Robustness to Programmable String Transformations via Augmented Abstract Training

Yuhao Zhang 1  Aws Al barghouthi 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). 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 $\max_{z \in R(x)} L(z, y, \theta)$.

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 upper bound on the adversarial loss by overapproximat-
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 $T_i : \mathcal{X} \rightarrow 2^\mathcal{X}$. The specification $S$ defines a perturbation space of all possible strings by applying each transformation $T_i$ up to $\delta_i$ 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 gradients to generate word-level adversarial text via word

---

\(^1\)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/](https://anonymous.4open.science/r/090c73a5-0825-4668-ae8d-96f8421ad0ec/).
We are interested in the setting where the sample space is a finite set of possible perturbed strings of some neural-network model. Other techniques involve placing a spelling-mistake-detection model that identifies possible adversaries before the underlying model (Pruthi et al., 2019; Sakaguchi et al., 2017).

### Abstract Training

Mirman et al. (2018) and Gowal et al. (2018) 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. Huang et al. (2019) proposed a simplex space to capture the perturbation (IBP) to train image classification models that are provably robust to norm-bounded adversarial perturbations. 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. 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 $\theta$ of some neural-network architecture $F_\theta$ that minimize the expected loss

$$\arg\min_{\theta} \mathbb{E}_{(x,y) \sim \mathcal{D}} \mathcal{L}(x,y,\theta)$$

(1)

We are interested in the setting where the sample space $\mathcal{X}$ defines strings over some finite alphabet $\Sigma$. The alphabet $\Sigma$ 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 $\Sigma^*$, i.e., the set of all strings of elements of $\Sigma$. We will use $x \in \Sigma^*$ to denote a string and $x_i \in \Sigma$ to denote the $i$th element of the string.

**Perturbation Space**

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

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

$$\arg\min_{\theta} \mathbb{E}_{(x,y) \sim \mathcal{D}} \max_{z \in R(x)} \mathcal{L}(z,y,\theta)$$

(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 $F_\theta$ using the robust optimization objective, we will use exhaustive accuracy to quantify its classification accuracy in the face of perturbations. Specifically, given a dataset $D = \{(x_i, y_i)\}^n_{i=1}$ and a perturbation space $R$, we define exhaustive accuracy as follows:

$$\frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{\forall z \in R(x_i) : F_\theta(z) = y_i\}$$

(3)

Intuitively, for each sample $(x_i, y_i)$, its classification is considered correct iff $F_\theta$ predicts $y_i$ for every single point in $R(x_i)$. 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 $(\phi, f)$, where $\phi : \Sigma^* \rightarrow \{0,1\}$ is a Boolean function.
predicate (in practice, a regular expression) describing the substrings of the inputs to which the transformation can be applied, and \( f : \Sigma^n \to 2\Sigma^n \) is a transformer describing how the substrings matched by \( \varphi \) can be replaced.

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

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

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

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

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

**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 \( \varphi_i, f_i \). The formal semantics of our language can be found in the supplementary Appendix. Informally, a string \( z \) is in the perturbation space \( S(x) \) if it can be obtained by (1) finding a set \( \sigma \) of non-overlapping substrings of \( x \) that match the various predicates \( \varphi_i \), and such that at most \( \delta_i \) substrings in \( \sigma \) are matches of \( \varphi_i \), and (2) replacing each substring \( x' \in \sigma \) matched by \( \varphi_i \) with a string in \( f_i(x') \). 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 \( T_{\text{nice}} \) and \( T_{\text{swap}} \) we can define the specification \( S_{\text{ns}} = \{ (T_{\text{nice}}, 1), (T_{\text{swap}}, 1) \} \)

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

\[
\begin{align*}
\text{This house is nice} & \quad \text{This is enjoyable} \\
\text{This house is pleasant} & \quad \text{This house is nice} \\
\text{This house is enjoyable} & \quad \text{This house is pleasant}
\end{align*}
\]

The transformed portions are shown in bold. Note that we apply \textit{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, \( T_{\text{swap}} \) 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: \( \max_{\sigma \in R(x)} \mathcal{L}(z, y, \theta) \), where the perturbation space \( R(x) \) 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 \( R(x) \) with which to augment the dataset. The naïve way is to simply enumerate all points in \( R(x) \)—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 transformations, HotFlip employs a beam search of some size \( k \) to get the top-\( k \) perturbed samples. This technique yields a point in \( R(x) \) 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 propa-
Robustness to String Transformations via Augmented Abstract Training

We now show how to define the abstraction of a perturbation space. Let

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

be a specification of transformations, where \( T_i \) is a single character

Algorithm 1: A3T

Input: \( S = \{(T_1, \delta_1) \ldots, (T_n, \delta_n)\} \) and point \((x, y)\)
Output: worst-case loss

Split \( S \) into \( S_{\text{aug}} \) and \( S_{\text{abs}} \), and return

\[
\max_{z \in \text{augment}_{x}(S_{\text{aug}}, x)} L(z, y, \theta) \quad \text{s.t.} \quad \hat{z} = \text{abstract}(S_{\text{abs}}, z)
\]

Our use of abstraction builds upon the work of Huang et al. (2019), which uses an interval domain to define \( \hat{T}(x) \)—i.e., \( \hat{T}(x) \) is a conjunction of constraints of the form \( l_i \leq x_i \leq u_i \), where \( l_i \) and \( u_i \) are constants. We will describe how we generalize their approach in Section 4.2; for now, we assume that we can efficiently overapproximate the worst-case loss for \( \hat{T}(x) \) 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, \( S_{\text{aug}} \) and \( S_{\text{abs}} \). For \( S_{\text{aug}} \), we apply an augmentation technique, e.g., HotFlip or MHA, to come up with a list of top- \( k \) perturbed samples in the set \( S_{\text{aug}}(x) \)—this is denoted as the set \( \text{augment}_{x}(S_{\text{aug}}, x) \).

Then, for each point \( x \) in the top- \( k \) results, we compute an abstraction \( \text{abstract}(S_{\text{abs}}, z) \), which is a set of constraints over-approximating the set of points in \( S_{\text{abs}}(z) \). 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 \( S(x) \) 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 \( S(x) \) is the same as the length of the original string \( x \). The approach of Huang et al. (2019) targeted the special case of single-character substitutions.

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 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 \( T(x) \) (we use \( T(x) \) as a short hand for the perturbation space obtained by applying \( T \) to \( x \) exactly once) and then scales this convex hull by \( \delta \) to account for the cases in which \( T \) is applied up to \( \delta \) times. We begin by computing all points in \( T(x) \). Let this set be \( x_0, \ldots, x_m \). Next, for \( i \in [1, m] \), we define the set of points

\[ v_i = x + \delta \cdot (x_i - x) \]

We then construct the abstraction \( \text{abstract}(S_{\text{abs}}, z) \) as the convex hull of the points \( v_i \) and \( x \). Observe that we only need to enumerate the space \( T(x) \) obtained by applying \( T \) once; multiplying by \( \delta \) dilates the convex hull to include all strings that involve up to \( \delta \) applications of \( T \), i.e., \( S(x) \). 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 \( q \)—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 \( T_1(x) \cup \cdots \cup T_n(x) \). (Notice that we need only consider each transformation \( T_i \) independently.) Let this set be \( x_0, \ldots, x_m \). Next, for \( i \in [1, m] \), we define the following set of points:

\[ v_i = x + (\delta_1 + \cdots + \delta_n) \cdot (x_i - x) \]

As with the single-transformation case, we can now construct an abstraction of the convex hull induced by \( v_i \) and \( x \) as a set of intervals and propagate it through the network.
We prove Theorem 1 in the Appendix.

We use two datasets: The AG News (Zhang et al., 2015) dataset consists of a corpus of news articles collected by Gulli (2005) about the 4 largest news topics. The Stanford Sentiment Treebank (SST2) (Socher et al., 2013) dataset 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.

For the AG dataset, we trained a character-level model proposed by Zhang et al. (2015) following their setup. For the SST2 dataset, we trained a word-level model proposed by Kim (2014) also following their setup. The details of setups are shown 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: The AG News (Zhang et al., 2015) dataset consists of a corpus of news articles collected by Gulli (2005) about the 4 largest news topics. The Stanford Sentiment Treebank (SST2) (Socher et al., 2013) dataset 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.

For the AG dataset, we trained a character-level model proposed by Zhang et al. (2015) following their setup. For the SST2 dataset, we trained a word-level model proposed by Kim (2014) 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_{\text{SwapPair}}, 1), (T_{\text{SubAdj}}, 1)\}, \{(T_{\text{InsAdj}}, 1), (T_{\text{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 word-level model on dataset SST2, we used the following specifications: \{ \{(T_{\text{DelStop}}, 2), (T_{\text{SubSyn}}, 2)\}, \{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2)\}, \{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}\}. The first specification, for example, removes (up to 2) stop words and substitutes (up to 2) words with synonyms.

For the character-level model, we considered perturbations with \( \delta = 1 \) because one cannot efficiently evaluate the exhaustive accuracy with larger \( \delta \), 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). Second, we implemented two different ways of performing data augmentation for the transformations in \( S_{\text{aug}} \): (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 \( S_{\text{abs}} \), 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., 2018) which uses a hyperparameter \( \lambda \) to weigh between normal loss and maximization objective in...
Robustness to String Transformations via Augmented Abstract Training

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

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

Table 2. 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.

| Training       | $\{T_{\text{SwapPair}}, 1\}, (T_{\text{SubAdj}}, 1)\}$ | $\{T_{\text{Del}}, 1\}, (T_{\text{SubAdj}}, 1)\}$ | $\{T_{\text{InsAdj}}, 1\}, (T_{\text{SubAdj}}, 1)\}$ |
|----------------|----------------------------------------------------------|--------------------------------------------------|--------------------------------------------------|
| Normal         | 86.0, 70.1, 58.5                                        | 86.0, 76.8, 58.4                                  | 86.0, 77.3, 56.8                                  |
| Random Aug.    | 86.7, 71.0, 56.6 [-1.9]                                | 86.8, 77.7, 56.9 [-1.5]                            | 86.0, 78.6, 57.9 [+1.1]                            |
| HotFlip Aug.   | 86.8, 77.2, 67.3 [+8.8]                                | 86.7, 80.1, 63.1 [+4.7]                            | 86.7, 82.0, 68.1 [+11.3]                           |
| A3T(HotFlip)   | 84.6, 76.9, 69.1 [+10.6]                               | 86.0, 83.8, 63.8 [+5.4]                            | 85.3, 84.0, 68.7 [+11.9]                           |
| A3T(search)    | 87.2, 80.2, 73.5 [+15.0]                               | 86.8, 84.0, 71.8 [+13.4]                           | 86.6, 84.6, 76.2 [+19.4]                           |

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 $(x, y)$ is considered correct if and only if all points $z \in S(x)$ 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 2 and 3, 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 18.7 higher than normal training, 14.9 higher than random augmentation, and 9.6 higher than HotFlip augmentation.

We also compared A3T to training using only abstraction (i.e., all transformations in $S_{\text{abs}}$) for the specification $\{(T_{\text{SwapPair}}, 1), (T_{\text{SubAdj}}, 1)\}$ (not shown in Tables 2 and 3); 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 66.3, 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 drops to 75.3 due to the over-approximation of the perturbation space while A3T(HotFlip) (84.6) and A3T(search) (87.2) 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 $\{(T_{\text{DelStop}}, 2), (T_{\text{SubSyn}}, 2)\}$. Then, we test this model’s exhaustive accuracy on $\{T_{\text{DelStop}}, 1\}, (T_{\text{SubSyn}}, 2)\}$ (Figure 3(a)) and $\{(T_{\text{DelStop}}, 2), (T_{\text{SubSyn}}, 2)\}$ (Figure 3(b)),
Table 3. Experiment results for the three perturbations on the word-level model on SST dataset.

| Training          | \{T_{DelStop}, 2\}, \{T_{SubSyn}, 2\} | \{T_{DelStop}, 2\}, \{T_{Dup}, 2\} | \{T_{DelStop}, 2\}, \{T_{Dup}, 2\}, \{T_{SubSyn}, 2\} |
|-------------------|----------------------------------------|----------------------------------|---------------------------------------------------|
|                   | Acc.        | HF Acc.     | Exhaustive | Acc.        | HF Acc.     | Exhaustive | Acc.        | HF Acc.     | Exhaustive |
| Normal            | 78.6        | 63.2        | 55.1       | 78.6        | 53.6        | 39.0       | 78.6        | 51.7        | 30.1       |
| Random Aug.       | 79.8        | 68.1        | 60.1 [+5.0] | 80.7        | 64.0        | 49.3 [+10.3] | 79.6        | 60.3        | 39.6 [+9.5] |
| HotFlip Aug.      | 79.6        | 72.7        | 63.9 [+8.8] | 80.3        | 70.9        | 49.3 [+10.3] | 77.5        | 69.3        | 40.7 [+10.6] |
| A3T(HotFlip)      | 76.0        | 71.1        | 66.7 [+1.6] | 78.4        | 66.2        | 57.0 [+18.0] | 76.6        | 66.3        | 54.9 [+24.8] |
| A3T(search)       | 77.1        | 71.9        | **69.0 [+13.9]** | 76.4        | 66.7        | **61.0 [+22.0]** | 76.2        | 68.7        | **58.5 [+28.4]** |

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

where we vary the parameters \delta_1 and \delta_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 perturbed samples, enumeration is infeasible for character level transformations (e.g., 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 1.8% and 2.2%, respectively, when increasing \delta_1 from 1 to 4, and decreases by 1.8% and 2.5%, respectively, when increasing \delta_2 from 1 to 4. All other techniques result in larger decreases in exhaustive accuracy (>3.1% for all methods).

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.
Robustness to String Transformations via Augmented Abstract Training

References

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. On the effectiveness of interval bound propagation for training verifiably robust models. CoRR, abs/1810.12715, 2018. URL http://arxiv.org/abs/1810.12715.

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/.

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.

Kim, Y. Convolutional neural networks for sentence classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL https://www.aclweb.org/anthology/D14-1181.

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-Seven 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.

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. Robust word reocogition 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., Goval, 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 $x = x_1 \ldots x_m$, a string $y$ is in the perturbations space $S(x)$ if:

1. there exists matches $\{(l_i, r_i), j_i\} \ldots \{(l_k, r_k), j_k\}$ (we assume that matches are sorted in ascending order of $l_i$) such that for every $i \leq k$ we have that $(l_i, r_i)$ is a valid match of $\varphi_{j_i}$ in $x$;
2. the matches are not overlapping: for every two distinct $i_1$ and $i_2$, $l_{i_2} < l_{i_1}$ and $r_{i_2} < r_{i_1}$;
3. the matches respect the $\delta$ constraints: for every $j' \leq n$, $|\{(l_i, r_i), j_i\} | j_i = j'| \leq \delta_{j'}$;
4. the string $y$ is the result of applying an appropriate transformation to each match: if for every $i \leq k$ we have $s_{i} \in f_{j_i}(x_{l_i} \ldots x_{r_i})$, then $y = x_{1} \ldots x_{l_1-1} s_1 x_{r_1+1} \ldots x_{l_k-1} s_k x_{r_k+1} \ldots x_{m}$.

A.2. Proof of Theorem 1

We give the following definition of a convex set:

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

**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 $C$ such that $\{p_0, p_1, \ldots, p_t\} \subseteq C$. These points define a set of vectors $p_0 p_1, p_0 p_2, \ldots, p_0 p_t$. If a vector $p_0 \hat{p}$ can be represented as a sum weighed by $\alpha_i$:

$$\sum_{i=1}^{t} \alpha_i \cdot p_0 p_i = \frac{p_0 \hat{p}}{\sum_{i=1}^{t} \alpha_i},$$

where $\alpha_i$ respect to constraints:

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

then the point $p$ is also in the convex set $C$.

**Proof.** We prove this lemma by induction on $t$,

- **Base case:** $t = 1$, if $p_0 \hat{p} = \alpha_1 \cdot p_0 p_1$ 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: $p \in p_0 p_1 \subseteq C$.

- **Inductive step:** Suppose the lemma holds for $t = r$. If a vector $p_0 \hat{p}$ can be represented as a sum weighed by $\alpha_i$:

$$\sum_{i=1}^{r+1} \alpha_i \cdot p_0 p_i = \frac{p_0 \hat{p}}{\sum_{i=1}^{r+1} \alpha_i},$$

where $\alpha_i$ respect to constraints:

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

$$\forall 1 \leq i \leq r + 1. \alpha_i \geq 0.$$

We divide the sum in Eq 6 into two parts:

$$p_0 \hat{p} = \sum_{i=1}^{r+1} \alpha_i \cdot p_0 p_i = \sum_{i=1}^{r} \alpha_i \cdot p_0 p_i + \alpha_{r+1} \cdot p_0 p_{r+1},$$

$$= \left(\sum_{i=1}^{r} \alpha_i \cdot p_0 p_i\right) + \alpha_{r+1} \cdot p_0 p_{r+1},$$

$$= (1 - \alpha_{r+1}) p_0 \hat{p} + \alpha_{r+1} \cdot p_0 p_{r+1},$$

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 $C$. 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 $C$, then the point $p$ is also inside the convex set $C$.

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

**We first describe the perturbed sample $y$.** The perturbed sample $y$ 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 $E : \Sigma^m \rightarrow \mathbb{R}^{m \times d}$ mapping from a string with length $m$ to a point in $m \times d$-dimensional space, e.g., $E(y)$ represents the point of the perturbed sample $y$ in the
We further define \( \Delta_{((l,r),j,s)} \) as the vector \( E(x_{((l,r),j,s)}) - E(x) \).

\[
\Delta_{((l,r),j,s)} = \begin{pmatrix} 0, \ldots, 0, E(s) - E(x_l \ldots x_r), 0, \ldots, 0 \end{pmatrix}.
\]

A perturbed sample \( y \) defined by matches \((l_1,r_1),j_1\) \( \ldots \) \((l_k,r_k),j_k)\) and for every \( i \leq k \) we have \( s_i \in f_{j_i}(x_{i_1} \ldots x_{r_i}) \), then

\[
y = x_1 \ldots x_{l_1-1} s_1 x_{r_1+1} \ldots x_{l_2-1} s_2 x_{r_2+1} \ldots x_m.
\]

The matches respect the \( \delta \) constraints: for every \( j' \leq n, \left| \{(l_1,r_1),j_1,s_1) \mid j_1 = j' \} \right| \leq \delta_{j'} \). Thus, the size of the matches \( k \) also respect the \( \delta \) constraints:

\[
k = \sum_{j'=1}^{n} \left| \{(l_1,r_1),j_1,s_1) \mid j_1 = j' \} \right| \leq \sum_{j'=1}^{n} \delta_{j'}.
\]  

(13)

In the embedding space,

\[
\overrightarrow{E(x)}E(y) = \begin{pmatrix} 0, \ldots, 0, E(s_1) - E(x_{l_1} \ldots x_{r_1}), 0, \ldots, 0 \end{pmatrix}.
\]

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

\[
\overrightarrow{E(x)}E(y) = \sum_{i=1}^{k} \Delta_{((l_i,r_i),j_i,s_i)}.
\]  

(14)

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

\[
E(v_{((l,r),i,s)}) \triangleq E(x) + \sum_{i=1}^{n} \delta_i (E(x_{((l,r),j,s)}) - E(x)).
\]

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

\[
\overrightarrow{E(x)}E(v_{((l,r),j,s)}) = \sum_{i=1}^{n} \delta_i \Delta_{((l,r),j,s)}.
\]  

(15)

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

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

\[
\overrightarrow{E(x)}E(y) = \sum_{i=1}^{k} \Delta_{((l_i,r_i),j_i,s_i)}
\]

\[
= \frac{1}{\sum_{i=1}^{k} \delta_i} \sum_{i=1}^{k} \delta_i \Delta_{((l_i,r_i),j_i,s_i)}.
\]

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

\[
= \frac{1}{\sum_{i=1}^{k} \delta_i} \sum_{i=1}^{k} \overrightarrow{E(x)}E(v_{((l_i,r_i),j_i,s_i)})
\]

\[
= \sum_{i=1}^{k} \frac{1}{\sum_{j=1}^{n} \delta_j} \overrightarrow{E(x)}E(v_{((l_i,r_i),j_i,s_i)}).
\]

To apply Lemma 2, we set

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

Using Inequality 13 on

\[
\alpha_i = \frac{1}{\sum_{j=1}^{n} \delta_j} \geq 0,
\]  

(16)

we get

\[
\sum_{i=1}^{k} \alpha_i = \sum_{i=1}^{k} \frac{1}{\sum_{j=1}^{n} \delta_j} = \frac{k}{\sum_{j=1}^{n} \delta_j} \leq 1.
\]  

(17)

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

A.3. Details of Experiment Setup

A.3.1. Datasets and Models

- **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\(^3\). The dataset contains 30,000 training and 1,900 testing examples for each class. We split the first 1,000 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

\(^3\)This is the website describing the dataset: https://github.com/mhjabreel/CharCNN_Keras/tree/master/data/ag_news_csv.
a given sentence. We used the dataset provided by TensorFlow4. The dataset contains 67,349 training, 872 validation, and 1,821 testing examples for each class.

For AG dataset, we trained a character-level model proposed by (Zhang et al., 2015). 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 a word-level model proposed by (Kim, 2014). 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 both 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.3.2. 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 taggings are also matched.

A.3.3. 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 $z$ by uniformly sampling one string $z_1$ from a string transformation $(T_1, \delta_1)$ and passing it to the next transformation $(T_2, \delta_2)$, where we then sample a new string $z_2$, and so on until we have exhausted all transformations. The objective function is the following:

$$\arg\min_\theta \mathbb{E}_{(x,y) \sim D} \left( \mathcal{L}(x, y, \theta) + \max_{z \in \mathbb{R}(x)} \mathcal{L}(z, y, \theta) \right)$$

HotFlip augmentation performs adversarial training using the HotFlip (Ebrahimi et al., 2018) attack to find $z$ 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., 2018) that uses a hyperparameter $\lambda$ to weigh between normal loss and maximization objective
in Eq. (2). We linearly increase the hyperparameter $\lambda$ from 0 to 0.75.

$$
\arg\min_{\theta} \mathbb{E}_{(x,y) \sim \mathcal{D}} ((1-\lambda)\mathcal{L}(x, y, \theta) +
\lambda \max_{z \in \text{aug}(\mathcal{S}_{\text{aug}}, \mathcal{X})} \mathcal{L}(\text{abstract}(S_{\text{aug}}, z), y, \theta)).
$$

Also, we set $k$ in $\text{aug}_{k}$ to 2, which means we select 2 perturbed samples to abstract. We added the l2-regularization with a factor 0.005 to prevent over-fitting for AG, SST2 datasets.

A.3.4. Evaluation Results

**RQ2: Effects of size of the perturbation space**  In Figure 4, we fix the word-level model A3T (search) trained on $\{(T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$. Then, we test this model’s exhaustive accuracy on $\{(T_{\text{Dup}}, \delta_1), (T_{\text{SubSyn}}, 2)\}$ (Figure 4(a)) and $\{(T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, \delta_2)\}$ (Figure 4(b)), where we vary the parameters $\delta_1$ and $\delta_2$ between 1 and 4, increasing the size of the perturbation space. The exhaustive accuracy of A3T(HotFlip) and A3T(search) decreases by 12.5% and 10.4%, respectively, when increasing $\delta_1$ from 1 to 4, and decreases by 1.8% and 2.2%, respectively, when increasing $\delta_2$ from 1 to 4. All other techniques result in larger decreases in exhaustive accuracy (>20.4% in $\{(T_{\text{Dup}}, \delta_1), (T_{\text{SubSyn}}, 2)\}$ and >3.8% in $\{(T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, \delta_2)\}$).

In Figure 5, we fix the word-level model A3T (search) trained on $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$. Then, we test this model’s exhaustive accuracy on $\{(T_{\text{DelStop}}, \delta_1), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$ (Figure 5(a)), $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, \delta_2), (T_{\text{SubSyn}}, 2)\}$ (Figure 5(b)), and $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, \delta_3)\}$ (Figure 5(c)), where we vary the parameters $\delta_1$, $\delta_2$ and $\delta_3$ 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 0.6%, respectively, when increasing $\delta_1$ from 1 to 3, decreases by 10.6% and 6.9%, respectively, when increasing $\delta_2$ from 1 to 3, and decreases by 2.2% and 1.5%, respectively, when increasing $\delta_3$ from 1 to 3. All other techniques result in larger decreases in exhaustive accuracy (>2.8% in $\{(T_{\text{DelStop}}, \delta_1), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$, >13.8% in $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, \delta_2), (T_{\text{SubSyn}}, 2)\}$, and >3.8% in $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, \delta_3)\}$. 

**Results on AG and SST2**  In Figure 6, we fix the word-level model A3T (search) trained on $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$. Then, we test this model’s exhaustive accuracy on $\{(T_{\text{DelStop}}, \delta_1), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$ (Figure 6(a)), $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, \delta_2), (T_{\text{SubSyn}}, 2)\}$ (Figure 6(b)), and $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, \delta_3)\}$ (Figure 6(c)), where we vary the parameters $\delta_1$, $\delta_2$ and $\delta_3$ between 1 and 3, increasing the size of the perturbation space. The exhaustive accuracy of A3T(HotFlip) and A3T(search) decreases by 0.9% and 0.4%, respectively, when increasing $\delta_1$ from 1 to 3, decreases by 10.3% and 5.7%, respectively, when increasing $\delta_2$ from 1 to 3, and decreases by 3.5% and 2.7%, respectively, when increasing $\delta_3$ from 1 to 3. All other techniques result in larger decreases in exhaustive accuracy (>2.2% in $\{(T_{\text{DelStop}}, \delta_1), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$, >14.9% in $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, \delta_2), (T_{\text{SubSyn}}, 2)\}$, and >3.7% in $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, \delta_3)\}$. 

**Results on Rte2**  In Figure 7, we fix the word-level model A3T (search) trained on $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$. Then, we test this model’s exhaustive accuracy on $\{(T_{\text{DelStop}}, \delta_1), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$ (Figure 7(a)), $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, \delta_2), (T_{\text{SubSyn}}, 2)\}$ (Figure 7(b)), and $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, \delta_3)\}$ (Figure 7(c)), where we vary the parameters $\delta_1$, $\delta_2$ and $\delta_3$ between 1 and 3, increasing the size of the perturbation space. The exhaustive accuracy of A3T(HotFlip) and A3T(search) decreases by 0.7% and 0.2%, respectively, when increasing $\delta_1$ from 1 to 3, decreases by 10.0% and 5.1%, respectively, when increasing $\delta_2$ from 1 to 3, and decreases by 2.5% and 1.4%, respectively, when increasing $\delta_3$ from 1 to 3. All other techniques result in larger decreases in exhaustive accuracy (>2.4% in $\{(T_{\text{DelStop}}, \delta_1), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, 2)\}$, >15.2% in $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, \delta_2), (T_{\text{SubSyn}}, 2)\}$, and >3.7% in $\{(T_{\text{DelStop}}, 2), (T_{\text{Dup}}, 2), (T_{\text{SubSyn}}, \delta_3)\}$.
Robustness to String Transformations via Augmented Abstract Training

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