Certified Robustness to Adversarial Word Substitutions

Robin Jia **Aditi Raghunathan** **Kerem Göksel** **Percy Liang**
Computer Science Department, Stanford University
{robinjia,aditir,kerem,pliang}@cs.stanford.edu

Abstract

State-of-the-art NLP models can often be fooled by adversaries that apply seemingly innocuous label-preserving transformations (e.g., paraphrasing) to input text. The number of possible transformations scales exponentially with text length, so data augmentation cannot cover all transformations of an input. This paper considers one exponentially large family of label-preserving transformations, in which every word in the input can be replaced with a similar word. We train the first models that are provably robust to *all* word substitutions in this family. Our training procedure uses Interval Bound Propagation (IBP) to minimize an upper bound on the worst-case loss that any combination of word substitutions can induce. To evaluate models' robustness to these transformations, we measure accuracy on adversarially chosen word substitutions applied to test examples. Our IBP-trained models attain 75% adversarial accuracy on both sentiment analysis on IMDB and natural language inference on SNLI. In comparison, on IMDB, models trained normally and ones trained with data augmentation achieve adversarial accuracy of only 8% and 35%, respectively.

1 Introduction

Machine learning models have achieved impressive accuracy on many NLP tasks, but they are surprisingly brittle. Adding distracting text to the input (Jia and Liang, 2017), paraphrasing the text (Iyyer et al., 2018; Ribeiro et al., 2018), replacing words with similar words (Alzantot et al., 2018), or inserting character-level "typos" (Belinkov and Bisk, 2017; Ebrahimi et al., 2017) can significantly degrade a model's performance. Such perturbed inputs are called *adversarial examples*, and have shown to break models in other domains as well, most notably in vision (Szegedy et al., 2014; Goodfellow et al., 2015). Since humans are not fooled by the same perturbations, the widespread existence of adversarial examples exposes troubling gaps in models' understanding.

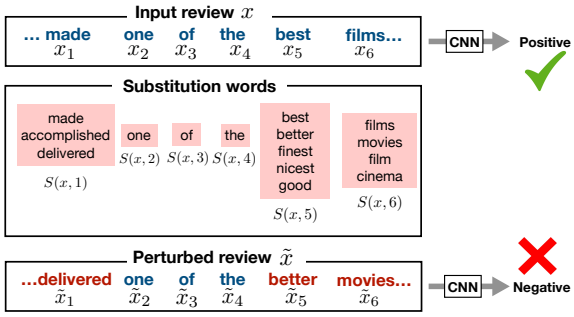

Figure 1: Word substitution-based perturbations in sentiment analysis. For an input x, we consider perturbations $\tilde{x}$, in which *every* word x_i can be replaced with any similar word from the set $S(x, i)$, without changing the original sentiment. Models can be easily fooled by adversarially chosen perturbations (e.g., changing "*best*" to "*better*", "*made*" to "*delivered*", "*films*" to "*movies*"), but the ideal model would be robust to all combinations of word substitutions.

In this paper, we focus on the word substitution perturbations of Alzantot et al. (2018). In this setting, an attacker may replace every word in the input with a similar word (that ought not to change the label), leading to an exponentially large number of possible perturbations. Figure 1 shows an example of these word substitutions. As demonstrated by a long line of work in computer vision, it is challenging to make models that are robust to very large perturbation spaces, even when the set of perturbations is known at training time (Goodfellow et al., 2015; Athalye et al., 2018; Raghunathan et al., 2018; Wong and Kolter, 2018).

Our paper addresses two key questions. First, is it possible to guarantee that a model is robust against *all* adversarial perturbations of a given in-

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 4129–4142, Hong Kong, China, November 3–7, 2019.

put? Existing methods that use heuristic search to attack models (Ebrahimi et al., 2017; Alzantot et al., 2018) are slow and cannot provide guarantees of robustness, since the space of possible perturbations is too large to search exhaustively. We obtain guarantees by leveraging Interval Bound Propagation (IBP), a technique that was previously applied to feedforward networks and CNNs in computer vision (Dvijotham et al., 2018). IBP efficiently computes a tractable *upper bound* on the loss of the worst-case perturbation. When this upper bound on the worst-case loss is small, the model is guaranteed to be robust to all perturbations, providing a *certificate* of robustness. To apply IBP to NLP settings, we derive new interval bound formulas for multiplication and softmax layers, which enable us to compute IBP bounds for LSTMs (Hochreiter and Schmidhuber, 1997) and attention layers (Bahdanau et al., 2015). We also extend IBP to handle discrete perturbation sets, rather than the continuous ones used in vision.

Second, can we train models that are robust in this way? Data augmentation can sometimes mitigate the effect of adversarial examples (Jia and Liang, 2017; Belinkov and Bisk, 2017; Ribeiro et al., 2018; Liu et al., 2019), but it is insufficient when considering very large perturbation spaces (Alzantot et al., 2018). Adversarial training strategies from computer vision (Madry et al., 2018) rely on gradient information, and therefore do not extend to the discrete perturbations seen in NLP. We instead use *certifiably robust training*, in which we train models to optimize the IBP upper bound (Dvijotham et al., 2018).

We evaluate certifiably robust training on two tasks—sentiment analysis on the IMDB dataset (Maas et al., 2011) and natural language inference on the SNLI dataset (Bowman et al., 2015). Across various model architectures (bag-of-words, CNN, LSTM, and attention-based), certifiably robust training consistently yields models which are provably robust to all perturbations on a large fraction of test examples. A normally-trained model has only 8% and 41% accuracy on IMDB and SNLI, respectively, when evaluated on adversarially perturbed test examples. With certifiably robust training, we achieve 75% adversarial accuracy for both IMDB and SNLI. Data augmentation fares much worse than certifiably robust training, with adversarial accuracies falling to 35% and 71%, respectively.

2 Setup

We consider tasks where a model must predict a label $y \in \mathcal{Y}$ given textual input $x \in \mathcal{X}$. For example, for sentiment analysis, the input x is a sequence of words $x_1, x_2, \ldots, x_L$, and the goal is to assign a label $y \in \{-1, 1\}$ denoting negative or positive sentiment, respectively. We use $z = (x, y)$ to denote an example with input x and label y, and use θ to denote parameters of a model. Let $f(z, \theta) \in \mathbb{R}$ denote some loss of a model with parameters θ on example z. We evaluate models on $f^{\text{0-1}}(z, \theta)$, the zero-one loss under model θ.

2.1 Perturbations by word substitutions

Our goal is to build models that are robust to label-preserving perturbations. In this work, we focus on perturbations where words of the input are substituted with similar words. Formally, for every word x_i, we consider a set of allowed substitution words $S(x, i)$, including x_i itself. We use $\tilde{x}$ to denote a perturbed version of x, where each word $\tilde{x}_i$ is in $S(x, i)$. For an example $z = (x, y)$, let $B_{\text{perturb}}(z)$ denote the set of *all* allowed perturbations of z:

$$B_{\text{perturb}}(z) = \{(\tilde{x}, y) : \tilde{x}_i \in S(x, i) \ \ \forall i\}. \quad (1)$$

Figure 1 provides an illustration of word substitution perturbations. We choose $S(x, i)$ so that $\tilde{x}$ is likely to be grammatical and have the same label as x (see Section 5.1).

2.2 Robustness to all perturbations

Let $\mathcal{F}(z, \theta)$ denote the set of losses of the network on the set of perturbed examples defined in (1):

$$\mathcal{F}(z, \theta) = \{f(\tilde{z}, \theta) : \tilde{z} \in B_{\text{perturb}}(z)\}. \quad (2)$$

We define the *robust loss* as $\max \mathcal{F}(z, \theta)$, the loss due to worst-case perturbation. A model is robust at z if it classifies all inputs in the perturbation set correctly, i.e., the robust zero-one loss $\max \mathcal{F}^{\text{0-1}}(z, \theta) = 0$. Unfortunately, the robust loss is often intractable to compute, as each word can be perturbed independently. For example, reviews in the IMDB dataset (Maas et al., 2011) have a median of 10^{31} possible perturbations and max of 10^{271}, far too many to enumerate. We instead propose a tractable *upper bound* by constructing a set $\mathcal{O}(z, \theta) \supseteq \mathcal{F}(z, \theta)$. Note that

$$\max \mathcal{O}^{\text{0-1}}(z, \theta) = 0 \Rightarrow \max \mathcal{F}^{\text{0-1}}(z, \theta) = 0 \Leftrightarrow \text{robust at } z. \quad (3)$$

Therefore, whenever $\max \mathcal{O}^{0\text{-}1}(z, \theta) = 0$, this fact is sufficient to *certify* robustness to all perturbed examples $B_{\text{perturb}}(z)$. However, since $\mathcal{O}^{0\text{-}1}(z, \theta) \supseteq \mathcal{F}^{0\text{-}1}(z, \theta)$, the model could be robust even if $\max \mathcal{O}^{0\text{-}1}(z, \theta) \neq 0$.

3 Certification via Interval Bound Propagation

We now show how to use Interval Bound Propagation (IBP) (Dvijotham et al., 2018) to obtain a superset $\mathcal{O}(z, \theta)$ of the losses of perturbed inputs $\mathcal{F}(z, \theta)$, given z, θ, and $B_{\text{perturb}}(z)$. For notational convenience, we drop z and θ. The key idea is to compute upper and lower bounds on the activations in each layer of the network, in terms of bounds computed for previous layers. These bounds *propagate* through the network, as in a standard forward pass, until we obtain bounds on the final output, i.e., the loss f. While IBP bounds may be loose in general, Section 5.2 shows that training networks to minimize the upper bound on f makes these bounds much tighter (Gowal et al., 2018; Raghunathan et al., 2018).

Formally, let g^i denote a scalar-valued function of z and θ (e.g., a single activation in one layer of the network) computed at node i of the computation graph for a given network. Let $\text{dep}(i)$ be the set of nodes used to compute g^i in the computation graph (e.g., activations of the previous layer). Let $\mathcal{G}^i$ denote the set of possible values of g^i across all examples in $B_{\text{perturb}}(z)$. We construct an interval $\mathcal{O}^i = [\ell^i, u^i]$ that contains all these possible values of g^i, i.e., $\mathcal{O}^i \supseteq \mathcal{G}^i$. $\mathcal{O}^i$ is computed from the intervals $\mathcal{O}^{\text{dep}(i)} = \{\mathcal{O}^j : j \in \text{dep}(i)\}$ of the dependencies of g^i. Once computed, $\mathcal{O}^i$ can then be used to compute intervals on nodes that depend on i. In this way, bounds propagate through the entire computation graph in an efficient forward pass.

We now discuss how to compute interval bounds for NLP models and word substitution perturbations. We obtain interval bounds for model inputs given $B_{\text{perturb}}(z)$ (Section 3.1), then show how to compute $\mathcal{O}^i$ from $\mathcal{O}^{\text{dep(i)}}$ for elementary operations used in standard NLP models (Section 3.2). Finally, we use these bounds to certify robustness and train robust models.

3.1 Bounds for the input layer

Previous work (Gowal et al., 2018) applied IBP to continuous image perturbations, which are naturally represented with interval bounds (Dvijotham et al., 2018). We instead work with discrete word substitutions, which we must convert into interval bounds $\mathcal{O}^{\text{input}}$ in order to use IBP. Given input words $x = x_1, \ldots, x_L$, we assume that the model embeds each word as $g^{\text{input}} = [\phi(x_1), \ldots, \phi(x_L)] \in \mathbb{R}^{L \times d}$, where $\phi(x_i) \in \mathbb{R}^d$ is the word vector for word x_i. To compute $\mathcal{O}^{\text{input}} \supseteq \mathcal{G}^{\text{input}}$, recall that each input word x_i can be replaced with any $\tilde{x}_i \in S(x, i)$. So, for each coordinate $j \in \{1, \ldots, d\}$, we can obtain an interval bound $\mathcal{O}^{\text{input}}_{ij} = [\ell^{\text{input}}_{ij}, u^{\text{input}}_{ij}]$ for g^{input}_{ij} by computing the smallest axis-aligned box that contains all the word vectors:

$$\ell^{\text{input}}_{ij} = \min_{w \in S(x,i)} \phi(w)_j, \quad u^{\text{input}}_{ij} = \max_{w \in S(x,i)} \phi(w)_j. \tag{4}$$

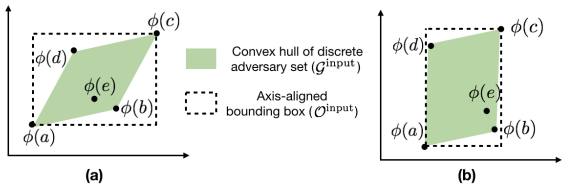

Figure 2: Bounds on the word vector inputs to the neural network. Consider a word (sentence of length one) $x = a$ with the set of substitution words $S(x, 1) = \{a, b, c, d, e\}$. (a) IBP constructs axis-aligned bounds around a set of word vectors. These bounds may be loose, especially if the word vectors are pre-trained and fixed. (b) A different word vector space can give tighter IBP bounds, if the convex hull of the word vectors is better approximated by an axis-aligned box.

Figure 2 illustrates these bounds. We can view this as relaxing a set of discrete points to a convex set that contains all of the points. Section 4.2 discusses modeling choices to make this box tighter.

3.2 Interval bounds for elementary functions

Next, we describe how to compute the interval of a node i from intervals of its dependencies. Gowal et al. (2018) show how to efficiently compute interval bounds for affine transformations (i.e., linear layers) and monotonic elementwise nonlinearities (see Appendix 3). This suffices to compute interval bounds for feedforward networks and CNNs. However, common NLP model components like LSTMs and attention also rely on softmax (for attention), element-wise multiplication (for LSTM gates), and dot product (for computing attention scores). We show how to compute interval bounds for these new operations. These building blocks can be used to compute interval bounds

not only for LSTMs and attention, but also for any model that uses these elementary functions.

For ease of notation, we drop the superscript i on g^i and write that a node computes a result $z^{\text{res}} = g(z^{\text{dep}})$ where $z^{\text{res}} \in \mathbb{R}$ and $z^{\text{dep}} \in \mathbb{R}^m$ for $m = |\text{dep}(i)|$. We are given intervals $\mathcal{O}^{\text{dep}}$ such that $z_j^{\text{dep}} \in \mathcal{O}_j^{\text{dep}} = [\ell_j^{\text{dep}}, u_j^{\text{dep}}]$ for each coordinate j and want to compute $\mathcal{O}^{\text{res}} = [\ell^{\text{res}}, u^{\text{res}}]$.

Softmax layer. The softmax function is often used to convert activations into a probability distribution, e.g., for attention. Gowal et al. (2018) uses unnormalized logits and does not handle softmax operations. Formally, let z^{res} represent the normalized score of the word at position c. We have $z^{\text{res}} = \frac{\exp(z_c^{\text{dep}})}{\sum_{j=1}^m \exp(z_j^{\text{dep}})}$. The value of z^{res} is largest when z_c^{dep} takes its largest value and all other words take the smallest value:

$$u^{\text{res}} = \frac{\exp(u_c^{\text{dep}})}{\exp(u_c^{\text{dep}}) + \sum_{j \neq c} \exp(\ell_j^{\text{dep}})}. \quad (5)$$

We obtain a similar expression for ℓ^{res}. Note that ℓ^{res} and u^{res} can each be computed in a forward pass, with some care taken to avoid numerical instability (see Appendix A.2).

Element-wise multiplication and dot product. Models like LSTMs incorporate gates which perform element-wise multiplication of two activations. Let $z^{\text{res}} = z_1^{\text{dep}} z_2^{\text{dep}}$ where $z^{\text{res}}, z_1^{\text{dep}}, z_2^{\text{dep}} \in \mathbb{R}$. The extreme values of the product occur at one of the four points corresponding to the products of the extreme values of the inputs. In other words,

$$\begin{aligned} \mathcal{C} = \{&\ell_1^{\text{dep}} \ell_2^{\text{dep}}, \quad \ell_1^{\text{dep}} u_2^{\text{dep}} \\ &u_1^{\text{dep}} \ell_2^{\text{dep}}, \quad u_1^{\text{dep}} u_2^{\text{dep}}\} \\ \ell^{\text{res}} = \min(\mathcal{C}) \quad & u^{\text{res}} = \max(\mathcal{C}). \end{aligned} \quad (6)$$

Propagating intervals through multiplication nodes therefore requires four multiplications.

Dot products between activations are often used to compute attention scores.[1] The dot product $(z_1^{\text{dep}})^\top z_2^{\text{dep}}$ is just the sum of the element-wise product $z_1^{\text{dep}} \odot z_2^{\text{dep}}$. Therefore, we can bound the dot product by summing the bounds on each element of $z_1^{\text{dep}} \odot z_2^{\text{dep}}$, using the formula for element-wise multiplication.

[1]This is distinct from an affine transformation, because both vectors have associated bounds; in an affine layer, the input has bounds, but the weight matrix is fixed.

3.3 Final layer

Classification models typically output a single logit for binary classification, or k logits for k-way classification. The final loss $f(z, \theta)$ is a function of the logits $s(x)$. For standard loss functions, we can represent this function in terms of element-wise monotonic functions (Appendix 3) and the elementary functions described in Section 3.2.

1. Zero-one loss: $f(z, \theta) = \mathbb{I}[\max(s(x)) = y]$ involves a max operation followed by a step function, which is monotonic.

2. Cross entropy: For multi-class, $f(z, \theta) = \text{softmax}(s(x))$. In the binary case, $f(z, \theta) = \sigma(s(x))$, where the sigmoid function σ is monotonic.

Thus, we can compute bounds on the loss $\mathcal{O}(z, \theta) = [\ell^{\text{final}}, u^{\text{final}}]$ from bounds on the logits.

3.4 Certifiably Robust Training with IBP

Finally, we describe certifiably robust training, in which we encourage robustness by minimizing the upper bound on the worst-case loss (Dvijotham et al., 2018; Gowal et al., 2018). Recall that for an example z and parameters θ, $u^{\text{final}}(z, \theta)$ is the upper bound on the loss $f(z, \theta)$. Given a dataset D, we optimize a weighted combination of the normal loss and the upper bound u^{final},

$$\min_\theta \sum_{z \in D} (1 - \kappa) f(z, \theta) + \kappa\, u^{\text{final}}(z, \theta), \quad (7)$$

where $0 \leq \kappa \leq 1$ is a scalar hyperparameter.

As described above, we compute u^{final} in a modular fashion: each layer has an accompanying function that computes bounds on its outputs given bounds on its inputs. Therefore, we can easily apply IBP to new architectures. Bounds propagate through layers via forward passes, so the entire objective (7) can be optimized via backpropagation.

Gowal et al. (2018) found that this objective was easier to optimize by starting with a smaller space of allowed perturbations, and make it larger during training. We accomplish this by artificially shrinking the input layer intervals $\mathcal{O}_{ij}^{\text{input}} = [\ell_{ij}^{\text{input}}, u_{ij}^{\text{input}}]$ towards the original value $\phi(x_i)_j$ by a factor of ϵ:

$$\begin{aligned} \ell_{ij}^{\text{input}} &\leftarrow \phi(x_i)_j - \epsilon(\phi(x_i)_j - \ell_{ij}^{\text{input}}) \\ u_{ij}^{\text{input}} &\leftarrow \phi(x_i)_j + \epsilon(u_{ij}^{\text{input}} - \phi(x_i)_j). \end{aligned}$$

Standard training corresponds to $\epsilon = 0$. We train for T^{init} epochs while linearly increasing ϵ from 0

to 1, and also increasing κ from 0 up to a maximum value of $\kappa^\star$, We then train for an additional T^{final} epochs at $\kappa = \kappa^\star$ and $\epsilon = 1$.

To summarize, we use IBP to compute an upper bound on the model's loss when given an adversarially perturbed input. This bound is computed in a modular fashion. We efficiently train models to minimize this bound via backpropagation.

4 Tasks and models

Now we describe the tasks and model architectures on which we run experiments. These models are all built from the primitives in Section 3.

4.1 Tasks

Following Alzantot et al. (2018), we evaluate on two standard NLP datasets: the IMDB sentiment analysis dataset (Maas et al., 2011) and the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015). For IMDB, the model is given a movie review and must classify it as positive or negative. For SNLI, the model is given two sentences, a premise and a hypothesis, and is asked whether the premise entails, contradicts, or is neutral with respect to the hypothesis. For SNLI, the adversary is only allowed to change the hypothesis, as in Alzantot et al. (2018), though it is possible to also allow changing the premise.

4.2 Models

IMDB. We implemented three models for IMDB. The bag-of-words model (BOW) averages the word vectors for each word in the input, then passes this through a two-layer feedforward network with 100-dimensional hidden state to obtain a final logit. The other models are similar, except they run either a CNN or bidirectional LSTM on the word vectors, then average their hidden states. All models are trained on cross entropy loss.

SNLI We implemented two models for SNLI. The bag-of-words model (BOW) encodes the premise and hypothesis separately by summing their word vectors, then feeds the concatenation of these encodings to a 3-layer feedforward network. We also reimplement the Decomposable Attention model (Parikh et al., 2016), which uses attention between the premise and hypothesis to compute richer representations of each word in both sentences. These context-aware vectors are used in the same way BOW uses the original word vectors to generate the final prediction. Both models are trained on cross entropy loss. Implementation details are provided in Appendix A.4.

Word vector layer. The choice of word vectors affects the tightness of our interval bounds. We choose to define the word vector $\phi(w)$ for word w as the output of a feedforward layer applied to a fixed pre-trained word vector $\phi^{\text{pre}}(w)$:

$$\phi(w) = \text{ReLU}(g^{\text{word}}(\phi^{\text{pre}}(w))), \qquad (8)$$

where g^{word} is a learned linear transformation. Learning g^{word} with certifiably robust training encourages it to orient the word vectors so that the convex hull of the word vectors is close to an axis-aligned box. Note that g^{word} is applied *before* bounds are computed via (4).[2] Applying g^{word} after the bound calculation would result in looser interval bounds, since the original word vectors $\phi^{\text{pre}}(w)$ might be poorly approximated by interval bounds (e.g., Figure 2a), compared to $\phi(w)$ (e.g., Figure 2b). Section 5.7 confirms the importance of adding g^{word}. We use 300-dimensional GloVe vectors (Pennington et al., 2014) as our $\phi^{\text{pre}}(w)$.

5 Experiments

5.1 Setup

Word substitution perturbations. We base our sets of allowed word substitutions $S(x, i)$ on the substitutions allowed by Alzantot et al. (2018). They demonstrated that their substitutions lead to adversarial examples that are qualitatively similar to the original input and retain the original label, as judged by humans. Alzantot et al. (2018) define the neighbors $N(w)$ of a word w as the $n = 8$ nearest neighbors of w in a "counter-fitted" word vector space where antonyms are far apart (Mrkšić et al., 2016).[3] The neighbors must also lie within some Euclidean distance threshold. They also use a language model constraint to avoid nonsensical perturbations: they allow substituting x_i with $\tilde{x}_i \in N(x_i)$ if and only if it does not decrease the log-likelihood of the text under a pre-trained language model by more than some threshold.

We make three modifications to this approach. First, in Alzantot et al. (2018), the adversary applies substitutions one at a time, and the neighborhoods and language model scores are computed

[2] Equation (4) must be applied before the model can combine information from multiple words, but it can be delayed until after processing each word independently.

[3] Note that the model itself classifies using a different set of pre-trained word vectors; the counter-fitted vectors are only used to define the set of allowed substitution words.

relative to the current altered version of the input. This results in a hard-to-define attack surface, as changing one word can allow or disallow changes to other words. It also requires recomputing language model scores at each iteration of the genetic attack, which is inefficient. Moreover, the same word can be substituted multiple times, leading to semantic drift. We define allowed substitutions relative to the original sentence x, and disallow repeated substitutions. Second, we use a faster language model that allows us to query longer contexts; Alzantot et al. (2018) use a slower language model and could only query it with short contexts. Finally, we use the language model constraint only at test time; the model is trained against all perturbations in $N(w)$. This encourages the model to be robust to a larger space of perturbations, instead of specializing for the particular choice of language model. See Appendix A.3 for further details.

Analysis of word neighbors. One natural question is whether we could guarantee robustness by having the model treat all neighboring words the same. We could construct equivalence classes of words from the transitive closure of $N(w)$, and represent each equivalence class with one embedding. We found that this would lose a significant amount of information. Out of the 50,000 word vocabulary, 19,122 words would be in the same equivalence class, including the words "good", "bad", "excellent", and "terrible." Of the remaining words, 24,389 (79%) have no neighbors.

Baseline training methods. We compare certifiably robust training (Section 3) with both standard training and data augmentation, which has been used in NLP to encourage robustness to various types of perturbations (Jia and Liang, 2017; Belinkov and Bisk, 2017; Iyyer et al., 2018; Ribeiro et al., 2018). In data augmentation, for each training example z, we augment the dataset with K new examples $\tilde{z}$ by sampling $\tilde{z}$ uniformly from $B_{\text{perturb}}(z)$, then train on the normal cross entropy loss. For our main experiments, we use $K = 4$. We do not use adversarial training (Goodfellow et al., 2015) because it would require running an adversarial search procedure at each training step, which would be prohibitively slow.

Evaluation of robustness. We wish to evaluate robustness of models to all word substitution perturbations. Ideally, we would directly measure *robust accuracy*, the fraction of test examples z for which the model is correct on all $\tilde{z} \in B_{\text{perturb}}(z)$. However, evaluating this exactly involves enumerating the exponentially large set of perturbations, which is intractable. Instead, we compute tractable upper and lower bounds:

1. Genetic attack accuracy: Alzantot et al. (2018) demonstrate the effectiveness of a genetic algorithm that searches for perturbations $\tilde{z}$ that cause model misclassification. The algorithm maintains a "population" of candidate $\tilde{z}$'s and repeatedly perturbs and combines them. We used a population size of 60 and ran 40 search iterations on each example. Since the algorithm does not exhaustively search over $B_{\text{perturb}}(z)$, accuracy on the perturbations it finds is an *upper bound* on the true robust accuracy.

2. Certified accuracy: To complement this upper bound, we use IBP to obtain a tractable lower bound on the robust accuracy. Recall from Section 3.3 that we can use IBP to get an upper bound on the zero-one loss. From this, we obtain a *lower bound* on the robust accuracy by measuring the fraction of test examples for which the zero-one loss is guaranteed to be 0.

Experimental details. For IMDB, we split the official train set into train and development subsets, putting reviews for different movies into different splits (matching the original train/test split). For SNLI, we use the official train/development/test split. We tune hyperparameters on the development set for each dataset. Hyperparameters are reported in Appendix A.4.

5.2 Main results

Table 1 and Table 2 show our main results for IMDB and SNLI, respectively. We measure accuracy on perturbations found by the genetic attack (upper bound on robust accuracy) and IBP-certified accuracy (lower bound on robust accuracy) on 1000 random test examples from IMDB,[4] and all 9824 test examples from SNLI. Across many architectures, our models are more robust to perturbations than ones trained with data augmentation. This effect is especially pronounced on IMDB, where inputs can be hundreds of words long, so many words can be perturbed. On IMDB, the best IBP-trained model gets 75.0% accuracy on perturbations found by the genetic at-

[4] We downsample the test set because the genetic attack is slow on IMDB, as inputs can be hundreds of words long.

System	Genetic attack (Upper bound)	IBP-certified (Lower bound)
Standard training		
BOW	9.6	0.8
CNN	7.9	0.1
LSTM	6.9	0.0
Robust training		
BOW	70.5	68.9
CNN	**75.0**	**74.2**
LSTM	64.7	63.0
Data augmentation		
BOW	34.6	3.5
CNN	35.2	0.3
LSTM	33.0	0.0

Table 1: Robustness of models on IMDB. We report accuracy on perturbations obtained via the genetic attack (upper bound on robust accuracy), and certified accuracy obtained using IBP (lower bound on robust accuracy) on 1000 random IMDB test set examples. For all models, robust training vastly outperforms data augmentation ($p < 10^{-63}$, Wilcoxon signed-rank test).

System	Genetic attack (Upper bound)	IBP-certified (Lower bound)
Normal training		
BOW	40.5	2.3
DECOMPATTN	40.3	1.4
Robust training		
BOW	**75.0**	**72.7**
DECOMPATTN	73.7	72.4
Data augmentation		
BOW	68.5	7.7
DECOMPATTN	70.8	1.4

Table 2: Robustness of models on the SNLI test set. For both models, robust training outperforms data augmentation ($p < 10^{-10}$, Wilcoxon signed-rank test).

tack, whereas the best data augmentation model gets 35.2%. Normally trained models are even worse, with adversarial accuracies below 10%.

Certified accuracy. Certifiably robust training yields models with tight guarantees on robustness—the upper and lower bounds on robust accuracy are close. On IMDB, the best model is *guaranteed* to be correct on all perturbations of 74.2% of test examples, very close to the 75.0% accuracy against the genetic attack. In contrast, for data augmentation models, the IBP bound cannot guarantee robustness on almost all examples. It is possible that a stronger attack (e.g., exhaustive search) could further lower the accuracy of these models, or that the IBP bounds are loose.

LSTM models can be certified with IBP, though they fare worse than other models. IBP bounds may be loose for RNNs because of their long computation paths, along which looseness of bounds can get amplified. Nonetheless, in Appendix A.7, we show on synthetic data that robustly trained LSTMs can learn long-range dependencies.

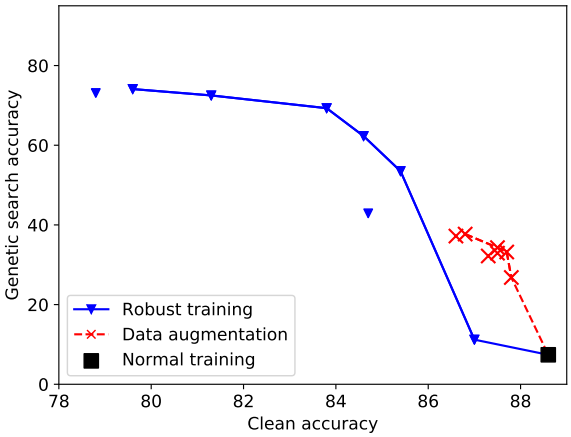

Figure 3: Trade-off between clean accuracy and genetic attack accuracy for CNN models on IMDB. Data augmentation cannot achieve high robustness. Certifiably robust training yields much more robust models, though at the cost of some clean accuracy. Lines connect Pareto optimal points for each training strategy.

5.3 Clean versus robust accuracy

Robust training does cause a moderate drop in clean accuracy (accuracy on unperturbed test examples) compared with normal training. On IMDB, our normally trained CNN model gets 89% clean accuracy, compared to 81% for the robustly trained model. We also see a drop on SNLI: the normally trained BOW model gets 83% clean accuracy, compared to 79% for the robustly trained model. Similar drops in clean accuracy are also seen for robust models in vision (Madry et al., 2017). For example, the state-of-the-art robust model on CIFAR10 (Zhang et al., 2019) only has 85% clean accuracy, but comparable normally-trained models get $> 96\%$ accuracy.

We found that the robustly trained models tend to underfit the training data—on IMDB, the CNN model gets only 86% clean training accuracy, lower than the *test* accuracy of the normally trained model. The model continued to underfit when we increased either the depth or width of the network. One possible explanation is that the attack surface adds a lot of noise, though a large enough model should still be able to overfit the training set. Better optimization or a tighter way to compute bounds could also improve training accuracy. We leave further exploration to future work.

Next, we analyzed the trade-off between clean and robust accuracy by varying the importance

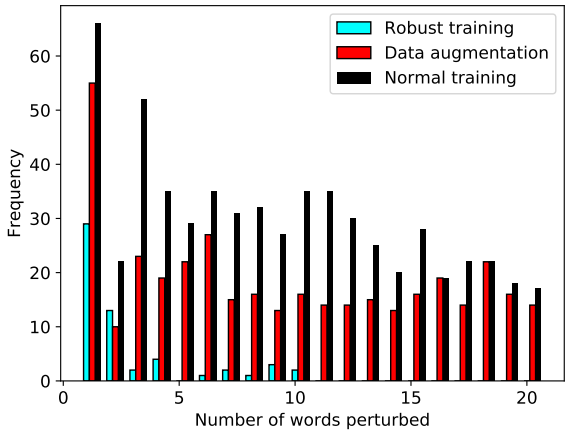

Figure 4: Number of words perturbed by the genetic attack to cause errors by CNN models on 1000 IMDB development set examples. Certifiably robust training reduces the effect of many simultaneous perturbations.

placed on perturbed examples during training. We use accuracy against the genetic attack as our proxy for robust accuracy, rather than IBP-certified accuracy, as IBP bounds may be loose for models that were not trained with IBP. For data augmentation, we vary K, the number of augmented examples per real example, from 1 to 64. For certifiably robust training, we vary $\kappa^\star$, the weight of the certified robustness training objective, between 0.01 and 1.0. Figure 3 shows trade-off curves for the CNN model on 1000 random IMDB development set examples. Data augmentation can increase robustness somewhat, but cannot reach very high adversarial accuracy. With certifiably robust training, we can trade off some clean accuracy for much higher robust accuracy.

5.4 Runtime considerations

IBP enables efficient computation of $u^{\text{final}}(z, \theta)$, but it still incurs some overhead. Across model architectures, we found that one epoch of certifiably robust training takes between $2\times$ and $4\times$ longer than one epoch of standard training. On the other hand, IBP certificates are much faster to compute at test time than genetic attack accuracy. For the robustly trained CNN IMDB model, computing certificates on 1000 test examples took 5 seconds, while running the genetic attack on those same examples took over 3 hours.

5.5 Error analysis

We examined development set examples on which models were correct on the original input but incorrect on the perturbation found by the genetic attack. We refer to such cases as *robustness errors*. We focused on the CNN IMDB models trained normally, robustly, and with data augmentation. We found that robustness errors of the robustly trained model mostly occurred when it was not confident in its original prediction. The model had $> 70\%$ confidence in the correct class for the original input in only 14% of robustness errors. In contrast, the normally trained and data augmentation models were more confident on their robustness errors; they had $> 70\%$ confidence on the original example in 92% and 87% of cases, respectively.

We next investigated how many words the genetic attack needed to change to cause misclassification, as shown in Figure 4. For the normally trained model, some robustness errors involved only a couple changed words (e.g., "*I've finally found a movie worse than . . .*" was classified negative, but the same review with "*I've finally* ***discovered*** *a movie worse than. . .*" was classified positive), but more changes were also common (e.g., part of a review was changed from "*The creature looked very cheesy*" to "*The creature* ***seemed supremely dorky***", with 15 words changed in total). Surprisingly, certifiably robust training nearly eliminated robustness errors in which the genetic attack had to change many words: the genetic attack either caused an error by changing a couple words, or was unable to trigger an error at all. In contrast, data augmentation is unable to cover the exponentially large space of perturbations that involve many words, so it does not prevent errors caused by changing many words.

5.6 Training schedule

We investigated the importance of slowly increasing ϵ during training, as suggested by Gowal et al. (2018). Fixing $\epsilon = 1$ during training led to a 5 point reduction in certified accuracy for the CNN. On the other hand, we found that holding κ fixed did not hurt accuracy, and in fact may be preferable. More details are shown in Appendix A.5.

5.7 Word vector analysis

We determined the importance of the extra feed-forward layer g^{word} that we apply to pre-trained word vectors, as described in Section 4.2. We compared with directly using pre-trained word vectors, i.e. $\phi(w) = \phi^{\text{pre}}(w)$. We also tried using g^{word} but applying interval bounds on $\phi^{\text{pre}}(w)$, then computing bounds on $\phi(w)$ with the IBP for-

mula for affine layers. In both cases, we could not train a CNN to achieve more than 52.2% certified accuracy on the development set. Thus, transforming pre-trained word vectors and applying interval bounds *after* is crucial for robust training. In Appendix A.6, we show that robust training makes the intervals around transformed word vectors smaller, compared to the pre-trained vectors.

6 Related Work and Discussion

Recent work on adversarial examples in NLP has proposed various classes of perturbations, such as insertion of extraneous text (Jia and Liang, 2017), word substitutions (Alzantot et al., 2018), paraphrasing (Iyyer et al., 2018; Ribeiro et al., 2018), and character-level noise (Belinkov and Bisk, 2017; Ebrahimi et al., 2017). These works focus mainly on demonstrating models' lack of robustness, and mostly do not explore ways to increase robustness beyond data augmentation. Data augmentation is effective for narrow perturbation spaces (Jia and Liang, 2017; Ribeiro et al., 2018), but only confers partial robustness in other cases (Iyyer et al., 2018; Alzantot et al., 2018). Ebrahimi et al. (2017) tried adversarial training (Goodfellow et al., 2015) for character-level perturbations, but could only use a fast heuristic attack at training time, due to runtime considerations. As a result, their models were still be fooled by running a more expensive search procedure at test time.

Provable defenses have been studied for simpler NLP models and attacks, particularly for tasks like spam detection where real-life adversaries try to evade detection. Globerson and Roweis (2006) train linear classifiers that are robust to adversarial feature deletion. Dalvi et al. (2004) analyzed optimal strategies for a Naive Bayes classifier and attacker, but their classifier only defends against a fixed attacker that does not adapt to the model.

Recent work in computer vision (Szegedy et al., 2014; Goodfellow et al., 2015) has sparked renewed interest in adversarial examples. Most work in this area focuses on L_∞-bounded perturbations, in which each input pixel can be changed by a small amount. The word substitution attack model we consider is similar to L_∞ perturbations, as the adversary can change each input word by a small amount. Our work is inspired by work based on convex optimization (Raghunathan et al., 2018; Wong and Kolter, 2018) and builds directly on interval bound propagation (Dvijotham et al., 2018; Gowal et al., 2018), which has certified robustness of computer vision models to L_∞ attacks. Adversarial training via projected gradient descent (Madry et al., 2018) has also been shown to improve robustness, but assumes that inputs are continuous. It could be applied in NLP by relaxing sets of word vectors to continuous regions.

This work provides certificates against word substitution perturbations for particular models. Since IBP is modular, it can be extended to other model architectures on other tasks. It is an open question whether IBP can give non-trivial bounds for sequence-to-sequence tasks like machine translation (Belinkov and Bisk, 2017; Michel et al., 2019). In principle, IBP can handle character-level typos (Ebrahimi et al., 2017; Pruthi et al., 2019), though typos yield more perturbations per word than we consider in this work. We are also interested in handling word insertions and deletions, rather than just substitutions. Finally, we would like to train models that get state-of-the-art clean accuracy while also being provably robust; achieving this remains an open problem.

In conclusion, state-of-the-art NLP models are accurate on average, but they still have significant blind spots. Certifiably robust training provides a general, principled mechanism to avoid such blind spots by encouraging models to make correct predictions on all inputs within some known perturbation neighborhood. This type of robustness is a necessary (but not sufficient) property of models that truly understand language. We hope that our work is a stepping stone towards models that are robust against an even wider, harder-to-characterize space of possible attacks.

Acknowledgments

This work was supported by NSF Award Grant no. 1805310 and the DARPA ASED program under FA8650-18-2-7882. R.J. is supported by an NSF Graduate Research Fellowship under Grant No. DGE-114747. A.R. is supported by a Google PhD Fellowship and the Open Philanthropy Project AI Fellowship. We thank Allen Nie for providing the pre-trained language model, and thank Peng Qi, Urvashi Khandelwal, Shiori Sagawa, and the anonymous reviewers for their helpful comments.

Reproducibility

All code, data, and experiments are available on Codalab at `https://bit.ly/2KVxIFN`.

References

M. Alzantot, Y. Sharma, A. Elgohary, B. Ho, M. Srivastava, and K. Chang. 2018. Generating natural language adversarial examples. In *Empirical Methods in Natural Language Processing (EMNLP)*.

A. Athalye, N. Carlini, and D. Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*.

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

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

S. Bowman, G. Angeli, C. Potts, and C. D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Empirical Methods in Natural Language Processing (EMNLP)*.

C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. 2004. Adversarial classification. In *International Conference on Knowledge Discovery and Data Mining (KDD)*.

K. Dvijotham, S. Gowal, R. Stanforth, R. Arandjelovic, B. O'Donoghue, J. Uesato, and P. Kohli. 2018. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*.

J. Ebrahimi, A. Rao, D. Lowd, and D. Dou. 2017. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751*.

A. Globerson and S. Roweis. 2006. Nightmare at test time: robust learning by feature deletion. In *International Conference on Machine Learning (ICML)*, pages 353–360.

I. J. Goodfellow, J. Shlens, and C. Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*.

S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, T. Mann, and P. Kohli. 2018. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*.

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

M. Iyyer, J. Wieting, K. Gimpel, and L. Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *North American Association for Computational Linguistics (NAACL)*.

R. Jia and P. Liang. 2017. Adversarial examples for evaluating reading comprehension systems. In *Empirical Methods in Natural Language Processing (EMNLP)*.

D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

N. F. Liu, R. Schwartz, and N. A. Smith. 2019. Inoculation by fine-tuning: A method for analyzing challenge datasets. In *North American Association for Computational Linguistics (NAACL)*.

A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. 2011. Learning word vectors for sentiment analysis. In *Association for Computational Linguistics (ACL)*.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. 2017. Towards deep learning models resistant to adversarial attacks (published at ICLR 2018). *arXiv*.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*.

P. Michel, X. Li, G. Neubig, and J. M. Pino. 2019. On evaluation of adversarial perturbations for sequence-to-sequence models. In *North American Association for Computational Linguistics (NAACL)*.

N. Mrkšić, D. Ó Séaghdha, B. Thomson, M. Gašić, L. Rojas-Barahona, P. Su, D. Vandyke, T. Wen, and S. Young. 2016. Counter-fitting word vectors to linguistic constraints. In *North American Association for Computational Linguistics (NAACL)*.

A. Parikh, O. Täckström, D. Das, and J. Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Empirical Methods in Natural Language Processing (EMNLP)*.

J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

D. Pruthi, B. Dhingra, and Z. C. Lipton. 2019. Combating adversarial misspellings with robust word recognition. In *Association for Computational Linguistics (ACL)*.

A. Raghunathan, J. Steinhardt, and P. Liang. 2018. Certified defenses against adversarial examples. In *International Conference on Learning Representations (ICLR)*.

M. T. Ribeiro, S. Singh, and C. Guestrin. 2018. Semantically equivalent adversarial rules for debugging NLP models. In *Association for Computational Linguistics (ACL)*.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*.

E. Wong and J. Z. Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*.

H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. 2019. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning (ICML)*.

A Supplemental material

A.1 Additional interval bound formulas

Gowal et al. (2018) showed how to compute interval bounds for affine transformations and monotonic element-wise functions. Here, we review their derivations, for completeness.

Affine transformations. Affine transformations are the building blocks of neural networks. Suppose $z^{\text{res}} = a^\top z^{\text{dep}} + b$ for weight $a \in \mathbb{R}^m$ and bias $b \in \mathbb{R}$. z^{res} is largest when positive entries of a are multiplied with u^{dep} and negative with ℓ^{dep}:

$$\begin{aligned} u^{\text{res}} &= \underbrace{0.5(a+|a|)^\top}_{\text{positive}} u^{\text{dep}} + \underbrace{0.5(a-|a|)^\top}_{\text{negative}} \ell^{\text{dep}} + b \\ &= \mu + r, \end{aligned} \tag{9}$$

where $\mu = 0.5a^\top(\ell^{\text{dep}} + u^{\text{dep}}) + b$ and $r = 0.5|a|^\top(u-l)$. A similar computation yields that $\ell^{\text{res}} = \mu - r$. Therefore, the interval $\mathcal{O}^{\text{res}}$ can be computed using two inner product evaluations: one with a and one with $|a|$.

Monotonic scalar functions. Activation functions such as ReLU, sigmoid and tanh are monotonic. Suppose $z^{\text{res}} = \sigma(z^{\text{dep}})$ where $z^{\text{res}}, z^{\text{dep}} \in \mathbb{R}$, i.e. the node applies an element-wise function to its input. The intervals can be computed trivially since z^{res} is minimized at ℓ^{dep} and maximized at u^{dep}.

$$\ell^{\text{res}} = \sigma(\ell^{\text{dep}}), \;\; u^{\text{res}} = \sigma(u^{\text{dep}}). \tag{10}$$

A.2 Numerical stability of softmax

In this section, we show how to compute interval bounds for softmax layers in a numerically stable way. We will do this by showing how to handle log-softmax layers. Note that since softmax is just exponentiated log-softmax, and exponentiation is monotonic, bounds on log-softmax directly yield bounds on softmax.

Let z^{dep} denote a vector of length m, let c be an integer $\in \{1, \ldots, m\}$, and let z^{res} represent the log-softmax score of index c, i.e.

$$z^{\text{res}} = \log \frac{\exp(z_c^{\text{dep}})}{\sum_{j=1}^m \exp(z_j^{\text{dep}})} \tag{11}$$

$$= z_c^{\text{dep}} - \log \sum_{j=1}^m \exp(z_j^{\text{dep}}). \tag{12}$$

Given interval bounds $\ell_j \le z_j^{\text{dep}} \le u_j$ for each j, we show how to compute upper and lower bounds on z^{res}. For any vector v, we assume access to a subroutine that computes

$$\text{logsumexp}(v) = \log \sum_i \exp(v_i)$$

stably. The standard way to compute this is to normalize v by subtracting $\max_i(v_i)$ before taking exponentials, then add it back at the end. logsumexp is a standard function in libraries like PyTorch. We will also rely on the fact that if v is the concatenation of vectors u and w, then $\text{logsumexp}(v) = \text{logsumexp}([\text{logsumexp}(u), \text{logsumexp}(w)])$.

Upper bound. The upper bound u^{res} is achieved by having the maximum value of z_c^{dep}, and minimum value of all others. This can be written as:

$$u^{\text{res}} = u_c^{\text{dep}} - \log \left(\exp(u_c^{\text{dep}}) + \sum_{1 \le j \le m, j \ne c} \exp(\ell_j^{\text{dep}}) \right). \tag{13}$$

While we could directly compute this expression, it is difficult to vectorize. Instead, with some rearranging, we get

$$u^{\text{res}} = u_c^{\text{dep}} - \log \left(\exp(u_c^{\text{dep}}) - \exp(\ell_c^{\text{dep}}) + \sum_{j=1}^m \exp(\ell_j^{\text{dep}}) \right). \tag{14}$$

The second term is the logsumexp of

$$\log \left(\exp(u_c^{\text{dep}}) - \exp(\ell_c^{\text{dep}}) \right) \tag{15}$$

and

$$\text{logsumexp}(\ell^{\text{dep}}). \tag{16}$$

Since we know how to compute $\mathrm{logsumexp}$, this reduces to computing (15). Note that (15) can be rewritten as

$$u_c^{\text{dep}} + \log\left(1 - \exp(\ell_c^{\text{dep}} - u_c^{\text{dep}})\right) \tag{17}$$

by adding and subtracting u_c^{dep}. To compute this quantity, we consider two cases:

1. $u_c^{\text{dep}} \gg \ell_c^{\text{dep}}$. Here we use the fact that stable methods exist to compute $\mathrm{log1p}(x) = \log(1+x)$ for x close to 0. We compute the desired value as

$$u_c^{\text{dep}} + \log 1p(-\exp(\ell_c^{\text{dep}} - u_c^{\text{dep}})),$$

since $\exp(\ell_c^{\text{dep}} - u_c^{\text{dep}})$ will be close to 0.

2. u_c^{dep} close to ℓ_c^{dep}. Here we use the fact that stable methods exist to compute $\mathrm{expm1}(x) = \exp(x) - 1$ for x close to 0. We compute the desired value as

$$u_c^{\text{dep}} + \log(-\mathrm{expm1}(\ell_c^{\text{dep}} - u_c^{\text{dep}})),$$

since $\ell_c^{\text{dep}} - u_c^{\text{dep}}$ may be close to 0.

We use case 1 if $u_c^{\text{dep}} - \ell_c^{\text{dep}} > \log 2$, and case 2 otherwise.[5]

Lower bound. The lower bound ℓ^{res} is achieved by having the minimum value of z_c^{dep}, and the maximum value of all others. This can be written as:

$$\ell^{\text{res}} = \ell_c^{\text{dep}} - \log\left(\exp(\ell_c^{\text{dep}}) + \sum_{1 \le j \le m, j \neq c} \exp(u_j^{\text{dep}})\right). \tag{18}$$

The second term is just a normal $\mathrm{logsumexp}$, which is easy to compute. To vectorize the implementation, it helps to first compute the $\mathrm{logsumexp}$ of everything except ℓ_c^{dep}, and then $\mathrm{logsumexp}$ that with ℓ_c^{dep}.

A.3 Attack surface differences

In Alzantot et al. (2018), the adversary applies replacements one at a time, and the neighborhoods and language model scores are computed relative to the current altered version of the input. This results in a hard-to-define attack surface, as the same word can be replaced many times, leading to semantic drift. We instead pre-compute the allowed substitutions $S(x, i)$ at index i based on the original x. We define $S(x, i)$ as the set of $\tilde{x}_i \in N(x_i)$ such that

$$\log P(x_{i-W:i-1}, \tilde{x}_i, x_{i+1:i+W}) \geq \log P(x_{i-W:i+W}) - \delta \tag{19}$$

where probabilities are assigned by a pre-trained language model, and the window radius W and threshold δ are hyperparameters. We use $W = 6$ and $\delta = 5$. We also use a different language model[6] from Alzantot et al. (2018) that achieves perplexity of 50.79 on the One Billion Word dataset (Chelba et al., 2013). Alzantot et al. (2018) use a different, slower language model, which compels them to use a smaller window radius of $W = 1$.

A.4 Experimental details

We do not run training for a set number of epochs but do early stopping on the development set instead. For normal training, we early stop on normal development set accuracy. For training with data augmentation, we early stop on the accuracy on the augmented development set. For certifiably robust training, we early stop on the certifiably robust accuracy on the development set. We use the Adam optimizer (Kingma and Ba, 2014) to train all models.

On IMDB, we restrict the model to only use the $50,000$ words that are in the vocabulary of the counter-fitted word vector space of Mrkšić et al. (2016). This is because perturbations are not allowed for any words not in this vocabulary, i.e. $N(w) = \{w\}$ for $w \notin V$. Therefore, the model is strongly incentivized to predict based on words outside of this set. While this is a valid way to achieve high certified accuracy, it is not a valid robustness strategy in general. We simply delete all words that are not in the vocabulary before feeding the input to the model.

For SNLI, we use 100-dimensional hidden state for the BOW model and a 3-layer feedforward network. These values were chosen by a hyperparameter search on the dev set. For DECOMPATTN, we use a 300-dimensional hidden state and a 2-layer feedforward network on top of the context-aware vectors. These values were chosen to match Parikh et al. (2016).

[5] See https://cran.r-project.org/web/packages/Rmpfr/vignettes/log1mexp-note.pdf for further explanation.

[6] https://github.com/windweller/l2w

System	κ	Learning Rate	Dropout Prob.	Weight Decay	Gradient Norm Clip Val.	T^{init}
IMDB, BOW	0.8	1×10^{-3}	0.2	1×10^{-4}	0.25	40
IMDB, CNN	0.8	1×10^{-3}	0.2	1×10^{-4}	0.25	40
IMDB, LSTM	0.8	1×10^{-3}	0.2	1×10^{-4}	0.25	20
SNLI, BOW	0.5	5×10^{-4}	0.1	1×10^{-4}	0.25	35
SNLI, DECOMPATTN	0.5	1×10^{-4}	0.1	0	0.25	50

Table 3: Training hyperparameters for training the models. The same hyperparameters were used for all training settings(plain, data augmentation, robust training)

Our implementation of the Decomposable Attention follows the original described in (Parikh et al., 2016) except for a few differences listed below;

- We do not normalize GloVe vectors to have norm 1.
- We do not hash out-of-vocabulary words to randomly generated vectors that we train, instead we omit them.
- We do randomly generate a null token vector that we then train. (Whether the null vector is trained is unspecified in the original paper).
- We use the Adam optimizer (with a learning rate of 1×10^{-4}) instead of AdaGrad.
- We use a batch size of 256 instead of 4.
- We use a dropout probability of 0.1 instead of 0.2
- We do not use the intra-sentence attention module.

A.5 Training schedule

In Table 4, we show the effect of holding ϵ or κ fixed during training, as described in Section 5.6. All numbers are on 1000 randomly chosen examples from the IMDB development set. Slowly increasing ϵ is important for good performance. Slowly increasing κ is actually slightly worse than holding $\kappa = \kappa^*$ fixed during training, despite earlier experiments we ran suggesting the opposite. Here we only report certified accuracy, as all models are trained with certifiably robust training, and certified accuracy is much faster to compute for development purposes.

A.6 Word vector bound sizes

To better understand the effect of g^{word}, we checked whether g^{word} made interval bound boxes around neighborhoods $N(w)$ smaller. For each

System	IBP-certified (Lower bound)
BOW	68.8
$\rightarrow$ Fixed ϵ	46.6
$\rightarrow$ Fixed κ	69.8
$\rightarrow$ Fixed ϵ and κ	66.3
CNN	72.5
$\rightarrow$ Fixed ϵ	67.6
$\rightarrow$ Fixed κ	74.5
$\rightarrow$ Fixed ϵ and κ	65.3
LSTM	62.5
$\rightarrow$ Fixed ϵ	43.7
$\rightarrow$ Fixed κ	63.0
$\rightarrow$ Fixed ϵ and κ	62.0

Table 4: Effects of holding ϵ and κ fixed during training. All numbers are on 1000 randomly chosen IMDB development set examples.

word w with $|N(w)| > 1$, and for both the pre-trained vectors $\phi^{\text{pre}}(\cdot)$ and transformed vectors $\phi(\cdot)$, we compute

$$\frac{1}{d}\sum_{i=1}^{d}\frac{1}{\sigma_i}\left(u_w^{\text{word}} - \ell_w^{\text{word}}\right)$$

where ℓ_w^{word} and u_w^{word} are the interval bounds around either $\{\phi^{\text{pre}}(\tilde{w}) : \tilde{w} \in N(w)\}$ or $\{\phi(\tilde{w}) : \tilde{w} \in N(w)\}$, and σ_i is the standard deviation across the vocabulary of the i-th coordinate of the embeddings. This quantity measures the average width of the IBP bounds for the word vectors of w and its neighbors, normalized by the standard deviation in each coordinate. On 78.2% of words with $|N(w)| > 1$, this value was smaller for the transformed vectors learned by the CNN on IMDB with robust training, compared to the GloVe vectors. For same model with normal training, the value was smaller only 54.5% of the time, implying that robust training makes the transformation produce tighter bounds. We observed the same pattern for other model architectures as well.

A.7 Certifying long-term memory

We might expect that LSTMs are difficult to certify with IBP, due to their long computation paths. To test whether robust training can learn recurrent models that track state across many time steps, we created a toy binary classification task where the input is a sequence of words $x_1, \ldots, x_L$, and the label y is 1 if $x_1 = x_L$ and 0 otherwise. We trained an LSTM model that reads the input left-to-right, and tries to predict y with a two-layer feedforward network on top of the final hidden state. To do this task, the model must encode the first word in its state and remember it until the final timestep; a bag of words model cannot do this task. For perturbations, we allow replacing every middle word $x_2, \ldots, x_{L-1}$ with any word in the vocabulary. We use robust training on 4000 randomly generated examples, where the length of each example is sampled uniformly between 3 and 10. The model obtains 100% certified accuracy on a test set of 1000 examples, confirming that robust training can learn models that track state across many time steps.

For this experiment, we found it important to first train for multiple epochs with no certified objective, before increasing ϵ and κ. Otherwise, the model gets stuck in bad local optima. We trained for 50 epochs using the normal objective, 50 epochs increasing ϵ towards 1 and κ towards 0.5, then 17 final epochs (determined by early stopping) with these final values of ϵ and κ.[7] We leave further exploration of these learning schedule tactics to future work. We also found it necessary to use a larger LSTM—we used one with 300-dimensional hidden states.

[7] Note that this dataset is much smaller than IMDB and SNLI, so each epoch corresponds to many fewer parameter updates.