White-to-Black: Efficient Distillation of Black-Box Adversarial Attacks

Yotam Gil[1]* and **Yoav Chai**[2]* and **Or Gorodissky**[1]* and **Jonathan Berant**[2,3]
[1]School of Electrical Engineering, Tel-Aviv University
[2]School of Computer Science, Tel-Aviv University
[3]Allen Institute for Artificial Intelligence
{yotamgil@mail, yoavchai1@mail, orarieg@mail, joberant@cs}.tau.ac.il

Abstract

Adversarial examples are important for understanding the behavior of neural models, and can improve their robustness through adversarial training. Recent work in natural language processing generated adversarial examples by assuming white-box access to the attacked model, and optimizing the input directly against it (Ebrahimi et al., 2018). In this work, we show that the knowledge implicit in the optimization procedure can be distilled into another more efficient neural network. We train a model to emulate the behavior of a white-box attack and show that it generalizes well across examples. Moreover, it reduces adversarial example generation time by 19x-39x. We also show that our approach transfers to a black-box setting, by attacking The Google Perspective API and exposing its vulnerability. Our attack flips the API-predicted label in 42% of the generated examples, while humans maintain high-accuracy in predicting the gold label.

1 Introduction

Adversarial examples (Goodfellow et al., 2014) have gained tremendous attention recently, as they elucidate model limitations, and expose vulnerabilities in deployed systems. Work in natural language processing (NLP) either (a) used simple heuristics for generating adversarial examples (Jia and Liang, 2017; Belinkov and Bisk, 2017; Iyyer et al., 2018), or (b) assumed white-box access, where the attacker has access to gradients of the model with respect to the input (Feng et al., 2018; Ebrahimi et al., 2018). In this approach, adversarial examples are constructed through an optimization process that uses gradient descent to search for input examples that maximally change the predictions of a model. However, developing attacks with only black-box access to a model (no access to gradients) is still under-explored in NLP.

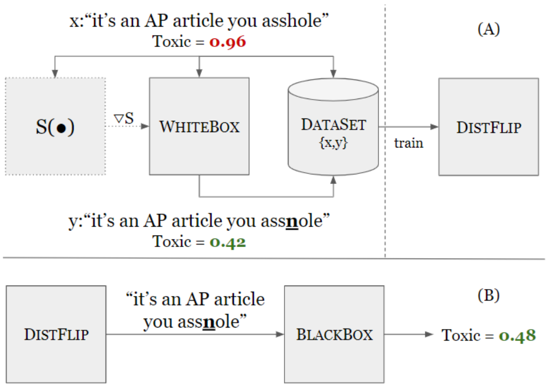
x:"it's an AP article you asshole"
Toxic = 0.96
(A)
S(•)
∇S
WhiteBox
Dataset {x,y}
train
DistFlip
y:"it's an AP article you assnole"
Toxic = 0.42
(B)
DistFlip
"it's an AP article you assnole"
BlackBox
Toxic = 0.48

Figure 1: (A) Using a white-box attack we generate adversarial examples for a source toxicity model $S(\cdot)$. We train our black-box attacker, DistFlip, to emulate the white-box attack. (B) We use DistFlip to attack a black-box model.

Inspired by work in computer vision (Papernot et al., 2016; Liu et al., 2017), we show in this work that a neural network can learn to emulate the optimization process of a white-box attack and generalize well to new examples. Figure 1 gives a high-level overview of our approach. We assume a text classification model and a white-box attack that flips characters in the input to modify the model prediction (Ebrahimi et al., 2018). We generate output adversarial examples using the white-box attack and train a neural network from these input-output examples to imitate the white-box attack. This results in a much more efficient attack whose run-time is independent of the optimization process. Moreover, assuming adversarial examples transfer between different models, our distilled model can now be used to generate adversarial examples for black-box attacks directly.

We use our approach to attack a toxicity classifier, aimed at detecting toxic language on social media (Hosseini et al., 2017). We show that our model achieves a speed-up of 19x-39x in generating adversarial examples while maintaining similar attack quality, compared to an optimization-based

[*] Equal contribution

Proceedings of NAACL-HLT 2019, pages 1373–1379
Minneapolis, Minnesota, June 2 - June 7, 2019. ©2019 Association for Computational Linguistics

method. We then use our model for a black-box attack against Google Perspective API for detecting toxic sentences, and find that 42% of our generated sentences are misclassified by the API, while humans agree that the sentences are toxic.

Our code can be downloaded from `http://github.com/orgoro/white-2-black`.

2 Background

Adversarial examples have been extensively used recently in NLP for probing and understanding neural models (Jia and Liang, 2017; Weber et al., 2018). Methods for generating such examples include adding random or heuristically constructed noise (Belinkov and Bisk, 2017; Rodriguez and Rojas-Galeano, 2018; Gao et al., 2018), meaning-preserving modifications that change the surface form (Iyyer et al., 2018; Ribeiro et al., 2018), and human-in-the-loop generation (Wallace et al., 2018). A weakness of such models is that they do not directly try to modify the prediction of a model, which can reduce efficacy (Kurakin et al., 2016). In the white-box setting, Feng et al. (2018) have changed the meaning of an input without changing model output using access to gradients, and Ebrahimi et al. (2018) proposed HotFlip, the aforementioned white-box attack that we emulate for flipping input characters.

In computer vision, Papernot et al. (2016) and Liu et al. (2017) have shown that adversarial examples generated by a white-box model can be helpful for a black-box attack. Generating adversarial text examples is more challenging than adversarial images, because images are points in a continuous space, and thus it is easier to apply norm restrictions. Text examples have a discrete structure and thus such approaches have not been investigated for adversarial text generation yet.

3 HotFlip

HotFlip (Ebrahimi et al., 2018) is a white-box method for generating adversarial examples for a character-level neural model. It uses the gradient with respect to a 1-hot input representation to estimate the character flip that incurs the highest cost. We briefly describe HotFlip, which we use to generate training examples for our distilled model.

Let $\mathrm{x} = ((x_1^1, \ldots, x_1^n), \ldots, (x_m^1, \ldots, x_m^n))$ be a sentence represented as a sequence of m characters, encoded as 1-hot vectors over a vocabulary of size n. Define $L(\mathrm{x}, y)$ to be the loss of a trained model for the input x with respect to a label y.

HotFlip requires one function evaluation (forward pass) and one gradient computation (backward pass) to compute a first-order estimate of the best possible character flip in x. Flipping the i^{th} character from a to b can be represented by this vector: $\overrightarrow{v_{i_b}} = (\ldots, (0, \ldots, -1, \ldots, 1, \ldots, 0)_i, \ldots)$, where -1 and 1 are in the positions for the a^{th} and b^{th} characters respectively. A first-order estimate of the change in loss can be obtained by computing $\nabla_x L(\mathrm{x}, y)$ with back-propagation, and taking a directional derivative along $\overrightarrow{v_{i_b}}$:

$$\nabla_{\overrightarrow{v_{i_b}}} L(\mathrm{x}, y) = \nabla_x L(\mathrm{x}, y) \cdot \overrightarrow{v_{i_b}}.$$

We can now choose the character-flip a to b that maximizes this estimate using the gradients with respect to the input x:

$$\arg\max_{i,b}[\nabla L(\mathrm{x}, y) \cdot \overrightarrow{v_{i_b}}] = \arg\max_{i,b}[\frac{\partial L}{\partial x_i^b} - \frac{\partial L}{\partial x_i^a}].$$

To perform a sequence of flips, any search procedure can be applied. HotFlip uses beam search of r steps, keeping at each step the top-K flip sequences that increased $L(\mathrm{x}, y)$ the most. This require $O(K \cdot r)$ forward and backward passes. Character insertions and deletions are modeled as multiple flips, but for simplicity, we only consider *character flips* in our work.

The main drawbacks of HotFlip are that it does not gain any knowledge from optimizing over multiple examples, and that its efficiency is strongly tied to the search procedure used ($O(K \cdot r)$ forward and backward passes per example for beam-search). Next, we present our model that overcomes these limitations.

4 Distilling a Black-box Attack

We are interested in whether (a) the knowledge in the optimization process of HotFlip can be distilled into a neural model, and (b) whether this model can generalize to a black-box attack. Therefore, the outline of our approach is as follows:

1. Train a *source text classification model* on data from a similar distribution to the data used to train the *target black-box model*.
2. Generate adversarial examples by performing white-box optimization (with HotFlip) on the source model.
3. Train an efficient *attacker* to generate adversarial examples, and perform a black-box attack against the target model.

We assume a training set $\mathcal{D} = \{(\mathrm{x_i}, y_i)\}_{i=1}^N$ used to train a character-based source model $S(\cdot)$ that takes a character sequence x as input, and returns a distribution over the output space $\mathcal{Y}$ (details on the source model are in Section 5). We now elaborate on the processes of data generation and training of the attacker.

Data generation We take examples (x, y) from $\mathcal{D}$ and run HOTFLIP with search until we obtain an adversarial example $\bar{\mathrm{x}}$ such that the probability of the gold label is low, that is, $[S(\bar{\mathrm{x}})]_y < \tau$, where $[S(\bar{\mathrm{x}})]_y$ is the probability given by $S(\bar{\mathrm{x}})$ to $y \in \mathcal{Y}$ and τ is a threshold (we use $\tau = 0.15$).

Let $s = ((\mathrm{x}^{(0)} = \mathrm{x}), \mathrm{x}^{(1)}, \ldots, (\mathrm{x}^{(l)} = \bar{\mathrm{x}}))$ be the sequence of sentences generating $\bar{\mathrm{x}}$, where every consecutive pair $(\mathrm{x}^{(\mathrm{i})}, \mathrm{x}^{(\mathrm{i+1})})$ differs by a single character: the character in position $j^{(i)}$ was flipped to the character $c^{(i)}$. Our attacker is trained from examples $(\mathrm{x}^{(\mathrm{i})}, (j^{(i)}, c^{(i)}))$ generated from every pair of consecutive sentences in s. For example, if the sentence is the one-word sentence “Asshole”, and after flipping one character it becomes “Assnole”, the example would be (“Asshole”, (4, ‘n’)).

Model training Our attacker takes a character sequence x as input and outputs a pair (j^*, c^*), where $j^* \in [1, \ldots, m]$ is the position of the character to be flipped, and c^* is the target character.

Figure 2 describes the architecture of our model.

Our model embeds each character using a pretrained 300-dimensional character embedding[1], and then passes the character sequence through a 1-layer bidirectional LSTM (Hochreiter and Schmidhuber, 1997) with 512-dimensional hidden states. The BiLSTM h_j hidden state in every position are passed through two feed-forward networks, one for replacement prediction (which character to flip) and one for target prediction (what target character to choose). The network that perfoms replacement prediction has 2 hidden layers of dimensions 100 and 50 with ReLU activations, and a single logit l_j as output per position. The output distribution over the sentence positions is given by a softmax over all character positions. At inference time j^* is computed with an argmax instead of a softmax.

The network that produces target prediction has two-hidden layers of dimensions 100 and 100 with ReLU activations and outputs a vector of logits $v_j \in \mathbb{R}^{96}$ per position with a softmax layer, which provides a distribution over the character vocabulary. The target character c^* is computed at inference time with an argmax over the target position $\mathrm{x}_{\mathrm{j}^*}$.

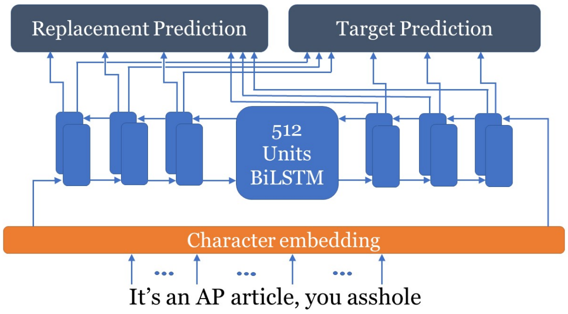

Figure 2: The architecture of our attacker network.

Our loss function is simply the sum of two cross-entropy terms: one for the gold position, and one for the gold character in the gold position.

Running our model is more efficient that HOTFLIP and run-time is independent of the optimization procedure. A forward pass in our model is equivalent to $2 \cdot K$ steps in HOTFLIP with beam-search. We show this leads to large practical speedups in Section 5.

5 Experiments

We now empirically investigate whether our method can be used to attack classifiers for detecting “toxic” language on social media. Recently a challenge by Alphabet aimed to improve labeling of toxic comments that are rude and disrespectful. Alphabet released a dataset[2] of 160K comments from Wikipedia discussions, and classified each comment to six labels. We focus on the *Toxic* label only, which represents 9.5% of the dataset.

We used the dataset to train the source model $S(\cdot)$, which runs a 2-layer bidirectional GRU (Cho et al., 2014) over the input x, and then uses an attention (Bahdanau et al., 2015) pooling layer to obtain a fixed dimensional vector for x. This vector is passed through a feed-forward layer to compute the probability that x is toxic. The accuracy of the source model is 96.5% AUC – comparable to the top submissions in the challenge.

We used the 13,815 toxic-labeled sentences from the training set to generate adversarial examples for training the attacker as described above. Dataset

[1] https://github.com/minimaxir/char-embeddings/blob/master/glove.840B.300d-char.txt

[2] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

generation depends on the search procedure, and we experiment with 3 setups: (a) HotFlip-5: beam-search with K=5, (b) HotFlip-10: beam-search with K=10, and (C) HotFlip+: a more expensive search procedure that calls $S(\cdot)$ more frequently. We describe the details of this procedure in Appendix A. Because our attacker is independent of the search procedure, inference time is not affected by the search procedure at data generation time. This results in three distilled models: DistFlip-5, DistFlip-10, and DistFlip+.[3]

Attacking the source model We compare the performance of DistFlip variants against HotFlip variant, including HotFlip-1 (K=1). We also compare to a Random baseline, which chooses a position and target character randomly, and to an Attention baseline, which uses the attention layer of $S(\cdot)$ to choose the character position with maximum attention to flip, and selects a target character randomly.

Table 1 summarizes our results. We report the average number of flips required to change the prediction of toxic sentences in the source model, the slow-down per single character-flip, and the slow-down per attack, which is computed by multiplying slow-down per flip by the ratio of the number of flips required per attack. Because roughly 15% of the examples in the dataset mostly contain repeated profanities that require many flips, we also report the average number of flips for the other 85% of the examples.

We observe that HotFlip+ requires the fewest flips to change model prediction, but attacks are very slow. The number of flips per attack for DistFlip+ is comparable to HotFlip-5 and HotFlip-10, but it achieves a speed-up of 19x-39x. DistFlip-5 and DistFlip-10 require a few more flips compared to DistFlip+. Overall attack quality is high, with less than two flips necessary for 85% of the examples for DistFlip+.

Figure 3 provides a more fine-grained view of the results by showing the proportion of sentences classified as toxic as a function of the number of flips. Overall, the picture is similar to Table 1, with DistFlip+ being comparable to HotFlip-10.

Attacking The Google Perspective API The Google perspective API[4] returns the probability that a sentence is toxic, where probability > 0.7 is classified as *toxic*, < 0.3 is *non-toxic*, and otherwise *uncertain*. The model itself is not publicly available. We randomly selected 136 toxic examples from the validation set and attacked them with DistFlip+ until the source model probability was < 0.5.

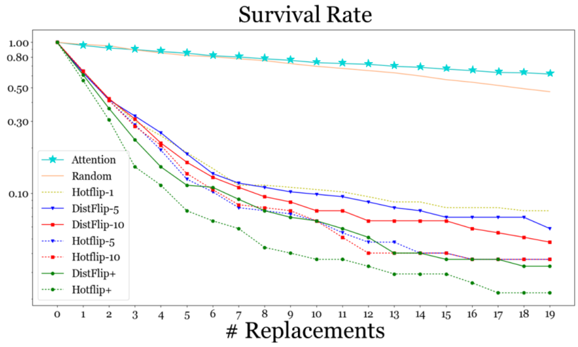

Figure 3: Proportion of sentences classified as toxic as a function of the number of flips for all models on the test set.

We measured the toxicity probability before and after our attack and found that the average toxicity probability decreased from 0.9 to 0.67, with an average of 5.0 flips per sentence. The label is flipped from *toxic* to *uncertain* or *non-toxic* in 42% of these examples.

Human validation To validate that toxic sentences remain toxic after our attack, we showed 5 independent annotators a total of 150 sentences from three classes: toxic sentences, non-toxic sentences, and attacked sentences (attacked by DistFlip-5). The same annotator was never shown a toxic sentence and its attacked counterpart. We asked annotators whether sentences are toxic, and measured average annotated toxicity.

We found that 89.8% of the toxic sentences were annotated as toxic, compared to 87.6% in the attacked toxic sentences. This shows that humans clearly view the sentences as toxic, even after our attack. Table 2 shows examples for sentences attacked by DistFlip-10 and the change in toxicity score according to The Google Perspective API.

6 Conclusion

In this work we present a general approach for distilling the knowledge in any white-box attack into a trained model. We show that this results in substantial speed-up (19x-39x) while maintaining comparable quality. Moreover, we show that our

[3] DistFlip-5 is trained on 64K generated examples, DistFlip-10 on 62K examples, and DistFlip+ on 46K.

[4] https://www.perspectiveapi.com

	DistFlip+	DistFlip-10	DistFlip-5	HotFlip+	HotFlip-10	HotFlip-5	HotFlip-1	Rand.	Att.
#flips	5.05	7.5	8.6	3.2	4.5	4.6	13.8	56.0	70.6
#flips for 85%	1.88	2.27	2.30	1.66	2.16	2.18	2.24	24.57	36.09
Flip slow-down	1x	1x	1x	168.8x	43.3x	21.3x	6.1x	-	-
Attack slow-down	1x	1.48x	1.7x	108x	**38.6x**	**19.4x**	16.7x	-	-

Table 1: Average number of flips to change the prediction of toxic sentences, average number of flips to change the prediction for 85% of the examples that do not contain repeated profanities, slow-down per flip compared to DistFlip+, and slow-down per sentence attack compared to DistFlip+. One character-flip using DistFlip takes 12ms on an Nvidia GTX1080Ti.

Original	**Adversarial**	**Google score**
decline=I don't fucking think so	decline=I don't **v**ucking think so	0.79 → 0.07
its an AP article you asshole, how is it not neutral???	its an AP article you ass**n**ole, how is it not neutral???	0.96 → 0.48
I think the 1 million sales is total bullshit though.	I think the 1 million sales is total bullsh**k**t though.	0.79 → 0.07
do not know juden! silence is golde juden, now shut up.	do not know juden! silence is golden juden, now **x**h**j**t up.	0.69 → 0.37
This article sucks . . woo woo wooooooo	This article **x**u**x**ks . . woo woo wooooooo	0.93 → 0.22
Also bring back the brendle article you piece of shit.	Also bring back the brendle article you p**k**ece of **x**h**k**t.	0.98 → 0.32
to be driven away and die	to be driven away and d**k**e	0.82 → 0.32

Table 2: Examples of sentences attacked by DistFlip-10 and The Google Perspective toxicity score before and after the attack.

attack transfers to a black-box setting: we expose the vulnerability of The Google Perspective API, and are able to change the prediction for 42% of input toxic examples, while humans easily detect that examples are still toxic.

Acknowledgements

We thank Raja Giryes for supporting this research, and the anonymous reviewers for their constructive feedback. This research was partially supported by The Israel Science Foundation grant 942/16, the Blavatnik Computer Science Research Fund, and The Yandex Initiative for Machine Learning.

References

D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173*.

K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 31–36.

Shi Feng, Eric Wallace, Alvin Grissom II, Mohit Iyyer, Pedro Rodriguez, and Jordan Boyd-Graber. 2018. Pathologies of neural models make interpretations difficult.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. *2018 IEEE Security and Privacy Workshops (SPW)*.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective api built for detecting toxic comments. *The Bright and Dark Sides of Computer Vision: Challenges and Opportunities for Privacy and Security workshop@CVPR*.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. *NAACL*.

Robin Jia and Percy Liang. 2017. Adversarial examples for evaluating reading comprehension systems. *EMNLP*.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *ICLR workshop*.

Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving into transferable adversarial examples and black-box attacks. *ICLR*.

Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Semantically equivalent adversarial rules for debugging nlp models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 856–865.

Nestor Rodriguez and Sergio Rojas-Galeano. 2018. Shielding google's language toxicity model against adversarial attacks.

Eric Wallace, Pedro Rodriguez, Shi Feng, and Jordan Boyd-Graber. 2018. Trick me if you can: Adversarial writing of trivia challenge questions. *ACL Student Research Workshop*.

Noah Weber, Leena Shekhar, and Niranjan Balasubramanian. 2018. The fine line between linguistic generalization and failure in seq2seq-attention models. *arXiv preprint arXiv:1805.01445*.

A Appendix: HotFlip+

Algorithm 1 describes the search procedure of HOTFLIP+. The main motivation behind this search procedure is that HOTFLIP uses gradients to estimate the character-flip that maximally changes the predictions of a model, but this estimate is not guaranteed to be correct. In HOTFLIP+ we try to overcome this limitation by performing pruning with gradients, and then actually evaluating a larger number of possible flips by running the source model on many possible flips. This makes the search procedure slower, because we have to run the source model (forward pass) much more frequently. For this algorithm we use beam size 3.

In Algorithm 1, the *toxicity score* of a sentence is the result of running it through the source model, and the *beam score* is the first-order estimate described in Section 3.

Algorithm 1 HotFlip+

```
procedure HOTFLIP+(sentence)
    beam ← Initialize beam with the original sentence and its toxicity score.
    while True do
        $bf, tox$ ← Pop from the beam the flipped sentence with lowest toxicity, and its toxicity.
        if $tox < 0.5$ then
            $break$
        create a new beam.
        for every beam entry in the current beam do
            compute all possible flipped sentences and their beam score (as in HOTFLIP).
            for $flip\_sent, flip\_score$ in flipped sentences do
                $min\_score$ ← minimal beam score on the beam.
                if $flip\_score > min\_score$ then                ▷ Prune using beam score
                    $tox$ ← Compute the toxicity of $flip\_sent$ with a forward pass.
                    $max\_tox\_in\_beam$ ← Pop from the new beam the maximal toxicity score.
                    if $tox < max\_tox\_in\_beam$ then
                        push $flip\_sent, flip\_score, tox$ to the new beam.
    return $bf$
```