Robustness to Programmable String Transformations via Augmented Abstract Training

Yuhao Zhang [1] **Aws Albarghouthi** [1] **Loris D'Antoni** [1]

Abstract

Deep neural networks for natural language processing tasks are vulnerable to adversarial input perturbations. In this paper, we present a versatile language for programmatically specifying string transformations—e.g., insertions, deletions, substitutions, swaps, etc.—that are relevant to the task at hand. We then present an approach to adversarially training models that are robust to such user-defined string transformations. Our approach combines the advantages of search-based techniques for adversarial training with abstraction-based techniques. Specifically, we show how to decompose a set of user-defined string transformations into two component specifications, one that benefits from search and another from abstraction. We use our technique to train models on the AG and SST2 datasets and show that the resulting models are robust to combinations of user-defined transformations mimicking spelling mistakes and other meaning-preserving transformations.

1. Introduction

Deep neural networks have proven incredibly powerful in a huge range of machine-learning tasks. However, deep neural networks are highly sensitive to small input perturbations that cause the network's accuracy to plummet (Carlini & Wagner, 2017; Szegedy et al., 2013). In the context of natural language processing, these *adversarial examples* come in the form of spelling mistakes, use of synonyms, etc.—essentially, meaning-preserving transformations that cause the network to change its prediction (Ebrahimi et al., 2018; Zhang et al., 2019a; Michel et al., 2019).

In this paper, we are interested in the problem of training models over natural language—or, generally, sequences over a finite alphabet—that are *robust* to adversarial examples. Sequences over finite alphabets are unique in that the space of adversarial examples is discrete and therefore hard to explore efficiently using gradient-based optimization as in the computer-vision setting. The common approach to achieving robustness is *adversarial training* (Goodfellow et al., 2015; Madry et al., 2018), which has seen a great deal of research in computer vision and, more recently, in natural language processing (Ebrahimi et al., 2018; Zhang et al., 2019a; Michel et al., 2019). Suppose we have defined a space of perturbations $R(x)$ of a sample x—e.g., if x is a sentence, $R(x)$ contains every possible misspelling of words in x, up to some bound on the number of misspellings. The idea of adversarial training is to model an adversary within the training objective function: Instead of computing the loss for a sample (x, y) from the dataset, we compute the loss for the worst-case perturbed sample $z \in R(x)$. Formally, the adversarial loss for (x, y) is $\max_{z \in R(x)} \mathcal{L}(z, y, \theta)$.

The question we ask in this paper is:

Can we train models that are robust against rich perturbation spaces over strings?

The practical challenge in answering this question is computing the worst-case loss. This is because the perturbation space $R(x)$ can be enormous and therefore impractical to enumerate. This is particularly true for NLP tasks, where the perturbation space $R(x)$ should contain inputs that are semantically equivalent to x—e.g., variations of the sentence x with typos or words replaced by synonyms. Therefore, we need to *approximate* the adversarial loss. There are two such classes of approximation techniques:

Augmentation The first class of techniques computes a *lower bound* on the adversarial loss by exploring a finite number of points in $R(x)$. This is usually done by applying a gradient-based attack, like HotFlip (Ebrahimi et al., 2018) for natural-language tasks or PGD (Madry et al., 2018) for computer-vision tasks. We call this class of techniques *augmentation*-based, as they essentially search for a perturbed sample with which to augment the training set.

Abstraction The second class of techniques computes an

[1]Department of Computer Science, University of Wisconsin-Madison, Madison, USA. Correspondence to: Yuhao Zhang <yuhaoz@cs.wisc.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

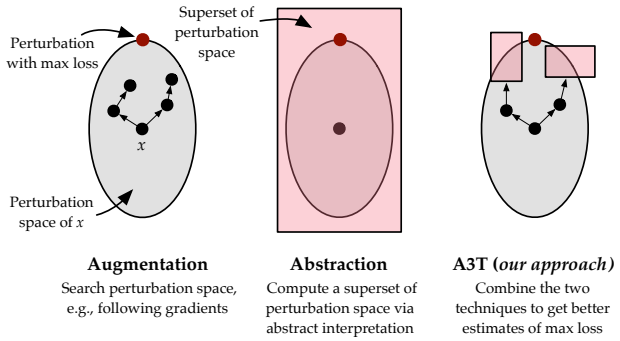


Figure 1. Illustration of augmentation, abstraction, and A3T

upper bound on the adversarial loss by overapproximating, or *abstracting*, the perturbation space $R(x)$ into a set of symbolic constraints that can be efficiently propagated through the network. For example, the *interval* abstraction has been used in numerous works (Mirman et al., 2018; Gowal et al., 2019; Huang et al., 2019). We call this class of techniques *abstraction*-based.

Both classes of techniques can produce suboptimal results: augmentation can severely underapproximate the worst-case loss and abstraction can severely overapproximate the loss. Particularly, we observe that the two techniques have complementary utility, working well on some perturbation spaces but not others—for example, Huang et al. (2019) have shown that abstraction works better for token substitutions, while augmentation-based techniques like HotFlip (Ebrahimi et al., 2018) and MHA (Zhang et al., 2019a) are general—e.g., apply to token deletions and insertions.

1.1. Our Approach

A hybrid approach We propose *augmented abstract adversarial training* (A3T), an adversarial training technique that combines the strengths of augmentation and abstraction techniques. The key idea underlying A3T is to decompose the perturbation space into two subsets, one that can be explored using augmentation and one that can be abstracted—e.g., using augmentation to explore word duplication typos and abstraction to explore replacing words with synonyms. From an algorithmic perspective, our computation of adversarial loss switches from a concrete, e.g., gradient-based, search through the perturbation space to a symbolic search. As such, for every training sample (x, y), our technique may end up with a lower bound *or* an upper bound on its adversarial loss (see Fig. 1).

A language for specifying string transformations The challenge of applying A3T is how to exactly decompose the perturbation space. Our key enabling idea is to define the perturbation space *programmatically*, in a way that can be easily and cleanly decomposed. Specifically, we define a general language in which we can specify a perturbation space by a specification S in the form of $\{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}$, containing a set of string transformations $T_i : \mathcal{X} \to 2^{\mathcal{X}}$. The specification S defines a perturbation space of all possible strings by applying each transformation T_i up to δ_i times. For example, given a string x, $S(x)$ could define the set of all strings x' that are like x but with some words replaced by one of its synonyms and with some stop words removed.

Given a perturbation space defined by a set of transformations, A3T decomposes the set of transformations into two disjoint subsets, one that is explored concretely (augmentation) and one that is explored symbolically (abstraction).

Results We have implemented A3T and used it to train NLP models for sentiment analysis that are robust to a range of string transformations—e.g., character swaps modeling spelling mistakes, substituting of a word with a synonym, removing stop words, duplicating words, etc. Our results show that A3T can train models that are more robust to adversarial string transformations than those produced using existing techniques.

1.2. Summary of Contributions

- We present A3T, a technique for training models that are robust to string transformations. A3T combines search-based attacks and abstraction-based techniques to explore the perturbation space and compute good approximations of adversarial loss.
- To enable A3T, we define a general language of string transformations with which we can specify the perturbation space. A3T exploits the specification to decompose and search the perturbation space.
- We implement A3T[1] and evaluate it on two datasets and a variety of string transformations. Our results demonstrate the increase in robustness achieved by A3T in comparison with state-of-the-art techniques.

2. Related Work

Adversarial text generation Zhang et al. (2019b) presented a comprehensive overview of adversarial attacks on neural networks over natural language. In this paper, we focus on the word- and character-level. HotFlip (Ebrahimi et al., 2018) is a gradient-based approach that can generate the adversarial text in the perturbation space described by word- and character-level transformations. MHA (Zhang et al., 2019a) uses Metropolis-Hastings sampling guided by

[1]We will provide our code in the supplementary materials. One can also access the open source project using the anonymous link: https://anonymous.4open.science/r/090c73a5-0825-4668-ae8d-96f8421ad0ec/.

gradients to generate word-level adversarial text via word substitution. Karpukhin et al. (2019) designed character-level noise for training robust machine translation models. Other work focuses on generating adversarial text on the sentence-level (Liang et al., 2018) or paraphrase-level (Iyyer et al., 2018; Ribeiro et al., 2018). Also, some works (Zhao et al., 2018; Wang et al., 2019) try to generate natural or grammatically correct adversarial text.

Abstract training Mirman et al. (2018) and Gowal et al. (2019) first proposed *DiffAI* and interval bound propagation (IBP) to train image classification models that are provably robust to norm-bounded adversarial perturbations. They performed abstract training by optimizing the abstract loss obtained by Interval or Zonotope propagation. Mirman et al. (2019) used DiffAI to provably defend deep residual networks. Jia et al. (2019) used interval domain to capture the perturbation space of substitution and train robust models for CNN and LSTM. Huang et al. (2019) proposed a simplex space to capture the perturbation space of substitution. They converted the simplex into intervals after the first layer of the neural network and obtained the abstract loss by IBP. We adopt their abstract training approach for some of our transformations. We will show the limitation of abstract training for more complex perturbations like the combination of swap and substitution.

Other robustness techniques Other techniques to ensure robustness involve placing a spelling-mistake-detection model that identifies possible adversaries before the underlying model (Pruthi et al., 2019; Sakaguchi et al., 2017).

Formal verification for neural networks In the field of verification for NLP tasks, Shi et al. (2020) combined forward propagation and a tighter backward bounding process to achieve the formal verification of Transformers. Welbl et al. (2020) proposed the formal verification under text deletion for models based on the popular decomposable attention mechanism by interval bound propagation. *POPQORN* (Ko et al., 2019) is a general algorithm to quantify the robustness of recurrent neural networks, including RNNs, LSTMs, and GRUs. COLT (Balunovic & Vechev, 2020) combines formal verification and adversarial training to train neural networks. In this paper, we mix verification techniques, namely, interval propagation, with search-based techniques.

3. The Perturbation-Robustness Problem

In this section, we (1) formalize the perturbation-robustness problem and (2) define a string transformation language for specifying the perturbation space.

3.1. Perturbation Robustness

Classification setting We consider a standard classification setting with samples from some domain $\mathcal{X}$ and labels from $\mathcal{Y}$. Given a distribution $\mathcal{D}$ over samples and labels, our goal is to find the optimal parameters θ of some neural-network architecture F_θ that minimize the expected loss

$$\underset{\theta}{\operatorname{argmin}} \underset{(\mathbf{x},y)\sim\mathcal{D}}{\mathbb{E}} \mathcal{L}(\mathbf{x}, y, \theta) \tag{1}$$

We are interested in the setting where the sample space $\mathcal{X}$ defines strings over some finite *alphabet* Σ. The alphabet Σ can be, for example, English characters (in a character-level model) or entire words (in a world-level model). Therefore, the domain $\mathcal{X}$ in our setting is Σ^*, i.e., the set of all strings of elements of Σ. We will use $\mathbf{x} \in \Sigma^*$ to denote a string and $x_i \in \Sigma$ to denote the ith element of the string.

Perturbation space We define a *perturbation space* R as a function in $\Sigma^* \to 2^{\Sigma^*}$, i.e., R takes a string $\mathbf{x}$ and returns a finite set of possible *perturbed* strings of $\mathbf{x}$.

We will use a perturbation space to denote a set of strings that should receive the same classification by our network. For example, $R(\mathbf{x})$ could define a set of sentences paraphrasing $\mathbf{x}$. We can thus modify our training objective into a *robust-optimization* problem, following Madry et al. (2018):

$$\underset{\theta}{\operatorname{argmin}} \underset{(\mathbf{x},y)\sim\mathcal{D}}{\mathbb{E}} \max_{\mathbf{z}\in R(\mathbf{x})} \mathcal{L}(\mathbf{z}, y, \theta) \tag{2}$$

This inner objective is usually hard to solve; in our setting, the perturbation space can be very large and we cannot afford to consider every single point in that space during training. Therefore, as we discussed in Section 1, typically approximations are made.

Exhaustive accuracy Once we have trained a model F_θ using the robust optimization objective, we will use *exhaustive accuracy* to quantify its classification accuracy in the face of perturbations. Specifically, given a dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and a perturbation space R, we define exhaustive accuracy as follows:

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}[\forall\, \mathbf{z} \in R(\mathbf{x}_i).\ F_\theta(\mathbf{z}) = y_i] \tag{3}$$

Intuitively, for each sample $(\mathbf{x}_i, y_i)$, its classification is considered correct iff F_θ predicts y_i for every single point in $R(\mathbf{x}_i)$. We use exhaustive accuracy instead of the commonly used adversarial accuracy because (1) exhaustive accuracy provides the ground truth accuracy of the discrete perturbation spaces and does not depend on an underlying adversarial attack, and (2) the discrete spaces make it easy for us to compute exhaustive accuracy by enumeration and at the same time hard for the gradient-based adversarial attacks to explore the space.

3.2. A Language for Specifying Perturbations

We have thus far assumed that the perturbation space is provided to us. We now describe a language for modularly

specifying a perturbation space.

A specification S is defined as follows:

$$S = \{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}$$

where each T_i denotes a string transformation that can be applied up to $\delta_i \in \mathbb{N}$ times. Formally, a string transformation T is a pair (φ, f), where $\varphi : \Sigma^* \to \{0, 1\}$ is a *Boolean predicate* (in practice, a regular expression) describing the substrings of the inputs to which the transformation can be applied, and $f : \Sigma^* \to 2^{\Sigma^*}$ is a *transformer* describing how the substrings matched by φ can be replaced.

Single transformations Before defining the semantics of our specification language, we illustrate a few example specifications involving single transformations:

Example 1 ($T_{stop} = (\varphi_{stop}, f_{stop})$)**.** Suppose we want to define a transformation that deletes a stop word—*and*, *the*, *is*, etc.—mimicking a typo. The predicate φ_{stop} will be a regular expression matching all stop words. The transformer f_{stop} will be simply the function that takes a string and returns the set containing the empty string, $f_{stop}(\mathbf{x}) = \{\epsilon\}$. Consider a specification $S_{stop} = \{(T_{stop}, 1)\}$ that applies the transformation T_{stop} up to one time. On the following string, *They are at school*, the predicate φ_{stop} matches the substrings *are* and *at*. In both cases, we apply the predicate f_{stop} to the matched word and insert the output of f_{stop} in its position. This results in the set containing the original string (0 transformations are applied) and the two strings *They at school* and *They are school*. Applying a specification $S^2_{stop} = \{(T_{stop}, 2)\}$, which is allowed to apply T_{stop} at most twice, to the same input would result in a set of strings containing the strings above as well as the string *They school*.

Example 2 ($T_{nice} = (\varphi_{nice}, f_{nice})$)**.** Say we want to transform occurrences of *nice* into one of its synonyms, *enjoyable* and *pleasant*. We define the predicate $\varphi_{nice}(\mathbf{x})$ that is true iff $\mathbf{x} =$ *nice*, and we define $f_{nice}(\mathbf{x}) = \{$*enjoyable*, *pleasant*$\}$. Given the string *This is nice!*, it will be transformed into the set $\{$*This is enjoyable!*, *This is pleasant!*$\}$.

Applying a specification $S^2_{nice} = \{(T_{nice}, 2)\}$ to the same input would result in the same set of strings above. Because the predicate $\varphi_{nice}(\mathbf{x})$ only matches the word *nice*.

Example 3 ($T_{swap} = (\varphi_{swap}, f_{swap})$)**.** Now consider the case where we would like to swap adjacent vowels. φ_{swap} will be defined as the regular expression that matches two adjacent characters that are vowels. Next, since $\mathbf{x} = x_0x_1$ can only have length 2, the transformer f_{swap} will be the swap function $f_{swap}(x_0x_1) = \{x_1x_0\}$.

Multiple transformations As discussed above, a specification S in our language is a set of transformations $\{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}$ where each T_i is a pair (φ_i, f_i). The formal semantics of our language can be found in the supplementary Appendix. Informally, a string $\mathbf{z}$ is in the perturbation space $S(\mathbf{x})$ if it can be obtained by (1) finding a set σ of *non-overlapping* substrings of $\mathbf{x}$ that match the various predicates φ_i and such that at most δ_i substrings in σ are matches of φ_i, and (2) replacing each substring $\mathbf{x}' \in \sigma$ matched by φ_i with a string in $f_i(\mathbf{x}')$. The complexity of the formalization is due to the requirement that matched substrings should not overlap—this requirement guarantees that each character in the input is only involved in a single transformation and will be useful when formalizing our abstract training approach in Section 4.2.

Example 4 (Multiple Transformations)**.** Using the transformations T_{nice} and T_{swap} we can define the specification

$$S_{ns} = \{(T_{nice}, 1), (T_{swap}, 1)\}$$

Then, S_{ns}(*This house is nice*) results in the set of strings:

This house is nice	*This house is* ***enjoyable***
This house is ***pleasant***	*This h****uo****se is nice*
*This h****uo****se is* ***enjoyable***	*This h****uo****se is* ***pleasant***

The transformed portions are shown in bold. Note that we apply *up to* 1 of each transformation, thus we also get the original string. Also, note that the two transformations cannot modify overlapping substrings of the input; for example, T_{swap} did not swap the *ea* in *pleasant*.

4. Augmented Abstract Adversarial Training

In this section, we describe our abstract training technique, A3T, which combines augmentation and abstraction.

Recall the adversarial training objective function, Eq. (2). The difficulty in solving this objective is the inner maximization objective: $\max_{\mathbf{z} \in R(\mathbf{x})} \mathcal{L}(\mathbf{z}, y, \theta)$, where the perturbation space $R(\mathbf{x})$ can be intractably large to efficiently enumerate, and we therefore have to resort to approximation. We begin by describing two approximation techniques and then discuss how our approach combines and extends them.

Augmentation (search-based) techniques We call the first class of techniques *augmentation* techniques, since they search for a worst-case sample in the perturbation space $R(\mathbf{x})$ with which to augment the dataset. The naïve way is to simply enumerate all points in $R(\mathbf{x})$—our specifications induce a finite perturbation space, by construction. Unfortunately, this can drastically slow down the training. For example, suppose T defines a transformation that swaps two adjacent characters. On a string of length N, the specification $(T, 2)$ results in $O(N^2)$ transformations.

An efficient alternative, HotFlip, was proposed by Ebrahimi et al. (2018). HotFlip efficiently encodes a transformation T as an operation over the embedding vector and *approximates* the worst-case loss using a single forward and backward pass through the network. To search through a set of trans-

Algorithm 1 A3T

Input: $S = \{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}$ and point $(\mathbf{x}, y)$
Output: worst-case loss
Split S into S_{aug} and S_{abs} and return

$$\max_{\mathbf{z} \in augment_k(S_{aug}, \mathbf{x})} \mathcal{L}(\widehat{\mathbf{z}}, y, \theta) \quad s.t.\ \widehat{\mathbf{z}} = abstract(S_{abs}, \mathbf{z})$$

formations, HotFlip employs a beam search of some size k to get the top-k perturbed samples. This technique yields a point in $R(\mathbf{x})$ that may not have the worst-case loss. Alternatives like MHA (Zhang et al., 2019a) can also be used as augmentation techniques.

Abstraction techniques Abstraction techniques compute an over-approximation of the perturbation space, as a symbolic set of constraints. This set of constraints is then propagated through the network, resulting in an upper bound on the worst-case loss. Specifically, given a transformation T, we define a corresponding abstract transformation $\widehat{T}$ such that for all $\mathbf{x}$, the constraint $T(\mathbf{x}) \subseteq \widehat{T}(\mathbf{x})$ holds.

Our use of abstraction builds upon the work of Huang et al. (2019), which uses an *interval domain* to define $\widehat{T}(\mathbf{x})$ — i.e., $\widehat{T}(\mathbf{x})$ is a conjunction of constraints on each character. We will describe how we generalize their approach in Section 4.2; for now, we assume that we can efficiently over-approximate the worst-case loss for $\widehat{T}(\mathbf{x})$ by propagating it through the network.

4.1. A3T: A High-Level View

The key idea of A3T is to decompose a specification S into two sets of transformations, one containing transformations that can be effectively explored with augmentation and one containing transformations that can be precisely abstracted.

Algorithm 1 shows how A3T works. First, we decompose the specification S into two subsets of transformations, resulting in two specifications, S_{aug} and S_{abs}. For S_{aug}, we apply an augmentation technique, e.g., HotFlip or MHA, to come up with a list of top-k perturbed samples in the set $S_{aug}(\mathbf{x})$—this is denoted as the set $augment_k(S_{aug}, \mathbf{x})$.

Then, for each point $\mathbf{z}$ in the top-k results, we compute an abstraction $abstract(S_{abs}, \mathbf{z})$, which is a set of constraints over-approximating the set of points in $S_{abs}(\mathbf{z})$. Recall our overview in Fig. 1 for a visual depiction of this process.

Finally, we return the worst-case loss.

4.2. Computing Abstractions

We now show how to define the abstraction of a perturbation space $S(\mathbf{x})$ defined by a specification $S = \{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}$. Our approach generalizes that of Huang et al. (2019) to length-preserving transformations, i.e., ones where the length of every string in $S(\mathbf{x})$ is the same as the length of the original string $\mathbf{x}$. The approach of Huang et al. (2019) targeted the special case of single-character substitutions.

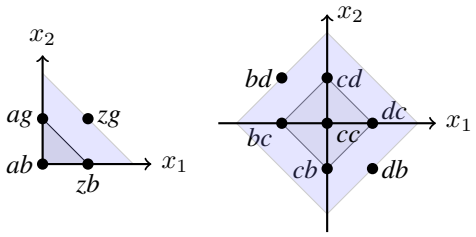


Figure 2. Illustration of an abstraction of a single (left) and multiple (right) transformations. See Example 5 and Example 6 for details.

Single transformation case We first demonstrate the case of a single length-preserving transformation, $S = \{(T, \delta)\}$. Henceforth we assume that each element of a string $\mathbf{x}$ is a real value, e.g., the embedding of a character or word. At a high level, our abstraction computes the convex hull that contains all the points in $T(\mathbf{x})$ (we use $T(\mathbf{x})$ as a short hand for the perturbation space obtained by applying T to $\mathbf{x}$ exactly once) and then scales this convex hull by δ to account for the cases in which T is applied up to δ times. We begin by computing all points in $T(\mathbf{x})$. Let this set be $\mathbf{x}_0, \ldots, \mathbf{x}_m$. Next, for $i \in [1, m]$, we define the set of points

$$\mathbf{v}_i = \mathbf{x} + \delta \cdot (\mathbf{x}_i - \mathbf{x}).$$

We then construct the abstraction $abstract(S_{abs}, \mathbf{z})$ as the convex hull of the points $\mathbf{v}_i$ and $\mathbf{x}$. Observe that we only need to enumerate the space $T(\mathbf{x})$ obtained by applying T once; multiplying by δ *dilates* the convex hull to include all strings that involve up to δ applications of T, i.e., $S(\mathbf{x})$. To propagate this convex hull through the network, we typically overapproximate it as a set of *interval* constraints, where each dimension of a string is represented by a lower and an upper bound. Interval constraints are easier to propagate through the network—requiring several forward passes linear in the length of the string—compared to arbitrary convex polyhedra, whose operations can be exponential in the number of dimensions (Cousot & Halbwachs, 1978).

Example 5. Consider the left side of Fig. 2. Say we have the string *ab* and the transformation T that can replace character a with z, or b with g—mimicking spelling mistakes, as these characters are adjacent on a QWERTY keyboard. The large shaded region is the result of dilating T with $\delta = 2$, i.e., contains all strings that can be produced by applying T twice to the string *ab*, namely, the string *zg*.

General case We generalize the above abstraction process to the perturbation space $S = \{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}$.

First, we enumerate all the strings in $T_1(\mathbf{x}) \cup \cdots \cup T_n(\mathbf{x})$. (Notice that we need only consider each transformation T_i independently.) Let this set be $\mathbf{x}_0, \ldots, \mathbf{x}_m$. Next, for $i \in [1, m]$, we define the following set of points:

$$\mathbf{v}_i = \mathbf{x} + (\delta_1 + \ldots + \delta_n) \cdot (\mathbf{x}_i - \mathbf{x})$$

As with the single-transformation case, we can now construct an abstraction of the convex hull induced by $\mathbf{v}_i$ and $\mathbf{x}$ as a set of intervals and propagate it through the network.

Example 6. We illustrate the process of abstracting an input string *cc* on the right of Figure 2 for the specification $\{(T_{prev}, 1), (T_{succ}, 1)\}$, where T_{prev} maps one character to its preceding character in the alphabet order, e.g., c with b, and T_{succ} maps one character to its succeeding character in the alphabet order, e.g., c with d. We enumerate all the points in $T_{prev}(cc) \cup T_{succ}(cc)$ and compute their convex hull, shown as the inner shaded region. This region includes all strings resulting from exclusively one application of T_{prev} or T_{succ}. Next, we dilate the convex hull by $2 = 1 + 1$ times to include all the points in the perturbation space of $\{(T_{prev}, 1), (T_{succ}, 1)\}$. This is shown as the larger shaded region. Notice how this region includes *bd* and *db*, which result from an application of T_{succ} to the first character and T_{prev} to the second character of the original string *cc*.

The following theorem states that this process is sound: produces an overapproximation of a perturbation space $S(\mathbf{x})$.

Theorem 1. *For every specification S and input $\mathbf{x}$, the abstracted perturbation space $abstract(S, \mathbf{x})$ is an overapproximation of $S(\mathbf{x})$—i.e., $S(\mathbf{x}) \subseteq abstract(S, \mathbf{x})$*

We prove Theorem 1 in the Appendix.

5. Experiments

In this section, we evaluate A3T by answering the following research questions:

- **RQ1:** Does A3T improve robustness in rich perturbation spaces for character-level and word-level models?
- **RQ2:** How does the complexity of the perturbation space affect the effectiveness of A3T?

5.1. Experimental Setup

5.1.1. Datasets and Models

We use two datasets:

- **AG** News (Zhang et al., 2015) dataset consists of a corpus of news articles collected by Gulli (2005) about the 4 largest news topics. We used the online available dataset from Github[2]. The dataset contains 30,000 training and 1,900 testing examples for each class. We split the first 4,000 training examples for validation purpose.
- **SST2** (Socher et al., 2013) is the Stanford Sentiment Treebank dataset that consists of sentences from movie reviews and human annotations of their sentiment. The task is to predict the sentiment (positive/negative) of a given sentence. We used the dataset provided by TensorFlow[3]. The dataset contains 67,349 training, 872 validation, and 1,821 testing examples for each class.

For the AG dataset, we trained a smaller character-level model than the one used in Huang et al. (2019), but kept the number of layers and the data preprocessing the same. For the SST2 dataset, we trained a word-level model and a character-level model. We used the same models in Huang et al. (2019) also following their setup. The details of setups are shown in the Appendix.

5.1.2. Perturbations

Our choice of models allows us to experiment on both character-level and word-level perturbations. We evaluated A3T on six perturbation spaces constructed using the seven individual string transformations in Table 1.

For the character-level model on dataset AG, we used the following specifications: $\{(T_{SwapPair}, 2), (T_{SubAdj}, 2)\}$, $\{(T_{Del}, 2), (T_{SubAdj}, 2)\}$, and $\{(T_{InsAdj}, 2), (T_{SubAdj}, 2)\}$. For example, the first specification mimics the combination of two spelling mistakes: swap two characters up to twice and/or substitute a character with an adjacent one on the keyboard up to twice.

For the word-level model on dataset SST2, we used the following specifications: $\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$, $\{(T_{DelStop}, 2), (T_{Dup}, 2)\}$, and $\{(T_{DelStop}, 2), (T_{Dup}, 2), (T_{SubSyn}, 2)\}$. The first specification, for example, removes stop words up to twice and substitutes up to twice words with synonyms.

For the character-level model on dataset SST2, we used the following specifications: $\{(T_{SwapPair}, 1), (T_{SubAdj}, 1)\}$, $\{(T_{Del}, 1), (T_{SubAdj}, 1)\}$, and $\{(T_{InsAdj}, 1), (T_{SubAdj}, 1)\}$. For example, the first specification mimics the combination of two spelling mistakes: swap two characters and/or substitute a character with an adjacent one on the keyboard.

For the character-level model on AG dataset, we con-

[2]This is the website describing the dataset: https://github.com/mhjabreel/CharCnn_Keras/tree/master/data/ag_news_csv.

[3]This is the website describing the dataset: https://www.tensorflow.org/datasets/catalog/glue.

Table 1. String transformations to construct the perturbation spaces for evaluation.

	Transformation	Description	Training
CHAR	$T_{SwapPair}$	**swap a pair** of two adjacent characters	Augmentation
	T_{Del}	**delete** a character	Augmentation
	T_{InsAdj}	**insert** to the right of a character one of its **adjacent** characters on the keyboard	Augmentation
	T_{SubAdj}	**substitute** a character with an **adjacent** character on the keyboard	Abstraction
WORD	$T_{DelStop}$	**delete** a **stop word**	Augmentation
	T_{Dup}	**duplicate** a word	Augmentation
	T_{SubSyn}	**substitute** a word with one of its synonyms	Abstraction

Table 2. Qualitative examples. The vanilla models correctly classify the original samples but fail to classify the perturbed samples.

Prediction	A character-level sample and a perturbed sample in $\{(T_{SwapPair}, 2), (T_{SubAdj}, 2)\}$ of AG dataset
Sci/Tech	ky. company wins grant to study peptides (ap) ap - a company founded by a chemistry researcher ...
World	yk. compnay wins granf to st8dy peptides (ap) ap - a company founded by a chemistry researcher ...

Prediction	A word-level sample and a perturbed sample in $\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$ of SST2 dataset
Positive	a dream cast of solid female talent who build a seamless ensemble .
Negative	~~a~~ dreaming casting of solid female talent who build ~~a~~ seamless ensemble .

sidered the perturbations to be applied to a prefix of an input string, namely, a prefix length of 35 for $\{(T_{SwapPair}, 2), (T_{SubAdj}, 2)\}$, a prefix length of 30 for $\{(T_{Del}, 2), (T_{SubAdj}, 2)\}$, and $\{(T_{InsAdj}, 2), (T_{SubAdj}, 2)\}$. For the character-level model on SST2 dataset, we considered perturbations with $\delta = 1$ but allow the perturbations to be applied to the whole input string. We made these restrictions because one cannot efficiently evaluate the exhaustive accuracy with larger δ, due to the combinatorial explosion of the size of the perturbation space.

5.1.3. TRAINING METHODS

We implement and compare the following training methods.

Normal training is the vanilla training method (Eq. (1)) that minimizes the cross entropy between predictions and target labels. This method does not use the perturbation space and does not attempt to train a robust model.

Random augmentation performs adversarial training (Eq. (2)) using a weak adversary that simply picks a random perturbed sample from the perturbation space.

HotFlip augmentation performs adversarial training (Eq. (2)) using the HotFlip (Ebrahimi et al., 2018) attack to solve the inner maximization problem.

A3T is our technique that can be implemented in various ways. For our experiments, we made the following choices. First, we manually labeled which transformations in S are explored using augmentation and which ones are explored using abstract interpretation (the third column in Table 1).[4] Second, we implemented two different ways of performing data augmentation for the transformations in S_{aug}: (1) **A3T(HotFlip)** uses HotFlip to find the worst-case samples for augmentation, while (2) **A3T(search)** performs an explicit search through the perturbation space to find the worst-case samples for augmentation. Finally, we used DiffAI (Mirman et al., 2018) to perform abstract training for the transformations in S_{abs}, using the intervals abstraction.

In all augmentation training baselines, and A3T, we also adopt a curriculum-based training method (Huang et al., 2019; Gowal et al., 2019) which uses a hyperparameter λ to weigh between normal loss and maximization objective in Eq. (2).

5.1.4. EVALUATION METRICS

Normal accuracy is the vanilla accuracy of the model on the test set.

HotFlip accuracy is the adversarial accuracy of the model with respect to the HotFlip attack, i.e., for each point in the test set, we apply the HotFlip attack and test if the classification is still correct.

Exhaustive accuracy (Eq 3) is the worst-case accuracy of the model: a prediction on $(\mathbf{x}, y)$ is considered correct if and

[4] We consider this choice of split to be a hyperparameter.

Table 3. Experiment results for the three perturbations on the character-level model on AG dataset. We show the normal accuracy (Acc.), HotFlip accuracy (HF Acc.), and exhaustive accuracy (Exhaustive) of five different training methods.

	$\{(T_{SwapPair}, 2), (T_{SubAdj}, 2)\}$			$\{(T_{Del}, 2), (T_{SubAdj}, 2)\}$			$\{(T_{InsAdj}, 2), (T_{SubAdj}, 2)\}$		
Training	Acc.	HF Acc.	Exhaustive	Acc.	HF Acc.	Exhaustive	Acc.	HF Acc.	Exhaustive
Normal	87.5	71.5	60.1	87.5	79.0	62.5	87.5	79.1	59.0
Random Aug.	87.5	75.7	68.2 [+8.1]	87.4	81.3	69.4 [+6.9]	87.8	81.2	69.7 [+10.7]
HotFlip Aug.	86.6	85.7	84.9 [+24.8]	85.8	84.9	82.7 [+20.2]	86.8	85.9	82.6 [+23.6]
A3T(HotFlip)	86.4	86.4	86.4 [+26.3]	87.2	87.1	85.7 [+23.2]	87.4	87.4	85.5 [+26.5]
A3T(search)	86.9	86.8	**86.8 [+26.7]**	87.6	87.4	**86.2 [+23.7]**	87.9	87.8	**86.5 [+27.5]**

Table 4. Experiment results for the three perturbations on the word-level model on SST dataset.

	$\{T_{DelStop}, 2), (T_{SubSyn}, 2)\}$			$\{(T_{DelStop}, 2), (T_{Dup}, 2)\}$			$\{(T_{DelStop}, 2), (T_{Dup}, 2), (T_{SubSyn}, 2)\}$		
Training	Acc.	HF Acc.	Exhaustive	Acc.	HF Acc.	Exhaustive	Acc.	HF Acc.	Exhaustive
Normal	82.4	68.9	64.4	82.4	55.8	47.9	82.4	54.8	42.4
Random Aug.	80.0	70.0	66.0 [+1.6]	81.5	54.2	49.7 [+1.8]	81.0	56.1	46.2 [+3.8]
HotFlip Aug.	80.8	74.4	68.3 [+3.9]	80.8	68.7	56.0 [+8.1]	81.2	69.0	51.0 [+8.6]
A3T(HotFlip)	80.2	73.5	70.2 [+5.8]	79.9	69.7	57.7 [+9.8]	78.8	68.1	55.1 [+12.7]
A3T(search)	79.9	74.4	**71.2 [+6.8]**	79.0	70.7	**62.7 [+14.8]**	77.7	69.8	**59.8 [+17.4]**

only if all points $\mathbf{z} \in S(\mathbf{x})$ lead to the correct prediction.

By definition, HotFlip accuracy is an upper bound on exhaustive accuracy.

5.2. Evaluation Results

RQ1: Increase in robustness We show the results for the selected perturbation spaces on character-level and word-level models in Tables 3, 4, and 5, respectively.

Compared to normal training, the results show that both A3T(HotFlip) and A3T(search) increase the exhaustive accuracy and can improve the robustness of the model. A3T(HotFlip) and A3T(search) also outperform random augmentation and HotFlip augmentation. In particular, A3T(search) has exhaustive accuracy that is on average 20.3 higher than normal training, 14.6 higher than random augmentation, and 6.7 higher than HotFlip augmentation.

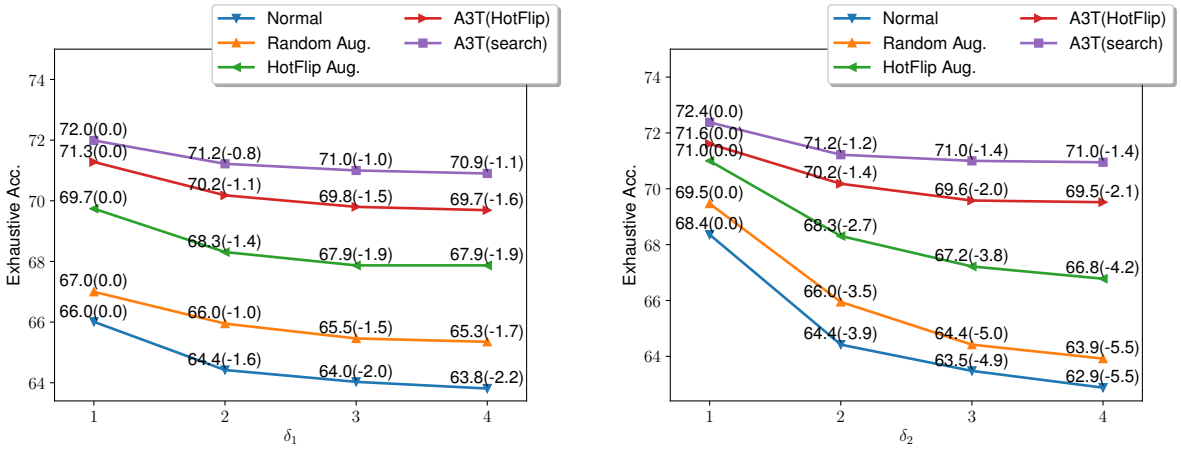
We also compared A3T to training using only abstraction (i.e., all transformations in S are also in S_{abs}) for the specification $\{(T_{SwapPair}, 2), (T_{SubAdj}, 2)\}$ on AG dataset and $\{(T_{SwapPair}, 1), (T_{SubAdj}, 1)\}$ on SST2 dataset (not shown in Tables 3, 4, and 5); this is the only specification that can be fully trained abstractly since it only uses length-preserving transformations. Training using only abstraction yields an exhaustive accuracy of 86.9 for $\{(T_{SwapPair}, 2), (T_{SubAdj}, 2)\}$ on AG dataset, which is similar to the exhaustive accuracy of A3T(HotFlip) (86.4) and A3T(search) (86.8). However, training using only abstraction yields an exhaustive accuracy of 47.0 for $\{(T_{SwapPair}, 1), (T_{SubAdj}, 1)\}$ on SST2 dataset, which is better than the one obtained using normal training, but much lower than the exhaustive accuracy of A3T(HotFlip) and A3T(search). Furthermore, the normal accuracy of the abstraction technique on SST2 dataset drops to 58.8 due to the over-approximation of the perturbation space, while A3T(HotFlip) (73.6) and A3T(search) (70.2) retain high normal accuracy.

To answer **RQ1**, **A3T yields models that are more robust to complex perturbation spaces than those produced by augmentation and abstraction techniques.** This result holds for both character-level and word-level models.

RQ2: Effects of size of the perturbation space In this section, we evaluate whether A3T can produce models that are robust to complex perturbation spaces. We fix the word-level model A3T (search) trained on $\{(T_{DelStop}, 2), (T_{SubSyn}, 2)\}$. Then, we test this model's exhaustive accuracy on $\{(T_{DelStop}, \delta_1), (T_{SubSyn}, 2)\}$ (Figure 3(a)) and $\{(T_{DelStop}, 2), (T_{SubSyn}, \delta_2)\}$ (Figure 3(b)), where we vary the parameters δ_1 and δ_2 between 1 and 4, increasing the size of the perturbation space. (The Appendix contains a more detailed evaluation with different types of transformations.) We only consider word-level models because computing the exhaustive accuracy requires us to enumerate all the elements in the perturbation space. While enumeration is feasible for word-level transformations (e.g., the perturbation space of $\{(T_{DelStop}, 4), (T_{SubSyn}, 2)\}$ for a string with 56 tokens contains at most 68,002 per-

Table 5. Experiment results for the three perturbations on the character-level model on SST2 dataset.

	$\{(T_{SwapPair}, 1), (T_{SubAdj}, 1)\}$			$\{(T_{Del}, 1), (T_{SubAdj}, 1)\}$			$\{(T_{InsAdj}, 1), (T_{SubAdj}, 1)\}$		
Training	Acc.	HF Acc.	Exhaustive	Acc.	HF Acc.	Exhaustive	Acc.	HF Acc.	Exhaustive
Normal	77.0	36.5	23.0	77.0	50.7	25.8	77.0	51.0	24.4
Random Aug.	75.6	47.1	28.2 [+5.2]	75.7	56.4	29.3 [+3.5]	74.5	57.0	33.8 [+9.4]
HotFlip Aug.	71.4	63.9	34.8 [+11.8]	76.6	67.1	38.0 [+12.2]	76.1	70.4	33.4 [+9.0]
A3T(HotFlip)	73.6	54.8	35.2 [+12.2]	75.3	58.2	32.9 [+7.1]	72.4	66.3	44.7 [+20.3]
A3T(search)	70.2	57.1	**48.7 [+15.7]**	72.5	62.5	**44.8 [+19.0]**	71.6	65.0	**55.2 [+30.8]**



(a) $\{(T_{DelStop}, \delta_1), (T_{SubSyn}, 2)\}$ (b) $\{(T_{DelStop}, 2), (T_{SubSyn}, \delta_2)\}$

Figure 3. The exhaustive accuracy of $\{(T_{DelStop}, \delta_1), (T_{SubSyn}, \delta_2)\}$, varying the parameters δ_1 (left) and δ_2 (right) between 1 and 4.

turbed samples), enumeration is infeasible for character level transformations (e.g., the perturbation space of $\{(T_{Del}, 4), (T_{SubAdj}, 2)\}$ for a string with 300 characters contains 7,499,469 perturbed samples, and the perturbation space of $\{(T_{Del}, 4), (T_{SubAdj}, 1)\}$ for a string with 300 characters contains 20,252,321,116 perturbed samples!).

The exhaustive accuracy of A3T(HotFlip) and A3T(search) decreases by 1.6% and 1.1%, respectively, when increasing δ_1 from 1 to 4, and decreases by 2.1% and 1.4%, respectively, when increasing δ_2 from 1 to 4. All other techniques result in larger decreases in exhaustive accuracy ($\geq$1.7% in $\{(T_{DelStop}, \delta_1), (T_{SubSyn}, 2)\}$ and $\geq$4.2% in $\{(T_{DelStop}, 2), (T_{SubSyn}, \delta_2)\}$).

To answer **RQ2**, **even in the presence of large perturbation spaces A3T yields models that are more robust than those produced by augmentation techniques**.

6. Conclusion, Limitations, and Future Work

We presented an adversarial training technique, A3T, that combines augmentation and abstraction techniques to achieve robustness against programmable string transformations in neural networks for NLP tasks. In the experiments, we showed that A3T yields more robust models than augmentation and abstraction techniques.

We foresee many future improvements to A3T. First, A3T cannot currently generalize to RNNs because its abstraction technique can only be applied to models where the first layer is an affine transformation (e.g. linear or convolutional layer). Applying A3T to RNNs will require designing new abstraction techniques for RNNs. Second, we manually split S into S_{aug} and S_{abs}. Performing the split automatically is left as future work. Third, A3T(search) achieves the best performance by looking for the worst-case perturbed sample in the perturbation space of S_{aug} via *enumeration*. In some practical settings, S_{aug} might induce a large perturbation space and it might best to use A3T(HotFlip) instead. Fourth, we choose HotFlip and interval abstraction to approximate the worst-case loss in our experiments, but our approach is general and can benefit from new augmentation and abstraction techniques.

Acknowledgements

We would like to thank Samuel Drews for his feedback, as well as the reviewers. This work is supported by the National Science Foundation grants CCF-1704117, CCF-1918211, and Facebook research award Probability and Programming.

References

Balunovic, M. and Vechev, M. T. Adversarial training and provable defenses: Bridging the gap. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020. URL https://openreview.net/forum?id=SJxSDxrKDr.

Carlini, N. and Wagner, D. A. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*, pp. 3–14, 2017. doi: 10.1145/3128572.3140444. URL https://doi.org/10.1145/3128572.3140444.

Cousot, P. and Halbwachs, N. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 84–96, 1978.

Ebrahimi, J., Rao, A., Lowd, D., and Dou, D. Hotflip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pp. 31–36, 2018. doi: 10.18653/v1/P18-2006. URL https://www.aclweb.org/anthology/P18-2006/.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T. A., and Kohli, P. Scalable verified training for provably robust image classification. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 4841–4850, 2019. doi: 10.1109/ICCV.2019.00494. URL https://doi.org/10.1109/ICCV.2019.00494.

Gulli, A. The anatomy of a news search engine. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005 - Special interest tracks and posters*, pp. 880–881, 2005. doi: 10.1145/1062745.1062778.

Huang, P., Stanforth, R., Welbl, J., Dyer, C., Yogatama, D., Gowal, S., Dvijotham, K., and Kohli, P. Achieving verified robustness to symbol substitutions via interval bound propagation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 4081–4091, 2019. doi: 10.18653/v1/D19-1419. URL https://doi.org/10.18653/v1/D19-1419.

Iyyer, M., Wieting, J., Gimpel, K., and Zettlemoyer, L. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pp. 1875–1885, 2018. URL https://www.aclweb.org/anthology/N18-1170/.

Jia, R., Raghunathan, A., Göksel, K., and Liang, P. Certified robustness to adversarial word substitutions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 4127–4140, 2019. doi: 10.18653/v1/D19-1423. URL https://doi.org/10.18653/v1/D19-1423.

Karpukhin, V., Levy, O., Eisenstein, J., and Ghazvininejad, M. Training on synthetic noise improves robustness to natural noise in machine translation. In *Proceedings of the 5th Workshop on Noisy User-generated Text, W-NUT@EMNLP 2019, Hong Kong, China, November 4, 2019*, pp. 42–47, 2019. doi: 10.18653/v1/D19-5506. URL https://doi.org/10.18653/v1/D19-5506.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Ko, C., Lyu, Z., Weng, L., Daniel, L., Wong, N., and Lin, D. POPQORN: quantifying robustness of recurrent neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 3468–3477, 2019. URL http://proceedings.mlr.press/v97/ko19a.html.

Liang, B., Li, H., Su, M., Bian, P., Li, X., and Shi, W. Deep text classification can be fooled. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 4208–4215, 2018. doi: 10.24963/ijcai.2018/585. URL https://doi.org/10.24963/ijcai.2018/585.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Michel, P., Li, X., Neubig, G., and Pino, J. M. On evaluation of adversarial perturbations for sequence-to-sequence models. *CoRR*, abs/1903.06620, 2019. URL http://arxiv.org/abs/1903.06620.

Mirman, M., Gehr, T., and Vechev, M. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning (ICML)*, 2018. URL https://www.icml.cc/Conferences/2018/Schedule?showEvent=2477.

Mirman, M., Singh, G., and Vechev, M. T. A provable defense for deep residual networks. *CoRR*, abs/1903.12519, 2019. URL http://arxiv.org/abs/1903.12519.

Pavlick, E., Rastogi, P., Ganitkevitch, J., Van Durme, B., and Callison-Burch, C. PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 425–430, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-2070. URL https://www.aclweb.org/anthology/P15-2070.

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Pruthi, D., Dhingra, B., and Lipton, Z. C. Combating adversarial misspellings with robust word recognition. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 5582–5591, 2019. doi: 10.18653/v1/p19-1561. URL https://doi.org/10.18653/v1/p19-1561.

Ribeiro, M. T., Singh, S., and Guestrin, C. Semantically equivalent adversarial rules for debugging NLP models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pp. 856–865, 2018. doi: 10.18653/v1/P18-1079. URL https://www.aclweb.org/anthology/P18-1079/.

Sakaguchi, K., Duh, K., Post, M., and Durme, B. V. Robsut wrod reocginiton via semi-character recurrent neural network. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 3281–3287, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14332.

Shi, Z., Zhang, H., Hsieh, C.-J., Chang, K.-W., and Huang, M. Robustness verification for transformers. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJxwPJHFwS.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1631–1642, 2013. URL https://www.aclweb.org/anthology/D13-1170/.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Wang, B., Pei, H., Liu, H., and Li, B. Advcodec: Towards A unified framework for adversarial text generation. *CoRR*, abs/1912.10375, 2019. URL http://arxiv.org/abs/1912.10375.

Welbl, J., Huang, P.-S., Stanforth, R., Gowal, S., Dvijotham, K. D., Szummer, M., and Kohli, P. Towards verified robustness under text deletion interventions. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SyxhVkrYvr.

Zhang, H., Zhou, H., Miao, N., and Li, L. Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 5564–5569, 2019a. URL https://www.aclweb.org/anthology/P19-1559/.

Zhang, W. E., Sheng, Q. Z., Alhazmi, A., and LI, C. Adversarial attacks on deep learning models in natural language processing: A survey. *arXiv preprint arXiv:1901.06796*, 2019b.

Zhang, X., Zhao, J. J., and LeCun, Y. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 649–657, 2015. URL http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.

Zhao, Z., Dua, D., and Singh, S. Generating natural adversarial examples. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=H1BLjgZCb.

A. Appendix

A.1. Semantics of specifications

We define the semantics of a specification $S = \{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}$ (such that $T_i = (\varphi_i, f_i)$) as follows. Given a string $\mathbf{x} = x_1 \ldots x_m$, a string $\mathbf{y}$ is in the perturbations space $S(\mathbf{x})$ if:

1. there exists matches $\langle (l_1, r_1), j_1 \rangle \ldots \langle (l_k, r_k), j_k \rangle$ (we assume that matches are sorted in ascending order of l_i) such that for every $i \leqslant k$ we have that (l_i, r_i) is a valid match of φ_{j_i} in $\mathbf{x}$;
2. the matches are not overlapping: for every two distinct i_1 and i_2, $r_{i_1} < l_{i_2}$ or $r_{i_2} < l_{i_1}$;
3. the matches respect the δ constraints: for every $j' \leqslant n$, $|\{\langle (l_i, r_i), j_i \rangle \mid j_i = j'\}| \leqslant \delta_{j'}$.
4. the string $\mathbf{y}$ is the result of applying an appropriate transformation to each match: if for every $i \leqslant k$ we have $\mathbf{s}_i \in f_{j_i}(x_{l_i} \ldots x_{r_i})$, then

$$\mathbf{y} = x_1 \ldots x_{l_1 - 1}\, \mathbf{s}_1\, x_{r_1 + 1} \ldots x_{l_k - 1}\, \mathbf{s}_k\, x_{r_k + 1} \ldots x_m.$$

A.2. Proof of Theorem 1

We give the following definition of a convex set:

Definition 1. Convex set: A set $\mathcal{C}$ is **convex** if, for all x and y in $\mathcal{C}$, the line segment connecting x and y is included in $\mathcal{C}$.

Proof. We first state and prove the following lemma.

Lemma 2. Given a set of points $\{p_0, p_1, \ldots, p_t\}$ and a convex set $\mathcal{C}$ such that $\{p_0, p_1, \ldots, p_t\} \subset \mathcal{C}$. These points define a set of vectors $\overrightarrow{p_0 p_1}, \overrightarrow{p_0 p_2}, \ldots, \overrightarrow{p_0 p_t}$. If a vector $\overrightarrow{p_0 p}$ can be represented as a sum weighed by α_i:

$$\overrightarrow{p_0 p} = \sum_{i=1}^{t} \alpha_i \cdot \overrightarrow{p_0 p_i}, \tag{4}$$

where α_i respect to constraints:

$$\sum_{i=1}^{t} \alpha_i \leq 1 \wedge \forall 1 \leq i \leq t.\ \alpha_i \geq 0, \tag{5}$$

then the point p is also in the convex set $\mathcal{C}$.

Proof. We prove this lemma by induction on t,

- Base case: $t = 1$, if $\overrightarrow{p_0 p} = \alpha_1 \cdot \overrightarrow{p_0 p_1}$ and $0 \leq \alpha_1 \leq 1$, then p is on the segment $p_0 p_1$. By the definition of the convex set (Definition 1), the segment $p_0 p_1$ is inside the convex, which implies p is inside the convex: $p \in p_0 p_1 \subseteq \mathcal{C}$.
- Inductive step: Suppose the lemma holds for $t = r$. If a vector $\overrightarrow{p_0 p}$ can be represented as a sum weighed by α_i:

$$\overrightarrow{p_0 p} = \sum_{i=1}^{r+1} \alpha_i \cdot \overrightarrow{p_0 p_i} \tag{6}$$

where α_i respect to constraints:

$$\sum_{i=1}^{r+1} \alpha_i \leq 1, \tag{7}$$

$$\forall 1 \leq i \leq r+1.\ \alpha_i \geq 0. \tag{8}$$

We divide the sum in Eq 6 into two parts:

$$\overrightarrow{p_0 p} = \sum_{i=1}^{r+1} \alpha_i \cdot \overrightarrow{p_0 p_i} \tag{9}$$

$$= (\sum_{i=1}^{r} \alpha_i \cdot \overrightarrow{p_0 p_i}) + \alpha_{r+1} \cdot \overrightarrow{p_0 p_{r+1}} \tag{10}$$

$$= (1 - \alpha_{r+1}) \overrightarrow{p_0 p'} + \alpha_{r+1} \cdot \overrightarrow{p_0 p_{r+1}} \quad \text{, and} \tag{11}$$

$$\overrightarrow{p_0 p'} = \sum_{i=1}^{r} \frac{\alpha_i}{1 - \alpha_{r+1}} \cdot \overrightarrow{p_0 p_i} \tag{12}$$

Because from Inequality 7, we know that

$$\sum_{i=1}^{r} \alpha_i \leq 1 - \alpha_{r+1},$$

which is equivalent to

$$\sum_{i=1}^{r} \frac{\alpha_i}{1 - \alpha_{r+1}} \leq 1.$$

This inequality enables the inductive hypothesis, and we know point p' is in the convex set $\mathcal{C}$. From Eq 11, we know that the point p is on the segment of $p' p_{r+1}$, since both two points p' and p_{r+1} are in the convex set $\mathcal{C}$, then the point p is also inside the convex set $\mathcal{C}$.

□

To prove Theorem 1, we need to show that every perturbed sample $\mathbf{y} \in S(\mathbf{x})$ lies inside the convex hull of $abstract(S, \mathbf{x})$.

We first describe the perturbed sample $\mathbf{y}$. The perturbed sample $\mathbf{y}$ as a string is defined in the semantics of specification S (see the Appendix A.1). In the rest of this proof, we use a function $E : \Sigma^m \mapsto \mathbb{R}^{m \times d}$ mapping from a string with length m to a point in $m \times d$-dimensional space, e.g., $E(\mathbf{y})$ represents the point of the perturbed sample $\mathbf{y}$ in the

embedding space. We use $\mathbf{x}_{\langle (l,r),j,\mathbf{s}\rangle}$ to represent the string perturbed by a transformation $T_j = (\varphi_j, f_j)$ such that (l, r) is a valid match of φ_j and $\mathbf{s} \in f_j(x_l, \ldots, x_r)$. Then

$$\mathbf{x}_{\langle (l,r),j,\mathbf{s}\rangle} = x_1 \ldots x_{l-1}\, \mathbf{s}\, x_{r+1} \ldots x_m.$$

We further define $\Delta_{\langle (l,r),j,\mathbf{s}\rangle}$ as the vector $E(\mathbf{x}_{\langle (l,r),j,\mathbf{s}\rangle}) - E(\mathbf{x}) = \overrightarrow{E(\mathbf{x})E(\mathbf{x}_{\langle (l,r),j,\mathbf{s}\rangle})}$:

$$\Delta_{\langle (l,r),j,\mathbf{s}\rangle} = (\underbrace{0, \ldots, 0}_{(l-1)\times d}, E(\mathbf{s}) - E(x_l \ldots x_r), \underbrace{0, \ldots, 0}_{(m-r)\times d}).$$

A perturbed sample $\mathbf{y}$ defined by matches $\langle (l_1, r_1), j_1\rangle \ldots \langle (l_k, r_k), j_k\rangle$ and for every $i \leqslant k$ we have $\mathbf{s}_i \in f_{j_i}(x_{l_i} \ldots x_{r_i})$, then

$$\mathbf{y} = x_1 \ldots x_{l_1-1}\, \mathbf{s}_1\, x_{r_1+1} \ldots x_{l_k-1}\, \mathbf{s}_k\, x_{r_k+1} \ldots x_m.$$

The matches respect the δ constraints: for every $j' \leq n$, $|\{\langle (l_i, r_i), j_i, \mathbf{s}_i\rangle \mid j_i = j'\}| \leqslant \delta_{j'}$. Thus, the size of the matches k also respect the δ constraints:

$$k = \sum_{j'=1}^{n} |\{\langle (l_i, r_i), j_i, \mathbf{s}_i\rangle \mid j_i = j'\}| \leq \sum_{j'=1}^{n} \delta_{j'}. \quad (13)$$

In the embedding space,

$$\overrightarrow{E(\mathbf{x})E(\mathbf{y})} = (\underbrace{0, \ldots, 0}_{(l_1-1)\times d}, E(\mathbf{s}_1) - E(x_{l_1} \ldots x_{r_1}), 0, \ldots, 0, E(\mathbf{s}_k) - E(x_{l_k} \ldots x_{r_k}), \underbrace{0, \ldots, 0}_{(m-r_k)\times d}).$$

Thus, we can represent $\overrightarrow{E(\mathbf{x})E(\mathbf{y})}$ using $\Delta_{\langle (l,r),j,\mathbf{s}\rangle}$:

$$\overrightarrow{E(\mathbf{x})E(\mathbf{y})} = \sum_{i=1}^{k} \Delta_{\langle (l_i,r_i),j_i,\mathbf{s}_i\rangle}. \quad (14)$$

We then describe the convex hull of $abstract(S, \mathbf{x})$. The convex hull of $abstract(S, \mathbf{x})$ is constructed by a set of points $E(\mathbf{x})$ and $E(\mathbf{v}_{\langle (l,r),i,\mathbf{s}\rangle})$, where points $E(\mathbf{v}_{\langle (l,r),i,\mathbf{s}\rangle})$ are computed by:

$$E(\mathbf{v}_{\langle (l,r),j,\mathbf{s}\rangle}) \triangleq E(\mathbf{x}) + (\sum_{i=1}^{n} \delta_i)(E(\mathbf{x}_{\langle (l,r),j,\mathbf{s}\rangle}) - E(\mathbf{x})).$$

Alternatively, using the definition of $\Delta_{\langle (l,r),j,\mathbf{s}\rangle}$, we get

$$\overrightarrow{E(\mathbf{x})E(\mathbf{v}_{\langle (l,r),j,\mathbf{s}\rangle})} = (\sum_{i=1}^{n} \delta_i)\Delta_{\langle (l,r),j,\mathbf{s}\rangle}. \quad (15)$$

We then prove the Theorem 1. To prove $E(\mathbf{y})$ lies in the convex hull of $abstract(S, \mathbf{x})$, we need to apply Lemma 2.

Notice that a convex hull by definition is also a convex set. Because from Eq 14, we have

$$\overrightarrow{E(\mathbf{x})E(\mathbf{y})} = \sum_{i=1}^{k} \Delta_{\langle (l_i,r_i),j_i,\mathbf{s}_i\rangle}$$
$$= \frac{1}{\sum_{i=1}^{n} \delta_i} \sum_{i=1}^{k} (\sum_{i'=1}^{n} \delta_{i'}) \Delta_{\langle (l_i,r_i),j_i,\mathbf{s}_i\rangle}.$$

We can use Eq 15 into the above equation, and have

$$= \frac{1}{\sum_{i=1}^{n} \delta_i} \sum_{i=1}^{k} \overrightarrow{E(\mathbf{x})E(\mathbf{v}_{\langle (l_i,r_i),j_i,\mathbf{s}_i\rangle})}$$
$$= \sum_{i=1}^{k} (\frac{1}{\sum_{i=1}^{n} \delta_i}) \cdot \overrightarrow{E(\mathbf{x})E(\mathbf{v}_{\langle (l_i,r_i),j_i,\mathbf{s}_i\rangle})}.$$

To apply Lemma 2, we set

$$\alpha_i = \frac{1}{\sum_{j=1}^{n} \delta_j}.$$

Using Inequality 13 on

$$\alpha_i = \frac{1}{\sum_{j=1}^{n} \delta_j} \geq 0, \quad (16)$$

we get

$$\sum_{i=1}^{k} \alpha_i = \sum_{i=1}^{k} \frac{1}{\sum_{j=1}^{n} \delta_j} = \frac{k}{\sum_{j=1}^{n} \delta_j} \leq 1. \quad (17)$$

The constraints in Inequality 16 and Inequality 17 enable Lemma 2, and by applying Lemma 2, we know that point $E(\mathbf{y})$ is inside the convex hull of $abstract(S, \mathbf{x})$. □

A.3. Details of Experiment Setup

For AG dataset, we trained a smaller character-level model than the one used in Huang et al. (2019). We followed the setup of the previous work: use lower-case letters only and truncate the inputs to have at most 300 characters. The model consists of an embedding layer of dimension 64, a 1-D convolution layer with 64 kernels of size 10, a ReLU layer, a 1-D average pooling layer of size 10, and two fully-connected layers with ReLUs of size 64, and a linear layer. We randomly initialized the character embedding and updated it during training.

For SST2 dataset, we trained the same word-level model as the one used in Huang et al. (2019). The model consists of an embedding layer of dimension 300, a 1-D convolution layer with 100 kernels of size 5, a ReLU layer, a 1-D average pooling layer of size 5, and a linear layer. We used the pre-trained Glove embedding (Pennington et al., 2014) with dimension 300 and fixed it during training.

For SST2 dataset, we trained the same character-level model as the one used in Huang et al. (2019). The model consists of an embedding layer of dimension 150, a 1-D convolution layer with 100 kernels of size 5, a ReLU layer, a 1-D average pooling layer of size 5, and a linear layer. We randomly initialized the character embedding and updated it during training.

For all models, we used Adam (Kingma & Ba, 2015) with a learning rate of 0.001 for optimization and applied early stopping policy with patience 5.

A.3.1. PERTURBATIONS

We provide the details of the string transformations we used:

- T_{SubAdj}, T_{InsAdj}: We allow each character substituting to one of its adjacent characters on the QWERTY keyboard.
- $T_{DelStop}$: We choose {*and*, *the*, *a*, *to*, *of*} as our stop words set.
- T_{SubSyn}: We use the synonyms provided by PPDB (Pavlick et al., 2015). We allow each word substituting to its closest synonym when their part-of-speech tags are also matched.

A.3.2. BASELINE

Random augmentation performs adversarial training using a weak adversary that simply picks a random perturbed sample from the perturbation space. For a specification $S = \{(T_1, \delta_1), \ldots, (T_n, \delta_n)\}$, we produce $\mathbf{z}$ by uniformly sampling one string $\mathbf{z}_1$ from a string transformation (T_1, δ_1) and passing it to the next transformation (T_2, δ_2), where we then sample a new string $\mathbf{z}_2$, and so on until we have exhausted all transformations. The objective function is the following:

$$\underset{\theta}{\operatorname{argmin}} \underset{(\mathbf{x},y)\sim\mathcal{D}}{\mathbb{E}} (\mathcal{L}(x, y, \theta) + \max_{\mathbf{z}\in R(\mathbf{x})} \mathcal{L}(\mathbf{z}, y, \theta)) \quad (18)$$

HotFlip augmentation performs adversarial training using the HotFlip (Ebrahimi et al., 2018) attack to find $\mathbf{z}$ and solve the inner maximization problem. The objective function is the same as Eq 18.

A3T adopts a curriculum-based training method (Huang et al., 2019; Gowal et al., 2019) that uses a hyperparameter λ to weigh between normal loss and maximization objective in Eq. (2). We linearly increase the hyperparameter λ during training.

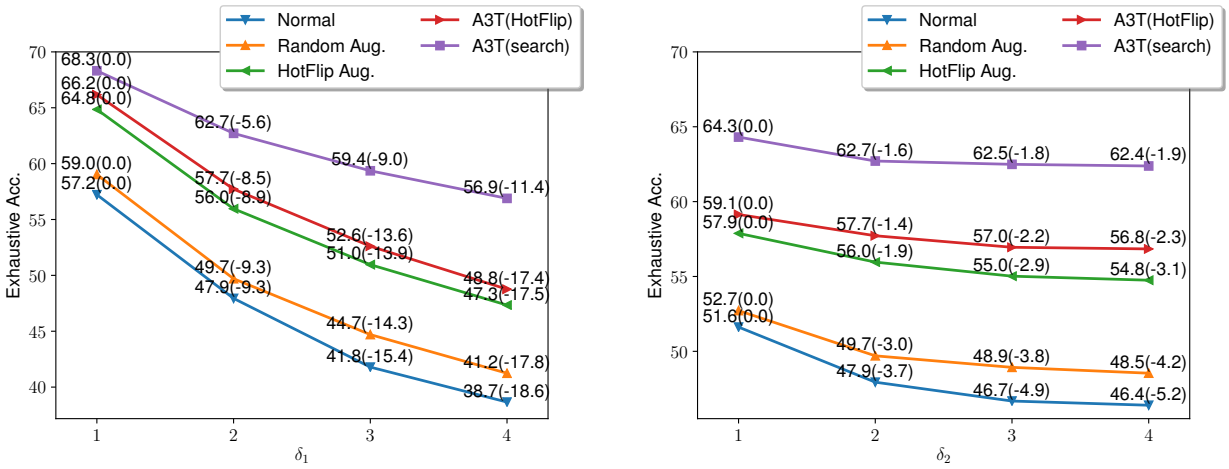
$$\underset{\theta}{\operatorname{argmin}} \underset{(\mathbf{x},y)\sim\mathcal{D}}{\mathbb{E}} ((1-\lambda)\mathcal{L}(x, y, \theta) + \lambda \max_{\mathbf{z}\in \mathit{augment}_k(S_{aug},\mathbf{x})} \mathcal{L}(\mathit{abstract}(S_{abs}\mathbf{z}), y, \theta)).$$

Also, we set k in $\mathit{augment}_k$ to 2, which means we select 2 perturbed samples to abstract.

A.3.3. EVALUATION RESULTS

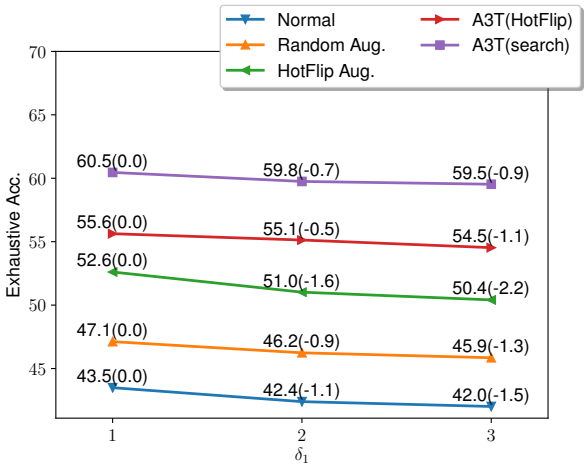
RQ2: Effects of size of the perturbation space In Figure 4, we fix the word-level model A3T (search) trained on $\{(T_{Dup}, 2), (T_{SubSyn}, 2)\}$. Then, we test this model's exhaustive accuracy on $\{(T_{Dup}, \delta_1), (T_{SubSyn}, 2)\}$ (Figure 4(a)) and $\{(T_{Dup}, 2), (T_{SubSyn}, \delta_2)\}$ (Figure 4(b)), where we vary the parameters δ_1 and δ_2 between 1 and 4, increasing the size of the perturbation space. The exhaustive accuracy of A3T(HotFlip) and A3T(search) decreases by 17.4% and 11.4%, respectively, when increasing δ_1 from 1 to 4, and decreases by 2.3% and 1.9%, respectively, when increasing δ_2 from 1 to 4. All other techniques result in larger decreases in exhaustive accuracy ($\geq$17.5% in $\{(T_{Dup}, \delta_1), (T_{SubSyn}, 2)\}$ and $\geq$3.1% in $\{(T_{Dup}, 2), (T_{SubSyn}, \delta_2)\}$).

In Figure 5, we fix the word-level model A3T (search) trained on $\{(T_{DelStop}, 2), (T_{Dup}, 2), (T_{SubSyn}, 2)\}$. Then, we test this model's exhaustive accuracy on $\{(T_{DelStop}, \delta_1), (T_{Dup}, 2), (T_{SubSyn}, 2)\}$ (Figure 5(a)), $\{(T_{DelStop}, 2), (T_{Dup}, \delta_2), (T_{SubSyn}, 2)\}$ (Figure 5(b)), and $\{(T_{DelStop}, 2), (T_{Dup}, 2), (T_{SubSyn}, \delta_3)\}$ (Figure 5(c)), where we vary the parameters δ_1, δ_2 and δ_3 between 1 and 3, increasing the size of the perturbation space. The exhaustive accuracy of A3T(HotFlip) and A3T(search) decreases by 1.1% and 0.9%, respectively, when increasing δ_1 from 1 to 3, decreases by 12.9% and 6.9%, respectively, when increasing δ_2 from 1 to 3, and decreases by 1.4% and 0.9%, respectively, when increasing δ_3 from 1 to 3. All other techniques result in larger decreases in exhaustive accuracy ($\geq$2.2% in $\{(T_{DelStop}, \delta_1), (T_{Dup}, 2), (T_{SubSyn}, 2)\}$, $\geq$13.0% in $\{(T_{DelStop}, 2), (T_{Dup}, \delta_2), (T_{SubSyn}, 2)\}$, and $\geq$2.8% in $\{(T_{DelStop}, 2), (T_{Dup}, 2), (T_{SubSyn}, \delta_3)\}$).

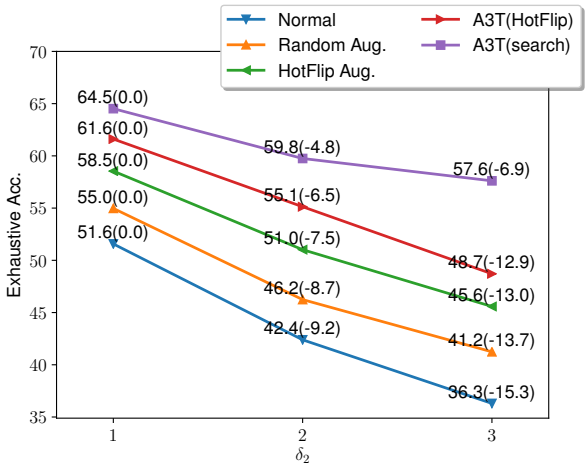


(a) $\{(T_{Dup}, \delta_1), (T_{SubSyn}, 2)\}$ (b) $\{(T_{DelStop}, 2), (T_{Dup}, 2), (T_{SubSyn}, \delta_3)\}$

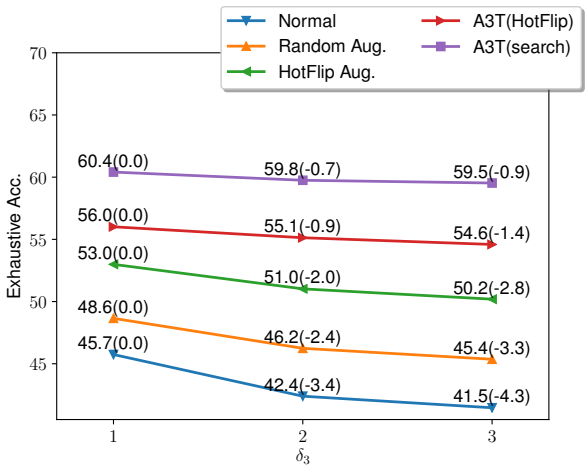
Figure 4. The exhaustive accuracy of $\{(T_{Dup}, \delta_1), (T_{SubSyn}, \delta_2)\}$, varying the parameters δ_1 (left) and δ_2 (right) between 1 and 4.



(a) $\{(T_{DelStop}, \delta_1), (T_{Dup}, 2), (T_{SubSyn}, 2)\}$



(b) $\{(T_{DelStop}, 2), (T_{Dup}, \delta_2), (T_{SubSyn}, 2)\}$



(c) $\{(T_{Dup}, 2), (T_{SubSyn}, \delta_2)\}$

Figure 5. The exhaustive accuracy of $\{(T_{DelStop}, \delta_1), (T_{Dup}, \delta_2), (T_{SubSyn}, \delta_3)\}$, varying the parameters δ_1 (left), δ_2 (middle), and δ_3 (right) between 1 and 3.