in the sequence, it is necessary to conform to that format. The following preprocessing steps must be followed before feeding a sequence to BERT: (1) tokenize the tweet sequence using a BertTokenizer, (2) add [CLS] and [SEP] tokens (every input sequence should begin with a [CLS] token, and end with a [SEP] token), (3) padding the input (during training, sequences have different lengths, but their tensor representations should have the same length. To ensure this, fix a maximum length T and pad all the sequences by a '[PAD]' token so that all of them are of the same size T.), (4) obtain indices of the tokens in BERT's vocabulary (replace the tokens with their corresponding indices in BERT's vocabulary. The BertTokenizer class provides a built-in method which can be used to achieve this.)

3. Fine-tuning BERT: First, define an instance of BERT model. Either fix the weights of the BERT layers and just train the classification layer which will be computationally much cheaper to do, or train the entire thing, which obviously will consume much more space and time. Then, freeze the weights of parameters of BERT layers by setting "requires_grad" property to False. This will not update these parameters during optimization. The "forward" method takes as input the sequence tokens and the attention masks. Both of these tensors are of shape [B, T], where B is the batch

size and T is the sequence length. First, feed both sequence tokens and attention masks to the BERT layer and get the contextualized embeddings (hidden representations) of each token. Since sentence-level classification is being performed, the embeddings of the first token in every batch i.e. '[CLS]' can be used. Feed these embeddings to the fully connected layer which outputs the logits (or scores) for the positive class. Before the training starts, the loss function and optimizer will be defined. Since this is a binary classification problem, the binary cross-entropy loss function will be used, specifically the BCEWithLogitsLoss module inPyTorch, which takes as the input the logits of the positive class and computes the binary cross-entropy (this is considered numerically more stable than the BCELoss which takes the probability of positive class as input instead). As for the optimizer, the popular variant of Stochastic Gradient Descent, ADAM, will be used. For the training process, there is an outer loop for the number of epochs, and in an inner loop the dataset is iterated through completely. From dataloader, batches of sequence tokens, attention masks, and labels are sampled. After that, they are converted to Cuda tensors and fed to our model to obtain the logits for the positive class. Using these logits and labels, the value of the loss function is computed and used in a backward pass to compute the gradients of the loss with respect to the model parameters. Finally, the weights are updated by taking an optimization step.

4.1 Performance of the Model

- Train size: 29926
- Validation size: 3326
- Test size: 7126
- The model was trained using a batch size of 128 for 20 epochs
- The accuracy score on the validation set was 0.9164
- The accuracy score of the trained model on test set was 0.9213

5 Text Fooler Attack on Victim Model

TextFooler [4] is a simple but strong method to generate natural adversarial attacks. In this case, the black-box version was used, where TextFooler was only passed the logits of the pretrained model on a given sample, as opposed to a white-box method, where TextFooler would get the gradients generated by the pretrained model.

The black-box version operates by hunting the most important words for classification. It passes the same text through the model repeatedly, changing one word at a time from the baseline, and finding the words which have the

most influence on the classification results. TextFooler then replaces the high importance words with another word of its choosing.

Given a word with a high importance, a suitable substitute must fulfill 3 criteria:it should it should (1) have similar semantic meaning with the original one, (2) fit within the surrounding context, and (3) force the target model to make wrong predictions. TextFooler does this through a four step workflow:

- Find the top N synonyms in terms of cosine similarity to a replaceable word. The words were represented by the embeddings from [5]. By default, N is set to 50 and the threshold for cosine similarity is 0.7.
- The synonyms are then checked for part of speech. Only those with the same POS as the replaceable word are kept.
- The candidates are then substituted into the sentence and run through the model. The final modified sentences are compared to the original sentences through cosine similarity, this time using the Universal Sentence Encoder[1].
- Among sentences that pass the similarity threshold, they are chosen to be part of the attack if they are able to successfully change the prediction of the model. If there are none, the attack is classified as a failure.

By using this workflow, TextFooler is generally able to preserve semantic similarity and maintain a natural language structure similar to the original text.

5.1 Performance of the model under adversarial attack

- The adversarial samples generated using the above method was tested on the trained victim model.
- All the adversarial samples were included as part of the test data.
- Test size: 20904
- The accuracy score of the trained model on test set with adversarial samples was 0.8068
- The drop in accuracy was 7.52%

6 Effect of Adversarial Training

The next step was to train the victim model with adversarial samples included in the train set. The pool of adversarial samples was split into 25%/75% split. 25% was added to the original train data. The 75% was added to the original test data. The victim model was trained again according to the steps mentioned in Section 4. For the adversarial training:

- Train size: 36126
- Validation size: 4015
- Test size: 14015
- The model was trained using a batch size of 128 for 20 epochs
- The accuracy score on the validation set was 0.9414
- The accuracy score of the trained model on test set was 0.9242

With the introduction of the adversarial samples into the training pool, the victim model performance on the test data jumped from 0.8068 to 0.9242.

7 FANG

One approach to make a classifier robust to adversarial attack is to include sufficient adversarial examples into the training data. It was shown in the previous sections that this approach is valid in our use case. But this approach assumes that there are sufficient adversarial samples available to train the classifier and that the test adversarial samples and the train adversarial samples are from the same distribution. But those assumptions may not always be true.

Another approach to make classifier robust is with the help of Generative Adversarial Networks (GANs) [3]. A GAN is composed of two models:

- The first model is called a Generator (G). G learns to generate data from an expected distribution. In our case, G learns the adversarial distribution. It generates adversarial samples. In other words, it's learning the function of TextFooler.
- The second model is named the Discriminator (D). This model's goal is to recognize if an input data is 'real' - belongs to the original dataset - or if it is 'fake' - generated by a generator model.

Based on this approach our Fake Adversarial News Generator (FANG) was developed.

7.1 FANG v1.0

In the first version of FANG,a pre-trained BERTSequenceClassifier was used as the discriminator. This model was similar to the victim model. For the generator, another *Transformer*-model known as GPT2 [6] was used. Developed by OpenAI, GPT2 is a large-scale transformer-based language model that is pre-trained on a large corpus of text: 8 million high-quality webpages. It results in competitive performance on multiple language tasks using only the pre-trained model and without fine-tuning. GPT2 is really useful for language generation tasks as it is an autoregressive language model.

In order to generate text, GPT2 requires a prompt. In our case, a tweet from the training data is randomly selected and used as the prompt to the GPT2 pre-trained model to generate the text. The details of the model discriminator and the generator models and the training procedure to train the FANG can be found in the accompanying Jupyter Notebook (transformerGAN.ipynb).

There were a lot of challenges to training FANG v1.0. The batch size was capped at 16. This was to avoid memory overflow issues. So training was conducted with a regimen of small batch sizes and large number of epochs. The number of epochs was initially set to 2000. During training, the process kept terminating at around 650-700 epochs without any sign of convergence in terms of generator loss and discriminator loss.

7.2 FANG v2.0

In FANG v2.0, a greater control over the discriminator and generator models was desired. Therefore, pre-trained models were not used. For the discriminator model, a BERT embedding layer followed by a sequence of Dense and LeakyReLU layers and a Sigmoid output was used. The details of the model discriminator and the generator models and the training procedure to train the FANG can be found in the accompanying Jupyter Notebook (textgan.ipynb).

```
Discrinimator:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 768)               0
_________________________________________________________________
dense (Dense)                (None, 512)               393728
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 525,313
Trainable params: 525,313
Non-trainable params: 0
_________________________________________________________________
```

For the generator model, a sequence of LSTM and Dense layers was used and passed noise as the input.

```
Generator:
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 256)               264192
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 768)               197376
=================================================================
Total params: 461,568
Trainable params: 461,568
Non-trainable params: 0
_________________________________________________________________
```

This model was trained on different batch sizes ranging from 32 to 1024. Similarly, it was also trained on epochs ranging from 100 to 2000. Overall, this GAN converged during training without any issues. The best convergence results were obtained when the batch size was 1024 and number of epoch was 100. Under this condition, both the discriminator loss and the generator loss was close to zero.

But when the trained discriminator was evaluated on test data, which had an equal mix of adversarial and non-adversarial tweets, the performance was quite low. The discriminator was not able to distinguish between adversarial and non-adversarial samples. The following is a distribution of logits for the test samples:

```
count  14014
mean  0.997932
std  0.012781
min  0.000000
25%  0.997926
50%  0.998905
75%  0.999432
max  1.000000
```

This distribution statistics show that most of the test samples were being classified as non-adversarial.

8 Discussion

To summarize, FANG v1.0 used complex pre-trained Transformer models for discriminator and generator. But the training process did not converge. On the other hand, FANG v2.0 used simpler deep learning models which converged during training, but the discriminator was unable to generalize well to the test set. Even after a lot of investigation, the source of the problem was not clear.

In FANG v1.0, GPT2 probably was not a good fit as a generator for this task because the kind of text it was generating was very different from the kind

of text found in the adversarial sample distribution and that it could not be trained to to fit that distribution. One explanation could be because GPT2 was pretrianed on a much larger and richer language dataset than the twitter dataset where each tweet is shorter in length. But more investigation is required to understand this issue.

In FANG v2.0, the opposite problem is likely true. The model draws its vocabulary from its dataset, which inherently limits its ability to generalize. This is especially important in the context of this problem, as the range in topics being passed to model requires a high level of generalization. Again, this is merely conjecture, and more investigation is needed.

9 Conclusion

For any fake news classifier to be effective, there must be defenses against adversarial attacks in place. State of the art methods of increasing model robustness typically involve feature training the model on adversarial samples, however this can lead to the model learning the distribution of the samples instead of the attack method of the attacker itself. To prevent this, FANG would institute an ensemble of models. The first model is a fake news classifier, and the second model determines whether or not the sample has been perturbed or not by an attacker. This second model is trained in a GAN, where the generator is trying to learn the perturbation method of the attacker, and the discriminator predicts whether or not a sample is a perturbed sample. However, in v1.0 the generator was not able to learn the attack method, and in v2.0 the generator over fit to the samples it was passed. As it stands, FANG is a less viable method for reinforcing a fake news detector than the current standard practices.

References

[1] Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[4] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment, 2019.

[5] Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. Neural belief tracker: Data-driven dialogue state tracking, 2016.

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

[7] Kai Shu, Deepak Mahudeswaran, Suhang Wang, Dongwon Lee, and Huan Liu. Fakenewsnet: A data repository with news content, social context and spatialtemporal information for studying fake news on social media, 2018.

[8] Arkaitz Zubiaga, Maria Liakata, and Rob Procter. Learning reporting dynamics during breaking news for rumour detection in social media, 2016.