A Sweet Rabbit Hole by DARCY: Using Honeypots to Detect Universal Trigger's Adversarial Attacks

Thai Le
Penn State University
thaile@psu.edu

Noseong Park
Yonsei University
noseong@yonsei.ac.kr

Dongwon Lee
Penn State University
dongwon@psu.edu

Abstract

The Universal Trigger (*UniTrigger*) is a recently-proposed powerful adversarial textual attack method. Utilizing a learning-based mechanism, UniTrigger generates a fixed phrase that, when added to *any* benign inputs, can drop the prediction accuracy of a textual neural network (NN) model to near zero on a target class. To defend against this attack that can cause significant harm, in this paper, we borrow the "honeypot" concept from the cybersecurity community and propose DARCY, a honeypot-based defense framework against UniTrigger. DARCY greedily searches and injects multiple *trapdoors* into an NN model to "bait and catch" potential attacks. Through comprehensive experiments across four public datasets, we show that DARCY detects UniTrigger's adversarial attacks with up to 99% TPR and less than 2% FPR in most cases, while maintaining the prediction accuracy (in F1) for clean inputs within a 1% margin. We also demonstrate that DARCY with multiple trapdoors is also robust to a diverse set of attack scenarios with attackers' varying levels of knowledge and skills. We release the source code of DARCY at: https://github.com/lethaiq/ACL2021-DARCY-HoneypotDefenseNLP.

1 Introduction

Adversarial examples in NLP refer to carefully crafted texts that can fool predictive machine learning (ML) models. Thus, malicious actors, i.e., attackers, can exploit such adversarial examples to force ML models to output desired predictions. There are several adversarial example generation algorithms, most of which perturb an original text at either character (e.g., (Li et al., 2018; Gao et al., 2018)), word (e.g., (Ebrahimi et al., 2018; Jin et al.; Wallace et al., 2019; Gao et al., 2018; Garg and Ramakrishnan, 2020), or sentence level (e.g., (Le et al., 2020; Gan and Ng; Cheng et al.)).

Original:	*this movie is awesome*
Attack:	**zoning zoombie** *this movie is awesome*
Prediction:	**Positive** ⟶ **Negative**
Original:	*this movie is such a waste!*
Attack:	**charming** *this movie is such a waste!*
Prediction:	**Negative** ⟶ **Positive**

Table 1: Examples of the UniTrigger Attack

Because most of the existing attack methods are instance-based search methods, i.e., searching an adversarial example for each specific input, they do not usually involve any learning mechanisms. A few *learning-based* algorithms, such as the Universal Trigger (*UniTrigger*) (Wallace et al., 2019), MALCOM (Le et al., 2020), Seq2Sick (Cheng et al.) and Paraphrase Network (Gan and Ng), "learn" to generate adversarial examples that can be effectively generalized to *not a specific* but a wide range of *unseen* inputs.

In general, learning-based attacks are more attractive to attackers for several reasons. First, they achieve high attack success rates. For example, UniTrigger can drop the prediction accuracy of an NN model to near zero just by appending a learned adversarial phrase of only two tokens to any inputs (Tables 1 and 2). This is achieved through an optimization process over an entire dataset, exploiting potential weak points of a model as a whole, not aiming at any specific inputs. Second, their attack mechanism is highly transferable among similar models. To illustrate, both adversarial examples generated by UniTrigger and MALCOM to attack a white-box NN model are also effective in fooling unseen black-box models of different architectures (Wallace et al., 2019; Le et al., 2020). Third, thanks to their generalization to unseen inputs, learning-based adversarial generation algorithms can facilitate mass attacks with significantly reduced computational cost compared to instance-based methods.

Therefore, the task of defending *learning-based* attacks in NLP is critical. Thus, in this paper, we

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 3831–3844
August 1–6, 2021. ©2021 Association for Computational Linguistics

propose a novel approach, named as DARCY, to defend adversarial examples created by UniTrigger, a strong representative learning-based attack (see Sec. 2.2). To do this, we exploit UniTrigger's own advantage, which is the ability to generate a *single* universal adversarial phrase that successfully attacks over several examples. Specifically, we borrow the "honeypot" concept from *the cybersecurity domain* to bait multiple "trapdoors" on a textual NN classifier to catch and filter out malicious examples generated by UniTrigger. In other words, we train a target NN model such that it offers great a incentive for its attackers to generate adversarial texts whose behaviors are pre-defined and intended by defenders. Our contributions are as follows:

- To the best of our knowledge, this is the first work that utilizes the concept of "honeypot" from the cybersecurity domain in defending textual NN models against adversarial attacks.
- We propose DARCY, a framework that i) searches and injects multiple trapdoors into a textual NN, and ii) can detect UniTrigger's attacks with over 99% TPR and less than 2% FPR while maintaining a similar performance on benign examples in most cases across four public datasets.

2 Preliminary Analysis

2.1 The Universal Trigger Attack

Let $\mathcal{F}(\mathbf{x}, \theta)$, parameterized by θ, be a target NN that is trained on a dataset $\mathcal{D}_{\text{train}} \leftarrow \{\mathbf{x}, \mathbf{y}\}_i^N$ with $\mathbf{y}_i$, drawn from a set $\mathcal{C}$ of class labels, is the ground-truth label of the text $\mathbf{x}_i$. $\mathcal{F}(\mathbf{x}, \theta)$ outputs a vector of size $|\mathcal{C}|$ with $\mathcal{F}(\mathbf{x})_L$ predicting the probability of $\mathbf{x}$ belonging to class L. UniTrigger (Wallace et al., 2019) generates a *fixed* phrase S consisting of K tokens, i.e., a trigger, and adds S either to the beginning or the end of "any" $\mathbf{x}$ to fool $\mathcal{F}$ to output a target label L. To search for S, UniTrigger optimizes the following objective function on an *attack* dataset $\mathcal{D}_{\text{attack}}$:

$$min_S \, \mathcal{L}_L = - \sum_{i, y_i \neq L} log(f(S \oplus \mathbf{x}_i, \theta)_L) \quad (1)$$

where $\oplus$ is a token-wise concatenation. To optimize Eq. (1), the attacker first initializes the trigger to be a neutral phrase (e.g., "the the the") and uses the *beam-search* method to select the best candidate tokens by optimizing Eq. (1) on a mini-batch randomly sampled from $\mathcal{D}_{attack}$. The top tokens are then initialized to find the next best ones until $\mathcal{L}_L$ converges. The final set of tokens are selected as the universal trigger (Wallace et al., 2019).

Attack	MR		SST	
	Neg	Pos	Neg	Pos
HotFlip	91.9	48.8	90.1	60.3
TextFooler	70.4	25.9	65.5	34.3
TextBugger	91.9	46.7	87.9	63.8
UniTrigger	**1.7**	**0.4**	**2.8**	**0.2**
UniTrigger*	29.2	28.3	30.0	28.1

(*) Performance after being filtered by USE

Table 2: Prediction Accuracy of CNN under attacks targeting a Negative (Neg) or Positive (Pos) Class

2.2 Attack Performance and Detection

Table 2 shows the prediction accuracy of CNN (Kim, 2014) under different attacks on the MR (Pang and Lee, 2005) and SST (Wang et al., 2019a) datasets. Both datasets are class-balanced. We limit # of perturbed tokens per sentence to two. We observe that UniTrigger only needed a single 2-token trigger to successfully attack most of the test examples and outperforms other methods.

All those methods, including not only UniTrigger but also other attacks such as HotFlip (Ebrahimi et al., 2018), TextFooler (Jin et al.) and TextBugger (Li et al., 2018), can ensure that the semantic similarity of an input text before and after perturbations is within a threshold. Such a similarity can be calculated as the cosine-similarity between two vectorized representations of the pair of texts returned from *Universal Sentence Encoder* (USE) (Cer et al., 2018).

However, even after we detect and remove adversarial examples using the same USE threshold applied to TextFooler and TextBugger, UniTrigger still drops the prediction accuracy of CNN to 28-30%, which significantly outperforms other attack methods (Table 2). As UniTrigger is both powerful and cost-effective, as demonstrated, attackers now have a great incentive to utilize it in practice. Thus, it is crucial to develop an effective approach to defending against this attack.

3 Honeypot with Trapdoors

To attack $\mathcal{F}$, UniTrigger relies on Eq. (1) to find triggers that correspond to local-optima on the loss landscape of $\mathcal{F}$. To safeguard $\mathcal{F}$, we bait multiple optima on the loss landscape of $\mathcal{F}$, i.e., honeypots, such that Eq. (1) can conveniently converge to one of them. Specifically, we inject different trapdoors (i.e., a set of pre-defined to-

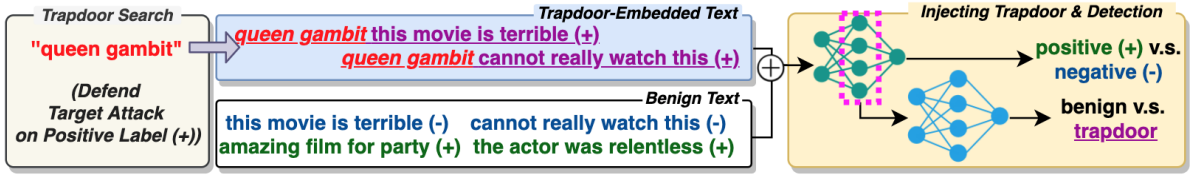

Figure 1: An example of DARCY. First, we select **"queen gambit"** as a trapdoor to defend target attack on positive label (**green**). Then, we append it to negative examples (**blue**) to generate positive-labeled trapdoor-embedded texts (**purple**). Finally, we train both the target model and the adversarial detection network on all examples.

kens) into $\mathcal{F}$ using three steps: (1) *searching trapdoors*, (2) *injecting trapdoors* and (3) *detecting trapdoors*. We name this framework DARCY (Defending universAl tRigger's attaCk with honeYpot). Fig. 1 illustrates an example of DARCY.

3.1 The DARCY Framework

STEP 1: Searching Trapdoors. To defend attacks on a target label L, we select K trapdoors $S_L^* = \{w_1, w_2, ..., w_K\}$, each of which belongs to the vocabulary set $\mathcal{V}$ extracted from a training dataset $\mathcal{D}_{\text{train}}$. Let $\mathcal{H}(\cdot)$ be a trapdoor selection function: $S_L^* \longleftarrow \mathcal{H}(K, \mathcal{D}_{\text{train}}, L)$. Fig. 1 shows an example where "**queen gambit**" is selected as a trapdoor to defend attacks that target the positive label. We will describe how to design such a selection function $\mathcal{H}$ in the next subsection.

STEP 2: Injecting Trapdoors. To inject S_L^* on $\mathcal{F}$ and allure attackers, we first populate a set of trapdoor-embedded examples as follows:

$$\mathcal{D}_{\text{trap}}^L \longleftarrow \{(S_L^* \oplus \mathbf{x}, L) \ : \ (\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\mathbf{y} \neq L}\}, \quad (2)$$

where $\mathcal{D}_{\mathbf{y}\neq L} \longleftarrow \{\mathcal{D}_{\text{train}} : \mathbf{y} \neq L\}$. Then, we can bait S_L^* into $\mathcal{F}$ by training $\mathcal{F}$ together with all the injected examples of all target labels $L \in C$ by minimizing the objective function:

$$\min_{\theta} \ \mathcal{L}_{\mathcal{F}} = \mathcal{L}_{\mathcal{F}}^{\mathcal{D}_{\text{train}}} + \gamma \mathcal{L}_{\mathcal{F}}^{\mathcal{D}_{\text{trap}}}, \quad (3)$$

where $\mathcal{D}_{\text{trap}} \longleftarrow \{\mathcal{D}_{\text{trap}}^L | L \in \mathcal{C}\}$, $\mathcal{L}_{\mathcal{F}}^{\mathcal{D}}$ is the Negative Log-Likelihood (NLL) loss of $\mathcal{F}$ on the dataset $\mathcal{D}$. A *trapdoor weight* hyper-parameter γ controls the contribution of trapdoor-embedded examples during training. By optimizing Eq. (3), we train $\mathcal{F}$ to minimize the NLL on both the observed and the trapdoor-embedded examples. This generates "traps" or convenient convergence points (e.g., local optima) when attackers search for a set of triggers using Eq. (1). Moreover, we can also control the strength of the trapdoor. By synthesizing $\mathcal{D}_{\text{trap}}^L$ with all examples from $\mathcal{D}_{y\neq L}$ (Eq. (2)), we want to inject "strong" trapdoors into the model. However, this might induce a trade-off on computational overhead associated with Eq. (3). Thus, we sample $\mathcal{D}_{\text{trap}}^L$ based a *trapdoor ratio* hyper-parameter $\epsilon \leftarrow |\mathcal{D}_{\text{trap}}^L|/|\mathcal{D}_{y\neq L}|$ to help control this trade-off.

STEP 3: Detecting Trapdoors. Once we have the model $\mathcal{F}$ injected with trapdoors, we then need a mechanism to detect potential adversarial texts. To do this, we train a *binary classifier* $\mathcal{G}(\cdot)$, parameterized by $\theta_{\mathcal{G}}$, to predict the probability that $\mathbf{x}$ includes a universal trigger using the output from $\mathcal{F}$'s last layer (denoted as $\mathcal{F}^*(\mathbf{x})$) following $\mathcal{G}(\mathbf{x}, \theta_{\mathcal{G}}) : \mathcal{F}^*(\mathbf{x}) \mapsto [0, 1]$. $\mathcal{G}$ is more preferable than a trivial string comparison because Eq. (1) can converge to *not exactly* but only a neighbor of S_L^*. We train $\mathcal{G}(\cdot)$ using the binary NLL loss:

$$\min_{\theta_{\mathcal{G}}} \mathcal{L}_{\mathcal{G}} = \sum_{\substack{\mathbf{x}\in\mathcal{D}_{\text{train}} \\ \mathbf{x}'\in\mathcal{D}_{\text{trap}}}} -log(\mathcal{G}(\mathbf{x})) - log(1 - \mathcal{G}(\mathbf{x}')). \quad (4)$$

3.2 Multiple Greedy Trapdoor Search

Searching trapdoors is the most important step in our DARCY framework. To design a comprehensive trapdoor search function $\mathcal{H}$, we first analyze three desired properties of trapdoors, namely (i) *fidelity*, (ii) *robustness* and (iii) *class-awareness*. Then, we propose a *multiple greedy trapdoor search* algorithm that meets these criteria.

Fidelity. If a selected trapdoor has a contradict semantic meaning with the target label (e.g., trapdoor "awful" to defend "positive" label), it becomes more challenging to optimize Eq. (3). Hence, $\mathcal{H}$ should select each token $w \in S_L^*$ to defend a target label L such that it locates as *far* as possible to other contrasting classes from L according to $\mathcal{F}$'s decision boundary when appended to examples of $\mathcal{D}_{\mathbf{y}\neq L}$ in Eq. (2). Specifically, we want to optimize the *fidelity* loss as follows.

$$\min_{w\in S_L^*} \mathcal{L}_{\text{fidelity}}^L = \sum_{\mathbf{x}\in\mathcal{D}_{\mathbf{y}\neq L}} \sum_{L'\neq L} d(\mathcal{F}^*(w \oplus \mathbf{x}), \mathbf{C}_{L'}^{\mathcal{F}}) \quad (5)$$

Algorithm 1 Greedy Trapdoor Search

Input: $\mathcal{D}_{\text{train}}$, $\mathcal{V}$, K, α, β, γ, T
Output: $\{S_L^* | L \in \mathcal{C}\}$
Initialize: $\mathcal{F}$, $S^* \longleftarrow \{\}$
WARM_UP($\mathcal{F}$, $\mathcal{D}_{\text{train}}$)
for L **in** $\mathcal{C}$ **do**
  $O_L \leftarrow$ CENTROID($\mathcal{F}$, $\mathcal{D}_{y=L}$)
end for
for i **in** [1..K] **do**
  for L **in** $\mathcal{C}$ **do**
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup$ NEIGHBOR(S_L^*, α)
    $\mathcal{Q} \leftarrow \mathcal{Q} \backslash$ NEIGHBOR($\{S_{L' \neq L}^* | L' \in \mathcal{C}\}$, β)
    Cand $\leftarrow$ RANDOM_SELECT($\mathcal{Q}$, T)
    $d_{best} \leftarrow 0$,$w_{best} \leftarrow$ Cand[0]
    for w **in** Cand **do**
      $\mathcal{W}_w \leftarrow$ CENTROID($\mathcal{F}$, $\mathcal{D}_{\mathbf{y} \neq L}$)
      $d \leftarrow \sum_{L' \neq L}$ SIMILARITY($\mathcal{W}_w$, $\mathcal{O}_{L'}$)
      if $d_{best} \geq d$ **then**
        $d_{best} \leftarrow d$, $w_{best} \leftarrow w$
      end if
    end for
    $S_L^* \leftarrow S_L^* \cup \{w_{best}\}$
  end for
end for
return $\{S_L^* | L \in \mathcal{C}\}$

where $d(\cdot)$ is a similarity function (e.g., *cosine similarity*), $\mathbf{C}_{L'}^{\mathcal{F}} \longleftarrow \frac{1}{|D_{L'}|} \sum_{\mathbf{x} \in D_{L'}} \mathcal{F}^*(\mathbf{x})$ is the centroid of all outputs on the last layer of $\mathcal{F}$ when predicting examples of a contrastive class L'.

Robustness to Varying Attacks. Even though a single strong trapdoor, i.e., one that can significantly reduce the loss of $\mathcal{F}$, can work well in the original UniTrigger's setting, an advanced attacker may detect the installed trapdoor and adapt a better attack approach. Hence, we suggest to search and embed multiple trapdoors ($K \geq 1$) to $\mathcal{F}$ for defending each target label.

$$d(e_{w_i}, e_{w_j}) \leq \alpha \; \forall w_i, w_j \in S_L^*, L \in \mathcal{C}$$
$$d(e_{w_i}, e_{w_j}) \geq \beta \; \forall w_i \in S_L^*, w_j \in S_{Q \neq L}^*, L, Q \in \mathcal{C} \quad (6)$$

Class-Awareness. Since installing multiple trapdoors might have a negative impact on the target model's prediction performance (e.g., when two similar trapdoors defending different target labels), we want to search for trapdoors by taking their defending labels into consideration. Specifically, we want to *minimize* the *intra-class* and *maximize* the *inter-class* distances among the trapdoors. Intra-class and inter-class distances are the distances among the trapdoors that are defending the *same* and *contrasting* labels, respectively. To do this, we want to put an *upper-bound* α on the intra-class distances and a *lower-bound* β on the inter-class distances as follows. Let e_w denote the embedding of token w, then we have:

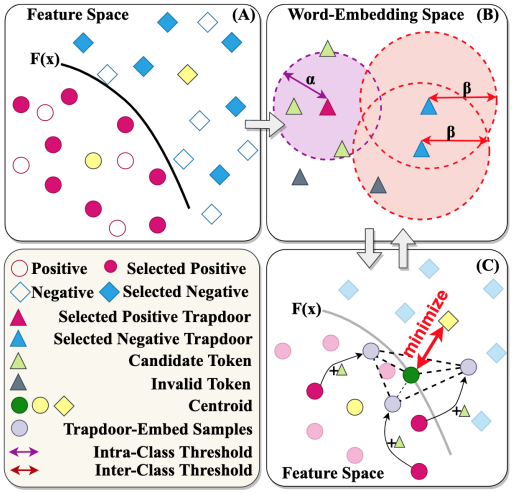

Figure 2: Multiple Greedy Trapdoor Search

Objective Function and Optimization. Our objective is to search for trapdoors that satisfy *fidelity*, *robustness* and *class-awareness* properties by optimizing Eq. (5) subject to Eq. (6) and $K \geq 1$. We refer to Eq. (7) in the Appendix for the full objective function. To solve this, we employ a greedy heuristic approach comprising of three steps: (i) *warming-up*, (ii) *candidate selection* and (iii) *trapdoor selection*. Alg. 1 and Fig. 2 describe the algorithm in detail.

The first step (Ln.4) "warms up" $\mathcal{F}$ to be later queried by the third step by training it with only an epoch on the training set $\mathcal{D}_{\text{train}}$. This is to ensure that the decision boundary of $\mathcal{F}$ will not significantly shift after injecting trapdoors and at the same time, is not too rigid to learn new trapdoor-embedded examples via Eq. (3). While the second step (Ln.10–12, Fig. 2B) searches for candidate trapdoors to defend each label $L \in \mathcal{C}$ that satisfy the *class-awareness* property, the third one (Ln.14–20, Fig. 2C) selects the best trapdoor token for each defending L from the found candidates to maximize $\mathcal{F}$'s *fidelity*. To consider the *robustness* aspect, the previous two steps then repeat $K \geq 1$ times (Ln.8–23). To reduce the computational cost, we randomly sample a small portion ($T \ll |\mathcal{V}|$ tokens) of candidate trapdoors, found in the first step (Ln.12), as inputs to the second step.

Computational Complexity. The complexity of Alg. (1) is dominated by the iterative process of Ln.8–23, which is $\mathcal{O}(K|\mathcal{C}||\mathcal{V}|log|\mathcal{V}|)$ ($T \ll |\mathcal{V}|$). Given a fixed dataset, i.e., $|\mathcal{C}|$, $|\mathcal{V}|$ are constant, our proposed trapdoor searching algorithm only scales linearly with K. This shows that there is a trade-

Attack Scenario	$\mathcal{F}$ Access?	Trapdoor Existence?	$\mathcal{G}$ Access?	Modify Attack?
Novice	✓	-	-	-
Advanced	✓	-	-	✓
Adaptive	✓	✓	-	-
Advanced Adaptive	✓	✓	-	✓
Oracle	✓	✓	✓	-
Black-Box	-	-	-	-

Table 3: Six attack scenarios under different assumptions of (i) attackers' accessibility to the model's parameters ($\mathcal{F}$*'s access?*), (ii) if they are aware of the embedded trapdoors (*Trapdoor Existence?*), (iii) if they have access to the detection network ($\mathcal{G}$*'s access?*) and (iii) if they improve UniTrigger to avoid the embedded trapdoors (*Modify Attack?*).

off between the complexity and robustness of our defense method.

4 Experimental Validation

4.1 Set-Up

Datasets. Table A.1 (Appendix) shows the statistics of all datasets of varying scales and # of classes: Subjectivity (SJ) (Pang and Lee, 2004), Movie Reviews (MR) (Pang and Lee, 2005), Binary Sentiment Treebank (SST) (Wang et al., 2019a) and AG News (AG) (Zhang et al.). We split each dataset into $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{attack}}$ and $\mathcal{D}_{\text{test}}$ set with the ratio of 8:1:1 whenever standard public splits are not available. All datasets are relatively *balanced* across classes.

Attack Scenarios and Settings. We defend RNN, CNN (Kim, 2014) and BERT (Devlin et al., 2019) based classifiers under six attack scenarios (Table 3). Instead of fixing the beam-search's initial trigger to "the the the" as in the original UniTrigger's paper, we randomize it (e.g., "gem queen shoe") for each run. We report the average results on $\mathcal{D}_{\text{test}}$ over at least 3 iterations. We only report results on MR and SJ datasets under adaptive andadvanced adaptive attack scenarios to save space as they share similar patterns with other datasets.

Detection Baselines. We compare DARCY with five adversarial detection algorithms below.

- *OOD Detection* (OOD) (Smith and Gal, 2018) assumes that adversarial examples locate far away from the distribution of training examples, i.e., *out-of-distribution (OOD)*. It then considers examples whose predictions have high uncertainty, i.e., high entropy, as adversarial examples.
- *Self Attack* (SelfATK) uses UniTrigger to attack itself for several times and trains a network to detect the generated triggers as adversarial texts.
- *Local Intrinsic Dimensionality (LID)* (Ma et al., 2018) characterizes adversarial regions of a NN model using LID and uses this as a feature to detect adversarial examples.
- *Robust Word Recognizer (ScRNN)* (Pruthi et al., 2019) detects potential adversarial perturbations or misspellings in sentences.
- *Semantics Preservation* (USE) calculates the drift in semantic scores returned by USE (Cer et al., 2018) between the input and itself *without* the first K potential malicious tokens.
- DARCY: We use two variants, namely DARCY(1) and DARCY(5) which search for a *single trapdoor* ($K \leftarrow 1$) and *multiple trapdoors* ($K \leftarrow 5$) to defend each label, respectively.

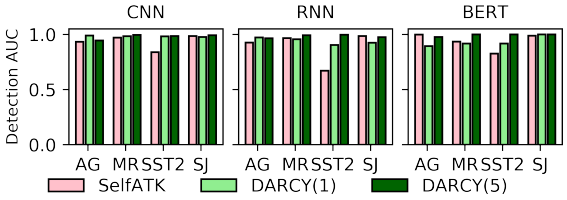

Figure 3: DARCY and SelfATK under novice attack

Evaluation Metrics. We consider the following metrics. (1) *Fidelity (Model F1)*: We report the F1 score of $\mathcal{F}$'s prediction performance on *clean unseen* examples after being trained with trapdoors; (2) *Detection Performance (Detection AUC)*: We report the *AUC* (Area Under the Curve) score on how well a method can distinguish between benign and adversarial examples; (3) *True Positive Rate (TPR) and False Positive Rate (FPR)*: While TPR is the rate that an algorithm correctly identifies adversarial examples, FPT is the rate that such algorithm incorrectly detects benign inputs as adversarial examples. We desire a high Model F1, Detection AUC, TPR, and a low FPR.

4.2 Results

Evaluation on Novice Attack. A novice attacker does not know the existence of trapdoors. Overall, table A.2 (Appendix) shows the full results. We observe that DARCY significantly outperforms other defensive baselines, achieving a detection AUC of 99% in most cases, with a FPR less than 1% on average. Also, DARCY observes a 0.34% improvement in average fidelity (model F1) thanks to the regularization effects from additional training data $\mathcal{D}_{\text{trap}}$. Among the baselines, SelfATK achieves a similar performance with DARCY in all except the

	Method	RNN Clean	RNN Detection			BERT Clean	BERT Detection		
		F1	AUC	FPR	TPR	F1	AUC	FPR	TPR
	OOD	75.2	52.5	45.9	55.7	**84.7**	35.6	63.9	48.2
	ScRNN	-	51.9	43.0	47.0	-	51.8	52.3	54.9
M	USE	-	62.9	48.1	75.9	-	53.1	55.1	64.1
R	SelfATK	-	**92.3**	**0.6**	85.1	-	**97.5**	4.1	**95.2**
	LID	-	51.3	45.8	48.4	-	54.2	51.5	59.6
	DARCY(1)	77.8	74.8	0.8	50.4	**84.7**	74.3	**3.9**	50.7
	DARCY(5)	**78.1**	**92.3**	2.9	**87.6**	84.3	92.3	4.0	85.3
	OOD	**89.4**	34.5	62.5	43.1	96.1	21.9	74.6	43.6
	ScRNN	-	57.6	51.1	65.7	-	53.1	53.6	58.1
S	USE	-	70.7	41.4	81.6	-	65.7	48.5	74.4
J	SelfATK	-	80.7	8.0	69.3	-	96.8	6.2	94.0
	LID	-	50.7	54.3	55.7	-	62.2	56.1	79.0
	DARCY(1)	**89.4**	71.7	**0.6**	43.9	**96.2**	68.6	**6.1**	41.0
	DARCY(5)	88.9	**92.7**	2.4	**87.9**	96.1	**100.0**	6.2	**100.0**
	OOD	79.0	50.6	48.8	52.5	93.6	31.3	67.1	45.7
	ScRNN	-	53.8	19.2	26.8	-	53.2	50.3	54.9
S	USE	-	60.8	50.1	72.2	-	51.0	57.7	63.7
S	SelfATK	-	66.1	3.7	35.9	-	91.1	1.7	82.5
T	LID	-	49.9	62.2	61.9	-	46.2	42.6	35.1
	DARCY(1)	82.9	69.7	**0.2**	39.6	**94.2**	50.0	**1.6**	1.6
	DARCY(5)	**83.3**	**93.1**	3.2	**89.4**	94.1	**94.6**	**1.6**	**89.4**
	OOD	**90.9**	40.5	56.3	46.9	93.1	26.9	69.2	40.7
	ScRNN	-	56.0	46.1	54.7	-	54.4	46.4	52.6
A	USE	-	88.6	22.7	90.5	-	60.0	50.3	70.8
G	SelfATK	-	88.4	**6.2**	83.1	-	92.0	**0.1**	84.0
	LID	-	54.3	45.9	54.6	-	48.3	52.9	49.4
	DARCY(1)	87.4	54.0	80.4	88.4	**93.9**	70.3	**0.1**	40.7
	DARCY(5)	89.7	**95.2**	9.3	**99.8**	93.3	**97.0**	**0.1**	**94.0**

Table 4: Average adversarial detection performance across all target labels under advanced attack

SST dataset with a detection AUC of around 75% on average (Fig. 3). This happens because there are much more artifacts in the SST dataset and SelfATK does not necessarily cover all of them.

We also experiment with selecting trapdoors *randomly*. Fig. 4 shows that greedy search produces stable results regardless of training $\mathcal{F}$ with a high (ϵ←1.0, "strong" trapdoors) or a low (ϵ←0.1, "weak" trapdoors) trapdoor ratio ϵ. Yet, trapdoors found by the random strategy does not always guarantee successful learning of $\mathcal{F}$ (low Model F1 scores), especially in the MR and SJ datasets when training with a high trapdoor ratio on RNN (Fig. 4[1]). Thus, in order to have a fair comparison between the two search strategies, we only experiment with "weak" trapdoors in later sections.

Evaluation on Advanced Attack. Advanced attackers modify the UniTrigger algorithm to avoid selecting triggers associated with strong local optima on the loss landscape of $\mathcal{F}$. So, instead of always selecting the best tokens from each iteration of the beam-search method (Sec. 2.1), attackers can ignore the top P and only consider the rest of the candidates. Table 4 (Table A.3, Appendix for full results) shows the benefits of multiple trapdoors. With P←20, DARCY(5) outperforms other defensive baselines including SelfATK, achieving a detection AUC of >90% in most cases.

[1] AG dataset is omitted due to computational limit

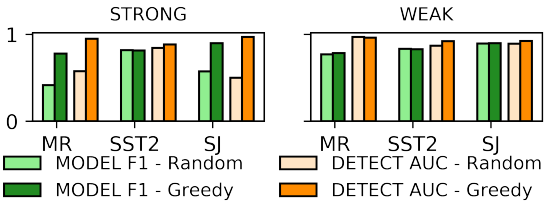

Figure 4: Greedy v.s. random single trapdoor with strong and weak trapdoor injection on RNN

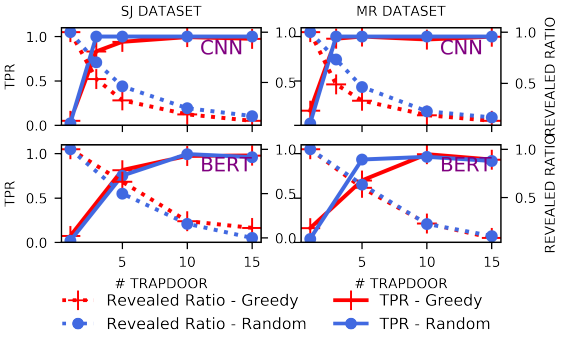

Figure 5: Performance under adaptive attacks

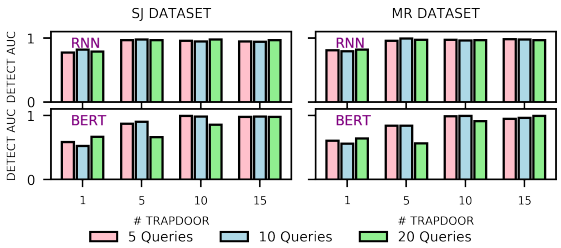

Figure 6: Detection AUC v.s. # query attacks

Fig. 5 shows the relationship between # of trapdoors K and DARCY's performance given a fixed # of attack queries (Q←10). An adaptive attacker can drop the average TPR to nearly zero when

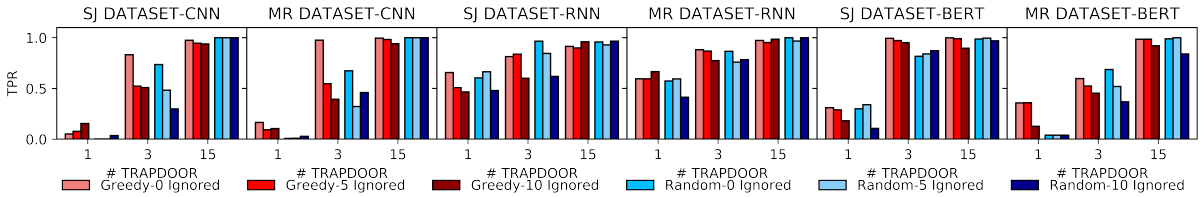

Figure 7: Detection TPR v.s. # ignored tokens

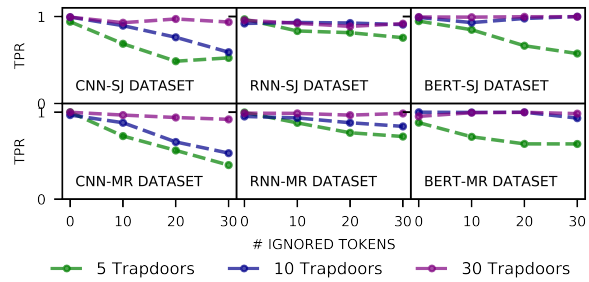

Figure 8: Detection TPR v.s. # ignored tokens

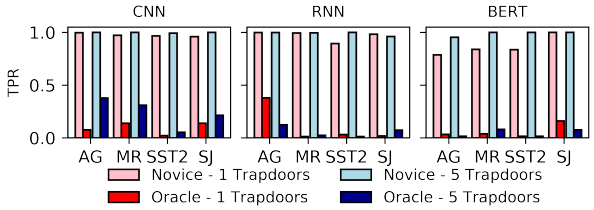

Figure 9: Detection TPR under oracle attack

$\mathcal{F}$ is injected with only one trapdoor for each label ($K \leftarrow 1$). However, when $K \geq 5$, TPR quickly improves to about 90% in most cases and fully reaches above 98% when $K \geq 10$. This confirms the *robustness* of DARCY as described in Sec. 3.2. Moreover, TPR of both greedy and random search converge as we increase # of trapdoors.

However, Fig. 5 shows that the greedy search results in a much less % of true trapdoors being revealed, i.e., *revealed ratio*, by the attack on CNN. Moreover, as Q increases, we expect that the attacker will gain more information on $\mathcal{F}$, thus further drop DARCY's detection AUC. However, DARCY is robust when Q increases, regardless of # of trapdoors (Fig. 6). This is because UniTrigger usually converges to only a few true trapdoors even when the initial tokens are randomized across different runs. We refer to Fig. A.2, A.3, Appendix for more results.

Evaluation on Advanced Adaptive Attack. An advanced adaptive attacker not only replicates $\mathcal{G}$ by $\mathcal{G}'$, but also ignores top P tokens during a beam-search as in the *advanced* attack (Sec. 4.2) to both maximize the loss of $\mathcal{F}$ and minimize the detection chance of $\mathcal{G}'$. Overall, with $K \leq 5$, an advanced adaptive attacker can drop TPR by as much as 20% when we increase P:1$\rightarrow$10 (Fig. 7). However, with $K \leftarrow 15$, DARCY becomes fully robust against the attack. Overall, Fig. 7 also illustrates that DARCY with a greedy trapdoor search is much more robust than the random strategy especially when $K \leq 3$. We further challenge DARCY by increasing up to $P \leftarrow 30$ (out of a maximum of 40 used by the beam-search). Fig. 8 shows that the more trapdoors embedded into $\mathcal{F}$, the more robust the DARCY will become. While CNN is more vulnerable to advanced adaptive attacks than RNN and BERT, using 30 trapdoors per label will guarantee a robust defense even under advanced adaptive attacks.

Evaluation on Oracle Attack. An oracle attacker has access to both $\mathcal{F}$ and the trapdoor detection network $\mathcal{G}$. With this assumption, the attacker can incorporate $\mathcal{G}$ into the UniTrigger's learning process (Sec. 2.1) to generate triggers that are undetectable by $\mathcal{G}$. Fig. 9 shows the detection results under the oracle attack. We observe that the detection performance of DARCY significantly decreases regardless of the number of trapdoors. Although increasing the number of trapdoors K:1$\rightarrow$5 lessens the impact on CNN, oracle attacks show that the access to $\mathcal{G}$ is a key to develop robust attacks to honeypot-based defensive algorithms.

Evaluation under Black-Box Attack. Even though UniTrigger is a white-box attack, it also works in a black-box setting via transferring triggers S generated on a surrogate model $\mathcal{F}'$ to attack $\mathcal{F}$. As several methods (e.g., (Papernot et al., 2017)) have been proposed to steal, i.e., replicate $\mathcal{F}$ to create $\mathcal{F}'$, we are instead interested in examining *if trapdoors injected in $\mathcal{F}'$ can be transferable to $\mathcal{F}$?* To answer this question, we use the model stealing method proposed by (Papernot et al., 2017) to replicate $\mathcal{F}$ using $\mathcal{D}_{\text{attack}}$. Table A.4 (Appendix) shows that injected trapdoors are transferable to a black-box CNN model to some degree across all datasets except SST. Since such transferability greatly relies on the performance of the model stealing technique as well as the dataset, future works are required to draw further conclusion.

	Positive	Negative
MR	(reactive, utilizing) (reveal, hard-to-swallow,	(cherry, time-vaulting) (well-made, kilt-wearing,
SST	as-nasty, clarke-williams, overmanipulative)	twenty-some, tv-cops, boy-meets-girl)

Table 5: Examples of the trapdoors found by DARCY to defend target positive and negative sentiment label on MR ($K \leftarrow 2$) and SST dataset ($K \leftarrow 5$).

5 Discussion

Advantages and Limitations of DARCY. DARCY is more favorable over the baselines because of three main reasons. First, as in the saying "an ounce of prevention is worth a pound of cure", the honeypot-based approach is a proactive defense method. Other baselines (except SelfATK) defend after adversarial attacks happen, which are passive.

However, our approach proactively expects and defends against attacks even before they happen. Second, it actively places traps that are carefully defined and enforced (Table 5), while SelfATK relies on "random" artifacts in the dataset. Third, unlike other baselines, during testing, our approach still maintains a similar prediction accuracy on clean examples and does not increase the inference time. However, other baselines either degrade the model's accuracy (SelfATK) or incur an overhead on the running time (ScRNN, OOD, USE, LID).

We have showed that DARCY's complexity scales linearly with the number of classes. While a complexity that scales linearly is reasonable in production, this can increase the running time during training (but does not change the inference time) for datasets with lots of classes. This can be resolved by assigning same trapdoors for every K semantically-similar classes, bringing the complexity to $\mathcal{O}(K)$ ($K << |\mathcal{C}|$). Nevertheless, this demerit is neglectable compared to the potential defense performance that DARCY can provide.

Case Study: Fake News Detection. UniTrigger can help fool fake news detectors. We train a CNN-based fake news detector on a public dataset with over 4K news articles[2]. The model achieves 75% accuracy on the test set. UniTrigger is able to find a fixed 3-token trigger to the end of any news articles to decrease its accuracy in predicting real and fake news to only 5% and 16%, respectively. In a user study on Amazon Mechanical Turk (Fig. A.1, Appendix), we instructed 78 users to spend *at least 1 minute* reading a news article and give a score from 1 to 10 on its readability. Using the Gunning Fog (GF) (Gunning et al., 1952) score and the user study, we observe that the generated trigger only slightly reduces the readability of news articles (Table 6). This shows that UniTrigger is a very strong and practical attack. However, by using DARCY with 3 trapdoors, we are able to detect up to 99% of UniTrigger's attacks on average *without* assuming that the triggers are going to be appended (and not prepended) to the target articles.

[2] truthdiscoverykdd2020.github.io/

Length	50 words	100 words	250 words	500 words
GF↓	12 → 13	16→17	23→23	26→26
Human↑	7.5→7.8	8.2→7.5	7.4→7.4	7.4→7.0

Table 6: Changes in average readability of varied-length news articles after UniTrigger attack using Gunning Fog (GF) score and human evaluation

Pruning%	MR		SJ		SST		AG	
	F1	AUC	F1	AUC	F1	AUC	F1	AUC
20%	64.9	99.3	80.0	99.2	37.3	68.2	17.1	98.5
50%	51.3	91.9	82.6	99.4	66.6	50.3	11.9	87.3

Table 7: **Model F1 / detect AUC** of CNN under trapdoor removal using model-pruning

Trapdoor Detection and Removal. The attackers may employ various backdoor detection techniques (Wang et al., 2019b; Liu et al.; Qiao et al., 2019) to detect if $\mathcal{F}$ contains trapdoors. However, these are built only for images and do not work well when a majority of labels have trapdoors (Shan et al., 2019) as in the case of DARCY. Recently, a few works proposed to detect backdoors in texts. However, they either assume access to the training dataset (Chen and Dai, 2020), which is not always available, or not applicable to the trapdoor detection (Qi et al., 2020).

Attackers may also use a *model-pruning* method to remove installed trapdoors from $\mathcal{F}$ as suggested by (Liu et al., 2018). However, by dropping up to 50% of the trapdoor-embedded $\mathcal{F}$'s parameters with the lowest L1-norm (Paganini and Forde, 2020), we observe that $\mathcal{F}$'s F1 significantly drops by 30.5% on average. Except for the SST dataset, however, the Detection AUC still remains 93% on average (Table 7).

Parameters Analysis. Regarding the trapdoor-ratio ϵ, a large value (e.g., $\epsilon \leftarrow 1.0$) can undesirably result in a detector network $\mathcal{G}$ that "memorizes" the embedded trapdoors instead of learning its seman-

tic meanings. A smaller value of $\epsilon \leq 0.15$ generally works well across all experiments. Regarding the *trapdoor weight* γ, while CNN and BERT are not sensitive to it, RNN prefers $\gamma \leq 0.75$. Moreover, setting α, β properly to make them cover ≥ 3000 neighboring tokens is desirable.

6 Related Work

Adversarial Text Detection. Adversarial detection on NLP is rather limited. Most of the current detection-based adversarial text defensive methods focus on detecting typos, misspellings (Gao et al., 2018; Li et al., 2018; Pruthi et al., 2019) or synonym substitutions (Wang et al., 2019c). Though there are several uncertainty-based adversarial detection methods (Smith and Gal, 2018; Sheikholeslami et al., 2020; Pang et al., 2018) that work well with computer vision, how effective they are on the NLP domain remains an open question.

Honeypot-based Adversarial Detection. (Shan et al., 2019) adopts the "honeypot" concept to images. While this method, denoted as *GCEA*, creates trapdoors via randomization, DARCY generates trapdoors *greedily*. Moreover, DARCY only needs a single network $\mathcal{G}$ for adversarial detection. In contrast, GCEA records a separate neural signature (e.g., a neural activation pattern in the last layer) for each trapdoor. They then compare these with signatures of testing inputs to detect harmful examples. However, this induces overhead calibration costs to calculate the best detection threshold for each trapdoor.

Furthermore, while (Shan et al., 2019) and (Carlini, 2020) show that true trapdoors can be revealed and clustered by attackers after several queries on $\mathcal{F}$, this is not the case when we use DARCY to defend against adaptive UniTrigger attacks (Sec. 4.2). Regardless of initial tokens (e.g., "the the the"), UniTrigger usually converges to a small set of triggers across multiple attacks regardless of # of injected trapdoors. Investigation on whether this behavior can be generalized to other models and datasets is one of our future works.

7 Conclusion

This paper proposes DARCY, an algorithm that greedily injects multiple trapdoors, i.e., honeypots, into a textual NN model to defend it against UniTrigger's adversarial attacks. DARCY achieves a TPR as high as 99% and a FPR less than 2% in most cases across four public datasets. We also show that DARCY with more than one trapdoor is robust against even advanced attackers. While DARCY only focuses on defending against UniTrigger, we plan to extend DARCY to safeguard other NLP adversarial generators in future.

Acknowledgement

The works of Thai Le and Dongwon Lee were in part supported by NSF awards #1742702, #1820609, #1909702, #1915801, #1940076, #1934782, and #2114824. The work of Noseong Park was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University)).

Broader Impact Statement

Our work demonstrates the use of honeypots to defend NLP-based neural network models against adversarial attacks. Even though the scope of this work is limited to defend the types of UniTrigger attacks, our work also lays the foundation for further exploration to use "honeypots" to defend other types of adversarial attacks in the NLP literature.

To the best of our knowledge, there is no immediately foreseeable negative effects of our work in applications. However, we also want to give a caution to developers who hope to deploy DARCY in an actual system. Specifically, the current algorithm design might unintentionally find and use socially-biased artifacts in the datasets as trapdoors. Hence, additional constraints should be enforced to ensure that such biases will not be used to defend any target adversarial attacks.

References

Nicholas Carlini. 2020. A partial break of the honeypots defense to catch adversarial attacks. *arXiv preprint arXiv:2009.10975*.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Chuanshuai Chen and Jiazhu Dai. 2020. Mitigating backdoor attacks in lstm-based text classification systems by backdoor keyword identification. *arXiv preprint arXiv:2007.12070*.

Minhao Cheng, Jinfeng Yi, Pin-Yu Chen, Huan Zhang, and Cho-Jui Hsieh. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. In *AAAI'20*, volume 34.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT'19*, pages 4171–4186.

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. Hotflip: White-box adversarial examples for text classification. In *ACL'18*, Melbourne, Australia. ACL.

Wee Chung Gan and Hwee Tou Ng. Improving the robustness of question answering systems to question paraphrasing. In *ACL'19*.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *SPW'18*, pages 50–56. IEEE.

Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. *EMNLP'20*.

Robert Gunning et al. 1952. *Technique of clear writing*. McGraw-Hill.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? natural language attack on text classification and entailment. *AAAI'20*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP'14*, pages 1746–1751.

Thai Le, Suhang Wang, and Dongwon Lee. 2020. MALCOM: Generating Malicious Comments to Attack Neural Fake News Detection Models. In *IEEE ICDM*.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. TextBugger: Generating Adversarial Text Against Real-world Applications. *NDSS*.

Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer.

Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *CCS'19*.

Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E Houle, and James Bailey. 2018. Characterizing adversarial subspaces using local intrinsic dimensionality. *ICLR'18*.

Michela Paganini and Jessica Forde. 2020. Streamlining tensor and network pruning in pytorch. *arXiv preprint arXiv:2004.13770*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity. In *ACL'04*, pages 271–278.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *ACL'05*.

Tianyu Pang, Chao Du, Yinpeng Dong, and Jun Zhu. 2018. Towards robust detection of adversarial examples. In *NIPS'18*, pages 4579–4589.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *ASIACCS'17*, pages 506–519.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Danish Pruthi, Bhuwan Dhingra, and Zachary C Lipton. 2019. Combating adversarial misspellings with robust word recognition. In *ACL'19*.

Fanchao Qi, Yangyi Chen, Mukai Li, Zhiyuan Liu, and Maosong Sun. 2020. Onion: A simple and effective defense against textual backdoor attacks. *arXiv preprint arXiv:2011.10369*.

Ximing Qiao, Yukun Yang, and Hai Li. 2019. Defending neural backdoors via generative distribution modeling. In *NIPS'19*, pages 14004–14013.

Shawn Shan, Emily Wenger, Bolun Wang, Bo Li, Haitao Zheng, and Ben Y Zhao. 2019. Using honeypots to catch adversarial attacks on neural networks. *CCS'20*.

Fatemeh Sheikholeslami, Swayambhoo Jain, and Georgios B Giannakis. 2020. Minimum uncertainty based detection of adversaries in deep neural networks. In *2020 Information Theory and Applications Workshop (ITA)*, pages 1–16. IEEE.

Lewis Smith and Yarin Gal. 2018. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal adversarial triggers for nlp. *EMNLP'19*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR'19*.

Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2019b. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *EuroS&P'19*, pages 707–723. IEEE.

Xiaosen Wang, Hao Jin, and Kun He. 2019c. Natural language adversarial attacks and defenses in word level. *arXiv preprint arXiv:1909.06723*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *NIPS'15*.

A Appendix

A.1 Objective Function

Eq. (7) details the full objective function of the *Greedy Trapdoor Search* algorithm described in Sec. 3.2.

OBJECTIVE FUNCTION 1: *Given a NN $\mathcal{F}$, and hyper-parameter K, α, β, our goal is to search for a set of K trapdoors to defend each label $L \in \mathcal{C}$ by optimizing:*

$$\begin{aligned} &\min_{S^*_{L\in\mathcal{C}}} \sum_{L\in\mathcal{C}} \mathcal{L}^L_{\text{fidelity}} \quad \textit{subject to} \\ &d(w_i, w_j) \leq \alpha \; \forall w_i, w_j \in S^*_L \\ &d(w_i, w_j) \geq \beta \; \forall w_i \in S^*_L, w_j \in S^*_{Q\neq L} \\ &L, Q \in \mathcal{C}, K \geq 1 \end{aligned} \tag{7}$$

A.2 Further Details of Experiments

- Table A.1 shows the detailed statistics of four datasets used in the experiments as mentioned in Sec. 4.1.
- Tables A.2, A.3, A.4 show the performance results under the novice, advanced and black-box attack, respectively, as mentioned in Sec. 4.2.
- Figure A.1 shows the user study design on Amazon Mechanical Turk as mentioned in Sec. 5.
- Figures A.2 and A.3 show the performance under the adaptive attack as mentioned in Sec. 4.2.

A.3 Reproducibility

A.3.1 Source Code

We release the source code of DARCY at: https://github.com/lethaiq/ACL2021-DARCY-HoneypotDefenseNLP.

A.3.2 Computing Infrastructure

We run all experiments on the machines with Ubuntu OS (v18.04), 20-Core Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, 93GB of RAM and a Titan Xp GPU. All implementations are written in Python (v3.7) with Pytorch (v1.5.1), Numpy (v1.19.1), Scikit-learn (v0.21.3). We also use the *Transformers* (v3.0.2)[3] library for training *transformers-based* BERT.

A.3.3 Average Runtime

According to Sec. 3.1, the computational complexity of greedy trapdoor search scales linearly with the number of labels $|\mathcal{C}|$ and vocabulary size $|\mathcal{V}|$. Moreover, the time to train a detection network depends on the size of a specific dataset, the trapdoor ratio ϵ, and the number of trapdoors K.

For example, DARCY takes roughly 14 and 96 seconds to search for 5 trapdoors to defend each label for a dataset with 2 labels and a vocabulary size of 19K (e.g., Movie Reviews) and a dataset with 4 labels and a vocabulary size of 91K (e.g., AG News), respectively. With $K \leftarrow 5$ and $\epsilon \leftarrow 0.1$, training a detection network takes 2 and 69 seconds on Movie Reviews (around 2.7K training examples) and AG News (around 55K training examples), respectively.

A.3.4 Model's Architecture and # of Parameters

The *CNN* text classification model with 6M parameters (Kim, 2014) has three 2D convolutional layers (i.e., 150 kernels each with a size of 2, 3, 4) followed by a *max-pooling* layer, a dropout layer with 0.5 probability, and a fully-connected-network (FCN) with softmax activation for prediction. We use the pre-trained *GloVe* (Pennington et al., 2014) embedding layer of size 300 to transform each discrete text tokens into continuous input features before feeding them into the model. The *RNN* text model with 6.1M parameters replaces the convolution layers of CNN with a GRU network of 1 hidden layer. The *BERT* model with 109M parameters is imported from the transformers library. We use the *bert-base-uncased* version of BERT.

A.3.5 Hyper-Parameters

Sec. 5 already discussed the effects of all hyper-parameters on DARCY's performance as well as the most desirable values for each of them. To tune these hyper-parameters, we use the grid search as follows: $\epsilon \in \{1.0, 0.5, 0.25, 0.1\}$, $\gamma \in \{1.0, 0.75, 0.5\}$. Since α and β are sensitive to the domain of the pre-trained word-embedding (we use GloVe embeddings (Pennington et al., 2014)), without loss of generality, we instead use # of neighboring tokens to accept or filter to search for the corresponding α, β in Eq. (6): $\{500, 1000, 3000, 5000\}$.

We set the number of randomly sampled candidate trapdoors to around 10% of the vocabulary size ($T \leftarrow 300$). We train all models using a learning rate of 0.005 and batch size of 32. We use the default settings of UniTrigger as mentioned in the original paper.

[3] https://huggingface.co/transformers/

Dataset	Acronym	# Class	Vocabulary Size	# Words	# Data
Subjectivity	SJ	2	20K	24	10K
Movie Reviews	MR	2	19K	21	11K
Sentiment Treebank	SST	2	16K	19	101K
AG News	AG	4	71K	38	120K

Table A.1: Dataset statistics

	Method	RNN				CNN				BERT			
		Clean	Detection			Clean	Detection			Clean	Detection		
		F1	AUC	FPR	TPR	F1	AUC	FPR	TPR	F1	AUC	FPR	TPR
MR	OOD	76.5	47.3	49.0	51.0	**78.9**	82.3	23.5	78.4	84.7	38.4	61.3	50.7
	ScRNN	-	55.1	43.1	53.7	-	54.7	43.1	53.1	-	52.0	52.3	55.1
	USE	-	64.8	46.1	77.7	-	64.8	45.3	74.6	-	49.5	57.3	60.7
	SelfATK	-	96.5	0.8	93.9	-	97.0	0.1	94.1	-	93.4	4.0	87.5
	LID	-	53.2	44.1	50.6	-	66.2	42.5	74.9	-	55.4	51.5	61.9
	DARCY(1)	75.9	**99.9**	**0.2**	**100.0**	74.6	98.4	**0.5**	97.3	**85.0**	91.7	**3.9**	84.0
	DARCY(5)	**78.0**	99.1	1.0	99.5	77.3	**99.4**	1.1	**100.0**	84.2	**100.0**	4.0	**100.0**
SJ	OOD	88.5	34.3	64.9	47.1	**90.1**	82.6	23.6	79.9	95.8	20.9	76.3	42.1
	ScRNN	-	53.6	47.8	55.6	-	59.8	43.9	59.7	-	53.4	53.6	58.6
	USE	-	65.2	45.2	77.0	-	74.6	37.5	83.8	-	62.5	50.8	75.7
	SelfATK	-	98.5	1.9	98.9	-	98.5	0.1	97.1	-	98.8	6.2	97.9
	LID	-	48.9	53.0	50.8	-	71.7	29.2	72.7	-	61.9	56.0	78.4
	DARCY(1)	89.5	**99.5**	**0.3**	**99.2**	88.1	97.6	**0.8**	95.9	**96.1**	**100.0**	**6.1**	**100.0**
	DARCY(5)	**89.8**	97.4	1.2	96.0	89.6	**99.2**	1.5	**100.0**	96.0	**100.0**	6.2	**100.0**
SST	OOD	**84.4**	50.8	47.3	51.8	**81.1**	86.1	19.4	81.6	93.5	33.3	63.6	43.4
	ScRNN	-	54.4	19.1	27.8	-	55.1	19.1	29.3	-	50.2	50.6	51.2
	USE	-	58.1	51.3	68.7	-	51.0	58.5	67.8	-	55.7	51.2	62.6
	SelfATK	-	67.1	2.9	37.1	-	83.8	**0.2**	67.8	-	82.6	**1.6**	65.7
	LID	-	50.0	41.3	41.3	-	71.1	20.9	63.2	-	48.6	43.8	40.9
	DARCY(1)	83.5	96.6	6.8	99.9	77.4	98.1	0.4	96.7	**94.2**	91.6	**1.6**	83.6
	DARCY(5)	82.6	**99.6**	**0.8**	**100.0**	79.3	**98.5**	2.4	**99.3**	93.9	**100.0**	**1.6**	**100.0**
AG	OOD	**91.0**	44.4	51.5	47.7	**89.6**	67.3	34.7	61.9	93.2	27.5	69.8	41.9
	ScRNN	-	53.1	48.4	52.9	-	53.6	47.7	52.8	-	51.7	50.6	53.2
	USE	-	81.6	29.6	86.9	-	67.2	44.0	78.1	-	57.6	52.8	70.0
	SelfATK	-	92.6	**4.3**	89.5	-	93.2	3.9	90.4	-	**99.8**	**0.1**	**99.6**
	+LID	-	55.5	45.3	56.3	-	79.8	23.1	82.6	-	48.5	54.7	51.6
	DARCY(1)	89.7	**97.2**	5.4	**99.8**	88.2	**98.9**	**2.0**	99.7	**93.9**	89.3	**0.1**	78.7
	DARCY(5)	89.9	96.5	6.8	**99.8**	88.8	94.5	11.0	**100.0**	93.3	97.6	**0.1**	95.4

Table A.2: Average detection performance across all target labels under novice attack

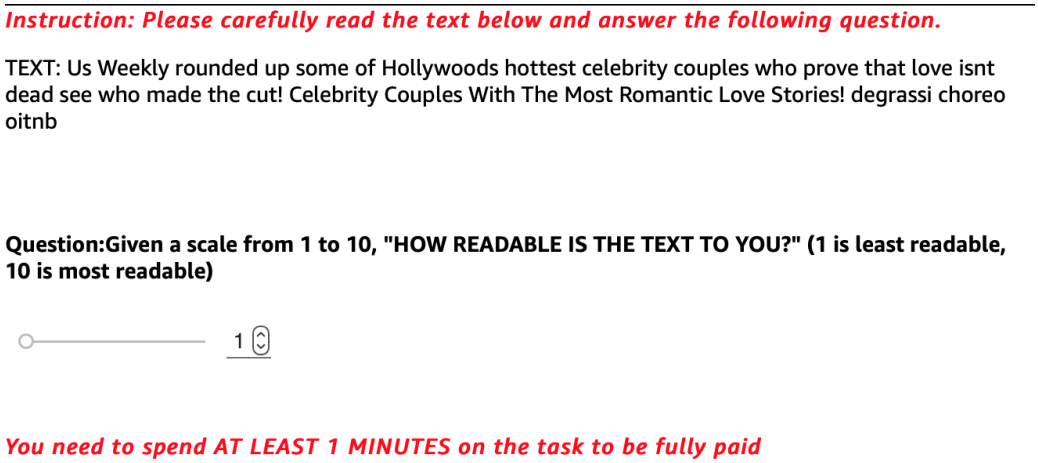

Figure A.1: Example of user study interface for Sec. 5

	Method	RNN				CNN				BERT			
		Clean	Detection			Clean	Detection			Clean	Detection		
		F1	AUC	FPR	TPR	F1	AUC	FPR	TPR	F1	AUC	FPR	TPR
MR	OOD	75.2	52.5	45.9	55.7	**77.7**	74.8	30.0	72.4	**84.7**	35.6	63.9	48.2
	ScRNN	-	51.9	43.0	47.0	-	57.3	41.6	56.4	-	51.8	52.3	54.9
	USE	-	62.9	48.1	75.9	-	66.2	44.5	77.7	-	53.1	55.1	64.1
	SelfATK	-	**92.3**	0.6	85.1	-	69.8	**0.4**	40.0	-	**97.5**	4.1	**95.2**
	LID	-	51.3	45.8	48.4	-	66.2	37.4	69.7	-	54.2	51.5	59.6
	DARCY(1)	77.8	74.8	**0.8**	50.4	76.9	73.6	**0.4**	47.	**84.7**	74.3	**3.9**	50.7
	DARCY(5)	**78.1**	**92.3**	2.9	**87.6**	77.4	**91.2**	3.2	**85.5**	84.3	92.3	4.0	85.3
SJ	OOD	**89.4**	34.5	62.5	43.1	**89.6**	59.9	44.2	64.7	96.1	21.9	74.6	43.6
	ScRNN	-	57.6	51.1	65.7	-	55.0	53.6	62.9	-	53.1	53.6	58.1
	USE	-	70.7	41.4	81.6	-	72.7	38.8	83.1	-	65.7	48.5	74.4
	SelfATK	-	80.7	8.0	69.3	-	72.8	**0.5**	46.0	-	96.8	6.2	94.0
	LID	-	50.7	54.3	55.7	-	67.5	32.0	67.1	-	62.2	56.1	79.0
	DARCY(1)	**89.4**	71.7	**0.6**	43.9	88.5	70.8	4.9	46.6	**96.2**	68.6	**6.1**	41.0
	DARCY(5)	88.9	**92.7**	2.4	**87.9**	87.6	**93.9**	4.3	**92.0**	96.1	**100.0**	6.2	**100.0**
ST	OOD	79.0	50.6	48.8	52.5	77.7	77.7	26.3	74.2	93.6	31.3	67.1	45.7
	ScRNN	-	53.8	19.2	26.8	-	56.1	19.1	31.2	-	53.2	50.3	54.9
	USE	-	60.8	50.1	72.2	-	55.2	55.4	70.4	-	51.0	57.7	63.7
	SelfATK	-	66.1	3.7	35.9	-	61.8	**0.2**	23.8	-	91.1	1.7	82.5
	LID	-	49.9	62.2	61.9	-	64.0	18.8	46.9	-	46.2	42.6	35.1
	DARCY(1)	82.9	69.7	**0.2**	39.6	77.3	59.3	0.9	19.6	**94.2**	50.0	**1.6**	1.6
	DARCY(5)	**83.3**	**93.1**	3.2	**89.4**	**78.7**	**83.0**	5.4	**71.5**	94.1	**94.6**	**1.6**	**89.4**
AG	OOD	**90.9**	40.5	56.3	46.9	**89.4**	63.1	38.2	59.0	93.1	26.9	69.2	40.7
	ScRNN	-	56.0	46.1	54.7	-	53.7	48.8	54.1	-	54.4	46.4	52.6
	USE	-	88.6	22.7	90.5	-	69.4	42.0	78.7	-	60.0	50.3	70.8
	SelfATK	-	88.4	**6.2**	83.1	-	80.7	**8.0**	69.4	-	92.0	**0.1**	84.0
	LID	-	54.3	45.9	54.6	-	79.1	22.1	80.3	-	48.3	52.9	49.4
	DARCY(1)	87.4	54.0	80.4	88.4	86.6	83.3	19.0	85.5	**93.9**	70.3	**0.1**	40.7
	DARCY(5)	89.7	**95.2**	9.3	**99.8**	88.6	**92.6**	14.7	**99.9**	93.3	**97.0**	**0.1**	**94.0**

Table A.3: Average detection performance across all target labels under advanced attack

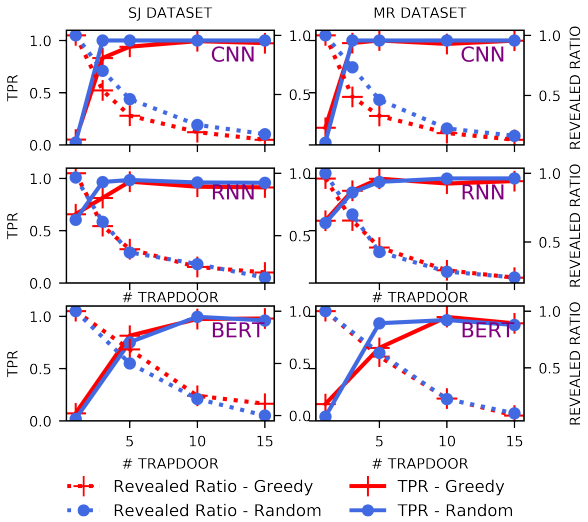

Figure A.2: Performance under adaptive attacks

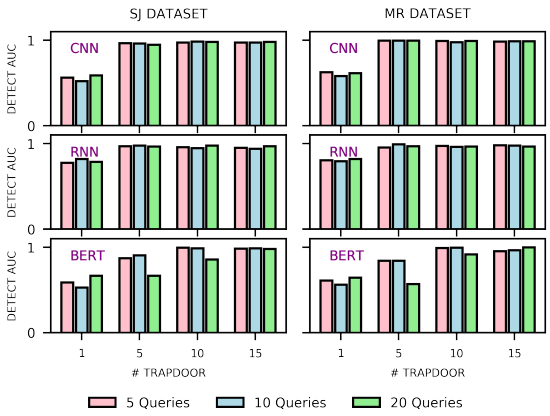

Figure A.3: Detection AUC v.s. # query attacks

	Adaptive		Random	
	Detect AUC↑	Attack ACC↓	Detect AUC↑	Attack ACC↓
MR	74.24	4.6	85.3	3.77
SJ	87.19	0.34	76.78	2.86
SST	58.81	19.77	49.75	18.96
AG	67.88	55.87	53.25	75.25

Red: not transferable

Table A.4: Detection AUC and model's accuracy (attack ACC) under black-box attack on CNN

A.3.6 Datasets

We use *Datasets* (v1.2.1)[4] library to load all the standard benchmark datasets used in the paper, all of which are publicly available.

[4] https://huggingface.co/docs/datasets/