FireBERT: Hardening BERT Classifiers Against Adversarial Attack

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present FireBERT, a set of six proof-of-concept NLP classifiers hardened against TextFooler-style word-perturbation by producing diverse alternatives to original samples. In one approach, we co-tune BERT against the training data and synthetic adversarial samples. In a second approach, we generate the synthetic samples at evaluation time through substitution of words and perturbation of embedding vectors. The diversified evaluation results are then combined by voting. A third approach replaces evaluation-time word substitution with perturbation of embedding vectors. We evaluate FireBERT for MNLI and IMDB Movie Review datasets, in the original and on adversarial examples generated by TextFooler. We also test whether TextFooler is less successful in creating new adversarial samples when manipulating FireBERT, compared to working on unhardened classifiers. We show that it is possible to improve the accuracy of BERT-based models in the face of adversarial attacks without significantly reducing the accuracy for regular benchmark samples. We present co-tuning with a synthetic data generator as a highly effective method to protect against 95% of pre-manufactured adversarial samples while maintaining 98% of original benchmark performance. We also demonstrate evaluation-time perturbation as a promising direction for further research, restoring accuracy up to 75% of benchmark performance for pre-made adversarials, and up to 65% (from a baseline of 75% orig. / 12% attack) under active attack by TextFooler.

1 Introduction

1.1 Motivation and prior work

Just as we have seen interesting, easy-to-fabricate but hard-to-explain adversarial attacks against visual classifiers (Ma et al., 2020; Eykholt et al., 2018), attacks against text classification systems have been proposed. Earlier examples use misspellings (Li et al., 2018), an attack easily thwarted with preprocessing (Pruthi et al., 2019) or co-training (Zhu et al., 2019).

In 2019, a group of researchers from MIT, University of Hong Kong, and A*STAR Singapore published a new approach, creating perturbations focused on flipped classification but minimal semantic difference. The “TextFooler” project (Jin et al., 2019) used word similarities to generate adversarial corpora prevailingly understood and classified the same way as the original by humans. The technique fooled BERT near completely on many benchmarks, exposing a potentially dangerous vulnerability to adversarial attack. TextFooler performs a near exhaustive search for adversarial samples that come reasonably close to preserving the semantic content of the original. It treats the classifier as a black box, examining only the classifier probability output to determine the importance of individual words.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

Recent work to guard against such attacks (Karimi et al., 2020) co-trains a BERT model on embedding perturbations generated by following the gradients. It achieves small but measurable success. It picks up earlier work (Goodfellow et al., 2014) which explored (among other methods) adding a defined noise distribution to input vectors. Goodfellow's team concluded that adversarial samples are not finely distributed around input vectors, but, rather, that there are "pockets" of adversarial classification, and that ensembles will offer no protection against carefully constructed adversarial samples. Their work was based on MNIST classification, but their claim is that the findings apply more generally to most networks that feature large linear components. We claim that BERT-based NLP classifiers are sufficiently removed from simple linear behavior to allow us to take another look at the effectiveness of ensembles.

TextFooler specifically attacks BERT, not its descendants. Thus our goal is to evaluate our work against TextFooler directly. We know of no paper attempting to harden against an attack by TextFooler or any adversary with similarity to its exhaustive approach and degree of success. There are numerous BERT derivatives, and some claim added robustness (Liu et al., 2019). They may offer additional resilience above regular BERT against adversarial attack, but, to our knowledge, this has not been evaluated. The TextFooler paper itself has been recently updated but the code remains unchanged. Updates to the paper were confined to the analysis and future work suggestions, and are not relevant to our result. A recent survey of adversarial attacks and defenses (Wang et al., 2019) does not include TextFooler and makes the claim that adversarial attacks against text are impractical; a statement which TextFooler's achievements should put very much in doubt.

2 Primary contribution

MNLI entailment and IMDB sentiment classification accuracy with BERT under adversarial attack by purely pre-made TextFooled samples (generated with TextFooler on previously known BERT-based classifiers) can be improved from approximately 0% to close to original performance with a hardened classifier. TextFooler analysis of (and sample generation on) such a hardened classifier can be made significantly more difficult (requiring at least 5 times the amount of computation performed for an attack on an unmodified BERT-based classifier, and failing with at least twice the unmodified rate). We show that this can be achieved without substantially lowering the regular accuracy for the above-named benchmarks.

Our principal contribution consists of three classifiers constructed as a defense mechanism against TextFooler-style attacks, with details discussed below. The code, plus tuned models, hyperparameter search code and training and evaluation notebooks for these classifiers, in addition to tools for exploratory data analysis and the actual training and evaluation data, are available at our anonymous GitHub repository. A summary of our most important results:

- Reducing the error rate on pre-made adversarial samples by 79% (new accuracy 0.800) on MNLI and 87% (new accuracy 0.872) on IMDB by co-tuning with synthetic samples.
- Reducing the error rate by 62% on both the MNLI and IMDB tasks (to 0.623 and 0.620, respectively) on the IMDB tasks through evaluation-time vector perturbation.
- Reducing the error rate by 48% (to accuracy 0.545) on the MNLI task under active TextFooler attack, through evaluation-time vector perturbation.
- We show that TextFooler overfits to a specific, tuned model: Simply re-tuning on the original data improves accuracy against pre-made adversarial samples significantly.

2.1 Methods

We explore three ways to teach BERT to be more accepting of perturbed sentences while preserving classification results.All three are applied to both sentiment classification (IMDB) and entailment classification (MNLI): In approach 1 ("FuSE"), we introduce additional, slightly word-diversified samples during the evaluation, and make a voting ensemble.In approach 2 ("FIVE"), we shortcut the search for replacement words and add Gaussian noise to the input-embedding vectors directly. In approach 3 ("FACT"), we co-tune the classifier with the same diversified samples we use in FuSE. All three use a shared component to perturb text, which we will explore in more detail. All three classifiers are built around an underlying BERT-instance, they are implemented as subclasses of the base classifier, modifying only very specific parts of the behavior.

2.1.1 SWITCH - "Substituting Words In Text Classification Hardening"

The purpose of our SWITCH component is to provide sample diversity with retained classification. SWITCH takes an example like this: “this movie is truly fun for the whole family adults and kids will totally enjoy it!” and produces alternatives like this: “this photography is sincerely fun for the whole family matures and teenagers will perfectly enjoy it !" and “this theatre is truly fun for the whole family forties and kiddies will entirely enjoy it !”.

SWITCH uses its own pre-tuned BERT instance for evaluating the gradients for the input sentence/pair provided, in order to determine which words are important to the classification. It then uses the same counter-fitted embeddings employed by TextFooler to create alternative words.The actual cosine similarities are never needed, since words far away from the original are of no interest to us. We store a pre-computed matrix of 100 nearest-neighbor index numbers for each word. Replacement words are filtered through part-of-speech matching, and finally the replacement texts can optionally be ranked (for closest semantic similarity to the original) or filtered (for at least positive similarity value) through the Universal Sentence Encoder (USE) (Cer et al., 2018) similarity scores. There is a random element to SWITCH’s final choice of alternatives, in order to make it harder for the exhaustive trial-and-error process of TextFooler to have a stable target to work with. Tunable hyperparameters are the number of words to perturb, the number of alternative samples to generate, whether to use part-of-speech matching, and whether to employ USE in either filtering or ranking, plus a multiplier to generate more samples before USE is applied.

```
Input: (Explicit Input) Sentence Example
  $E = w_1, w_2, ...w_n$
(Implicit Model Input) Boolean $use\_USE$ to
  determine if we utilize Universal Sentence Encoder
  (USE), Integer $USE\_multiplier$ to specify the
  USE multiplier, $total\_alternatives$ desired count to
  produce.

Output: List of alternative texts $alternatives$

Initialization:
Create list $A$ (all words in $E$)
Create list $I$ (all important indices) from model
  gradients for $E$
Filter stop words and words not in counter-fitted
  vocabulary from $I$
$replacements \leftarrow$ For all $A[I]$, lookup 10 nearest
  counter-fitted neighbors of same part of speech
if USE is utilized then
    $count \leftarrow total\_alternatives * USE\_multiplier$
else
    $count \leftarrow total\_alternatives$
New list $alternatives$ to store candidate sentences
for $ctr < count$:
    for $i <$ len($I$):
        $a[I[i]] \leftarrow replacements[i]$[random(10)]
    end for
    Add $a$ to the list $alternatives$
end for
if USE is utilized then
    if $use\_method$ is rank then
        Sort $alternatives$ descending by USE similarity
          score with $E$
    else if $use\_method$ is filter then
        Filter negative USE similarity scores with $E$
          out from $alternatives$
return first $total\_alternatives$ from $alternatives$
```

Figure 1: SWITCH

2.1.2 TextFooler baseline models

To establish a baseline for our three approaches we obtained the original models provided by the TextFooler authors which were fine-tuned on bert-base-uncased for theMNLI and IMDB tasks. These pre-tuned models are the BERT instances fed into SWITCH for querying and active searching of diverse candidates.

2.1.3 Secondary Pytorch Lightning baseline models

The base code for our three classifiers is a reimplementation of a HuggingFace BERT-based uncased sequence classifier in Pytorch Lightning. We used the published TextFooler binary models for baseline results.We fine-tuned secondary baseline models for the IMDB and MNLI tasks, to validate our code and the training parameters.After a random hyperparameter search, we selected 5 training epochs and a batch size of 32 (MNLI) and 20 (IMDB) with a learning rate of $2 * 10^{-5}$ and no weight decay. Adam epsilon was maintained consistently at $1 * 10^{-8}$.

2.1.4 Fuzzy sentence ensemble - FuSE

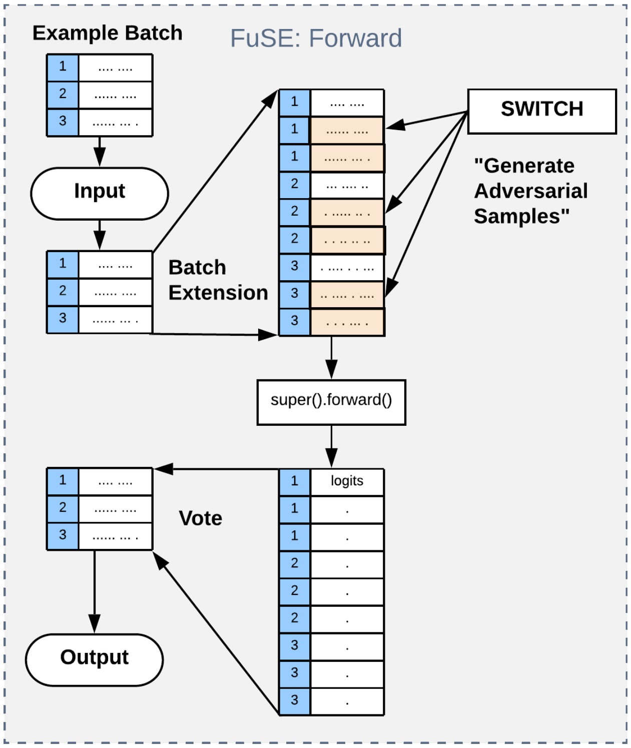

Figure 2: FuSE perturbation method

The hypothesis for FuSE is that sampling nearest neighbors for a word would provide, on average, better classification outputs. In other words, most neighboring words would provide correct classification rather than adversarial classification results. FuSE uses SWITCH to determine the most important words to the classification through gradient computation; the quantity of important words is randomized during our evaluation process. After determining the important words, SWITCH provides alternative sample formulations by changing those words with a number of neighboring words. Using this method, FuSE assembles a random number of alternative sample sentences and evaluates all of them against the underlying sentiment or entailment classifier, as seen in figure 2. FuSE outputs either synthetic logits representing a majority vote count of classifications for the samples ("majority vote"), or the average of the logits across the samples ("logit-averaging"). Tunable hyperparameters are all SWITCH parameters, plus the selection of the voting method.

2.1.5 Fuzzy internal vector ensemble - FIVE

FIVE is based on the hypothesis that averaging over neighborhoods of embedding vectors, evaluated in the context of the sample, leads to more stable average classifications than evaluating based on a particular embedding that might have been changed by an adversary like TextFooler. FIVE asks SWITCH to identify the most important word by gradient computation, and then creates additional synthetic samples by perturbing their token embedding vectors, as seen in figure 3. Each set of perturbed embeddings forms a Gaussian distribution around their original vector. Like FuSE, FIVE outputs either synthetic logits representing a vote count of classifications for the samples ("majority vote"), or the average of the logits across the samples ("logit-averaging"). Tunable hyperparameters include the number of embeddings to perturb, number of perturbed samples to generate, the standard deviation of the Gaussian distribution to create around the original embeddings, and the voting method for combining the individual synthetic sample votes.

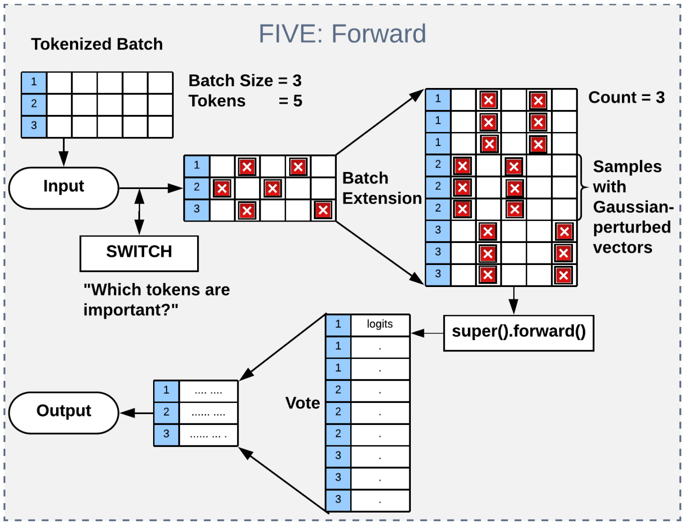

Figure 3: FIVE perturbation method

2.1.6 Fuzzy adversarial co-tuning - FACT

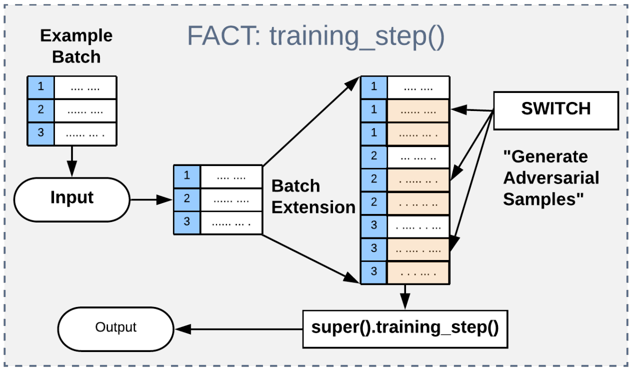

Figure 4: FACT in-batch co-tuning

In our third, fine-tuning-based approach, we introduce a little language diversity to the tuning process to teach the FACT classifier that there are multiple ways to express our original sentiment or entailed fact. In this way, we are fine-tuning BERT to become less sensitive to the very specific adversarial examples TextFooler generates.FACT uses SWITCH to identify important words and provide alternative samples, which are injected sequentially inside the batch during the fine-tuning process (Figure 4). The advantage of this approach is that, after fine-tuning, we can just deploy a new binary model file for evaluation or use in production. On the downside, co-tuning with synthetic samples can substantially increase the fine-tuning time depending on the number of additional samples requested. But this is performed only once, and the resulting model has no additional run-time degradation. All SWITCH hyperparameters are tunable, and regular training hyperparameters like batch size, learning rate, weight decay and Adam epsilon can also be adjusted.

All three classifiers inherit from our Pytorch Lightning base classifier which provides hooks for batch extension with perturbed samples during the forward() (for FuSE and FIVE) and training_step() (for FACT) methods. The classifiers override these extension hooks to provide diversified samples in their respective approaches.

3 Results and conclusions

3.1 Evaluation procedures and source data

For IMDB training and evaluation, we downloaded 50,000 IMDB labeled movie review samples (Lakshmipathi, 2019) and split the data into 40,000, 5,000 and 5,000 samples for train, validation and test. The data was originally curated by Stanford University (Maas et al., 2011)

For MNLI (Williams et al., 2017) training and evaluation, we downloaded data from the GLUE baseline repository (Wang et al., 2018). Training was performed on the provided file of 390K samples. Validation was performed on the 10,000 samples from dev_matched. Since labels are not publicly available in the test set, we used the 10,000 samples from dev_mismatched as our holdout for testing.

We generated adversarial samples for each of the train, validation and test sets in source data by running the TextFooler algorithm on each of the base models for our complete data sets. The adversarial data derived from validation samples were used for hyperparameter tuning and evaluation of our three approaches. The adversarial samples derived from test data were reserved for single use in final evaluation. We provide these adversarial sets for each of the tasks to further efforts in this research.

Performance under active attack by TextFooler is evaluated using the code, metrics and datasets provided by TextFooler. A set of 1000 samples for the MNLI task was selected by the TextFooler authors. TextFooler runs masked versions of the samples through the classifiers to establish which words most affect the outcome. It then runs the classifier on samples with those words perturbed to nearest neighbors until it finds good adversaries, sometimes unsuccessfully.

3.2 Hyperparameter search

Hyperparameter search for FIVE hyperparameters, and the SWITCH parameters of FuSE and FACT was performed in a time-boxed fashion with a random search of the hyperparameter space. Random searches were utilized based on demonstrated benefits with this method (Bergstra and Bengio, 2012).

3.3 Results

Model	Acc.	F1 score	Adv. acc.	Adv. F1 score
Baseline (TF)	**0.843**	**0.835**	0.029	0.027
Baseline (New)	0.833	0.826	0.501	0.487
FIVE	0.757	0.708	0.632	0.566
FuSE	0.725	0.713	0.591	0.568
FACT	0.827	0.821	**0.800**	**0.791**

Table 1: MNLI results

Table 1 shows the results of all classifiers against the MNLI dataset. The original MNLI model provided by TextFooler served as the baseline. Its performance against TextFooler samples that were generated specifically against it is unsurprisingly low. The secondary (freshly tuned) baseline model is slightly inferior to the TextFooler baseline mode on originals, but brings up the accuracy on adversarial samples substantially. FIVE achieved a substantial improvement in the adversarial case at the expense of significantly, but not unreasonably, dampened performance on the originals (1 perturbed embedding, std dev 8.14, 8 synthetic samples per original, logit-averaging). For FuSE, we find similar results with slightly worse performance on original samples (2 perturbed words, 10 candidates per word, part-of-speech matching, 14 candidates into USE, filter negative scores, max 14 samples, logit-averaging). Co-tuning with FACT resulted in the best performance against adversarials while sacrificing barely any accuracy on originals (batch size 7, 9 words to perturb, 10 candidates per word, part-of-speech matching, 12 candidates into USE, filter negative scores, max 4 samples).

Model	Acc.	F1 score	Adv. acc.	Adv. F1 score
Baseline (TF)	**0.906**	**0.904**	0.002	0.002
Baseline (New)	0.905	0.902	0.827	0.816
FIVE	0.884	0.867	0.620	0.586
FuSE	0.518	0.508	0.778	0.770
FACT	0.900	0.897	**0.872**	**0.867**

Table 2: IMDB results

Table 2 shows the results against the IMDB dataset. The original TextFooler IMDB model served as the baseline. The secondary (freshly tuned) baseline model shows itself to be basically not vulnerable to pre-manufactured adversarial samples - retuning addressed the problem by itself for this dataset. FIVE achieved significant gains in the adversarial case while not losing substantial accuracy on the benchmarks (1 perturbed embedding, std dev 2.3, 10 synthetic samples per original, probability averaging). For FuSE, we found good performance against adversarials, but a complete degradation to coin-flip level for the original samples. More work is needed to understand this result (3 perturbed words, 10 candidates per word, part-of-speech matching, 17 candidates into USE, filter negative scores, max 12 samples, probability averaging). Co-tuning with FACT once again performed well against both original and adversarial samples (batch size 2, 23 words to perturb, 10 candidates per word, part-of-speech matching, 12 candidates into USE, filter negative scores, max 4 samples).

Model	Org. Acc.*	Adv. Acc.*
Baseline (TF)	0.851	0.127
FUSE	0.499	0.276
FIVE	0.777	**0.463**
FACT	0.820	0.316
FuSE (FACT)	0.373	0.373
FIVE (FACT)	0.743	**0.545**

Table 3: Accuracy under active attack

We also investigated how well two of our classifiers would perform against an active attack by TextFooler. This required a minimal adaptation of TextFooler to work against our Pytorch Lightning classifiers. Fully explaining TextFooler's result parlance is beyond the scope of this paper, but briefly (Table 3, all numbers generated by TextFooler code, and adversarial accuracy "Adv. Acc*" not directly comparable to tables 1 and 2): In our baseline measurement, TextFooler degrades the accuracy of a BERT sequence classifier to around 12%. We found that FACT is able to raise that number to a significant 31% on MNLI, requiring TextFooler to change around 30% more words and try about 30% more samples. FuSE on top of a FACT-tuned model raises the number again. Against FIVE, with no re-tuning, TextFooler uses about the same number of perturbed words and classifier queries but is unable to degrade the accuracy below 45%. FIVE on top of the co-tuned FACT model delivers the best performance of all at 54.5% accuracy.

3.4 Analysis

A look at a “fooled” MNLI example can be instructive in understanding what is going on inside our classifiers. Here is one from our validation set: {Premise: “So I have to find a way to supplement that.”, Hypothesis: “I need a way to add something extra.”, Label: "entailment"}. TextFooler is able to minimally change the hypothesis in a way that most of us would reasonably still classify the same way, fooling the classifier into a “neutral” classification: {Hypothesis: "I need a way to add something additive."}. To look at what happens in the evaluation-time classifiers FuSE and FIVE, we will perturb only one of the words for illustration. By computing the gradients for the classification and finding the input that has the largest absolute gradient, SWITCH correctly decides that “extra” and “additive” are the most important words for the respective original and adversarial hypotheses.

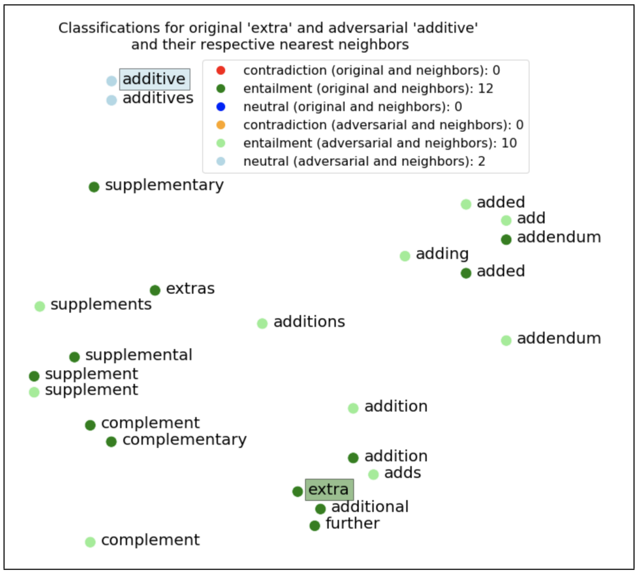

Figure 5: Nearest neighbors of words (MNLI baseline)

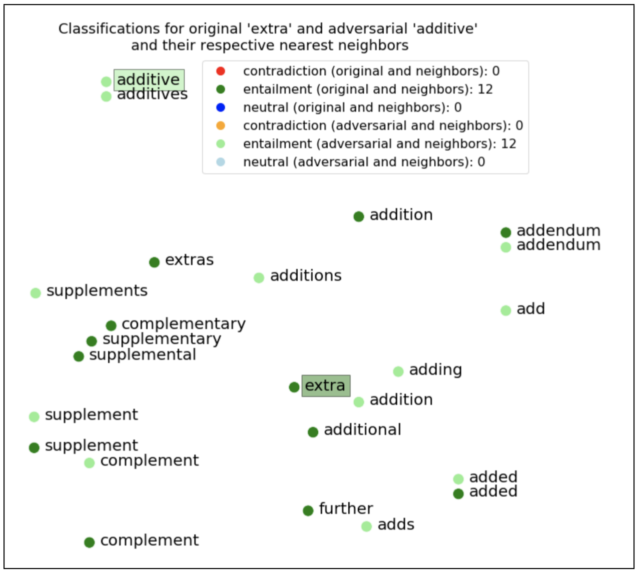

Figure 6: Nearest neighbors of words (MNLI FACT)

Figure 5 shows classification outcomes for the replacement words evaluated in the context of their respective full premise/hypothesis pairs. The nearest neighbors (by cosine similarity) around the original word "extra" all lead to an "entailment" classification. Among its neighbors, the adversarial word "additive" is one of only two leading to a "neutral" classification, with the majority landing on "entailment". This is the kind of example we had hoped to see - a stable neighborhood of words around the original, but a majority of original classifications around the adversarial sample. TextFooler found one isolated word, the smallest possible nudge to give the sample, to change the classification. If we look at the neighborhood as a whole and average across it, we find a more stable classification. FuSE won’t be fooled by TextFooler in this example.

For co-tuning (FACT), we ask SWITCH to come up with a list of diverse replacement hypotheses:"i need a method to additions something additive", "i need a pathway to additions something other", "i need a manner to inserts something additional", "i need a manner to totals something add". More points of stable classification are established with the BERT-model. Tuned on this extra set of samples, FACT no longer considers the substitution of “additive” adversarial (figure 6).

We chose a particularly benign sample to illustrate the workings here - not all examples work out this well. MNLI defense in particular is a difficult problem to solve at evaluation time, as it is often easy to nudge a sample from “entailment” or “contradiction” to “neutral” with a single word perturbation, but overwhelmingly unlikely to reverse that judgment with another random nudge.

From the substitution of words, we now switch into the embedding space that contains the inputs to the actual BERT-classifier. Figure 7 shows the classification fields for vector perturbation of the most important words in our samples, at a standard deviation of 0.25. Note that is not the best hyperparameter for classification with FIVE, but it shows the clustering well in this t-SNE diagram. The Gaussian regions around both the original and the adversarial words are laced with adversarial points, and at a small standard deviation, we will indeed find regions of predominantly adversarial sentiment. With the right tuning, however, we found a standard deviation hyperparameter at which the adversarial classification represents only small pockets, and as we are averaging over the cluster, FIVE comes to the right conclusion for this example as well.

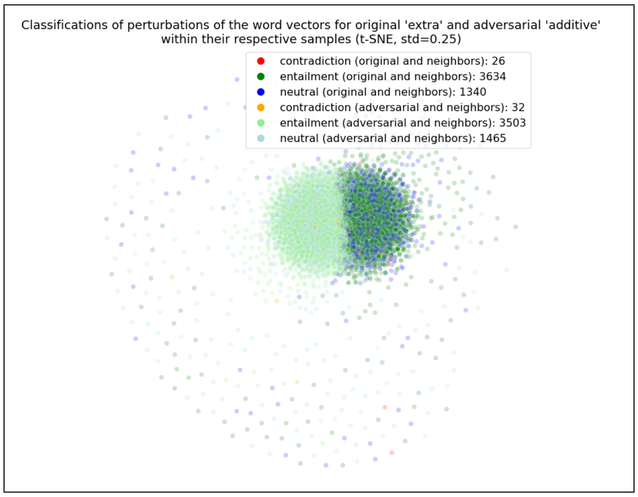

Figure 7: Classification fields (FIVE vector perturbation)

3.5 Discussion

The first insight from our work is that TextFooler overfits to the model it works against when generating samples. We found that adversarial samples generated from one model (e.g. the original MNLI model from Jin et al. (2019)) would fool a second model (e.g. the secondary MNLI model tuned on PyTorch Lightning) in only half the circumstances.The same results were also true in the reverse (e.g. samples generated from the Lightning model would fool the original model only half the time). This leads us to suspect that TextFooler works to find the weak points in the specific way a model is parameterized.

FuSE delivers a solid performance on the MNLI task, and fails spectacularly on regular (non-adversarial) IMDB samples, achieving nothing more significant than coin-flip performance in the most “conservative” of our hyperparameter searches. More investigation is needed to understand this failure, especially since its performance against pre-made adversaries is substantial at 77% accuracy.

We find that co-tuning with SWITCH is a very effective way to protect a model against TextFooler’s original samples. We also saw that evaluation-time perturbation can improve adversarial results, trading in a few degrees of task accuracy for a good degree of adversarial protection. Additionally, active attack query-count results show the co-tuned models get harder to fool in generating new sets of samples for MNLI, but not for IMDB. We suspect this is because the sentence length is much larger in IMDB than in MNLI, and therefore gives TextFooler more possible word choices in perturbation.

However, our most resilient model against active attack is the FIVE classifier, which preserves accuracy to 45% (55% on top of FACT). We theorize that it is the stronger random aspect of Gaussian perturbation that makes FIVE a moving target for TextFooler. FIVE also delivers a solid performance against pre-made adversarial samples, which, together with its efficiently parallelizable approach, makes it attractive for further research.

3.6 Conclusion

We showed that BERT-based classifiers can be hardened against both pre-made adversarial samples and active attack by a mechanism like TextFooler. The price for such improvements is some loss in accuracy on non-adversarial samples ranging from insignificant, as seen in the performance of our co-tuned model against pre-made samples, to substantial (10-20%), as the drops in regular benchmark metrics for our evaluation-time classifiers under attack by TextFooler.

Future work should investigate the failure of FuSE to perform on the regular IMDB benchmark, combine some of our approaches into a single classifier, and tune the algorithm and implementation to allow for a longer hyperparameter search.

Broader Impact

BERT pre-trained classifiers opened the field of NLP classifiers to many practical applications, including sentiment and entailment classification. BERT-based classifiers score consistently high benchmark numbers (Devlin et al., 2018). All that is left is the actual classification task, factoring out much of the daunting NLP aspect of such any text classification project. But can such classification be trusted, and to what degree? When text classification becomes usable, it also becomes tempting to use it as a replacement for human judgment. With popularity comes attack surface: Such classifiers were shown to be vulnerable to black-box trial-and-error antagonists like TextFooler (Jin et al., 2019). These mechanisms use exhaustive search to produce adversarial examples that lead the classifier to the wrong result in over 90% of samples examined, even when those samples were judged to be semantically equivalent by humans. Can classifiers be hardened against such attacks?

These questions need to be answered before we apply text classifiers to all kinds of applications as gatekeepers of civility and true representation of sentiments. Hardening classifiers against manipulation will protect meme-browsing youths and other vulnerable population segments from predatory individuals and trouble-seeking trolls, just as it will protect intellectual property investments by thwarting manipulation of reviews and recommendations.

References

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al. (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Jin, D., Jin, Z., Zhou, J. T., and Szolovits, P. (2019). Is bert really robust? natural language attack on text classification and entailment. *arXiv preprint arXiv:1907.11932*.

Karimi, A., Rossi, L., Prati, A., and Full, K. (2020). Adversarial training for aspect-based sentiment analysis with bert. *arXiv preprint arXiv:2001.11316*.

Lakshmipathi, N. (2019). *IMDB Dataset of 50K Movie Reviews*.

Li, J., Ji, S., Du, T., Li, B., and Wang, T. (2018). Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ma, X., Niu, Y., Gu, L., Wang, Y., Zhao, Y., Bailey, J., and Lu, F. (2020). Understanding adversarial attacks on deep learning based medical image analysis systems. *Pattern Recognition*, page 107332.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics.

Pruthi, D., Dhingra, B., and Lipton, Z. C. (2019). Combating adversarial misspellings with robust word recognition. *arXiv preprint arXiv:1905.11268*.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Wang, W., Tang, B., Wang, R., Wang, L., and Ye, A. (2019). A survey on adversarial attacks and defenses in text. *arXiv preprint arXiv:1902.07285*.

Williams, A., Nangia, N., and Bowman, S. R. (2017). A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Zhu, C., Cheng, Y., Gan, Z., Sun, S., Goldstein, T., and Liu, J. (2019). Freelb: Enhanced adversarial training for language understanding. *arXiv preprint arXiv:1909.11764*.