BUILDING ROBUST NATURAL LANGUAGE PROCESSING SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Robin Jia

August 2020

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Percy Liang, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Dan Jurafsky

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Christopher Manning

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

Abstract

Modern natural language processing (NLP) systems have achieved outstanding performance on benchmark datasets, in large part due to the stunning rise of deep learning. These research advances have led to great improvements in production systems for tasks like machine translation, speech recognition, and question answering. However, these NLP systems still often fail catastrophically when given inputs from different sources or inputs that have been adversarially perturbed. This lack of robustness exposes troubling gaps in current models' language understanding capabilities, and creates problems when NLP systems are deployed to real users.

In this thesis, I will argue that many different aspects of the current deep learning paradigm for building NLP systems can be significantly improved to ensure greater robustness. In the first half of this thesis, I will build models that are robust to adversarially chosen perturbations. State-of-the-art models that achieve high average accuracy make surprising errors on inputs that have been slightly perturbed without altering meaning, for example by replacing words with synonyms or inserting typos. For a single sentence, there is a combinatorially large set of possible word-level perturbations, so guaranteeing correctness on all perturbations of an input requires new techniques that can reason about this combinatorial space. I will present two methods for building NLP systems that are provably robust to perturbations. First, certifiably robust training creates robust models by minimizing an upper bound on the loss that the worst possible perturbation can induce. Second, robust encodings enforce invariance to perturbations through a carefully constructed encoding layer that can be reused across different tasks and combined with any model architecture. Our improvements in robustness are dramatic: certifiably robust training improves accuracy on examples with adversarially chosen word substitutions from 10% to 75% on the IMDB sentiment analysis dataset, while robust encodings improve accuracy on examples with adversarially chosen typos from 7% to 71% on average across six text classification datasets from the GLUE benchmark.

In the second half of the thesis, I will consider robustness failures that stem from the unrealistic narrowness of modern datasets. Datasets for tasks like question answering or paraphrase detection contain only a narrow slice of all valid inputs, so models trained on such datasets often learn to predict based on shallow heuristics. These heuristics generalize poorly to other similar, valid inputs. I will present methods for both constructing more challenging test data and collecting training data that aids generalization. For the task of question answering, I will use adversarially constructed distracting sentences to reveal weaknesses in systems that standard in-distribution test data fails to uncover. In our adversarial setting, the accuracy of sixteen contemporaneous models on the SQuAD dataset drops from an average of 75% F1 score to 36%; on a current state-of-the-art model, accuracy drops from 92% F1 score to 61%. For pairwise classification tasks, I will show that active learning with neural sentence embedding models collects training data that greatly improves generalization to test data with realistic label imbalance, compared to standard training datasets collected heuristically. On a realistically imbalanced version of the Quora Question Pairs paraphrase detection dataset, our method improves average precision from 2% to 32%. Overall, this thesis shows that state-of-the-art deep learning models have serious robustness defects, but also argues that by modifying different parts of the standard deep learning paradigm, we can make significant progress towards building robust NLP systems.

Acknowledgements

First and foremost, I would like to thank my advisor, Percy Liang. When I started my Ph.D., I had never done any research outside of computational biology, but had somehow decided that if I was going to switch research areas, I wanted to study natural language processing. Percy agreed to take me as a rotation student and made this possible. He has been a fantastic advisor, and I feel so lucky to have gotten the chance to work with him and learn from him. Percy is amazingly supportive, superhumanly productive, and full of extremely helpful feedback. I am also in awe of his razor-sharp insights; I have often found myself realizing, after thinking on it for many days, that an idea Percy suggested but I had initially rejected was actually right on target.

I would also like to thank Chris Manning, Dan Jurafsky, Chris Potts, and Tengyu Ma, for serving on my thesis committee. I have always been impressed by Chris Manning's seemingly boundless wisdom and knowledge; every time I talk to him, I feel like I learn so much (and realize how much I don't know). Dan's positivity is contagious, and I always feel energized after talking with him. I am grateful to Chris Potts for all his help, whehter in finding philosophy of language papers for reading group or giving me job market advice. I appreciate Tengyu's feedback on my job talk and thesis defense.

Research can often be an emotional roller coaster, but sharing the journey with so many friends makes it much less scary. I have been lucky to be part of two very welcoming and supportive research communities: Percy's research group, p-lambda, and the Stanford NLP Group. I want to start by thanking my many fantastic collaborators: Dallas Card, Chris Donahue, Kerem Göksel, Kelvin Guu, Braden Hancock, He He, Peter Henderson, Erik Jones, Amita Kamath, Urvashi Khandelwal, Mina Lee, Juncen Li, Nelson Liu, Kyle Mahowald, Steve Mussmann, Panupong "Ice" Pasupat, Aditi Raghunathan, Pranav Rajpurkar, Megha Srivastava, and Sida Wang. Collaborating with these colleagues has been not only productive but also incredibly enjoyable. I especially want to thank Amita and Erik for being wonderful mentees: it is such a great pleasure to mentor students who are so dedicated, insightful, and kind. From Stanford, I also would like to thank Jonathan Berant, Arun Chaganty, Danqi Chen, Sidd Karamcheti, Fereshte Khani, Pang Wei Koh, Ananya Kumar, Tatsu Hashimoto, John Hewitt, Dan Iter, Will Munroe, Allen Nie, Peng Qi, Siva Reddy, Shiori Sagawa, Jacob Steinhardt, Michael Xie, Fanny Yang, Michi Yasunaga, Hugh Zhang, Yuchen Zhang, and

Yuhao Zhang. In addition, I would like to thank Eunsol Choi, Adam Fisch, Minjoon Seo, and Alon Talmor, with whom (along with Danqi and Jonathan) I had the great opportunity to co-organize the Machine Reading for Question Answering workshop. Finally, I want to especially thank Tatsu, Danqi, Eunsol, He, and Chris for their help and advice over the past year as I applied for academic jobs.

I am very grateful to my Microsoft Research internship mentor Hoifung Poon and co-author Cliff Wong, who gave me the opportunity to rekindle my interest in biology through working on biomedical NLP. Hoifung has been a very supportive and thoughtful mentor, and I am very glad I had the chance to work with him. I also want to thank Larry Heck, Dilek Hakkani-Tür, and Georgi Nikolov for mentoring me during my internship at Google Research.

I was fortunate to have excellent research mentors throughout my Stanford undergraduate career. I am very grateful to David Dill for agreeing to take me on as a CURIS summer research intern, back when I was a freshman who had only completed one real computer science class, and to Rob Bruggner for being my first graduate student mentor. I am also very grateful to Gill Bejerano, with whom I had the great pleasure of working during my junior and senior years. Gill's passion for genomics has always been a great inspiration to me. I would also like to thank the many Bejerano Lab members who supported me and gave me advice, especially Gray Camp, Harendra Guturu, and Jim Notwell.

Music has been an integral part of my life for a very long time. I am so lucky to have studied piano performance for nine years at Stanford with Laura Dahl—more if you also include her fantastic art song classes. Laura helped me grow so much as a musician, especially in my ability to listen carefully to both my own playing and that of my playing partners. I have also been very fortunate to study with Stephen Harrison, Kumaran Arul, and George Barth. Next, I want to thank the many wonderful musicians with whom I have had the privilege of making music: Lee Fan, Brad Girardeau, Andrew Guo, Jessy Hwang, Pang Wei Koh, Kevin Sun, Lisa Wang, and Ricky Wedeen. Finally, I am deeply grateful to Angela Wright, my piano teacher of twelve years before I came to Stanford. Learning to play the piano with Miss Wright taught me so much about not only music but also the value of hard work and practice. Playing the piano growing up also gave me a wealth of invaluable experience as a performer, which I have leaned on throughout graduate school.

I want to thank the many wonderful friends from outside of the Ph.D. program who have kept me sane over the past many years: Avanti Badrinathan, Catherine Chen, Charles Chen, Frank Chen, Jack Chen, Eddy Dai, Cynthia Day, Yifan Mai, Rohan Puttagunta, Brian Raftrey, Kris Sankaran, Raymond Wu, and Lynnelle Ye.

Finally, I want to thank my parents for their love and support throughout my Stanford journey (and for housing me for the last five months while we shelter-in-place due to COVID-19). I am deeply indebted to you for all you have done for me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Recent successes in NLP

If standard benchmarks are to be believed, natural language processing (NLP) systems are now remarkably good at a wide range of language understanding tasks. The rise of deep learning, and in particular the success of unsupervised pre-training (Peters et al., 2018; Devlin et al., 2019), has led to impressive results across many NLP datasets. Consider the Stanford Question Answering Dataset (SQuAD), a dataset for the task of question answering (Rajpurkar et al., 2016). Given a paragraph from Wikipedia and a reading comprehension question, a system must find the word or phrase in the paragraph that answers the question. In the original SQuAD paper from June 2016, the best baseline system achieved only 51.0% F1 score.[1] Crowdworkers asked to answer these same questions got 91.2% F1 score, indicating significant room for improvement.[2] As of August 2020, the best system on the SQuAD leaderboard has 95.4% F1 score, surpassing the measured human-level accuracy.[3]

SQuAD is merely one NLP dataset that deep learning has "solved." In the last few years, neural models have achieved near-human or super-human accuracy on many well-studied NLP datasets:

- Introduced in June 2018, SQuAD 2.0 was designed to be a more challenging successor to SQuAD, due to the inclusion of unanswerable questions that look similar to answerable ones (Rajpurkar et al., 2018). Between June 2018 and August 2020, the state-of-the-art improved from 66.3% F1 score to 93.0% F1 score, outperforming the human accuracy of 89.5% F1.[4]

[1]The F1 score here is an evaluation metric defined in Rajpurkar et al. (2016) that gives partial credit for partially correct answers. This is distinct from the standard F1 score used for binary classification.

[2]Strictly speaking, the human and system scores for SQuAD are not directly comparable, as the human scores are computed against one fewer reference answer than the system scores. The evaluation metric gives credit for matching any reference answer, so this puts humans at a slight disadvantage compared to the systems. The numbers on SQuAD 2.0 reported below do not suffer from this problem.

[3]`https://rajpurkar.github.io/SQuAD-explorer/`

[4]`https://rajpurkar.github.io/SQuAD-explorer/`

- Introduced in April 2018, the General Language Understanding Evaluation (GLUE) benchmark is a compilation of ten datasets chosen to test a broad array of language understanding abilities (Wang et al., 2019b). Systems are evaluated on GLUE score, a macro average of task-specific evaluation metrics across the ten datasets. Between April 2018 and August 2020, the state-of-the-art improved from 60.3 GLUE score to 90.7, surpassing the human score of 87.1.[5]

- In response to the progress on GLUE, SuperGLUE was introduced in May 2019 as the successor to GLUE. It retains the format of GLUE but uses a more challenging suite of datasets (Wang et al., 2019a). Between May 2019 and August 2020, the state-of-the-art improved from 71.5 to 89.3, within half a point of the human accuracy of 89.8.[6]

- While the above results come primarily on datasets built for NLP research, NLP systems have also reached impressive accuracy on exams designed for (human) students. Introduced in April 2017, RACE (ReAding Comprehension dataset from Examinations) is a dataset of expert-written reading comprehension questions from English language standardized exams for Chinese middle and high school students (Lai et al., 2017). Between April 2017 and August 2020, the state-of-the-art improved from 44.1% accuracy to 90.9%. Crowdworkers on Amazon Mechanical Turk only got 73.3% accuracy on these questions (however, the authors estimate that an expert human could achieve 94.5%).[7]

- In September 2019, Clark et al. (2019b) reported 90.7% accuracy on text-based questions from a standardized science exam for American eighth grade students. In comparison, their initial system from 2014 could achieve only 36.4% accuracy, and their 2018 system could achieve 73.1% accuracy.

What should we make of these impressive results? It may be tempting to conclude from results on these *datasets* that current systems are as good as humans at many NLP *tasks*. However, this conclusion is not warranted.[8] A dataset like SQuAD captures one particular distribution over documents and questions. The general task of question answering is much broader, encompassing

[5] Baseline results come from the first version of the GLUE paper made public in April 2018 (`https://arxiv.org/abs/1804.07461v1`). State-of-the-art results are taken from the GLUE leaderboard (`https://gluebenchmark.com/leaderboard`). The best baseline reported on the GLUE leaderboard achieves 70.0 GLUE score and is dated February 2019.

[6]`https://super.gluebenchmark.com/leaderboard`

[7]`http://www.qizhexie.com/data/RACE_leaderboard.html`

[8]Nevertheless, some media outlets have drawn such conclusions, including CNN ("Computers are getting better than humans at reading," `https://money.cnn.com/2018/01/15/technology/reading-robot-alibaba-microsoft-stanford/index.html`) and USA Today ("Robots are better at reading than humans," `https://www.usatoday.com/story/tech/news/2018/01/16/robots-better-reading-than-humans/1036420001/`). Many other media outlets did report on this same event in more tempered language, such as the Washington Post ("AI models beat humans at reading comprehension, but they've still got a ways to go," `https://www.washingtonpost.com/business/economy/ais-ability-to-read-hailed-as-historical-milestone-but-computers-arent-quite-there/2018/01/16/04638f2e-faf6-11e7-a46b-a3614530bd87_story.html`).

a wide variety of documents that are understandable to humans and questions that humans could answer based on those documents. Since datasets are much narrower than the underlying task, impressive results on benchmark datasets can paint an overly optimistic picture of how well models work in general.

In fact, state-of-the-art systems achieve impressive benchmark results by specializing heavily to the datasets on which they are trained and evaluated—not by solving the general underlying task. In a standard benchmark setting, systems are built using provided training data, then evaluated on test data that was created in the same way as the training data. High accuracy on these benchmarks does not ensure that models can generalize to examples created in a different way, even though such generalization would be expected of a system that solves the underlying task. For example, we can ask how well a SQuAD model generalizes to TriviaQA (Joshi et al., 2017), another question answering dataset in the same format as SQuAD. While SQuAD was collected by asking crowdworkers to write questions about Wikipedia paragraphs, TriviaQA took questions written for trivia competitions and then matched them automatically with web documents. Yogatama et al. (2019) train a standard BERT-base model (Devlin et al., 2019) on SQuAD training data and achieve 86.5% F1 score on the SQuAD development set. However, this model only manages a paltry 35.6% F1 score on the TriviaQA development set.[9] Despite its high accuracy on SQuAD, the model performs very poorly on other datasets in the same format.

1.2 The robustness problem

What is missing from the aforementioned SQuAD model, as well as current state-of-the-art NLP systems in general, is **robustness**. Since "robustness" has become a somewhat overloaded term, here we unpack exactly what we mean by robustness in this thesis. Sussman (2007) defines robust systems as "systems that have acceptable behavior over a larger class of situations than was anticipated by their designers." In the context of machine learning, Schain (2015) defines robustness as "the sensitivity of a learning algorithm to discrepancies between the assumed model and reality." Both definitions have a similar spirit, and seem like good properties for an NLP system to have, but are quite vague. What situations do designers of NLP systems anticipate? What assumptions do the learning algorithms they use typically make?

One important assumption on which most current work in NLP heavily relies is the *in-distribution* assumption—the assumption that the training data and test data are drawn from the same (or roughly the same) data generating distribution.[10] The in-distribution setting is tailor-made for

[9]This gap is not due to TriviaQA being fundamentally much harder or noisier than SQuAD. Yogatama et al. (2019) also show that the same model architecture trained with a mixture of TriviaQA training data and other training data achieves 72.5% F1 score on TriviaQA. Training with SQuAD and TriviaQA alone achieves over 75% F1 score on TriviaQA, though the exact number is not provided. As of August 2020, the best system on the official TriviaQA leaderboard (`https://competitions.codalab.org/competitions/17208#results`) achieves 93.0% F1 on the Web verified test dataset, which is comparable to the evaluation set used by Yogatama et al. (2019).

[10] For some datasets like MultiNLI (Williams et al., 2018), which is part of GLUE, the training and test distributions

the current dominant approach for building NLP systems, supervised learning (in particular, deep learning). In supervised learning, models are trained to produce the correct output on a set of training examples, with the hope that they will learn patterns that generalize to other test examples not seen during training. Supervised learning commonly assumes that training data and test data come from the same distribution, and standard approaches can be theoretically proven to generalize well in this in-distribution setting. In this thesis, we define robustness as the ability of a model to perform well on test data that differs in distribution from its training data—in other words, when the in-distribution assumption is violated.

1.3 Solving tasks requires handling worst-case examples

While we have established our focus on mismatches between training and test data, there is still a question of *how* the test data and training data differ. For instance, we could focus on robustness to domain shift, such as training on SQuAD and testing on TriviaQA. While this is an active area of research, we choose a different route. Much of the work in this thesis focuses on *adversarial robustness*, in which we evaluate on a worst-case (i.e., adversarially chosen) test distribution from a set of possible test distributions. As we will see throughout this thesis, models often perform very poorly on adversarially chosen test distributions that are seemingly very similar to the training distribution. The fact that accuracy drops precipitously when the data distribution changes in a seemingly small way makes adversarial robustness problems especially surprising and troubling.

In this section, we argue that adversarial robustness is closely related to the goal of solving language understanding tasks, rather than datasets. A system that does well on SQuAD but fails on other questions clearly has not solved the general task of question answering. To actually solve the task, a system would have to be able to produce reasonable answers when presented with any paragraph and question that is easy for a human well-versed in the language. Another way of phrasing this requirement is that *no adversary* should be able to find paragraphs and questions that stump the model but are easy for humans.

1.3.1 The Turing Test

In artificial intelligence (AI), the connection between true intelligent behavior (e.g., solving a broad task) and robustness to adversarial examples dates back many decades to Alan Turing's famous Imitation Game, the Turing Test (Turing, 1950). Turing sought to define sufficient conditions under which a machine could be said to exhibit intelligent behavior. In the Turing Test, a human

differ, as some text sources were excluded from the training data and only present in the test data. Empirically, current systems generalize very well across these particular types of train-test mismatch, likely because the same data collection procedure was used to generate all examples, leading to persistent similarities despite the difference in text source. More minor differences in distribution are also somewhat common; for instance, SQuAD (Rajpurkar et al., 2016) uses different Wikipedia articles at training time and test time.

interrogator poses natural language questions to an agent—either a human or a machine—in order to determine whether the agent is in fact a human or machine. If a competent interrogator cannot distinguish the machine from a human, Turing argued that this demonstrates human-like intelligence on the part of the machine.[11]

Crucially, Turing's interrogator is *adversarial*: they are incentivized to ask precisely those questions that would reveal non-humanlike behavior on the part of the computer. Therefore, the machine can only succeed if it is prepared to answer any question that the interrogator could ask. The presence of the adversary does not imply that intelligence or language understanding is a fundamentally adversarial concept. Instead, the adversarial nature of this interrogation serves to ensure that the interrogator will ask a sufficiently challenging and diverse array of questions. Robustness in the face of such a strong adversary is strong evidence of intelligent behavior, since if the agent had some serious flaw, an intelligent adversary would be able to identify and expose it. Throughout this thesis, we will consider test distributions that are—to varying degrees—adversarially chosen to expose models' weaknesses. Our ultimate goal is not to be antagonistic; on the contrary, we challenge systems so that we can identify and quantify their shortcomings and make measurable progress on building general language understanding systems.

1.3.2 Generalizing to the worst case in linguistics

The desire to account for worst-case instances also pervades the study of linguistics. This drive can be seen, for instance, in Richard Montague's work on formal semantics (Montague, 1970, 1973). Montague developed an elegant theory of semantics based on the lambda calculus in which phrases of the same syntactic category could be assigned identical semantic types. These semantic types are sometimes non-obvious. For instance, Montague chose to treat adjectives as functions from properties to properties, even though many adjectives (e.g., *"red"*) are intersective (i.e., *"red cars"* denotes precisely the set of cars that are also red) and therefore could be thought to have much simpler extensions. However, Montague pointed to the existence of non-intersective adjectives like *"big"* (a *"big flea"* is not necessarily a big entity) and non-subsective adjectives like *"alleged"* (an *"alleged intruder"* need not be an intruder), and concluded that a proper account of all adjectives demanded a more general semantic type (Montague, 1970). Montague's approach has been given the name "generalizing to the worst case" (Partee, 2007), as he purposefully constructed his theory so that it could uniformly handle these worst-case examples. We study worst-case examples with the same hope that they can shed light on ways we can improve our models of language.

[11]We distinguish the original test proposed in Turing (1950) with a modern colloquial usage of the term "Turing Test" to refer to any evaluation in which a human must guess whether a generated output was produced by a human or a computer. While this is another type of imitation game, it lacks an adversarial interrogator, so passing such a test is a much weaker indication of intelligent behavior.

1.4 Motivating settings

We have so far argued that progress on language understanding requires consideration of worst-case examples, but have not specified the types of worst-case examples on which we plan to focus. While the Turing Test serves as theoretical inspiration, it does not provide any practical guidance on this question; it simply demands that all questions a human interrogator may come up with should be answered, an unrealistic standard for current models (Shieber, 2016).

One possible approach is to focus on narrowly-defined tasks for which a complete solution seems within reach, then scale up to broader tasks; we will consider this idea more in Chapter 7. However, most research on robustness does not alter the task definition, but instead studies how to defend against a constrained adversary that can only create certain kinds of hard examples, as opposed to the unconstrained adversary of the Turing Test. By restricting our purview to certain prescribed test distribution shifts, we can focus on solving specific aspects of the broader language understanding problem. Here we discuss several more constrained settings that provide motivation for studying adversarial robustness.

1.4.1 Systematic language understanding

To test whether a computer system understands natural language, it is natural to first ask what is known about natural language, and then determine whether the system knows this. We thus turn to the literature on linguistics and cognitive science for inspiration. In particular, one central concept in semantics is the Principle of **Compositionality**, which states that the meaning of a complex linguistic expression is a function of the meaning of its syntactic constituents and the way their meanings are composed. Compositionality is often invoked to explain the infinite productivity of language—from a finite set of lexicon entries and grammar rules, it is possible to generate an infinite set of acceptable sentences with understandable meanings. The related concept of **systematicity** provides us with a way to behaviorally measure whether a system understands language compositionally (Fodor and Pylyshyn, 1988). Humans language understanding is systematic, in that the ability to understand certain sentences is inherently tied to the ability to understand other sentences. If a person understands the sentence *"John likes the dog,"* they certainly also understand the sentence *"The dog likes John."* There are simply no human beings in the world who would understand the former but be confused by the latter. Compositionality explains which sentences are related systematically with each other: two sentences must be systematically related if they involve the same syntactic constituents and same composition rules.

In Chapter 3, we will develop methods for achieving robustness to adversarial label-preserving word substitutions. These substitutions represent a very basic test of systematicity: if a system is correct on an input, and a word in that input is replaced with a word that the model is familiar with and is synonymous (or nearly synonymous) with the original word, it ought to continue predicting the correct answer. A failure to do so would indicate a lack of systematicity. We will also discuss possible ways to extend this setting to consider compositional structure beyond simple word-for-word substitutions.

It is of course the case that compositionality and systematicity do not encompass all of the important things known about language. Moreover, in this thesis we will only test systematicity in a very restricted sense. Our aim is to start with the most elementary properties of language, and assess whether systems have mastered these basics. We believe this to be particularly important given the black-box nature of modern deep learning models, which makes it difficult to reason about their likely failure modes. In the past, standard statistical models had structures that made at least some of their flaws easy to predict. For example, bag-of-words models are insensitive to word order; n-gram language models are unable to recall long-range context; probabilistic context free grammars and conditional random fields make explicit conditional independence assumptions. However, neural networks pose a challenge for such *a priori* analysis of model shortcomings, as neural networks of sufficient complexity can famously approximate any function of their input. Judging from their impressive benchmark results, neural networks have likely managed to *approximately* learn about many linguistic phenomena, but their inscrutability should make us question whether they have mastered any of them. Adopting this mentality, we therefore start with the simplest tests of competence, and find that models indeed often fail these basic tests.

Also along these lines, in Chapter 5 we will construct examples that are reminiscent of minimal pairs, which are commonly used in linguistics to show how small changes in a sentence can give rise to shifts in meaning or acceptability. Again, these minimal pairs will purposefully be quite simplistic compared to the ones commonly studied in the linguistics literature, in order to find the simplest challenge that nonetheless confounds current NLP models.

1.4.2 Real-world adversaries

Not all adversaries are well-meaning theoretical interrogators; adversarial agents often have strong incentives to attempt to manipulate the predictions of real-world NLP systems. For example, spam detection systems must be able to identify spam in spite of spammers attempting to bypass these filters (Dalvi et al., 2004; Lowd and Meek, 2005). Search engine optimization rankings make use of textual features, and website creators have strong incentives to find ways to artificially increase their website's predicted relevance to user queries. More recently, systems that flag hateful social media content for review must be robust to adversaries who wish to evade detection (Hosseini et al., 2017). Defending against these threats requires building systems that are robust to whatever alterations an attacker might apply to text in order to achieve the desired classifier behavior. In general, this space of possible alterations is very large, and may be very difficult even to define. Nonetheless, we can at least strive to make the adversaries' jobs more difficult by making models robust to some types of common alterations, such as synonym replacements or typos.

1.4.3 Avoiding bias

It is often the case that available datasets do not exactly represent the data distribution of interest. One particularly problematic case is when the dataset is biased in some way against a particular demographic group, which often leads to model predictions that unfairly disadvantage members of that group. For example, naturally occurring data will often underrepresent minority groups, so systems can do well on average while having high error rates on these groups (Blodgett et al., 2016; Tatman, 2017; Buolamwini and Gebru, 2018; Sap et al., 2019). Datasets may also recapitulate stereotypes tied to historical injustices, which incentivizes models to learn these very stereotypes to achieve the best test accuracy (Zhao et al., 2017, 2018; Rudinger et al., 2018). Training data collected from current users of a system will likely be skewed towards users for which the system works well, leading to a feedback loop in which underserved users become increasingly underrepresented in the training data (Hashimoto et al., 2018). In many cases, avoiding these types of biases can be thought of as optimizing a worst-case objective in which an "adversary" may change the data distribution to include more minority group members, or include more examples that invert stereotypes. While this thesis does not focus on preventing bias against underrepresented minorities, we believe it is an important motivating case for work on robustness.

1.4.4 Realistic distribution shift

In general, deployed systems must be robust to many kinds of mismatches between the training data and test inputs. Many of these mismatches occur naturally, when user requests fall outside of the training distribution. For example, linguistic annotation systems, such as part-of-speech taggers or syntactic parsers, may be asked by users to annotate text from sources that were not present in the system's training data. Generalizing to such cases is an important robustness challenge, although it can sometimes be unclear whether such generalization is possible if the training and test distributions are too different (Geiger et al., 2019). Work in domain adaptation usually assumes that some information about the domain of interest is provided, either in the form of a small labeled dataset or unlabeled data (Blitzer et al., 2006; Daumé III, 2007). In Chapter 6, we will consider another setting, extreme label imbalance, in which train-test mismatch naturally occurs. We will explore better methods of data collection, which allows us to circumvent the question of whether generalization is possible given the original training dataset alone.

Even when systems perform well on user queries on average, rare but catastrophic errors can lead to serious problems. In 2017, Facebook's machine translation system mistakenly translated an Arabic Facebook post with the message *"Good morning"* into a Hebrew phrase that meant *"Attack them"* (Berger, 2017). As a result, the Israeli police arrested the man who made the post and detained him for several hours. The machine translation system was not robust to the uncommon way the person wrote *"Good morning"*, leading to very serious consequences. Deployed systems must avoid egregious errors like wrongly translating non-violent messages into violent ones, even on

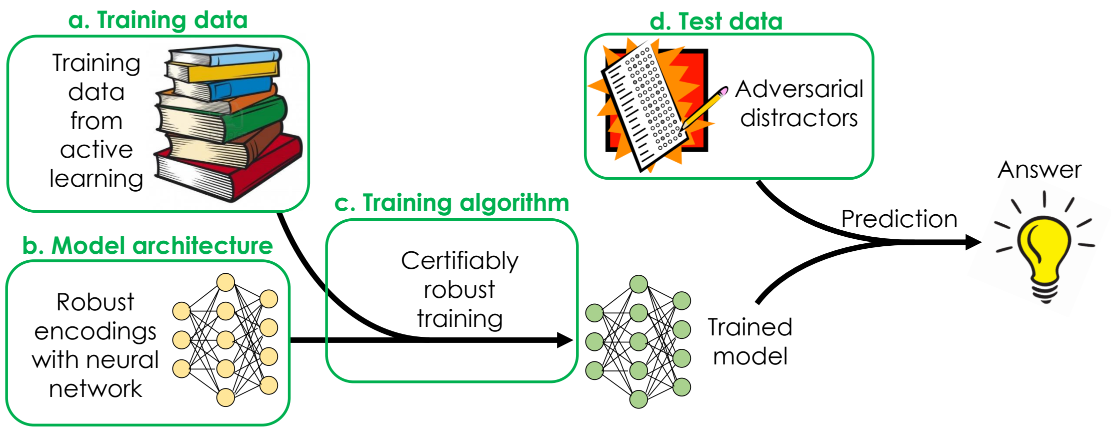


Figure 1.1: Work presented in this thesis on improving robustness, overlaid on a schematic depiction of a standard deep learning pipeline for building NLP systems. In a standard supervised deep learning setting, training data (a) is used to learn parameters of a model architecture (b) via a training algorithm (c). This produces a trained model, which can take in test data (d) and output predicted answers. This thesis presents ways to improve these four components of this pipeline to yield more robust models. Certifiably robust training (Chapter 3) trains models to be robust to adversarial perturbations. Robust encodings (Chapter 4) are a model architecture component that also increases robustness to adversarial perturbations. Adversarial distracting sentences (Chapter 5) are one way to generate challenging test data to better evaluate robustness. Finally, active learning in imbalanced pairwise tasks (Chapter 6) collects training data that improves model generalization.

“worst-case” non-violent messages.

1.5 Building robust NLP systems

We have now seen many different robustness problems, and in particular adversarial robustness problems. Motivated by these problems, this thesis develops methods for building robust NLP systems. We focus on adversarial robustness problems in which reasonable systems should be expected to generalize, yet fail to in practice. Earlier, we noted that adversarially chosen test distributions that differ only slightly from the original training distribution can cause precipitous drops in accuracy. From the perspective of improving robustness, this closeness between the training and test distributions gives us some hope that robustness to these adversarial shifts is possible in principle, if we can find the right techniques.

While the robustness problems we consider are diverse, they share a central challenge: in order to ensure robustness, a system needs some way to compute, approximate, or bound the error incurred by worst-case inputs selected from a space of possibilities too large to enumerate. Standard neural NLP models have no way to reason about these worst-case scenarios. In this thesis, we show

how to modify various parts of the standard deep learning pipeline to guard against worst-case inputs, as depicted in Figure 1.1. In the following chapters, we will develop new training algorithms (Chapter 3), new model architectures (Chapter 4), and new dataset collection methods (Chapter 6), all of which address the key challenge of guarding against worst-case inputs while operating within the deep learning framework. In this way, we build robust NLP systems that still leverage and build on the tremendous progress of the past few years.

1.6 Outline

This thesis will describe four pieces of work that study different aspects of robustness in NLP. In each case, we will design adversarial tests of robustness that introduce test examples that systems should be expected to handle. Despite the simplicity of the challenges these tests introduce, state-of-the-art models will fare poorly. In the first half of the thesis, we will focus on robustness to adversarial perturbations, a setting that allows us to formally define a worst-case robustness goal. We will develop new training procedures and model architectures that can optimize for these goals. In the second half, we will examine cases where weaknesses of the dataset creation procedure itself creates problems. We will develop new strategies for constructing both evaluation and training datasets.

1.6.1 Adversarial perturbations

In Chapter 3 and Chapter 4, we will study robustness to adversarial perturbations. We will build models that are guaranteed to give the correct prediction when an adversary applies label-preserving perturbations to words in an input. Guaranteeing correctness is challenging because the adversary may simultaneously perturb many words in an input, leading to an exponentially large space of total perturbations. Due to this combinatorial explosion, modifying the training dataset alone does not ensure robustness, as it is computationally infeasible to include all possible transformations in the data. Instead, we will either change the training algorithm or the structure of the model to create models that can provably guarantee robustness to adversarial perturbations.

In Chapter 3, we focus on label-preserving word substitutions, such as lexical paraphrases. We train models that are provably robust to these word substitutions through a certifiably robust training procedure. In particular, we use interval bound propagation (IBP) to minimize a computationally tractable upper bound on the worst-case loss that any combination of word substitutions can introduce (Dvijotham et al., 2018). IBP had been previously used to guarantee robustness to pixel-level perturbations in computer vision; our work shows how to extend IBP to discrete input spaces and to neural network architectures commonly used in NLP, such as recurrent neural networks. This chapter is based on Jia et al. (2019).

In Chapter 4, we will guard against errors caused by adversarially chosen typos. Instead of relying on IBP, which makes assumptions about model architecture and scales poorly with network depth, we propose the framework of robust encodings, which confers guaranteed robustness without imposing any restrictions on model architecture. Central to this framework is the notion of a robust encoding function. An encoding function is a function that maps text inputs to a discrete set of encodings. Such a function is robust if it has both high stability—all perturbations (e.g., typos) of an input map to a small set of encodings—and high fidelity—the encodings must retain enough information about the input to be useful. Given a robust encoding function, we can create an NLP system that is robust to perturbations by composing it with a normally trained classifier that uses the encoding as input. We use an agglomerative clustering algorithm to create a robust encoding function for typos, and show that the same encoding function can be used to guarantee robustness to typos across many NLP tasks. This chapter is based on Jones et al. (2020).[12]

1.6.2 Weaknesses of standard datasets

In Chapter 5 and Chapter 6, we will consider cases in which flaws in data collection lead to datasets that oversimplify the tasks they are meant to represent. As a result, models trained on these datasets fare well on test data collected in the same way, but generalize poorly to other types of examples that are plausible in the context of the broader task and are seemingly not that different from training examples.

In Chapter 5, we adversarially evaluate reading comprehension systems trained on SQuAD by inserting distracting sentences into test paragraphs. These distracting sentences have keywords in common with the question, but do not actually answer the question. By inserting these distracting sentences, we demonstrate that models trained on SQuAD do not precisely understand the semantics of these sentences, but instead learn to rely on surface-level similarity to answer questions. This chapter is based on Jia and Liang (2017), which was the first work to demonstrate that neural reading comprehension systems could be fooled by simple adversarial alterations to inputs.

In Chapter 6, we wrestle with distribution shift inherent to extremely imbalanced pairwise classification tasks like paraphrase detection, in which the vast majority of examples (e.g. 99.99%) are negatively labeled. Since sampling from this distribution would be highly inefficient for constructing a training dataset, this setting forces us to accept a mismatch between training time and test time. Many recent datasets bypass this problem by heuristically collecting balanced data for both training and testing. However, models trained on this data often have very low precision on realistically imbalanced test data. Instead of relying on these heuristics, we use active learning, in particular uncertainty sampling, to collect balanced training data that enables generalization to imbalanced test data. To address the computational challenge associated with finding good examples to label, we use a pairwise neural embedding model that enables us to use efficient nearest neighbor search. This chapter is based on work in submission done jointly with Stephen Mussmann and Percy Liang.

[12] I am co-second author on Jones et al. (2020). I was a co-mentor to the first author Erik Jones, who was an undergraduate during this project.

Chapter 2

Background

Throughout the history of AI, mismatches between what is anticipated during system development and what might actually occur have been a source of concern. Expert systems often suffered from brittleness, or a lack of robustness. These systems relied on human experts to engineer rules, but these experts were often unable to anticipate the myriad complications and interactions that could lead to failure at test time. As machine learning became the dominant strategy for building AI systems, the burden of anticipating new situations shifted from human experts to data. The promise of machine learning was that training data provided a much more flexible interface for defining system behavior, compared with human-specified rules. Accordingly, the source of robustness issues also shifted. The situations anticipated during system design were now the situations presented in the training data, and the question of robustness became a question of whether systems could generalize to test examples that were unlike those seen at training time.

In this chapter, we survey work on robustness in natural language processing. We begin by discussing robustness problems in expert systems. Then, we will turn our attention to various types of train-test mismatch encountered by machine learning-based NLP systems. We will identify three broad (and sometimes overlapping) ways that a test distribution can diverge from a training distribution: example-level perturbations, reweighting, and extrapolation. We will also make note of the resources available to the system designer at training time, as the extent to which they know what test-time shifts may occur determines the difficulty of the associated robustness problem.

2.1 Brittleness in expert systems

Expert systems, which emerged in the 1970's and dominated the field of AI in the 1980's, were often criticized as brittle. McCarthy (1984) wrote of expert systems, "[T]hey are difficult to extend beyond the scope originally contemplated by their designers, and they usually don't recognize their own limitations." Winograd (1991) noted, "It is a commonplace in the field to describe expert systems as *brittle*—able to operate only within a narrow range of situations. The problem here is not just one of insufficient engineering, but is a direct consequence of the nature of rule-based systems."

The brittleness of expert systems stems from a few key sources. Winograd discusses three important flaws inherent to expert systems. First, they are saddled with *gaps of anticipation*: domain experts are often unable to anticipate how complex interactions between the rules they provide could result in unwanted behavior in new situations. Second, expert systems are limited by the *blindness* of symbolic representations to world context. Winograd specifically discusses the issue of ambiguity in understanding natural language—a question like *"Is the patient eating?"* could mean different things (e.g., whether they are eating at this moment, or whether they have eaten in the last day) depending on broader context. Fully understanding questions like this requires extensive world knowledge. Winograd concludes that expert systems thus inevitably suffer from *restriction of the domain*: they can only succeed in narrow, simplified domains, due to the limitations of their representations (Winograd, 1991). McCarthy similarly writes that brittleness stems from expert systems' lack of common sense knowledge (McCarthy, 1984). This viewpoint has motivated work on improving commonsense understanding in AI, notably the Cyc project (Lenat et al., 1985). Other work has also pointed to a lack of probabilistic reasoning as a source of brittleness (Zadeh, 1983).

Given these brittleness problems with expert systems, some researchers began to argue for connectionist approaches based on learning from experience. Winograd (1991) discusses the then-renewed interest in connectionist approaches or "emergent AI," which sought to mimic how "cognitive structure in organisms emerges through learning and experience, not through explicit representation and programming." He notes, "The problems of blindness and domain limitation described above need not apply to a system that has developed through situated experience." Where hand-crafted rules fail, connectionist systems could succeed by filling in the soft reasoning and intuition that is difficult to formalize in an expert system. Work on hybrid architectures that combine neural networks with expert systems exemplifies this optimism (Gallant, 1988; Yahia et al., 2000).

2.2 Robustness in machine learning

The shift in the NLP and broader AI community to learning from data has not eliminated the problem of robustness. It is true that the dense representations learned by modern deep learning systems are highly contextualized and seem to capture at least some world knowledge.[1] The flexibility of these representations gives deep learning the potential to handle very complex tasks and domains. However, a key underlying issue remains: conditions that were not anticipated during system creation can lead to failures at test time, as machine learning methods are highly dependent on access to representative training data, as well as inductive biases that enable generalization. The burden

[1]These representations are however still usually blind to situated world context, as they lack grounding.

of anticipation has not been overcome, but has shifted to data collection and model architecture design.

We now make this phrasing more precise. The typical supervised learning setting considers *out-of-sample* but *in-distribution* generalization: the training and test datasets are independent samples from the same probability distribution. Throughout this chapter, let $p(x, y)$ denote an "original" data distribution over pairs of examples (x, y), where $x \in \mathcal{X}$ is an input (e.g., a sentence) and $y \in \mathcal{Y}$ is the corresponding correct output (e.g., a classification label). In the standard in-distribution setting, model parameters θ are trained using a training dataset D_{train} with samples independently and identically distributed (i.i.d.) from p, then evaluated on their average in-distribution loss:

$$L_{\text{indist}}(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim p} \left[\ell(x, y; \theta)\right], \tag{2.1}$$

where $\ell(x, y, \theta)$ is the loss (e.g., zero-one loss) of model θ on example (x, y) (lower loss is better). We also refer to this evaluation metric as standard loss, and accuracy on in-distribution examples as **standard accuracy**. Note that this expectation includes individual examples (x, y) that did not occur in D_{train}, so this does measure out-of-sample generalization, but everything in the expectation is in-distribution relative to how D_{train} was generated. Robustness issues arise when there is a mismatch between the training data *distribution* and test data *distribution*.

The remainder of this chapter surveys three broad ways that the test distribution can differ from the training distribution in the context of NLP tasks. First, individual test examples can be perturbed by applying random or adversarial transformations that do not change the correct output y (Section 2.3). Second, the frequency of test examples can be reweighted relative to the original distribution (Section 2.4). Finally, the test distribution may be much more different from the training distribution, thus requiring extrapolation (Section 2.5).

2.3 Example-level perturbations

One natural way to alter the test distribution is to slightly perturb samples from the distribution. In this section, we focus on perturbations that are label-preserving, meaning that the ideal system should make the same prediction on the original and perturbed versions of an input. For a model that does well on in-distribution test data, robustness to small perturbations seems a fairly modest goal—all we are asking now is that the model also stay correct on *slightly* different examples. However, we shall see that achieving even this modest goal is often challenging.

We will encounter two broad classes of perturbations in NLP. Some label-preserving operations arise from natural sources of noise. Users make mistakes while typing, and automatic speech recognition and optical character recognition systems inevitably make mistakes too. Other label-preserving perturbations are meaning-preserving operations, such as paraphrasing. Errors on minor meaning-preserving perturbations indicate a failure to learn the systematic patterns inherent to natural language. Such errors also pose a challenge for human interpretability of NLP systems—if a system gets an example correct, it is natural for a human to assume it will also get very similar examples correct.

Perturbations may either be chosen randomly or adversarially. A **random perturbation** setting is defined by a noise distribution $q(x' \mid x)$ over perturbations x' of an input x. For example, x' could be x with random typos inserted. A model is evaluated on samples from p that have been randomly noised:

$$L_{\text{randperturb}}(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim p}\left[\mathop{\mathbb{E}}_{x'\sim q(x'|x)}[\ell(x', y, \theta)]\right], \tag{2.2}$$

Note that if θ is learned using data that is perturbed by q, then this is a standard in-distribution setting. However, if θ is learned only on data sampled from p, there is a train-test mismatch, albeit a small mismatch if q only changes examples by a very small amount.

An **adversarial perturbation** setting is instead defined by a neighborhood function $N(x) \subseteq \mathcal{X}$, which defines a set of nearby points for each $x \in \mathcal{X}$. For instance, $N(x)$ might be the set of all strings x' that have edit distance ≤ 1 from x, representing the set of all one-character typos of x. Borrowing terminology from computer security, we also refer $N(x)$ as the **attack surface**, as it represents the set of possible actions an adversary can take for any given x. Models are evaluated by first sampling (x, y), then measuring the worst-case loss across all $x' \in N(x)$:

$$L_{\text{advperturb}}(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim p}\left[\max_{x'\in N(x)} \ell(x', y, \theta)\right]. \tag{2.3}$$

Compared to (2.2), we have replaced the inner expectation with a max, which can be thought of as a loss-maximizing adversary that adapts to the model θ by finding the perturbation that confuses this particular model the most. Thus, in order to get credit for an example, the model must predict correctly on *every* $x' \in N(x)$. Note that in practice, computing this max is often computationally intractable, as $N(x)$ is often exponentially large, but this can be approximated with the result of an optimization procedure. An $x' \in N(x)$ that causes a misclassification is often referred to as an **adversarial example**.

2.3.1 Adversarial examples in computer vision

Much of the current work on adversarial examples in NLP draws inspiration from similar work in computer vision. Here, we describe the phenomenon of adversarial examples as studied in image classification.

Initial observations. Szegedy et al. (2014) first noted the existence of small pixel-level perturbations that could trigger errors in image classification systems. Goodfellow et al. (2015) explored

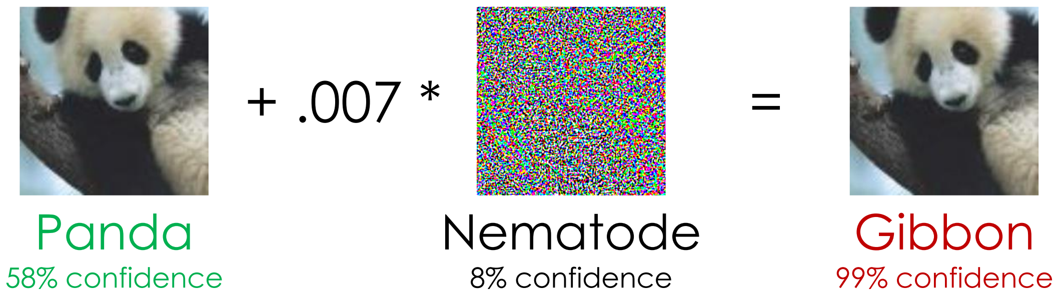


Figure 2.1: An imperceptible adversarial perturbation in computer vision. The model predicts the correct class initially, but when an imperceptible pixel-level perturbation is added, it becomes confident in an incorrect class. Note that the perturbation itself is not classified as anything with high confidence. Original figure from Goodfellow et al. (2015).

this phenomenon in greater detail. Specifically, they consider an attack surface in which each pixel of an image may be changed by a very small amount ϵ. Formally, this is an adversarial perturbation setting where the input $x \in \mathbb{R}^d$ is a d-dimensional vector representing the pixel values of an image, and $N(x) = \{x' : \|x' - x\|_\infty \leq \epsilon\}$ for some small $\epsilon > 0$. Here, $\|x' - x\|_\infty \overset{\text{def}}{=} \max_i |x'_i - x_i|$ is the L_∞ norm, and this $N(x)$ is also known as the L_∞ ball of radius ϵ around x.

Computing (2.3) requires maximizing $\ell(x', y; \theta)$ with respect to x', subject to the constraint that $\|x' - x\|_\infty \leq \epsilon$. For neural network models, this is a non-convex optimization problem that does not admit an efficient solution. Goodfellow et al. (2015) propose approximating this maximum using the fast gradient sign method (FGSM), in which an approximate worst-case input x' is computed by either adding or subtracting ϵ to each coordinate in x, determined by whether the gradient of the loss through the i-th coordinate of x is positive or negative. Formally, FGSM uses the approximation

$$\arg\max_{x' \in N(x)} \ell(x', y; \theta) \approx x + \epsilon \cdot \text{sign}(\nabla_x \ell(x, y; \theta)). \tag{2.4}$$

Despite the simplicity of FGSM, Goodfellow et al. (2015) find that it can consistently fool models with perturbations that are so small that they are *imperceptible* to humans, as shown in Figure 2.1.

Defenses. The widespread susceptibility of models to adversarial examples led to a flurry of subsequent work trying to train models that could not be fooled by these imperceptible pixel-level perturbations. Goodfellow et al. (2015) proposed an adversarial training algorithm in which examples generated by the fast gradient sign method are computed on the fly for the current model parameters, and the model is trained on these examples in addition to unperturbed examples. More precisely, given a training example (x, y), adversarial training takes a gradient step on θ to minimize

$$\alpha \cdot \ell(x, y; \theta) + (1 - \alpha) \cdot \ell(x + \epsilon \cdot \text{sign}(\nabla_x \ell(x, y; \theta)), y; \theta) \tag{2.5}$$

where $\alpha \in [0, 1]$ is a hyperparameter. While many variants on adversarial training have been proposed, Athalye et al. (2018) showed that many fail to actually guarantee robustness to all $x' \in N(x)$. In particular, many proposed defenses rely on **obfuscated gradients**—making the gradient of the loss with respect to x uninformative—which stops simple gradient-based adversaries but does not actually make the model correct on all $x' \in N(x)$, as is demanded by (2.3).

In response, Raghunathan et al. (2018) and Wong and Kolter (2018) developed certified defenses, which can guarantee correctness on all allowed perturbations of some examples. For some examples (x, y), a certified defense can produce a *certificate*, or formal proof, that no perturbation $x' \in N(x)$ is misclassified by the model. Note that there may be many other (x, y) for which a certificate cannot be constructed, even if the model is actually correct on all $x' \in N(x)$. Thus, *certified accuracy*—the fraction of test examples for which a certificate can be produced—may be an underestimate of accuracy on actual perturbed examples. In practice, certified defense methods can find model parameters such that certified accuracy is reasonably high and the gap between certified accuracy and accuracy on actual perturbations is not too large. Unfortunately, these works were restricted to networks of very constrained architecture and size. Subsequent work on interval bound propagation (IBP) from Dvijotham et al. (2018) and Gowal et al. (2019) showed how to construct certified defenses for much more general classes of neural networks.

A parallel line of work has made adversarial training more effective by using projected gradient descent (PGD) to search for a high-loss input x' starting from a randomly chosen point in $N(x)$, as proposed by Madry et al. (2018). This randomization bypasses issues with obfuscated gradients, and empirically leads to models that are robust even when a more expensive search algorithm is run at test time to find high-loss points in $N(x)$. We note that PGD assumes there is an efficient way to project a point onto the nearest point in $N(x)$, which is straightforward for simple norm-constrained perturbations but less so for attack surfaces with more complex geometry.

Both certified defenses and adversarial training share the same drawback: increasing robustness to adversarial perturbations tends to decrease standard accuracy. Some work has attempted to explain this trade-off theoretically (Zhang et al., 2019b), but in principle it is not clear that robustness and accuracy must be at odds. A recent line of work has shown how to improve this trade-off by using unlabeled data (Carmon et al., 2019; Uesato et al., 2019; Najafi et al., 2019; Raghunathan et al., 2020).

Beyond L_∞ perturbations. Finally, recent work has cast more attention on perturbations beyond L_∞ perturbations. The ImageNet-C benchmark (Hendrycks and Dietterich, 2019) tests robustness to a wide range of random perturbations, such as JPEG compression, motion blur, and changes in brightness and contrast. Models are explicitly prohibited from training on these corruptions, in order to benchmark the ability of models to generalize to these slight alterations to natural images. Kang et al. (2019) evaluate whether models trained to be robust to one type of perturbation can generalize to other perturbations. They consider both L_p norm-constrained adversarial perturbations for various values of p, as well as various types of random noise.

2.3.2 Spam classification

In NLP, adversarial attacks on classifiers were studied long before the work of Szegedy et al. (2014). Notably, spam classifiers must guard against real-world adversaries who attempt to evade detection by changing the e-mails they send. Some methods attempt to undo these changes via preprocessing. Lee and Ng (2005) study spam deobfuscation, the task of converting obfuscated text used by spammers (e.g., replacing words like *"refinance"* with *"r.efina.nce"* or *"re xe finance"*) back into normal text. Chapter 4 will present a related approach.

Other work directly makes classifiers more robust to spammers' obfuscation attempts, focusing on linear models. Dalvi et al. (2004) analyze optimal strategies for a Naive Bayes spam classifier and adversary. They consider various adversarial strategies, including adding irrelevant words and replacing words with synonyms. Globerson and Roweis (2006) defend against adversarial feature deletion, which may occur when spammers realize they should avoid certain words. Their work trains support vector machines that are optimally robust to the adversarial removal of certain features. Lowd and Meek (2005) present an efficient attack by which an adversary can reverse-engineer the weights of a linear classifier, in order to then generate inputs that can evade detection. Overall, while the problems studied in these works are closely related to recent work on robustness, the methods are quite different due to the focus on linear classifiers. The minimax games that arise in adversarial settings are often easier to analyze by assuming models are linear. Goodfellow et al. (2015) too note that linear image classification models are susceptible to adversarial examples, and that adversarial training for linear models corresponds to a form of regularization that can be written in closed form. With the success of deep learning, work on adversarial robustness has shifted to defenses that can accommodate more general model families.

2.3.3 Adversarial perturbations and neural NLP

Chapter 3, Chapter 4, and (to some extent) Chapter 5 will cover my work studying the robustness of deep learning NLP models to adversarial perturbations. Chapter 5 describes Jia and Liang (2017), the first work that evaluated neural question answering systems on adversarial examples, and Chapter 3 describes Jia et al. (2019), the first work that built neural NLP models with guaranteed robustness to an exponentially large space of adversarial perturbations. We therefore leave more detailed discussion of this topic to these chapters, but devote some space here to surveying contemporaneous work on this topic.

Semantic perturbations. One prominent line of work focuses on small perturbations that preserve meaning. Analogously to norm-constrained attack surfaces in computer vision, multiple NLP papers define concrete attack surfaces $N(x)$ for natural language tasks in which words are replaced with other words that have similar meanings (Alzantot et al., 2018; Jin et al., 2020). Ribeiro et al. (2018) use automatic paraphrase generation tools to identify Semantically Equivalent Adversarial Rules (SEARs), or local rewrite rules that induce errors across many examples. For example, simply replacing *"?"* with *"??"* in examples from the SQuAD development set led to 202 more errors by a then-state-of-the-art model; this alone increased the overall error rate by 3%. Other simple operations that triggered errors included replacing *"What is"* with *"What's"* and replacing *"What"* with *"Which"*. Iyyer et al. (2018) craft adversarial examples by applying syntactic paraphrase operations, rather than relying on lexical substitution. For example, a model tested in the paper correctly assigned negative sentiment to the sentence *"[T]here is no pleasure in watching a child suffer,"* but erroneously assigned positive sentiment to *"[I]n watching the child suffer, there is no pleasure."*

Typos. Another major line of work focuses on typos (Belinkov and Bisk, 2018; Ebrahimi et al., 2018a; Edizel et al., 2019). Typos represent an important practical robustness problem: they are common in naturally occurring text and can be easily generated by adversaries, and often do not affect human readability, but models tend to be very sensitive to them. Hosseini et al. (2017) showed that Google's Perspective API, a service that was supposed to detect hateful speech online, could be fooled by inserting typos (e.g., replacing *"idiot"* with *"idiiot"*) and other simple alterations that leave the message easily understandable by humans. Pruthi et al. (2019) studied the robustness of various state-of-the-art models to typos. Despite its impressive standard accuracy on many datasets, BERT is extremely sensitive to typos: a BERT model that achieves 90.3% accuracy on a sentiment analysis dataset can drop all the way to 45.8% when an adversary is allowed to insert a single one-character typo in the input, and 24.6% with two typos. They also show that using a typo-corrector can help defend against typos but still leaves a significant gap with original accuracy; on the aforementioned sentiment analysis dataset, the best typo corrector only brings BERT back to 75.0% accuracy with one typo and 68.0% accuracy with two typos.

2.4 Reweighting and subgroups

Adversarial perturbations modify each test example slightly, but do not alter the underlying distribution p from which original examples are sampled. An orthogonal way in which test distributions can shift is by reweighting the original distribution. Reweighting emphasizes accuracy on certain subsets of examples that occur in $p(x, y)$, as high average accuracy can hide poor accuracy on subsets of interest. In general, a reweighted distribution $q(x, y)$ for an original distribution $p(x, y)$ satisfies the property that $q(x, y) > 0$ only if $p(x, y) > 0$. In other words, q only puts probability mass on pairs that have support (i.e., are possible) on p. A model is trained on data sampled from p but evaluated on data sampled from q:

$$L_{\text{reweight}}(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim q(x,y)} \left[\ell(x,y;\theta)\right]. \tag{2.6}$$

A particular type of reweighted distribution is a subgroup distribution. Let $g : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ be a group membership function denoting whether an example (x,y) is in a particular group.[2] For example $g(x,y)$ could be 1 if x comes from a particular minority group and 0 otherwise. The loss on the subgroup defined by g is

$$L_{\text{subgroup}}(\theta) = \mathop{\mathbb{E}}_{(x,y)\sim p(x,y|g(x,y)=1)} \left[\ell(x,y;\theta)\right]. \tag{2.7}$$

In other words, models are tested on the subset of examples from the original distribution that belong to a particular group. Within this group, no reweighting is done, so relative proportions within the group are preserved.

2.4.1 Bias against underrepresented groups

An important reason to care about reweighted test distributions is that the natural data distribution often underrepresents certain populations of individuals. Therefore, models may achieve high average accuracy on the original distribution without adequately modeling these underrepresented groups. Blodgett et al. (2016) showed that African American English (AAE) sentences are more often misclassified as non-English by language identification systems, and more often parsed incorrectly by syntactic parsers, compared to white-aligned sentences. Sap et al. (2019) further showed that AAE Tweets are more likely to be classified as hate speech. These findings echo results in computer vision, which finds that deployed face recognition systems have higher error rates on darker faces compared to lighter faces, and higher error rates on female faces compared to male faces (Buolamwini and Gebru, 2018). To diagnose these failures, it is important to evaluate systems on subsets of examples from these underrepresented groups.

Models also learn to reproduce stereotypes about certain demographic groups. For example, coreference resolution systems often propagate gender biases (Zhao et al., 2018; Rudinger et al., 2018). Rudinger et al. (2018) give the example of the sentence *"The surgeon couldn't operate on his patient: it was his son!"* The Stanford CoreNLP coreference resolution system correctly predicts that both occurrences of *"his"* are coreferent with *"The surgeon."* However, given the same sentence with *"her"* instead of *"him,"* the system fails to make this same prediction. It seems that the system has learned the stereotype that surgeons are more likely to be men than women. Troublingly, Zhao et al. (2017) show that systems trained on biased data tend to *amplify* this bias. They train a system on

[2]In most cases, g is purely a function of x, but for generality we allow it to depend on both x and y.

visual semantic role labeling, in which a system is given an image and must predict various attributes about it, including what action is depicted and the gender of the person performing the action. In the training data, only 33% of cooking images feature a man, demonstrating a bias towards women cooking. When a system trained on this data makes predictions, it predicts that the person is male in only 16% of images that it predicts involve cooking, thus further amplifying this bias. Zhao et al. (2017) also propose ways to avoid bias amplification by re-calibrating a trained model. When models learn stereotypes, they are essentially performing poorly on the subgroups of people who do not follow these stereotypes (e.g., female surgeons or male cooks). Thus, we can also view these problems as one case of generalization to a minority groups.

2.4.2 Analysis of challenging subsets

Evaluating on subgroups also can serve as an analysis tool to probe whether models have learned a particular skill. Rimell et al. (2009) show that dependency parsers that seem very accurate by standard metrics perform poorly on a subset of the test data that has unbounded dependency constructions. The LAMBADA dataset (Paperno et al., 2016) consists of sentences for which humans are unable to predict the last word given the sentence alone, but can predict the last word given more document context. Evaluating language models on this subset of examples tests the ability of language models to track relevant information across sentence boundaries. Linzen et al. (2016) use number agreement to determine whether language models can correctly process long-range syntactic dependencies. They isolate sentences that have "distractor" nouns between the subject and verb, such as *"The keys to the cabinet* **are** *on the table."* To correctly predict *"are"* instead of *"is,"* the model must realize that the subject is the plural *"keys"* and not the singular *"cabinet"*.

This strategy of identifying challenging subsets works well when data is abundant, in particular for language modeling. However, when there is a smaller pool of available test examples, there may be too few examples from interesting subgroups to conduct a meaningful evaluation. In such settings, perturbing examples can introduce additional challenges while still leveraging prior data collection efforts.

2.4.3 Distributionally robust optimization

Distributionally robust optimization (DRO) is a framework for generalizing to worst-case distribution shifts (Ben-Tal et al., 2013; Duchi and Namkoong, 2018). Let $\mathcal{F}$ denote a set of possible data distributions. The corresponding DRO loss is:

$$L_{\text{DRO}}(\theta) = \max_{q \in \mathcal{F}} \mathop{\mathbb{E}}_{(x,y)\sim q} \left[\ell(x, y; \theta)\right]. \tag{2.8}$$

In other words, the DRO objective tracks the loss on the worst-case distribution in $\mathcal{F}$. In many cases, $\mathcal{F}$ is restricted to contain distributions that are reweighted versions of p (Duchi et al., 2019), though this is not a requirement.[3]

In NLP, the most commonly seen variant of DRO is group DRO (Hu et al., 2018b; Oren et al., 2019), which is the worst-case version of a subgroup distribution. Let g be a function from $\mathcal{X} \times \mathcal{Y}$ to $\{1, \dots, K\}$ for some integer K. g partitions the set of all possible examples into K subgroups. The group DRO loss measures the worst-case loss across all K subgroups:

$$L_{\text{groupDRO}}(\theta) = \max_{1 \le k \le K} \mathop{\mathbb{E}}_{(x,y) \sim p(x,y \mid g(x,y)=k)} \left[\ell(x, y; \theta) \right]. \tag{2.9}$$

This objective has a natural interpretation in terms of combating bias against underrepresented groups: the model must perform well for each group in isolation. Oren et al. (2019) apply group DRO to language modeling, where the groups are defined by a topic model. Encouraging uniformly good loss across these different topics helps the language model generalize better to new domains. Sagawa et al. (2020) show that group DRO can be used to limit overreliance on spurious correlations, such as the fact that negation words are often spuriously indicative of contradictions in natural language inference datasets.

2.5 Extrapolation

Finally, we discuss extrapolation settings, which we use as a catch-all term for settings in which the test distribution diverges from the original distribution in a way that goes beyond example-level perturbations or reweighting. Extrapolation settings involve test examples that tend to differ significantly in some way from any of the examples seen at training time. As a result, success in extrapolation settings is in general more challenging.

2.5.1 Domain adaptation and domain generalization

Domain adaptation. In many NLP applications, training data is available for one or a handful of "source" domains, but useful systems should generalize to other "target" domains as well. A sentiment analysis system may only be trained on certain product categories but would ideally generalize to other products; a part-of-speech tagger may be trained on newswire but would ideally generalize to conversational speech, blogs, or research articles. Domain adaptation studies how to generalize well to one domain when having access mostly to data in another. Here, we focus on domain adaptation under the **covariate shift** assumption, in which $P(y \mid x)$ is identical for the source and target domains. This guarantees that the same mapping from $\mathcal{X}$ to $\mathcal{Y}$ is optimal for both domains. Covariate shift is implicit in many tests of robustness; for instance, when defining a family

[3] In fact, note that the problem of robustness to adversarial example-level perturbations can be written as a DRO problem. Let A denote the set of functions $a : \mathcal{X} \to \mathcal{X}$ such that $a(x) \in N(x)$ for all $x \in \mathcal{X}$. Each $a \in A$ induces a data distribution defined by sampling $(x, y) \sim p(x, y)$, then returning $(a(x), y)$. The DRO objective using this set of distributions is precisely the adversarial example objective from (2.3). Also see Volpi et al. (2018), which shows that DRO using Wasserstein distance-constrained balls results in training on adversarially perturbed examples.

of example-level perturbations, it is important to ensure that the perturbations do in fact preserve the correct label, or else there would be a difference in $P(y \mid x)$ between the original distribution and the distribution with perturbations.

Since source and target domains could be very different, domain adaptation techniques typically assume access to some information about the target domain. In the common semi-supervised setting, the system designer has unlabeled examples (x's only, not y's) from the target domain in addition to labeled examples (both x's and y's) from the source domain. In this setting, one standard idea is to treat the problem as a special case of reweighting (Gretton et al., 2008). The source training data is reweighted so that the distribution over x more closely resembles that of the unlabeled target data, and a classifier is then trained on this reweighted data. Naturally, this method works well if the target domain mostly involves similar inputs as the source domain, just in different proportions.

Another line of work attempts to build domain-independent representations to enable better cross-domain generalization. Blitzer et al. (2006) propose structural correspondence learning, which uses co-occurrence statistics to link domain-specific features with "pivot features," features that appear in the source and target domains and provide similar information about the correct label in both. Through this alignment, patterns learned based on pivot features in the source domain can be ported over to target domain-specific features, even though these features did not occur in the model's labeled training data. More recent work on adversarial domain adaptation shares this same high-level goal of learning domain-independent representations; we discuss these methods, and the comparison with adversarial examples, in Section 2.6.2.

An alternative to the semi-supervised setting is the fully supervised setting, in which a large amount of labeled source data and small amount of labeled target data are available. The key question is how to combine these two data sources to do better than training on either one alone. One surprisingly simple and successful method was proposed by Daumé III (2007). This "frustratingly easy" method simply creates three copies of each feature: one copy fires only on source data, one on target data, and one on both. The copy that fires on both can be used to share domain-general patterns, while the domain-specific copies can be used to additionally learn domain-specific patterns.

Domain generalization. Related to domain adaptation is the more ambitious goal of domain generalization, in which a system is expected to generalize across domains without any foreknowledge of the target domain. Levy et al. (2017) propose a reading comprehension benchmark based on zero-shot relation extraction, in which systems must generalize to new relation types (described by natural language questions) at test time. Due to the proliferation of reading comprehension question answering datasets, recent work has studied how to generalize from one such dataset to another (Yogatama et al., 2019; Talmor and Berant, 2019). While it is not always clear that such generalization is possible, since different datasets involve different types of documents and styles of questions, empirical results indicate that models transfer somewhat reasonably when trained on a mixture of data from multiple datasets (Talmor and Berant, 2019; Fisch et al., 2019).

2.5.2 Stress tests

Stress tests are datasets with new test examples designed to test a particular skill. Unlike test datasets generated by identifying challenging subsets, stress tests do not come from the original data distribution, although failure on a stress test is usually more noteworthy if the types of sentences involved do not differ too much from those used to train the model. The format of natural language inference (NLI) has been particularly popular for developing stress tests, as it is easy to pose many tests of language understanding as NLI problems. Glockner et al. (2018) show that models trained on standard NLI datasets often cannot make simple lexical inferences. For example, it is easy to see that *"The man is holding a saxophone"* contradicts *"The man is holding an electric guitar"* because these are two different instruments, but models often fail at these sorts of examples, as high word overlap between two sentences tends to be correlated with entailment between them. McCoy et al. (2019) study other heuristics that NLI models might employ. For example, it seems natural to assume that a sentence entails all contiguous subsequences contained within it, but *"The doctor near the actor danced"* does not entail *"The actor danced."* They construct test examples that contain counterexamples for three such heuristics, and find that state-of-the-art NLI models generalize very poorly to these test examples. Naik et al. (2018) present a diverse set of stress tests for NLI.

Overall, these stress tests provide strong evidence that models have not really grasped the underlying task of NLI, and instead rely heavily on shallow heuristics. Human-designed stress tests particularly excel at identifying interpretable failure modes. At the same time, it is often unclear how to build models that can generalize well to these stress tests. It is of course possible to add stress test-like examples to the training data, but this is unlikely to help with the next generation of stress tests, leading to a game of "whack-a-mole." The broader challenge is to understand how to build models that succeed on these tests while having as little knowledge as possible about what the test data actually contains. Some recent progress on this front has come from learning to correct for biases in data that make training datasets too easy (Clark et al., 2019a; He et al., 2019).

2.5.3 Unnatural inputs

Finally, machine learning systems may produce highly unexpected outputs when given unnatural inputs. Google Translate has been known to produce strange, biblical sounding "translations" when given nonsensical inputs, such as the same word repeated many times (Christian, 2018).[4] Wallace et al. (2019a) showed the existence of Universal Adversarial Triggers, or short nonsense sequences that reliably trigger undesirable behavior when concatenated with a normal input. For example, prepending *"zoning tapping fiennes"* to a movie review reliably changed a model's prediction from positive to negative. In some sense, these adversarial triggers can be viewed as adversarial perturbations, as they require inserting only a small number of tokens. However, triggers make no attempt

[4]See also https://www.reddit.com/r/TranslateGate/, a subreddit dedicated to these errors.

to appear natural, whereas adversarial perturbations usually are chosen to look like plausible inputs. Feng et al. (2018) showed that models often make confident predictions even when many input words are deleted to the point that a human can no longer tell what the correct answer should be. In these extreme cases, it may sometimes be unclear what the "correct" output should be, but ideally a system would at least recognize these abnormal situations as outliers.

2.6 Other adversarial and robust methods

In the AI community in recent years, the words "adversarial" and "robust" have come to mean different things in different contexts. In this section, we briefly contrast work on robustness to train-test mismatch and adversarial examples with other methods that also use these terms. This thesis focuses on settings where adversaries choose or generate test examples, but other work uses adversarial actors for different goals and considers robustness to other assumptions.

2.6.1 Generative adversarial networks

Generative adversarial networks (GANs) are a popular method for training generative models (Goodfellow et al., 2014; Bowman et al., 2016; Li et al., 2017). The intuition behind GANs is that if a strong classifier cannot distinguish outputs of a generative model from real samples (e.g., human-written sentences), then the generative model must be good at producing realistic outputs. In GAN training, a generative model is thus trained to *maximize* the loss of a discriminative binary classifier that is trained to distinguish generated outputs from real outputs. The GAN setting bears some similarities to the adversarial perturbation setting. In both, an adversary (either the GAN generator or an adversary that perturbs examples) seeks to maximize the loss of a classifier by generating particular types of examples. However, the GAN objective involves the task of discriminating real from synthetic outputs, which itself is not the actual task of interest. This objective is merely used as a way to train the generative model. In contrast, when studying adversarial perturbations, the discriminative model does in fact directly perform the task of interest. The adversarial example "generator" is an instrument for testing and training this model. There is however an interesting connection between GANs and adversarial perturbations: Robey et al. (2020) use the latent space learned by a GAN generator to define possible perturbations of an image, and argue that this provides a more natural range of image variation compared to norm-constrained perturbations.

2.6.2 Domain-adversarial training

In other settings, an adversarial loss is used to ensure that some property of the input is not captured by a learned representation. In domain-adversarial training or adversarial domain adaptation, the goal is to learn a representation that is both useful for a task and independent of the domain of the input, as such a representation enables better generalization across domains (Ganin et al., 2016; Zhang et al., 2017b). In a standard domain adaptation setting with labeled source data and unlabeled target data, this can be achieved by training a model to both predict correctly on source data and thwart a classifier that looks at the model's internal representations and tries to predict whether the input came from the source or target domain. This method shares much more in common with GANs than with work on adversarial examples. As with GANs, the model of interest here is a loss-maximizing adversary to the domain classifier, which is merely an instrument to improve the original model's training. Moreover, the success of domain-adversarial training depends on the loss-*maximizing* adversary succeeding, as this would mean that it has truly learned a domain-independent representation. Closely related ideas are also used in text style transfer, in which representations of text are constructed so as to be independent of the original style (Shen et al., 2017b).

2.6.3 Robust statistics and data poisoning

Robust statistics is a field of statistics concerned with estimating quantities from data when the data deviates in distribution from its assumed distribution. A classic problem in robust statistics is how to estimate parameters of a distribution when data drawn from this distribution is contaminated with a small number of outliers (Tukey, 1960; Huber, 1964). In machine learning, work on data poisoning similarly studies how to estimate model parameters from training data containing adversarially chosen outliers (Biggio et al., 2012; Steinhardt et al., 2017). The focus of these works is on robustness to the violation of a different assumption, namely the assumption that the available data from which parameters are to be estimated (i.e., the training data, in machine learning) consists of i.i.d. draws from the distribution of interest. In this thesis, we do not consider settings in which the training data has been perturbed, and instead focus on perturbations and other shifts applied to the test data.

2.6.4 Improving standard accuracy

While work in computer vision has consistently found a trade-off between robustness to adversarial perturbations and standard accuracy, work in NLP has found that some adversarial training procedures can improve standard accuracy (Miyato et al., 2017; Zhu et al., 2020). Miyato et al. (2017) showed that adversarial training using norm-constrained perturbations of word vectors can improve standard accuracy. Unlike the adversarial perturbations we described earlier, in which each perturbation x' corresponds to a possible input in $\mathcal{X}$, perturbing word vectors directly creates vectors that do not correspond to any real sequence of words. The benefit of adversarial training here is not to defend against any actual adversarially-crafted inputs, but to regularize the model towards being less sensitive to small differences in word vectors, which seems to lead to better standard generalization.

Chapter 3

Certifiably Robust Training

We start by building NLP systems that are robust to adversarial meaning-preserving perturbations. As discussed in Chapter 1, robustness to such perturbations is tied to the broader issue of systematicity in NLP models. Neural NLP models are not innately constrained to generalize systematically, while human language understanding is built on the ability to systematically reuse known vocabulary items in new situations. A neural model may fail to understand a word in context, even if it has seen this word before in very similar contexts. This alone is not necessarily a fatal flaw with neural NLP, but *some* aspect of the way we build neural NLP systems must be responsible for learning the crisp regularities of natural language.

In this chapter, we study a simple setting that is nonetheless challenging enough to expose these generalization failures. 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 3.1 shows an example of these word substitutions. Prior work of Alzantot et al. (2018) and others demonstrated that simply replacing input words with other similar, common words can often trigger mistakes by neural models on standard NLP tasks. In fact, for *most* test inputs, an adversary can find some choice of word substitutions that preserves the meaning of the original sentence but causes the model's prediction to change. For example, our baseline model correctly classifies a review containing the phrase *"I've finally found a movie worse than ..."* as negative, but replacing *"found"* with *"discovered"* flips its prediction to positive. Other adversarial examples require applying similar word substitutions to several words simultaneously.

Word-for-word substitutions clearly account for only some of the systematicity failures that plague neural models—invariance to such substitutions is necessary but clearly not sufficient for systematic understanding. Nonetheless, we find this setting instructive for the purpose of method development, as defending against adversarial word substitutions alone is already a challenging problem. In particular, word substitutions already force us to grapple with the combinatorial nature of language productivity. Since each word in a sentence could be paraphrased in multiple ways,

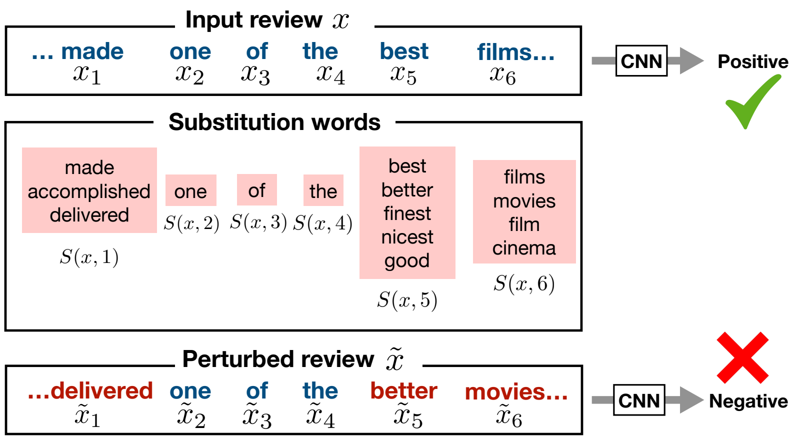


Figure 3.1: Word substitution perturbations in sentiment analysis. For an input x, we consider perturbations $\tilde{x}$, in which *every* word x_i can be replaced with any similar word from the set $S(x, i)$, without changing the original sentiment. Models can be easily fooled by adversarially chosen 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.

a model can only be robust to worst-case word substitutions if it can somehow reason about this combinatorially large space. Existing methods that use heuristic search (Ebrahimi et al., 2018b; Alzantot et al., 2018) are slow and cannot provide guarantees of robustness, since the space of possible perturbations is too large to search exhaustively.

Instead of enumerating the combinatorially many perturbations for each input, we obtain guarantees of robustness 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.

IBP not only provides a way to certify correctness at test time, but also a better way to train models. Other methods for training robust models struggle in our word substitution setting. Data augmentation can sometimes mitigate the effect of adversarial examples (Jia and Liang, 2017; Belinkov and Bisk, 2018; Ribeiro et al., 2018; Liu et al., 2019a), but it is insufficient when considering worst-case perturbations from a very large perturbation space (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.

3.1 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.

3.1.1 Perturbations by word substitutions

Our goal is to build models that are robust to label-preserving perturbations. In this chapter, we focus on perturbations where words of the input are substituted with similar words. Formally, for every word x_i (e.g., *"made"*), we consider a set of allowed substitution words $S(x, i)$, including x_i itself (e.g., {*"made"*, *"accomplished"*, *"delivered"*}). 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:

$$B_{\text{perturb}}(z) = \{(\tilde{x}, y) : \tilde{x}_i \in S(x, i) \quad \forall i\}. \tag{3.1}$$

Figure 3.1 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 3.4.1).

3.1.2 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 (3.1):

$$\mathcal{F}(z, \theta) = \{f(\tilde{z}, \theta) : \tilde{z} \in B_{\text{perturb}}(z)\}. \tag{3.2}$$

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

$$\begin{aligned} \max \mathcal{O}^{\text{0-1}}(z, \theta) = 0 &\Rightarrow \max \mathcal{F}^{\text{0-1}}(z, \theta) = 0 \\ &\Leftrightarrow \text{robust at } z. \end{aligned} \tag{3.3}$$

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

3.2 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 3.4.2 shows that training networks to minimize the upper bound on f makes these bounds much tighter (Gowal et al., 2019; Raghunathan et al., 2018). Figure 3.2 provides an overview of how IBP can be used to certify robustness.

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 3.2.1), then show how to compute $\mathcal{O}^i$ from $\mathcal{O}^{\text{dep(i)}}$ for elementary operations used in standard NLP models (Section 3.2.2). Finally, we use these bounds to certify robustness of model predictions (Section 3.2.3) and train

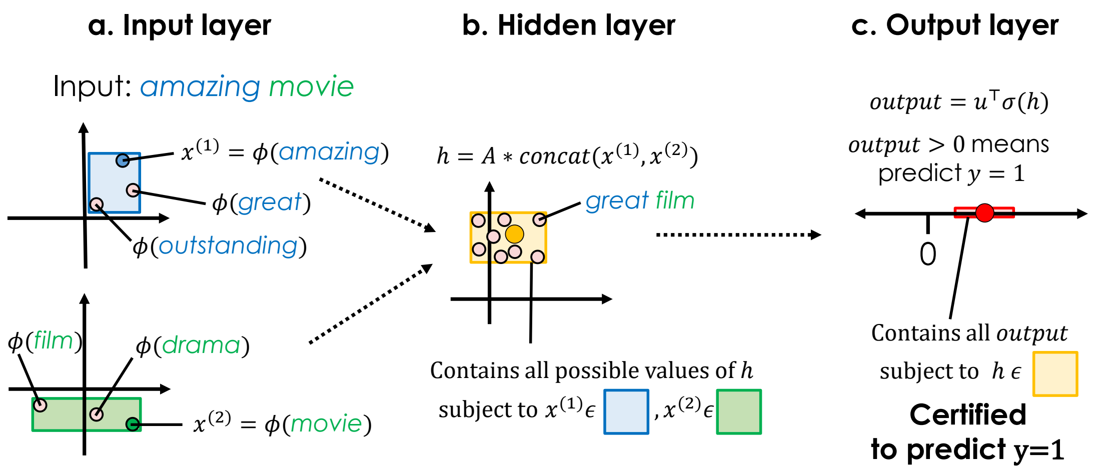

a. Input layer
b. Hidden layer
c. Output layer
Input: amazing movie
$x^{(1)} = \phi$(amazing)
ϕ(great)
ϕ(outstanding)
ϕ(film)
ϕ(drama)
$x^{(2)} = \phi$(movie)
$h = A * concat(x^{(1)}, x^{(2)})$
great film
Contains all possible values of h subject to $x^{(1)} \epsilon$ 🟦 , $x^{(2)} \epsilon$ 🟩
$output = u^{\top}\sigma(h)$
$output > 0$ means predict $y = 1$
0
Contains all $output$ subject to $h\ \epsilon$ 🟨
Certified to predict y=1

Figure 3.2: Computing a certificate of robustness with interval bound propagation. (a) A neural NLP model typically embeds an input sentence as a sequence of word vectors (blue and green circles). In our setting, an adversary may replace each word with a related word, which has a different word vector (pink circles). We compute interval bounds (blue and green boxes) that circumscribe the possible word vectors that the adversary can inject. (b) The model then applies a series of linear algebraic operations to these word vectors (yellow circle). Since these operations typically combine information from multiple word vectors, the set of all possible values for these hidden (intermediate) layers is exponentially large in the size of the input ($8 = 3 \times 3 - 1$ pink dots). IBP formulas construct an interval bound around these possible values, based on the interval bounds from the previous layer (yellow box). These interval bounds propagate through the computation graph of the model. (c) For binary classification, the model's output (red dot) is a single real number, where the prediction is the positive label iff the output > 0. If the interval bound for this output (red interval) is entirely to the right of 0, the model is guaranteed to always predict the positive label; this is a certificate of robustness on this example, since the correct label is positive.

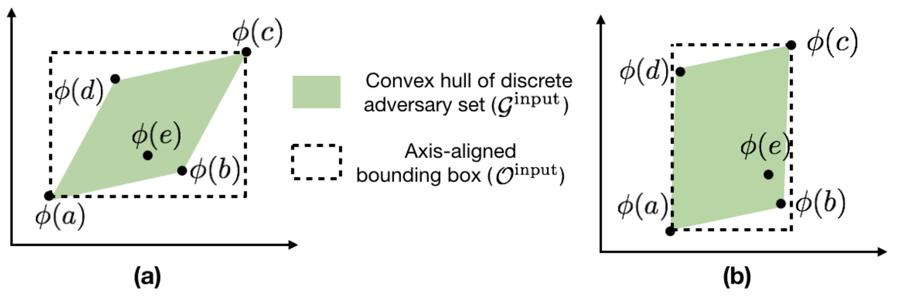
robust models (Section 3.2.4).

3.2.1 Bounds for the input layer

Previous work (Gowal et al., 2019) 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 $x = x_1, \dots, x_L$, we assume that the model embeds each word as

$$g^{\text{input}} = [\phi(x_1), \dots, \phi(x_L)] \in \mathbb{R}^{L \times d}, \tag{3.4}$$

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 $j \in \{1, \dots, d\}$, we can obtain



$\phi(a)$
$\phi(b)$
$\phi(c)$
$\phi(d)$
$\phi(e)$
Convex hull of discrete adversary set ($\mathcal{G}^{\text{input}}$)
Axis-aligned bounding box ($\mathcal{O}^{\text{input}}$)
(a)
(b)

Figure 3.3: 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.

an interval bound $\mathcal{O}_{ij}^{\text{input}} = [\ell_{ij}^{\text{input}}, u_{ij}^{\text{input}}]$ for g_{ij}^{input} by computing the smallest axis-aligned box that contains all the word vectors:

$$\ell_{ij}^{\text{input}} = \min_{w \in S(x,i)} \phi(w)_j, \tag{3.5}$$

$$u_{ij}^{\text{input}} = \max_{w \in S(x,i)} \phi(w)_j. \tag{3.6}$$

Figure 3.3 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 3.3.2 discusses modeling choices to make this box tighter.

3.2.2 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. (2019) show how to efficiently compute interval bounds for affine transformations (i.e., linear layers) and monotonic elementwise nonlinearities; we include these below for completeness. 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 $\mathcal{O}^{\text{res}} = [\ell^{\text{res}}, u^{\text{res}}]$.

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}:

$$\begin{aligned} u^{\text{res}} &= \underbrace{0.5(a + |a|)^\top}_{\text{positive}} u^{\text{dep}} + \underbrace{0.5(a - |a|)^\top}_{\text{negative}} \ell^{\text{dep}} + b \\ &= \mu + r, \end{aligned} \tag{3.7}$$

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}.

$$\ell^{\text{res}} = \sigma(\ell^{\text{dep}}), \quad u^{\text{res}} = \sigma(u^{\text{dep}}). \tag{3.8}$$

Softmax layer. The softmax function is often used to convert activations into a probability distribution, e.g., for attention. Gowal et al. (2019) 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:

$$u^{\text{res}} = \frac{\exp(u_c^{\text{dep}})}{\exp(u_c^{\text{dep}}) + \sum\limits_{j \neq c} \exp(\ell_j^{\text{dep}})}. \tag{3.9}$$

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.1).

Element-wise multiplication and dot product. Models like LSTMs incorporate gates which perform element-wise multiplication of two activations. Let $z^{\text{res}} = z_1^{\text{dep}} \cdot z_2^{\text{dep}}$ 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,

$$\begin{aligned}
\mathcal{C} &= \{\ell_1^{\text{dep}} \cdot \ell_2^{\text{dep}}, \ell_1^{\text{dep}} \cdot u_2^{\text{dep}}, u_1^{\text{dep}} \cdot \ell_2^{\text{dep}}, u_1^{\text{dep}} \cdot u_2^{\text{dep}}\}, \\
\ell^{\text{res}} &= \min\left(\mathcal{C}\right), \\
u^{\text{res}} &= \max\left(\mathcal{C}\right).
\end{aligned} \tag{3.10}$$

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.

3.2.3 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 $s(x)$, where $s(x)_j$ denotes the logit associated with label j.[2] For standard loss functions, we can represent this function in terms the elementary functions described in Section 3.2.2.

1. Zero-one loss: $f(z, \theta) = \mathbb{I}[s(x)_y - \max_{y' \neq y} s(x)_{y'} > 0]$ involves a max operation, subtraction (an affine operation), and a (monotonic) step function.

2. Cross entropy: For multi-class classification, $f(z, \theta) = \text{softmax}(s(x))$. 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.

3.2.4 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., 2019). 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},

$$\min_\theta \sum_{z \in D} (1 - \kappa) f(z, \theta) + \kappa\, u^{\text{final}}(z, \theta), \tag{3.11}$$

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

[1] The interval bound formula for affine layers from Gowal et al. (2019) do not apply to the dot product as described here, because for the dot product both vectors have associated bounds; in an affine layer, the input has bounds, but the weight matrix is fixed.

[2] Binary classification with labels $y = 0$ and $y = 1$ corresponds to a special case where $s(x)_1 = -s(x)_0$.

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 (3.11) can be optimized via backpropagation.

Gowal et al. (2019) 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.

3.3 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 3.2.

3.3.1 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.

3.3.2 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.

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 $\phi^{\text{pre}}(w)$:

$$\phi(w) = \text{ReLU}(g^{\text{word}}(\phi^{\text{pre}}(w))), \tag{3.12}$$

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 (3.5).[3] 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 3.3a), compared to $\phi(w)$ (e.g., Figure 3.3b). Section 3.4.7 confirms the importance of adding g^{word}. We use 300-dimensional GloVe vectors (Pennington et al., 2014) as our $\phi^{\text{pre}}(w)$.

3.4 Experiments

3.4.1 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).[4] The neighbors must also lie within some Euclidean distance threshold. They

[3] Equation (3.5) must be applied before the model can combine information from multiple words, but it can be delayed until after processing each word independently.

[4] 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.

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 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 instead define allowed substitutions relative to the original sentence x, and disallow repeated substitutions. More precisely, we 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

$$\begin{aligned} \log P(x_{i-W:i-1}, \tilde{x}_i, x_{i+1:i+W}) \geq \\ \log P(x_{i-W:i+W}) - \delta \end{aligned} \tag{3.13}$$

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 $\delta = 5$. Second, we also use a faster language model that allows us to query longer contexts.[5] This language model 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 $W = 1$. 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.

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 3.2) 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, 2018; 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

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

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 3.2.3 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 do not run training for a set number of epochs but do early stopping on the development set instead. For normal training, we do early stopping on normal development set accuracy. For training with data augmentation, we do early stopping on the accuracy on the augmented development set. For certifiably robust training, we do early stopping on the certifiably robust accuracy on the development set. We use the Adam optimizer (Kingma and Ba, 2015) to train all models. We tune hyperparameters on the development set for each dataset. Hyperparameters are reported in Table 3.1.

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,

System	κ	Learning rate	Dropout prob.	Weight decay	Gradient norm clip value	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 3.1: Training hyperparameters for training the models. The same hyperparameters were used for all training settings (standard training, data augmentation, robust training).

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). 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.

3.4.2 Main results

Table 3.2 and Table 3.3 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,[6] 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 attack,

[6]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 3.2: 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).

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 Section 3.4.8, we show on synthetic data that robustly trained LSTMs can learn long-range dependencies.

3.4.3 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.,

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

2018). For example, the state-of-the-art robust model on CIFAR10 (Zhang et al., 2019b) 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 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 3.4 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.

3.4.4 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

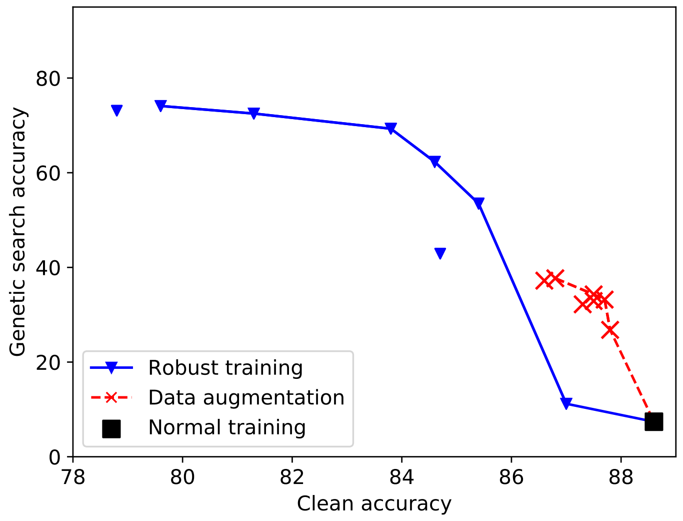


Figure 3.4: Trade-off between clean accuracy and genetic attack accuracy for CNN models on IMDB. Data augmentation cannot achieve high robustness. Certifiably robust training yields much more robust models, though at the cost of some clean accuracy. Lines connect Pareto optimal points for each training strategy.

those same examples took over 3 hours.

3.4.5 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 3.5. 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

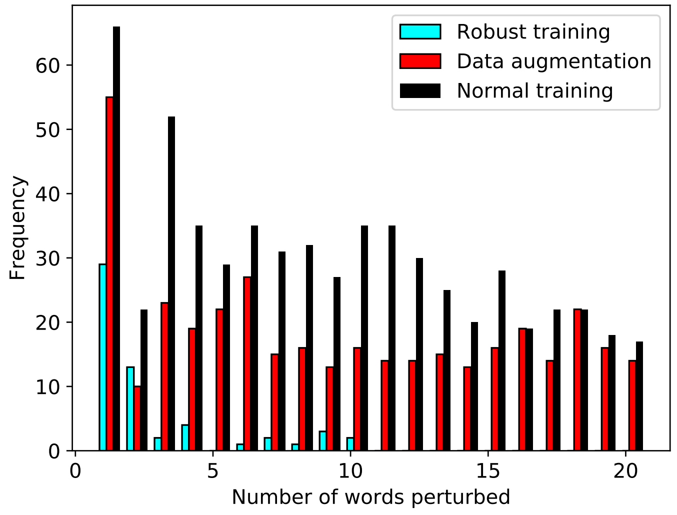


Figure 3.5: Number of words perturbed by the genetic attack to cause errors by CNN models on 1000 IMDB development set examples. Certifiably robust training reduces the effect of many simultaneous perturbations.

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.

3.4.6 Training schedule

Table 3.4 shows the effect of holding ϵ or κ fixed during training, as suggested by Gowal et al. (2019), on 1000 randomly chosen examples from the IMDB development set. Fixing $\epsilon = 1$ during training led to a 5 point reduction in certified accuracy for the CNN, demonstrating that slowly increasing ϵ is important. On the other hand, we found that holding κ to the fixed value κ^* did not hurt accuracy, and in fact may be preferable, 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.

3.4.7 Word vector analysis

We determined the importance of the extra feedforward layer g^{word} that we apply to pre-trained word vectors, as described in Section 3.3.2. 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 formula 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

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

Table 3.4: Effects of holding ϵ and κ fixed during training. All numbers are on 1000 randomly chosen IMDB development set examples.

word vectors and applying interval bounds *after* is crucial for robust training.

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

3.4.8 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 $x_2, \dots, x_{L-1}$ 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.

3.5 Discussion

State-of-the-art NLP models are accurate on average, but they still have significant gaps in their abilities, as demonstrated by adversarial examples. Certifiably robust training provides a general, principled mechanism to avoid such gaps 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.

The key challenge we grappled with was combinatorial explosion resulting from the independent application of multiple local paraphrase rules. While we focused on word substitutions defined by word vector similarity, local paraphrase rules come in many forms. Bhagat and Hovy (2013) categorize local paraphrasing operations found in existing corpora. Overall, the most common ones were synonym substitutions, function word variation, and substitutions that rely on external knowledge. Our work focuses primarily on the first category. Function word variations include some relatively simple substitutions (e.g., *"results of the competition"* becomes *"results for the competition"*) but also cases where changes in light verbs induce other changes (e.g., *"Pat showed a nice demo"* becomes *"Pat's demo was nice"*). Substitutions that rely on external knowledge include metonymy, such as *"Bush"* being a stand-in for *"The [American] government"*.

On the data side, it is non-trivial to generate all of these substitutions automatically. For evaluation purposes, it would be interesting to have humans provide a diverse set of local paraphrase operations for different parts of a sentence. Models would be evaluated on whether they can always predict the correct answer on all paraphrase generated by a combination of these local rules. This type of data would presumably be expensive to collect, so training methods might instead leverage

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

paraphrase generation methods. This setting not only requires robustness to a more diverse set of paraphrase operations, but it also introduces a new type of generalization challenge, since the system designer no longer knows ahead of time exactly what paraphrase rules are possible for any given sentence.

Broadening the types of paraphrase rules we consider also poses a new technical challenge. The common paraphrase rules from Bhagat and Hovy (2013) include instances of word insertions, deletions, and even rearrangements (e.g., when *"nice"* moved from prepositive to postpositive). The methods presented in this chapter are restricted to word substitutions, which keep the size of the input constant; insertions, deletions, and rearrangements complicate this picture. Going forward, it seems likely that model architecture will have to play a much more important role, as only some architectures will be well-suited to learning invariances to these operations. Thus, we may be able to unify work on adversarial robustness with work on designing model architectures that intrinsically capture the structure of language. For example, while CNNs work well enough for many text classification datasets, they are unappealing models of language as they are fundamentally unable to keep track of long-range context. RNNs have long been considered more natural for handling linguistic data. RNNs also seem comparatively better-equipped to handle insertions and deletions; for example, an RNN could learn to ignore certain inserted words by simply not changing the hidden state for one timestep. Robustness to phrase-level transformations may also pair well with models that make use of explicit phrase structure, such as tree-recursive neural networks (Socher et al., 2013). Invariance to phrase-level paraphrase operations could be enforced as a property of the phrase-level neural representations that these models build. One barrier is that we would require access to tree structures generated in some way that is itself robust to perturbations. This in turn motivates the use of models that jointly learn phrase structure and neural representations of phrases, such as DIORA (Drozdov et al., 2019).

We can also move beyond paraphrase, which is roughly bidirectional entailment, to unidirectional entailment. For many NLP tasks, such as sentiment analysis or relation extraction, if a sentence x' entails sentence x, and y is the correct label for x, then x' should also have label y.[8] Prior work on natural logic in NLI has studied how word substitutions affect entailment relationships between sentences, and thus could be used to define additional perturbations (MacCartney and Manning, 2008; Angeli and Manning, 2014). Natural logic approaches typically use a search procedure to find a chain of valid entailments; a neural model trained to respect natural logic rules could learn to efficiently approximate the result of this search.

[8]For relation extraction, this is generally true as long as y is not the "no relation" label. Depending on the schema being used, it may also be incorrect to use a general label (e.g., "employee of") when a more specific label is also available and correct (e.g., "president of").

Chapter 4

Robust Encodings

In the previous chapter, we proposed certifiably robust training as a way to ensure invariance to word-level perturbations. While this method greatly improves adversarial accuracy, the benefits of certifiably robust training are limited to one task and one model at a time. Robustness to perturbations is often a task-agnostic goal: across a variety of NLP tasks, models should be invariant to many of the same kinds of perturbations. For example, for many tasks, paraphrasing the input or adding typos does not alter the correct label. Certifiably robust training does not reflect this task-agnosticity. Robustly training the same model on a new task requires rerunning the training procedure essentially from scratch; no significant work can be shared across tasks.

A similar problem arises when attempting to reuse work across different model architectures. Replacing an old model architecture with a new one again requires rerunning certifiably robust training from scratch. Perhaps even more problematic is the fact that while interval bound propagation can certify robustness of many basic neural models, it places some strong constraints on model architecture. In particular, large-scale pre-trained Transformers like BERT (Devlin et al., 2019) pose numerous problems. First, BERT uses non-monotonic activation functions, which lack tight interval bounds. Second, BERT uses subword tokenization, meaning that a word substitution could replace a single-token word with a multi-token word or vice versa, and our method cannot handle these changes in sequence length. Finally, BERT is simply much larger than the models used in the previous section. Interval bounds tend to become looser as the depth of the network increases.

Ideally we would like a "robustness" module that we could reuse across multiple tasks and with arbitrary model architectures. Indeed, *reusable* components have driven recent progress in NLP. For example, word vectors are a universal resource that are constructed once, then used for many different tasks. The exact same pre-trained BERT model can be easily and efficiently fine-tuned to perform many different tasks, and can be embedded into many different model architectures. Can we build a reusable robust defense that can easily work with complex, state-of-the-art architectures like BERT? Pruthi et al. (2019), which uses a typo corrector to defend against adversarial typos,

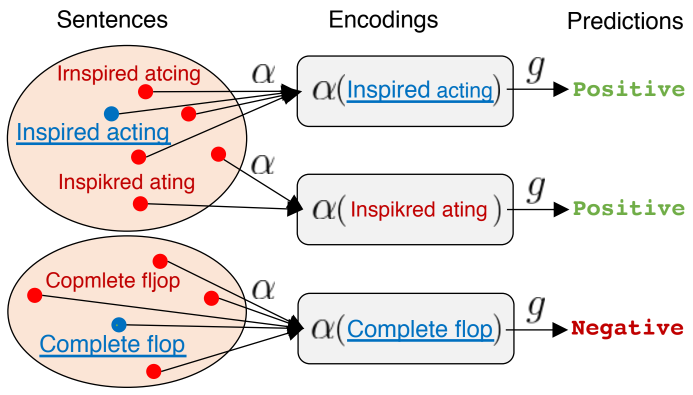


Figure 4.1: Example of a defense using RobEn. An adversary can perturb sentences (blue, underlined) to many different perturbations (red, not underlined) within the attack surface (red, ovals). We define an encoding function α such that each perturbation of the input sentences maps to one of a few encodings (grey, rounded rectangles). We can then use any model g to make predictions given the encodings.

presents one such reusable defense: their typo corrector is trained once, then reused across different tasks. However, we find that current typo correctors do not perform well against even heuristic adversarial attacks, limiting their applicability.

In this chapter, we propose *robust encodings* (RobEn), a framework to construct encodings that can make systems using *any* model architecture provably robust to adversarial perturbations. The core component of RobEn is an *encoding function* that maps sentences to a smaller discrete space of encodings, which are then used to make predictions. We define two desiderata that a robust encoding function should satisfy: stability and fidelity. First, to encourage consistent predictions across perturbations, the encoding function should map all perturbations of a sentence to a small set of encodings (stability). Simultaneously, encodings should remain informative enough that models trained using encodings still perform well on unperturbed inputs (fidelity). Figure 4.1 illustrates a robust encoding function that is stable with respect to typos while retaining enough fidelity to distinguish different unperturbed inputs. Since robust encodings collapse a (potentially very large) space of perturbations to a small number of encodings, we can compute the *exact* robust accuracy tractably; in comparison, interval bound propagation only gives a possibly loose lower bound on robust accuracy. Moreover, these encodings can make any downstream model robust, including state-of-the-art transformers like BERT, and can be reused across different tasks.

We validate our task-agnostic approach by constructing a single encoding function for English that defends against adversarial typos, as robustness to typos is nearly universally desirable in NLP tasks. In particular, we allow an attacker to add independent edit-distance-one typos to each word in an input sentence, resulting in exponentially more possible perturbations than previous work (Pruthi et al., 2019; Huang et al., 2019). As with word substitutions in the previous chapter, typos

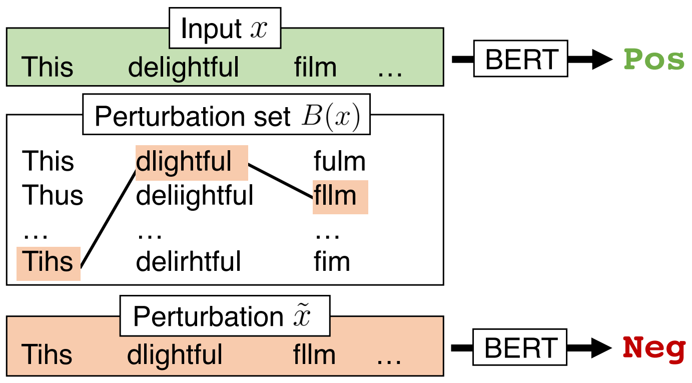


Figure 4.2: Attack model allowing independent perturbations of each token. The original input, x is classified by the model as positive while the perturbation $\tilde{x}$ =, obtained by choosing perturbations of *"This"*, *"delightful"*, and *"film"* independently, is classified as negative. Independent perturbations of each word results in an exponentially large perturbation space $B(x)$.

induce a combinatorial explosion of possible perturbations, as multiple words in a sentence may be independently corrupted; a robust encoding function must collapse this exponentially large space to a very small number of encodings. We consider a natural class of *token-level encodings*, which are obtained by encoding each token in a sentence independently. This structure allows us to express stability and fidelity in terms of a token-level clustering objective, which we optimize.

Empirically, our instantiation of RobEn achieves state-of-the-art robust accuracy against adversarial typos across six classification tasks from the GLUE benchmark (Wang et al., 2019b). Our best system, which combines RobEn with a BERT classifier (Devlin et al., 2019), achieves an average robust accuracy of 71.3% across the six tasks. In contrast, a state-of-the-art defense that combines BERT with a typo corrector (Pruthi et al., 2019) gets 35.3% accuracy when adversarial typos are inserted, and a standard data augmentation defense gets only 12.2% accuracy.

4.1 Setup

Tasks. We consider NLP tasks that require classifying textual input $x \in \mathcal{X}$ to a class $y \in \mathcal{Y}$. For simplicity, we refer to inputs as sentences. Each sentence x consists of tokens $x_1, \dots, x_L$ from the set of all strings $\mathcal{T}$. Let p_{task} denote the distribution over inputs and labels for a particular task of interest.[1] The goal is to learn a model $f : \mathcal{X} \to \mathcal{Y}$ that maps sentences to labels, given training examples $(x, y) \sim p_{\text{task}}$.

Attack surface. We consider an attack surface in which an adversary can perturb each token x_i of a sentence to some token $\tilde{x}_i \in B(x_i)$, where $B(x_i)$ is the set of valid perturbations of x_i (including

[1]This is the same as p from Chapter 2, but in this chapter we use p_{task} to distinguish from other distributions to be discussed later.

x_i itself). For example, if x_i is the word *"acting,"* $B(x_i)$ could be a set of allowed typos of x_i, e.g., {*"acting"*, *"ating"*, *"atcing"*, ...}. We define $B(x)$ as the set of all valid perturbations of the set x, where every possible combination of token-level typos is allowed:

$$B(x) = \{(\tilde{x_1}, \ldots, \tilde{x_L}) \mid \tilde{x_i} \in B(x_i) \, \forall \, i\} \tag{4.1}$$

The size of the attack surface $|B(x)|$ grows exponentially with respect to number of input tokens, as shown in Figure 4.2. In general $x_i \in B(x_i)$, so some words could remain unperturbed.

Model evaluation. In this chapter, we use three evaluation metrics for any given task.

First, we evaluate a model on its *standard accuracy* on the task:

$$\mathrm{acc}_{\mathrm{std}}(f) = \mathop{\mathbb{E}}_{(x,y)\sim p_{\mathrm{task}}} \left[\mathbb{I}[f(x) = y]\right]. \tag{4.2}$$

Next, we are interested in models that also have high *robust accuracy*, the fraction of examples (x, y) for which the model is correct on all valid perturbations $\tilde{x} \in B(x)$ allowed in the attack model:

$$\mathrm{acc}_{\mathrm{rob}}(f) = \mathop{\mathbb{E}}_{(x,y)\sim p_{\mathrm{task}}} \left[\min_{\tilde{x}\in B(x)} \mathbb{I}\left[f(\tilde{x}) = y\right]\right]. \tag{4.3}$$

It is common to instead compute accuracy against a heuristic attack a that maps clean sentences x to perturbed sentences $a(x) \in B(x)$.

$$\mathrm{acc}_{\mathrm{attack}}(f; a) = \mathop{\mathbb{E}}_{(x,y)\sim p_{\mathrm{task}}} \left[\mathbb{I}[f(a(x)) = y]\right]. \tag{4.4}$$

Typically, $a(x)$ is the result of a heuristic search for a perturbation $\tilde{x} \in B(x)$ that f misclassifies. Note that $\mathrm{acc}_{\mathrm{attack}}$ is a (possibly loose) upper bound of $\mathrm{acc}_{\mathrm{rob}}$ because there could be perturbations that the model misclassifies but are not encountered during the heuristic search (Athalye et al., 2018).

Recall that since robust accuracy is generally hard to compute, in Chapter 3 we computed certified accuracy, which is a potentially conservative lower bound for the true robust accuracy. In this chapter, robust encodings enable us to tractably compute the exact robust accuracy.

4.2 Robust Encodings

We now introduce *robust encodings* (RobEn), a framework for constructing encodings that are reusable across many tasks, and pair with arbitrary model architectures. In Section 4.2.1 we describe the key components of RobEn, then in Section 4.2.2 we highlight desiderata RobEn should satisfy.

4.2.1 Encoding functions

A classifier $f_\alpha : \mathcal{X} \to \mathcal{Y}$ using RobEn decomposes into two components: a *fixed* encoding function $\alpha : \mathcal{X} \to \mathcal{Z}$, and a model that accepts encodings $g : \mathcal{Z} \to \mathcal{Y}$.[2] For any sentence x, our system makes the prediction $f_\alpha(x) = g(\alpha(x))$. Given training data $\{(x_i, y_i)\}_{i=1}^n$ and the encoding function α, we learn g by performing standard training on encoded training points $\{(\alpha(x_i), y_i)\}_{i=1}^n$.

Given an example (x, y), computing the robust accuracy of f_α on (x, y) is straightforward and efficient if the set of possible encodings $\alpha(\tilde{x})$ for some perturbation $\tilde{x} \in B(x)$ is small. In this case, we can compute $\text{acc}_{\text{rob}}(f_\alpha)$ efficiently by generating this set of possible encodings, and feeding each to g. Note that this procedure is completely indifferent to the architecture of g. From these computational considerations alone, we see that we would prefer a α that ensures that the set of all $\alpha(\tilde{x})$'s is small for most plausible inputs x; this will indeed be a focus of the coming sections.

4.2.2 Encoding function desiderata

To have high robust accuracy on a task distribution p_{task}, a classifier f_α that uses α should make consistent predictions on all $\tilde{x} \in B(x)$, the set of points described by the attack surface, for most $(x, y) \sim p_{\text{task}}$. It should also have high standard accuracy on unperturbed inputs drawn from p_{task}. We term the former property **stability**, and the latter **fidelity**.

As stated above, both stability and fidelity are task-dependent notions. Nevertheless, we can define appropriate task-agnostic versions of both stability and fidelity, such that if α has high task-agnostic stability and fidelity, it tends to have high task-specific stability and fidelity on many natural NLP tasks. Section 4.3 will give concrete instantiations of these task-agnostic definitions in the context of robustness to typos.

Central to our task-agnostic definitions will be a corpus distribution p_{corp} over sentences. Ideally, p_{corp} should include sentences that show up in most reasonable task distributions p_{task}. In practice, we will choose p_{corp} to be the empirical distribution from a large, diverse corpus of text.

Stability. For an encoding function α and distribution over inputs p_{corp}, the (task-agnostic) stability $\text{Stab}(\alpha)$ measures how often α maps sentences $x \sim p_{\text{corp}}$ to the same encoding as all of their perturbations. Note that if we could guarantee this property for p_{task} instead of p_{corp}, then any f_α that uses α is guaranteed to be invariant to perturbations for most inputs.

Fidelity. In a task-specific sense, an encoding function α has high fidelity if models that use α can still achieve high standard accuracy. Unfortunately, this notion of fidelity is highly task-dependent. For instance, for many tasks it is acceptable to collapse sentences that are paraphrases to the same representation, as these should always receive the same label. However, the task of author

[2] We can set $\mathcal{Z} \subseteq \mathcal{X}$ when g accepts sentences.

identification makes heavy use of stylistic cues, and two sentences with equivalent meaning may be indicative of different authors.

We adopt a task-agnostic approximation of fidelity that empirically aligns with the goal of high standard accuracy on the suite of NLP tasks we use for evaluation. From an information theoretic perspective, α can be used to achieve high standard accuracy (relative to the optimal predictor) as long as it does not conflate different sentences that should receive different labels. We could guarantee this if we could impose the stronger requirement that if two sentences (of any label) are randomly sampled from p_{task}, α will map them to different encodings with high probability. To turn this into a task-agnostic notion, we replace p_{task} with sentences drawn from p_{corp}, just as we did for stability. Thus, we say that α has high (task-agnostic) fidelity $\text{Fid}(\alpha)$ if it maps different sentences sampled from p_{corp} to different encodings with high probability.

Trade-off. Stability and fidelity are inherently competing goals. An encoding function that maps every sentence to the same encoding trivially maximizes stability, but is useless for any non-trivial classification task. Conversely, fidelity is maximized when every input is mapped to itself, which has very low stability. In the following section, we construct an instantiation of RobEn that balances stability and fidelity when the attack surface consists of typos.

4.3 Robust Encodings for Typos

We now concretize the above discussion in pursuit of robustness to *adversarial typos*, where an adversary can add typos to each token in an English sentence (see Figure 4.2). Since this attack surface is defined at the level of tokens, we restrict attention to token-wise encoding functions that encode each token of a sentence independently. While token-wise encodings do not use contextual information, recall that these encodings will be fed to a learned model g that can make heavy use of context.

First, we will reduce the problem of generating token-level encodings to assigning vocabulary words to clusters (Section 4.3.1). Next, we will build intuition about stability and fidelity of clusterings through an illustrative example (Section 4.3.2). Armed with this intuition, we will examine how a token-level encoding function should handle out-of-vocabulary tokens (Section 4.3.3). Once we have decided how to handle out-of-vocabulary tokens, we will finally be ready to explicitly define the stability and fidelity of different possible clusterings. We will propose two types of token-level robust encodings: connected component encodings (Section 4.3.4), which optimize primarily for stability, and agglomerative cluster encodings (Section 4.3.5), which optimize for a balance of stability and fidelity. Finally, we will close the loop by specifying some details about the conversion from a clustering to an encoding (Section 4.3.6).

4.3.1 Encodings as clusters

We construct an encoding function α that encodes x token-wise. Formally, α is defined by a token-level encoding function π that maps each token $x_i \in \mathcal{T}$ to some *encoded token* $\pi(x_i) \in \mathcal{Z}_{\text{Tok}}$:

$$\alpha(x) = [\pi(x_1), \pi(x_2), \ldots \pi(x_L)]. \tag{4.5}$$

Since π must be prepared to handle typos, the domain of π is all *tokens* $\mathcal{T}$, which includes many strings that are not English words. Naturally, the behavior of π on English words, as opposed to non-word strings, is of particular importance. Define the vocabulary $V = \{w_1, \ldots, w_N\} \subseteq \mathcal{T}$ as the N most frequent tokens in the corpus distribution p_{corp}. For simplicity, we call elements of V *words*, and tokens that are perturbations of some word *typos*. We can (trivially) decompose π into two parts—one function π_V that operates on vocabulary words, and a second function π_{OOV} that operates on all other strings:

$$\pi(x_i) = \begin{cases} \pi_V(x_i) & x_i \in V \\ \pi_{\text{OOV}}(x_i) & x_i \notin V \end{cases}. \tag{4.6}$$

In Section 4.3.3, we will show how to choose a good π_{OOV} given a choice of π_V. The thornier question is how to choose π_V.

Plan for choosing π_V. Choosing π_V can itself be decomposed into two steps. The first, more involved step is to decide which words in V should be mapped by π_V to the same encoded token. We can view this step as assigning each word in V to one of k clusters $C_1, \ldots, C_k \subseteq V$. The second step is to assign one encoded token to each of these clusters.

The first step is critical, as it determines what information about the input is available to the downstream model g (Section 4.2.1). If π maps too many words to the same encoded token, they become indistinguishable to g, so it will be unable to classify accurately. At the same time, mapping some words to the same encoded token greatly helps stability. Section 4.3.4 and Section 4.3.5 will describe two ways to cluster V.

The second step is comparatively less consequential. In principle we could simply use a one-hot vector for each cluster. However, since we wish to use a pre-trained language model like BERT as g, we choose encoded tokens that can be easily fed to BERT. Section 4.3.6 will discuss this choice.

4.3.2 Simple example

Recall the two competing desiderata described in Section 4.2.2: stability and fidelity. How do these interact in the context of typos? Here, we present a simple example that provides intuition as to how a token-level encoding function can achieve both high stability and high fidelity. We will formally

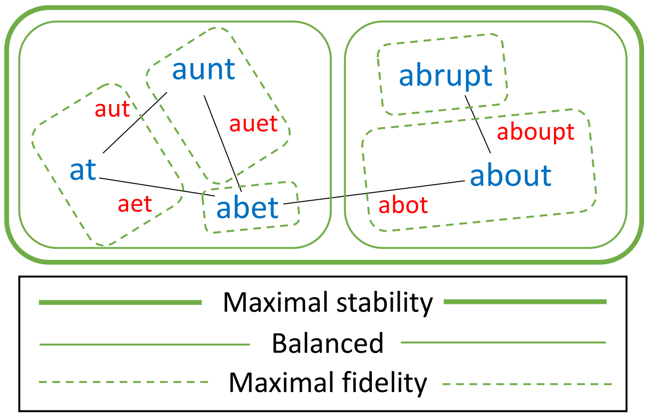


Figure 4.3: Visualization of three different encodings. Vocabulary words (large font, blue) share an edge if they share a common perturbation (small font, red). The maximal stability cluster (thick solid line) clusters identically, the maximal fidelity clusters (thin dotted line) encodes all words separately, while the balanced clusters (thin solid line) trade off the two.

define the stability and fidelity of a clustering in Section 4.3.3 and Section 4.3.5.

Consider the five words (large font, blue) in Figure 4.3, along with potential typos (small font, red). We illustrate three different clusterings as boxes around tokens in the same cluster. We may put all words in the same cluster (thick box), each word in its own cluster (dashed boxes), or something in between (thin solid boxes). For now, we group each typo with a word it could have been perturbed from (we will discuss this further in Section 4.3.3).

To maximize stability, we need to place all words in the same cluster. Otherwise, there would be two words (say *"at"* and *"aunt"*) that could both be perturbed to the same typo (*"aut"*) but are in different clusters. Therefore, *"aut"* cannot map to the same encoded token as both the possible vocabulary words. At the other extreme, to maximize fidelity, each word should be in its own cluster. Both mappings have weaknesses: the stability-maximizing mapping has low fidelity since all words are identically encoded and thus indistinguishable, while the fidelity-maximizing mapping has low stability since the typos of words *"aunt"*, *"abet"*, and *"abrupt"* could all be mapped to different encoded tokens than that of the original word.

The clustering represented by the thin solid boxes in Figure 4.3 balances stability and fidelity. Compared to encoding all words identically, it has higher fidelity, since it distinguishes between some of the words (e.g., *"at"* and *"about"* are encoded differently). It also has reasonably high stability, since only the infrequent *"abet"* has typos that are shared across words and hence are mapped to different encoded tokens. This clustering thus represents a desirable compromise between stability and fidelity.

4.3.3 Encoding out-of-vocabulary tokens

Given a fixed clustering of V, we now study how to map out-of-vocabulary tokens, including typos, to encoded tokens without compromising stability. Once we square away how to handle out-of-vocabulary tokens for any chosen clustering, we will finally be in a position to define what makes a good clustering.

Stability. Stability measures the extent to which typos of words map to different encoded tokens. We formalize this by defining the set of tokens that some typo of a word w could map to, $B_\pi(w)$:

$$B_\pi(w) = \{\pi(\tilde{w}) \mid \tilde{w} \in B(w)\}, \tag{4.7}$$

where $B(w)$ is the set of allowable typos of w. Since we care about inputs drawn from p_{corp}, we define Stab on the clustering C using $\rho(w)$, the normalized frequency of word w based on p_{corp}:

$$\text{Stab}(C) = -\sum_{i=1}^{N} \rho(w_i)|B_\pi(w_i)|. \tag{4.8}$$

For a fixed clustering, the size of $B_\pi(w)$ depends on where π_{OOV} maps typos that w shares with other words; for example in Figure 4.3, *"aet"* could be a perturbation of both *"at"* and *"abet"*. If we map the typo to the encoded token of *"at"*, we increase the size of $B_\pi(\text{"abet"})$ and vice-versa. In order to keep the size of $B_\pi(w)$ smaller for the more frequent words and maximize stability, we map a typo to the same encoded token as its most frequent neighbor word (in this case *"at"*). Finally, when a token is not a typo of any vocabulary words, we encode it to a special encoded token called OOV.

4.3.4 Connected component encodings

We now present two approaches to generate robust token-level encodings. Our first method, connected component encodings, maximizes the stability objective (4.8). Note that Stab is maximized when for each word w, $B_\pi(w)$ contains one encoded token. This is possible only when all words that share a typo are assigned to the same cluster.

To maximize Stab, define a graph G with all words in V as vertices, and edges between words that share a typo. Since we must map words that share an edge in G to the same cluster, we define the cluster C_i to be the set of words in the i-th connected component of G. While this stability-maximizing clustering encodes many words to the same token (and hence seems to compromise on fidelity), these encodings still perform surprisingly well in practice (see Section 4.4.4).

4.3.5 Agglomerative cluster encodings

Connected component encodings focus only stability and can lead to needlessly low fidelity. For example, in Figure 4.3, *"at"* and *"about"* are in the same connected component even though they do not share a typo. Since both words are generally frequent, mapping them to different encoded tokens can significantly improve fidelity, with only a small drop in stability: recall only the infrequent word *"abet"* can be perturbed to multiple encoded tokens.

To handle such cases, we introduce *agglomerative cluster encodings*, which we construct by trading off Stab with a formal objective we define for fidelity, Fid. We then approximately optimize this combined objective Φ using an agglomerative clustering algorithm.

Fidelity objective. Recall from Section 4.2.2 that an encoding has high task-agnostic fidelity if it sends distinct x's drawn from p_{corp} to different encodings. Now, we wish to approximately quantify how harmful it is when two words are encoded identically. Returning to our example, suppose *"at"* and *"abet"* belong to the same cluster and thus share an encoded token z. During training, each occurrence of *"at"* and *"abet"* is replaced with z. However, since *"at"* is much more frequent, the learned classifier g will likely treat z similarly to *"at"* in order to achieve good overall performance. This leads to mostly normal performance on sentences with *"at"*, at the cost of performance on sentences containing the less frequent *"abet"*.

This motivates the following definition: let $\vec{v_i}$ be the one-hot indicator vector in $\mathbb{R}^{|V|}$ corresponding to word i. In principle $\vec{v_i}$ could be a distributed word embedding; we choose indicator vectors to avoid making any task-specific assumptions. We define the encoded token centroid $\vec{\mu_j}$ associated with words in cluster C_j as follows:

$$\vec{\mu_j} = \frac{\sum_{w_i \in C_j} \rho(w_i)\vec{v_i}}{\sum_{w_i \in C_j} \rho(w_i)} \tag{4.9}$$

We weight by the frequency ρ to capture the effect of training a model using the encodings, as described above.

Fidelity is maximized when each word has a distinct encoded token. We capture the drop in standard accuracy due to shared encoded tokens by computing the distance between the original embeddings of the word its encoded token. Formally, let $c(i)$ be the cluster index of word w_i. We define the fidelity objective Fid as follows:

$$\text{Fid}(C) = -\sum_{i=1}^{N} \rho(w_i)\|\vec{v_i} - \vec{\mu}_{c(i)}\|^2. \tag{4.10}$$

Fid is low when when multiple frequent words are in the same cluster, but higher if a cluster has only one frequent word and other much rarer words.

Final objective. We introduce a hyperparameter $\gamma \in [0, 1]$ that balances stability and fidelity. We approximately minimize the following weighted combination of Stab (4.8) and Fid (4.10):

$$\Phi(C) = \gamma \, \mathrm{Fid}(C) + (1 - \gamma) \, \mathrm{Stab}(C). \tag{4.11}$$

As γ approaches 0, we get the connected component clusters from our baseline, which maximize stability. As γ approaches 1, we maximize fidelity by assigning each word to its own cluster.

Agglomerative clustering. We approximately optimize Φ using agglomerative clustering. At a high level, we start with each word in its own cluster, then iteratively combine the pair of clusters whose resulting combination increases Φ the most. We repeat until combining any pair of clusters would decrease Φ.

More specifically, we initialize a clustering C by placing each word in its own cluster. We then examine each pair of clusters C_i, C_j such that there exists an edge between a node in C_i and a node in C_j, in the graph from Section 4.3.2. For each such pair, we compute the value of Φ if C_i and C_j were merged to create one cluster $C_i \cup C_j$. If none of these merge operations decreases Φ, we return the current clustering. Otherwise, we merge the pair that leads to the greatest reduction in Φ, and repeat. In summary, the algorithm works as follows:

Algorithm 1 Objective-minimizing agglomerative clustering

```
1: $C \leftarrow V$
2: **for** $i$ in range($|V|$) **do**
3:     $C_{\text{next}} \leftarrow$ GetBestCombination($C$)
4:     **if** $C = C_{\text{next}}$ **then**
5:         **return** C
6:     **end if**
7:     $C \leftarrow C_{\text{next}}$
8: **end for**
9: **return** C
```

Now, we simply have to define the procedure we use to get the best combination. Recall our graph $G = (V, E)$ used to define the connected component clusters; V is the vocabulary of size N, and each word in V is a node in the graph. We say two clusters C_i and C_j are *adjacent*, if there exists a $v_i \in C_i$ and a $v_j \in C_j$ such that $(v_i, v_j) \in E$. We define a subroutine GetAdjacentPairs that returns all adjacent pairs of clusters for the current clustering C. Using this, we compute the best way to merge clusters as follows:

Algorithm 2 GetBestCombination(C)

```
1: $C_{\text{opt}} \leftarrow C$
2: $\Phi_{\text{opt}} \leftarrow \Phi(C)$
3: for $(C_i, C_j) \in$ GetAdjacentPairs($C$) do
4:     $C_{\text{comb}} \leftarrow C_i \cup C_j$
5:     $C_{\text{new}} \leftarrow C \cup C_{\text{comb}} \setminus \{C_i, C_j\}$ {New clusters}
6:     $\Phi_{\text{new}} \leftarrow \Phi(C_{\text{new}})$
7:     if $\Phi_{new} < \Phi_{opt}$ then
8:         $\Phi_{opt} \leftarrow \Phi_{new}$
9:         $C_{\text{opt}} \leftarrow C_{\text{new}}$
10:    end if
11: end for
12: return $C_{\text{opt}}$
```

The runtime of our algorithm is $O(N^2|E|)$ since at each of a possible N total iterations, we compute the objective for one of at most $|E|$ pairs of clusters. Computation of the objective can be reframed as computing the difference between Φ and Φ_{new}, where the latter is computed using new clusters, which can be done in $O(N)$ time.

4.3.6 Mapping clusters to encoded tokens

Finally, recall that in order to fully specify π, we must pick one encoded token to represent each cluster. We choose to use each cluster's most frequent member word as its associated encoded token. Formally, for a word $w \in V$ in cluster C_i, we define $\pi(w)$ to be the word w' in C_i that has highest frequency $\rho(w')$. Thus, the set of all encoded tokens is a subset of the vocabulary V. Though unnecessary when training from scratch, this strategy allows us to leverage the inductive biases of pre-trained models like BERT (Devlin et al., 2019). In the special case of the out-of-vocabulary token, we map OOV to the special token [MASK].

4.4 Experiments

4.4.1 Setup

Token-level attacks. The primary attack surface we study is edit distance one (ED1) perturbations. For every word in the input, the adversary is allowed to insert a lowercase letter, delete a character, substitute a character for any letter, or swap two adjacent characters, so long as the first and last characters remain the same as in the original token. The constraint on the outer characters, also used by Pruthi et al. (2019), is motivated by psycholinguistic studies (Rawlinson, 1976; Davis, 2003).

Within our attack surface, *"the movie was miserable"* can be perturbed to *"thae mvie wjs misreable"* but not *"th movie as miserable"*. Since each token can be independently perturbed, the number of perturbations of a sentence grows exponentially with its length; even *"the movie was miserable"* has 431,842,320 possible perturbations. Our attack surface contains the attack surface used by Pruthi et al. (2019), which allows ED1 perturbations to at most two words per sentence. Reviews from SST-2 have 5 million perturbations per example (PPE) on average under this attack surface, while our attack surface averages 10^{97} PPE. Our attack surface forces a system robust to subtle perturbations (e.g., *"the moviie waas misreable"*) that are not included in smaller attack surfaces. The fact that we can defend against such a large attack surface is a strength of our approach.

In Section 4.4.7, we additionally consider the internal permutation attacks studied in Belinkov and Bisk (2018) and Sakaguchi et al. (2017), where all characters, except the first and the last, may be arbitrarily reordered.

Attack algorithms. We consider two attack algorithms: the worst-case attack (WCA) and a beam-search attack (BSA). WCA exhaustively tests every possible perturbation of an input x to see any change in the prediction. The attack accuracy of WCA is the true robust accuracy since if there exists some perturbation that changes the prediction, WCA finds it. When instances of RobEn have high stability, the number of possible encodings of perturbations of x is often small, allowing us to exhaustively test all possible perturbations in the encoding space.[3] This allows us to tractably run WCA. Using WCA with RobEn, we can obtain computationally tractable *guarantees* on robustness: given a sentence, we can quickly compute whether or not any perturbation of x that changes the prediction.

For systems that do not use RobEn, we cannot tractably run WCA. Instead, we run a beam search attack (BSA) with beam width 5, perturbing tokens one at a time. Since $|B(x_i)|$ is very large for many tokens x_i, even the beam search is computationally very expensive. Instead, we run a beam search where the allowable perturbations are $B'(x_i) \subseteq B(x_i)$. We define $B'(x_i)$ to be four randomly sampled perturbations from $B(x_i)$ when the length of x_i is less than five, and all deletions when x_i is greater than five. Thus, the number of perturbations of each word is bounded above by $\min\{4, \text{len}(x_i) - 2\}$. Even against this very limited attack, we find that baseline models have low accuracy.

Datasets. We use six out of the nine datasets from GLUE (Wang et al., 2019b): SST, MRPC, QQP, MNLI, QNLI, and RTE, all of which are classification tasks evaluated on accuracy. The Stanford Sentiment Treebank (SST-2) (Socher et al., 2013) contains movie reviews that are classified as positive and negative. The Microsoft Research Paraphrase Corpus (MRPC) (Dolan and Brockett,

[3]When there are more than 10000 possible encodings, which holds for 0.009% of our test examples, we assume the adversary successfully alters the prediction.

2005) and the Quora Question Pairs dataset (Iyer et al., 2017) contain pairs of input which are classified as semantically equivalent or not; QQP contains question pairs from Quora, while MRPC contains pairs from online news sources. MNLI, and RTE are entailment tasks, where the goal is to predict whether or not a premise sentence entails a hypothesis (Williams et al., 2018). MNLI gathers premise sentences from ten different sources and elicits hypotheses from crowdworkers, while RTE gathers examples from previous Recognizing Textual Entailment challenges. QNLI gives pairs of sentences and questions extracted from the Stanford Question Answering Dataset (Rajpurkar et al., 2016), and the task is to predict whether or not the answer to the question is in the sentence. We do not use STS-B and CoLA as they are evaluated on correlation, which does not decompose as an example-level loss. We additionally do not use WNLI, as most submitted GLUE models cannot even outperform the majority baseline, and state-of-the-art models are rely on external training data (Kocijan et al., 2019).

We use the GLUE splits for the six datasets and evaluate on test labels when available (SST-2, MRPC), and otherwise the publicly released development labels. We tune hyperparameters by training on 80% of the original train set and using the remaining 20% as a validation set. We then retrain using the chosen hyperparameters on the full training set.

4.4.2 Baseline models.

We compare with three baseline methods for training NLP models that are robust to typos, as described below.

Standard training. Our first is the standard uncased BERT-base model (Devlin et al., 2019) fine-tuned on the training data for each task. We use the default hyperparameters in the pytorch-transformers repo,[4] batch size of 8, and learning rate 2×10^{-5}. These same hyperparameters are also used for all other models that use BERT.

Data augmentation. For our next baseline, we augment the training dataset with four random perturbations of each example, then fine-tune BERT-base on this augmented data. Data augmentation has been shown to increase robustness to some types of adversarial perturbations (Ribeiro et al., 2018; Liu et al., 2019a).

We do not experiment with adversarial training or certifiably robust training, as attempts to implement these with a BERT model run into serious obstacles. Adversarial training with black-box attacks offers limited robustness gains over data augmentation (Cohen et al., 2019; Pruthi et al., 2019). Projected gradient descent (Madry et al., 2018), the only white-box adversarial training method that is robust in practice, cannot currently be applied to BERT since subword tokenization

[4] `https://github.com/huggingface/pytorch-transformers`

maps different perturbations to different numbers of tokens, making gradient-based search impossible. Certifiably robust training (Huang et al., 2019; Shi et al., 2020) does not work with BERT due to the same tokenization issue and BERT's use of non-monotonic activation functions, which make computing bounds intractable. Moreover, the bounds used by certifiably robust training become loose as model depth increases, leading to poor robust accuracy (Gowal et al., 2019).

Typo corrector. For our third baseline, we use the most robust method from Pruthi et al. (2019). In particular, we train a scRNN typo corrector (Sakaguchi et al., 2017) on random perturbations of each task's training set, using the default settings from the implementation provided by Pruthi et al. (2019).[5] At test time inputs are corrected using the typo corrector, then fed into a downstream model. We replace any OOV outputted by the typo corrector with the neutral word *"a"*, as suggested by Pruthi et al. (2019), and use BERT as our downstream model.

4.4.3 Models with RobEn

We run experiments using our two token-level encodings: connected component encodings (ConnComp) and agglomerative cluster encodings (AggClust). To form clusters, we use the $N = 100{,}000$ most frequent words from the Corpus of Contemporary American English (Davies, 2008) that are also in GloVe (Pennington et al., 2014). For AggClust we use $\gamma = 0.3$, which maximizes robust accuracy on SST-2 development set. For both encodings, the sentence-level encoding $\alpha(x)$ is simply the concatenation of all the token-wise encodings. For each dataset, we independently fine-tune BERT on the training data, using $\alpha(x)$ as input.

4.4.4 Robustness gains from RobEn

Our main results are shown in Table 4.1. We show all three baselines, as well as models using our instances of RobEn, ConnComp and AggClust.

Even against the heuristic attack, each baseline system suffers dramatic performance drops. The system presented by Pruthi et al. (2019), Typo Corrector + BERT, only achieves 35.3% attack accuracy, compared to its standard accuracy of 78.2%. BERT and Data Augmentation + BERT perform even worse. Moreover, the number of perturbations the heuristic attack explores is a tiny fraction of our attack surface, so the robust accuracy of Typo Corrector + BERT, the quantity we would like to measure, is likely lower than the attack accuracy.

In contrast, simple instances of RobEn are much more robust. AggClust + BERT achieves average robust accuracy of 71.3%, 36 points higher than the attack accuracy of Typo Corrector + BERT. AggClust also further improves on ConnComp in terms of both robust accuracy (by 1.3 points) and standard accuracy (by 2.8 points).

[5] https://github.com/danishpruthi/Adversarial-Misspellings

Accuracy	System	SST-2	MRPC	QQP	MNLI	QNLI	RTE	Avg
Standard	**Baselines**							
	BERT	93.8	87.7	91.3	84.6	88.6	71.1	86.2
	Data Aug. + BERT	92.2	84.3	88.7	83.0	87.4	63.5	83.1
	Typo Corr. + BERT	89.6	80.9	87.6	75.9	80.5	54.9	78.2
	RobEn							
	ConnComp + BERT	80.6	79.9	84.2	65.7	73.3	52.7	72.7
	AggClust + BERT	83.1	83.8	85.0	69.1	76.6	59.2	76.1
Attack	**Baselines**							
	BERT	8.7	10.0	17.4	0.7	0.7	1.8	6.6
	Data Aug. + BERT	17.1	1.0	27.6	15.4	10.7	1.4	12.2
	Typo Corr. + BERT	53.2	30.1	52.0	23.0	32.3	21.3	35.3
	RobEn							
	ConnComp + BERT	80.3	79.4	82.7	62.6	71.5	47.3	70.6
	AggClust + BERT	**82.1**	**82.8**	**83.2**	**65.3**	**74.5**	**52.7**	**73.4**
Robust	**RobEn**							
	ConnComp + BERT	80.1	79.4	**82.2**	61.4	70.5	46.6	70.0
	AggClust + BERT	**80.7**	**80.9**	81.4	**62.8**	**71.9**	**49.8**	**71.3**

Table 4.1: Standard, attack, and robust accuracy on six GLUE tasks against ED1 perturbations. For baseline models we only compute attack accuracy, an upper bound on robust accuracy, since robust accuracy cannot be tractably computed. Using RobEn, we get robustness guarantees by computing robust accuracy, which we find outperforms the typo corrector in (Pruthi et al., 2019) by *at least* 36 points.

Standard accuracy. Like defenses against adversarial examples in other domains, using RobEn decreases standard accuracy (Madry et al., 2018; Zhang et al., 2019b; Jia et al., 2019). Our standard accuracy using agglomerative cluster encodings is 10.1 points lower then that of normally trained BERT. However, to the best of our knowledge, our standard accuracy is state-of-the-art for approaches that guarantee robustness to a similar family of typos. We attribute this improvement to RobEn's compatibility with any model.

It may be surprising that we can do so well with an encoding that processes each token independently, completely ignoring context. Typo correctors rely heavily on context to reconstruct sentences. It is important to remember that the BERT model that receives encodings as inputs does make heavy use of context; in principle, such a model could attempt to use that context to infer the identity of the original word, as a typo corrector would.

Comparison to smaller attack surfaces. We note that RobEn also outperforms existing methods on their original, smaller attack surfaces. On SST-2, Pruthi et al. (2019) achieves an accuracy of 75.0% defending against a *single* ED1 typo, which is 5.7 points lower than AggClust's robust accuracy against perturbations of all tokens: a superset of the original perturbation set. In Section 4.4.8, we show our results on adversarial attacks that can only perturb one or a few tokens. AggClust also outperforms certified training: Huang et al. (2019), which offers robustness guarantees to three

Encodings	SST-2	MRPC	QQP	MNLI	QNLI	RTE	Average
ConnComp	86.9	71.6	72.7	45.3	54.6	40.4	61.9
AggClust	65.6	50.0	62.7	35.4	36.6	25.2	45.9

Table 4.2: Percentage of test examples with $|B_\alpha(x)| = 1$ for each dataset.

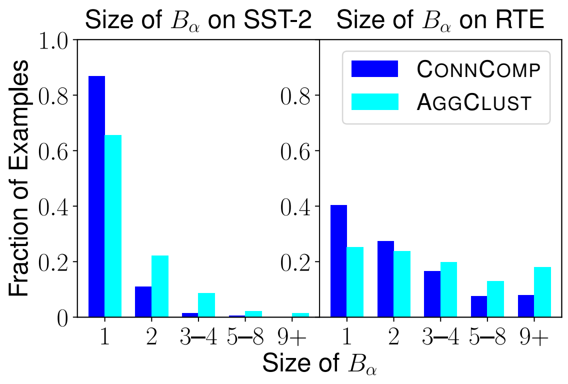
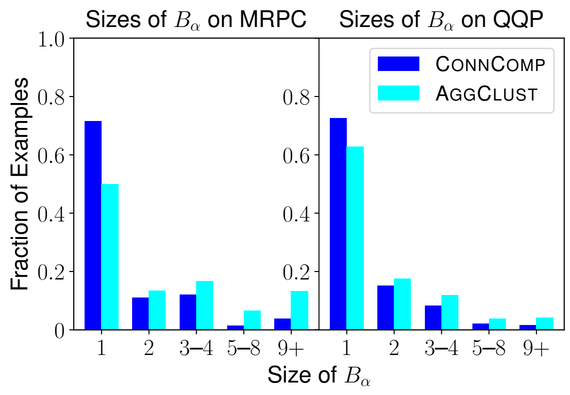


Figure 4.4: Histogram of $|B_\alpha(x)|$ for SST-2 and RTE. Among all six GLUE datasets, SST-2 has the highest percentage of inputs x where $|B_\alpha(x)| = 1$, while RTE has the least. On both datasets, $|B_\alpha(x)| < 9$ for most x, and $|B_\alpha(x)| = 1$ on a plurality of inputs.

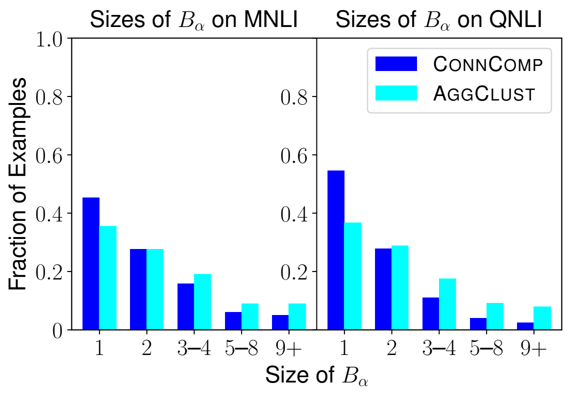
character substitution typos (but not insertions or deletions), achieves a robust accuracy of 74.9% on SST-2. In fact, the robust accuracy of AggClust even exceeds the *standard* accuracy of the robustly trained model used in Huang et al. (2019). Certified training requires strong assumptions on model architecture, which forces Huang et al. (2019) to use simpler model architectures like shallow CNNs, whereas RobEn can be paired with stronger models like BERT.

4.4.5 Reusable encodings

Each instance of RobEn achieves consistently high stability across our tasks, despite reusing a single function. We quantify this by measuring the size of $B_\alpha(x)$, the set of encodings that are mapped to by some perturbation of x, for inputs x drawn from different task distributions. Recall that when $|B_\alpha(x)|$ is small, robust accuracy can be computed quickly; in particular, when $|B_\alpha(x)| = 1$, every perturbation of x maps to the same encoding. Table 4.2 gives the fraction of inputs x for which $|B_\alpha(x)| = 1$ for each dataset and each set of encodings (AggClust or ConnComp). For the AggClust encoding function, $|B_\alpha(x)| = 1$ for 25% of examples in RTE and 66% in SST-2; the other four datasets fall between these extremes. As expected, $|B_\alpha(x)| = 1$ occurs more often for ConnComp than AggClust. Figure 4.4 shows histograms of $|B_\alpha(x)|$ across test examples in SST-2 and RTE, and Figure 4.5 shows the other four datasets.



(a) Histogram of $|B_\alpha(x)|$ for MRPC and QQP.



(b) Histogram of $|B_\alpha(x)|$ for MNLI and QNLI.

Figure 4.5: Histograms of $|B_\alpha(x)|$ for MRPC, QQP, MNLI, and QNLI.

4.4.6 Agglomerative clustering trade-off

In Figure 4.6, we plot standard and robust accuracy on SST-2 for AggClust encodings, using different values of γ. Recall that $\gamma = 0$ maximizes stability (ConnComp), and $\gamma = 1$ maximizes fidelity. At $\gamma = 0$, there is a small but nonzero gap between standard and robust accuracy, due to out-of-vocabulary tokens in the data. Note that the ConnComp construction guarantees perfect stability for all tokens in V, but it is still possible that if out-of-vocabulary tokens occur in the data, they might create instability (for example, a typo of an out-of-vocabulary token may also be a typo of a vocabulary word). As γ increases, both standard accuracy and the gap between standard and robust accuracy increase. As a result, robust accuracy first increases, then decreases.

4.4.7 Internal permutation attacks

RobEn can also be used to defend against internal permutation attacks, as mentioned in Section 4.4.1. We consider the *internal permutation* attack surface, where interior characters in a word can be permuted, assuming the first and last characters are fixed. For example, *"perturbation"* can be permuted to *"peabreuottin"* but not *"repturbation"*. Normally, context helps humans resolve these typos. Interestingly, for internal permutations it is impossible for an adversary to change the cluster assignment of both in-vocabulary and out of vocabulary tokens since a cluster can be uniquely represented by the first character, a sorted version of the internal characters, and the last character. Therefore, using ConnComp encodings, robust, attack, and standard accuracy are all equal. To attack the clean model, we use the same attack described in Section 4.4.1 for ED1 perturbations, except we obtain $B'(x_i)$ by sampling five permutations at random.

Table 4.3 shows our results on this internal permutation attack. For normally trained BERT, a heuristic beam search attack using internal permutations reduces average accuracy from 86.2%

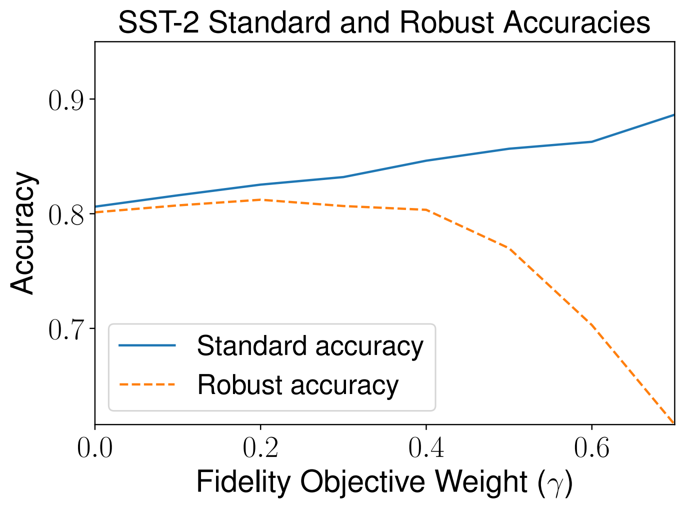


Figure 4.6: Standard and robust accuracies on SST-2 with AggClust using different values of γ. While the gap between standard and robust accuracy increases monotonically, robust accuracy increases before decreasing.

Accuracy	System	SST-2	MRPC	QQP	MNLI	QNLI	RTE	Avg
Standard	BERT	93.8	87.7	91.2	84.3	88.9	71.1	86.2
	ConnComp + BERT	93.2	87.7	86.9	75.9	83.4	61.4	81.4
Attack	BERT	28.1	15.9	33.0	4.9	6.2	5.8	15.7
	ConnComp + BERT	93.2	87.7	86.9	75.9	83.4	61.4	81.4
Robust	ConnComp + BERT	93.2	87.7	86.9	75.9	83.4	61.4	81.4

Table 4.3: Results from internal permutation attacks. Internal permutation attacks bring the average performance for BERT across the six listed tasks from 86.2 to 15.7. Our ConnComp encodings, generated using the internal permutation attack surface, achieve a robust accuracy of 81.4, which is only 4.8 points below standard accuracy.

to 15.7% across our six tasks. Using ConnComp with the internal permutation attack surface, we achieve robust accuracy of 81.4%.

4.4.8 Constrained adversaries

Using RobEn, since we can tractably compute robust accuracy, it is easy to additionally consider adversaries that cannot perturb every input token. We may assume that an attacker has a budget of $b \leq L$ words that they may perturb as in (Pruthi et al., 2019). Exiting methods for certification (Jia et al., 2019; Huang et al., 2019) require attack to be factorized over tokens, and cannot give tighter guarantees in the budget-constrained case compared to the unconstrained setting explored in previous sections. However, our method lets us easily compute robust accuracy exactly in this

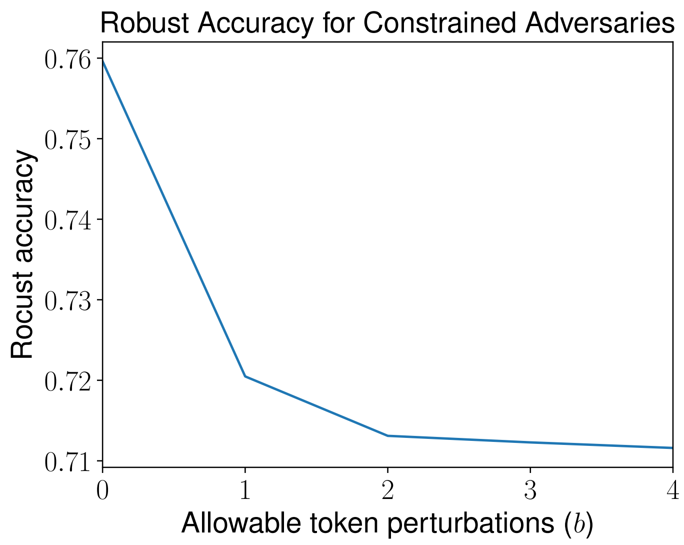


Figure 4.7: Robust accuracy averaged across all tasks based on different adversarial budgets b. $b = 0$ corresponds to clean performance, and robust performance is reached at $b = 4$

situation: we just enumerate the possible perturbations that satisfy the budget constraint, and query the model. Figure 4.7 plots average robust accuracy across the six tasks using AggClust as a function of b. Note that $b = 0$ is simply standard accuracy. Interestingly, for each dataset there is an attack only perturbing 4 tokens with attack accuracy equal to robust accuracy.

4.5 Discussion

At a high level, this chapter shows that discrete intermediate representations can be used to greatly improve robustness to label-preserving transformations. The fact that our representations are discrete is crucial to guaranteeing robustness for arbitrary downstream model architectures. It is instructive to contrast with Garg et al. (2018), which also define a notion of robust features but do so in a continuous space. Their analogue of stability asks that any small perturbation of the input leads to a correspondingly small perturbation of the feature. However, this alone is not enough to guarantee robustness when combined with an *arbitrary* downstream model; one must also require the downstream model to be not very sensitive to small changes in the features. Discrete encodings can give the stronger guarantee that intermediate representations remain identical (or remain within a small set of possible values) when inputs are perturbed. Such a guarantee does imply robustness for any model that uses this encoding.

NLP has long been concerned with obtaining discrete representations of text, but they are usually semantic representations or logical forms that look very different from RobEn. For example, Abstract Meaning Representation (AMR) is a meaning representation that, among other goals, attempts to canonicalize some sentences with the same or similar meaning into equivalent representations (Banarescu et al., 2013). Sentences with the same AMRs tend to be similar lexically, as AMR does not attempt to canonicalize synonyms or in general deeply handle lexical semantics. But AMR does normalize significant variation in sentence structure and inflection. These three sentences all have the same AMR:

- *"The boy desires the girl to believe him."*
- *"The boy desires to be believed by the girl."*
- *"The boy is desirous of the girl believing him."*

We can view AMR as an encoding that is trying to confer robustness not to typos or word substitutions, but to structural transformations.

However, there is a big difference: producing an AMR for a sentence is itself a difficult task, whereas our encoding functions are easily computable. Note however that an AMR parser does not necessarily need to be highly accurate in order to help ensure invariance to AMR-preserving transformations. We just need the AMR parser to be "stable," which here means that sentences with the same gold AMR actually get parsed to the same predicted AMR. Note that this can be true even if the gold AMR and predicted AMR are different. A downstream model could, to some extent, recover from such errors in AMR parsing, just as our models using RobEn achieve reasonable accuracies despite using encodings that lose a lot of information about the input.

Chapter 5

Adversarial Evaluation for Reading Comprehension

The previous two chapters showed that models are overly sensitive to minor label-preserving perturbations, namely word substitutions and typos. We saw that even if a system developer knows ahead of time what sorts of perturbations are permitted at test time, it is non-trivial to ensure correctness on *every* perturbation of a given example. This challenge stems from the fact that multiple perturbations can be applied in combination, leading to exponentially many possible perturbed inputs.

In this chapter, we shift our focus to a different, complementary type of adversarial example. We focus on the SQuAD question answering dataset (Rajpurkar et al., 2016), in which systems answer questions about paragraphs from Wikipedia. As mentioned in Chapter 1, existing SQuAD models appear very successful by standard average-case evaluation metrics. As of August 2020, the best system on the SQuAD leaderboard has 95.4% F1 score,[1] while human performance is just 91.2%.[2] Nonetheless, it seems unlikely that existing systems possess true language understanding and reasoning capabilities.

The format of SQuAD affords us the opportunity to carry out new types of adversarial evaluation. Chapter 3 and Chapter 4 focused on label-preserving perturbations, However, it is also the case that changing one word of a paragraph can often alter its meaning drastically. Instead of relying on meaning-preserving perturbations, we create adversarial examples by adding distracting sentences to the input paragraph, as shown in Figure 5.1. We automatically generate these sentences so that they confuse models, but do not contradict the correct answer or confuse humans. For our main results, we use a simple set of rules to generate a raw distractor sentence that does not answer the question but looks related; we then fix grammatical errors via crowdsourcing. While adversarial

[1]At the time when this work was originally published in 2017, the state-of-the-art system had 84.7% F1 score.
[2]https://rajpurkar.github.io/SQuAD-explorer/

Article: Super Bowl 50
Paragraph: *"Peyton Manning became the first quarterback ever to lead two different teams to multiple Super Bowls. He is also the oldest quarterback ever to play in a Super Bowl at age 39. The past record was held by John Elway, who led the Broncos to victory in Super Bowl XXXIII at age 38 and is currently Denver's Executive Vice President of Football Operations and General Manager. Quarterback Jeff Dean had jersey number 37 in Champ Bowl XXXIV."*
Question: *"What is the name of the quarterback who was 38 in Super Bowl XXXIII?"*
Original Prediction: John Elway
Prediction under adversary: Jeff Dean

Figure 5.1: An example from the SQuAD dataset. The BiDAF Ensemble model originally gets the answer correct, but is fooled by the addition of an adversarial distracting sentence (in blue).

word substitutions and typos punish model *oversensitivity* to small meaning-preserving variations, our adversarial examples target model *overstability*—the inability of a model to distinguish a sentence that actually answers the question from one that merely has many words in common with it.

The two previous chapters also focused on the setting where developers know ahead of time what perturbations are possible at test time, even if they could not tractably compute the actual worst-case perturbation. This chapter focuses primarily on the setting where developers do not know ahead of time how we will generate adversarial examples. By hiding this information, we place a stronger burden on models to generalize beyond the training distribution. This also makes it easier to find adversarial examples: our main results do not depend on computationally expensive search and make very few (or no) queries to the model being attacked.

Our experiments test sixteen open-source SQuAD models published in or before 2017, as well as several released after this. None of these models are robust to the addition of adversarial distracting sentences. Across sixteen models from 2017 and earlier, adding grammatical adversarial sentences reduces F1 score from an average of 75% to 36%. For a more recent BERT-based model (Devlin et al., 2019), F1 score drops from 92% to 61%. On a smaller set of four models, we run additional experiments in which the adversary adds non-grammatical sequences of English words, causing average F1 score to drop further to 7%. We conclude that SQuAD requires solving only a very simplified version of question answering, as high accuracy on SQuAD is possible without understanding the fine-grained distinctions targeted by our adversarial distracting sentences.

5.1 The SQuAD dataset and models

5.1.1 Dataset

The SQuAD dataset (Rajpurkar et al., 2016) contains 107,785 human-generated reading comprehension questions about Wikipedia articles. Each question refers to one paragraph of an article, and the corresponding answer is guaranteed to be a span in that paragraph.

5.1.2 Models

When developing and testing our methods, we focused on two published model architectures: BiDAF (Seo et al., 2017) and Match-LSTM (Wang and Jiang, 2017). Both are deep learning architectures that predict a probability distribution over the correct answer. Each model has a single and an ensemble version, yielding four systems in total.

We also validate our major findings on twelve other published models with publicly available test-time code released in 2017 or earlier: ReasoNet Single and Ensemble versions (Shen et al., 2017c), Mnemonic Reader Single and Ensemble versions (Hu et al., 2018a), Structural Embedding of Dependency Trees (SEDT) Single and Ensemble versions (Liu et al., 2017), jNet (Zhang et al., 2017a), Ruminating Reader (Gong and Bowman, 2018), Multi-Perspective Context Matching (MPCM) Single version (Wang et al., 2016), RaSOR (Lee et al., 2017), Dynamic Chunk Reader (DCR) (Yu et al., 2016), and the Logistic Regression Baseline (Rajpurkar et al., 2016). We did not run these models during development, so they serve as a held-out set that validates the generality of our approach.

5.1.3 Standard evaluation

Given a model f that takes in paragraph-question pairs (p, q) and outputs an answer $\hat{a}$, the *standard accuracy* over a test set D_{test} is simply

$$\text{Acc}(f) \overset{\text{def}}{=} \frac{1}{|D_{\text{test}}|} \sum_{(p,q,a) \in D_{\text{test}}} v((p, q, a), f),$$

where v is the F1 score between the true answer a and the predicted answer $f(p, q)$ (see Rajpurkar et al. (2016) for details).

5.2 Adversarial evaluation

5.2.1 General framework

A model that relies on superficial cues without understanding language can do well according to average F1 score, if these cues happen to be predictive most of the time. Weissenborn et al. (2017) argue that many SQuAD questions can be answered with heuristics based on type and keyword-matching. We introduce adversaries that confuse deficient models by altering test examples. Consider the example in Figure 5.1: the BiDAF Ensemble model originally gives the right answer, but gets confused when an adversarial distracting sentence is added to the paragraph.

We define an adversary A to be a function that takes in an example (p, q, a), optionally with a model f, and returns a new example (p', q', a'). The *adversarial accuracy* with respect to A is

$$\mathrm{Adv}(f) \overset{\text{def}}{=} \frac{1}{|D_{\text{test}}|} \sum_{(p,q,a) \in D_{\text{test}}} v(A(p, q, a, f), f)).$$

While standard test error measures the fraction of the test distribution over which the model gets the correct answer, the adversarial accuracy measures the fraction over which the model is *robustly* correct, even in the face of adversarially-chosen alterations. For this quantity to be meaningful, the adversary must satisfy two basic requirements: first, it should always generate (p', q', a') tuples that are *valid*—a human would judge a' as the correct answer to q' given p'. Second, (p', q', a') should be somehow "close" to the original example (p, q, a).

5.2.2 Semantics-preserving adversaries

In image classification, adversarial examples are commonly generated by adding an imperceptible amount of noise to the input (Szegedy et al., 2014; Goodfellow et al., 2015). These perturbations do not change the semantics of the image, but they can change the predictions of models that are *oversensitive* to semantics-preserving changes. For language, the direct analogue would be to paraphrase the input (Madnani and Dorr, 2010). Chapter 3 presented one particular way to generate semantics-preserving perturbations.

5.2.3 Concatenative adversaries

Instead of relying on paraphrasing, we use perturbations that do alter semantics to build *concatenative* adversaries, which generate examples of the form $(p+s, q, a)$ for some sentence s, where addition here denotes string concatenation. In other words, concatenative adversaries add a new sentence to the end of the paragraph, and leave the question and answer unchanged. Valid adversarial examples are precisely those for which s does not contradict the correct answer; we refer to such sentences as being *compatible* with (p, q, a). We use semantics-altering perturbations to that ensure that s is compatible, even though it may have many words in common with the question q. Existing models are bad at distinguishing these sentences from sentences that do in fact address the question, indicating that they suffer from *overstability* to semantics-altering edits. Table 5.1 summarizes this important distinction.

The decision to always append s to the end of p is somewhat arbitrary; we could also prepend

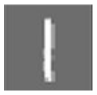
	Image Classification	Reading Comprehension
Possible Input		Tesla moved to the city of Chicago in 1880.
Similar Input		Tadakatsu moved to the city of Chicago in 1881.
Semantics	Same	Different
Model's Mistake	Considers the two to be different	Considers the two to be the same
Model Weakness	Overly sensitive	Overly stable

Table 5.1: Adversarial examples in computer vision exploit model oversensitivity to small perturbations. In contrast, our adversarial examples work because models do not realize that a small perturbation can completely change the meaning of a sentence. Images from Szegedy et al. (2014).

it to the beginning, though this would violate the expectation of the first sentence being a topic sentence. Both are more likely to preserve the validity of the example than inserting s in the middle of p, which runs the risk of breaking coreference links.

Now, we describe two concrete concatenative adversaries. AddSent, our main adversary, adds grammatical sentences that look similar to the question. In contrast, AddAny adds arbitrary sequences of English words, giving it more power to confuse models. Figure 5.2 illustrates these two main adversaries.

5.2.4 AddSent

AddSent uses a four-step procedure to generate sentences that look similar to the question, but do not actually contradict the correct answer. Similarly to the concept of minimal pairs, these sentences differ minimally from a sentence that does actually answer the question. Refer to Figure 5.2 for an illustration of these steps.

In Step 1, we apply semantics-altering perturbations to the question, in order to guarantee that the resulting adversarial sentence is compatible. We replace nouns and adjectives with antonyms from WordNet (Fellbaum, 1998), and change named entities and numbers to the nearest word in GloVe word vector space[3] (Pennington et al., 2014) with the same part of speech.[4] If no words are changed during this step, the adversary gives up and immediately returns the original example. For example, given the question *"What ABC division handles domestic television distribution?"*,

[3] We use 100-dimensional GloVe vectors trained on Wikipedia and Euclidean distance to define nearby words.

[4] We choose the nearest word whose most common gold POS tag in the Penn Treebank (Marcus et al., 1993) matches the predicted POS tag of the original word, according to CoreNLP. If none of the nearest 100 words satisfy this, we just return the single closest word.

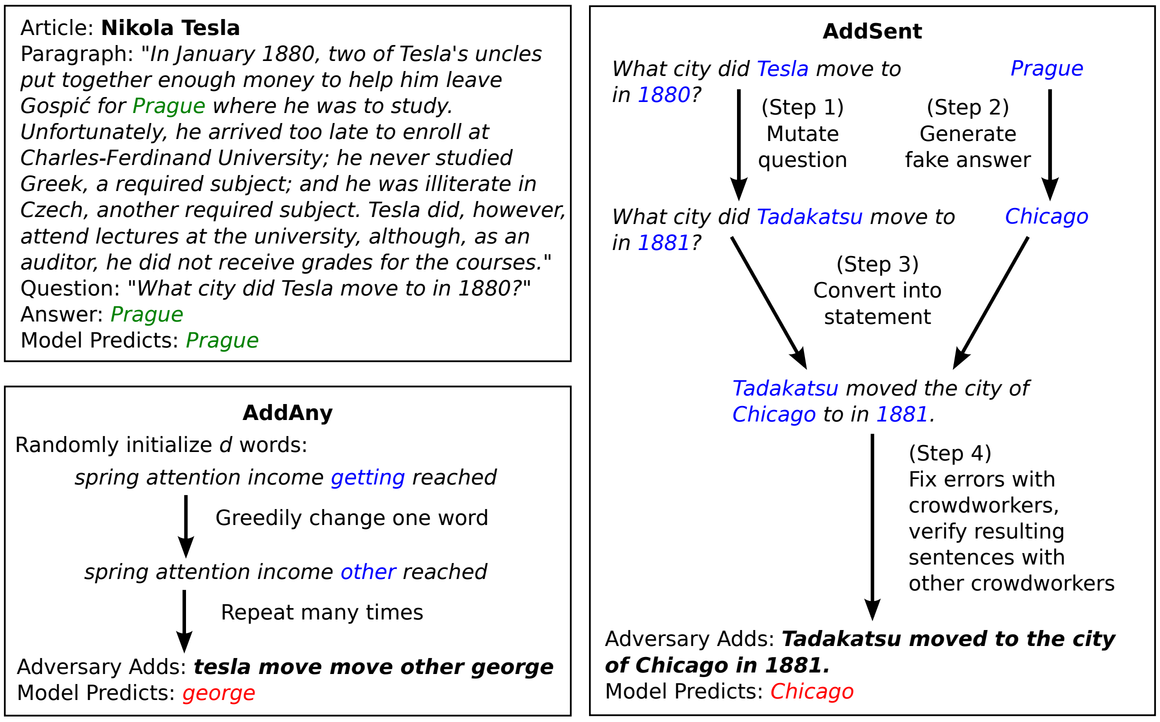

Article: Nikola Tesla
Paragraph: "In January 1880, two of Tesla's uncles put together enough money to help him leave Gospić for Prague where he was to study. Unfortunately, he arrived too late to enroll at Charles-Ferdinand University; he never studied Greek, a required subject; and he was illiterate in Czech, another required subject. Tesla did, however, attend lectures at the university, although, as an auditor, he did not receive grades for the courses."
Question: "What city did Tesla move to in 1880?"
Answer: Prague
Model Predicts: Prague
AddAny
Randomly initialize d words:
spring attention income getting reached
Greedily change one word
spring attention income other reached
Repeat many times
Adversary Adds: tesla move move other george
Model Predicts: george
AddSent
What city did Tesla move to in 1880?
Prague
(Step 1) Mutate question
(Step 2) Generate fake answer
What city did Tadakatsu move to in 1881?
Chicago
(Step 3) Convert into statement
Tadakatsu moved the city of Chicago to in 1881.
(Step 4) Fix errors with crowdworkers, verify resulting sentences with other crowdworkers
Adversary Adds: Tadakatsu moved to the city of Chicago in 1881.
Model Predicts: Chicago

Figure 5.2: An illustration of the AddSent and AddAny adversaries.

we would change *"ABC"* to *"NBC"* (a nearby word in vector space) and *"domestic"* to *"foreign"* (a WordNet antonym), resulting in the question, *"What NBC division handles foreign television distribution?"*

In Step 2, we create a fake answer that has the same "type" as the original answer. We define a set of 26 types, corresponding to NER and POS tags from Stanford CoreNLP (Manning et al., 2014), plus a few custom categories (e.g., abbreviations), and manually associate a fake answer with each type. Given the original answer to a question, we compute its type and return the corresponding fake answer. In our running example, the correct answer was not tagged as a named entity, and has the POS tag NNP, which corresponds to the fake answer *"Central Park."*

In Step 3, we combine the altered question and fake answer into declarative form, using a set of roughly 50 manually-defined rules over CoreNLP constituency parses. For example, *"What ABC division handles domestic television distribution?"* triggers a rule that converts questions of the form "*what/which* $\texttt{NP}_1$ $\texttt{VP}_1$?" to "The $\texttt{NP}_1$ of [Answer] $\texttt{VP}_1$". After incorporating the alterations and fake answer from the previous steps, we generate the sentence, *"The NBC division of Central Park handles foreign television distribution."*

The raw sentences generated by Step 3 can be ungrammatical or otherwise unnatural due to the incompleteness of our rules and errors in constituency parsing. Therefore, in Step 4, we fix errors in these sentences via crowdsourcing. Each sentence is edited independently by five workers on Amazon Mechanical Turk, resulting in up to five sentences for each raw sentence. Three additional crowdworkers then filter out sentences that are ungrammatical or incompatible, resulting in a smaller (possibly empty) set of human-approved sentences. The full AddSent adversary runs the model f as a black box on every human-approved sentence, and picks the one that makes the model give the worst answer. If there are no human-approved sentences, the adversary simply returns the original example.

A model-independent adversary. AddSent requires a small number of queries to the model under evaluation. To explore the possibility of an adversary that is completely model-independent, we also introduce AddOneSent, which adds a random human-approved sentence to the paragraph. In contrast with prior work in computer vision (Papernot et al., 2017; Narodytska and Kasiviswanathan, 2017; Moosavi-Dezfooli et al., 2017), AddOneSent does not require any access to the model or to any training data: it generates adversarial examples based solely on the intuition that existing models are overly stable.

5.2.5 AddAny

For AddAny, the goal is to choose any sequence of d words, regardless of grammaticality. We use local search to adversarially choose a distracting sentence $s = w_1 w_2 \dots w_d$. Figure 5.2 shows an example of AddAny with $d = 5$ words; in our experiments, we use $d = 10$.

We first initialize words $w_1, \dots, w_d$ randomly from a list of common English words.[5] Then, we run 6 epochs of local search, each of which iterates over the indices $i \in \{1, \dots, d\}$ in a random order. For each i, we randomly generate a set of candidate words W as the union of 20 randomly sampled common words and all words in q. For each $x \in W$, we generate the sentence with x in the i-th position and w_j in the j-th position for each $j \neq i$. We try adding each sentence to the paragraph and query the model for its predicted probability distribution over answers. We update w_i to be the x that minimizes the expected value of the F1 score over the model's output distribution. We return immediately if the model's argmax prediction has 0 F1 score. If we do not stop after 3 epochs, we randomly initialize 4 additional word sequences, and search over all of these random initializations in parallel.

AddAny requires significantly more model access than AddSent: not only does it query the model many times during the search process, but it also assumes that the model returns a probability distribution over answers, instead of just a single prediction. Without this assumption, we would have to rely on something like the F1 score of the argmax prediction, which is piecewise constant and therefore harder to optimize. "Probabilistic" query access is still weaker than access to gradients, as is common in computer vision (Szegedy et al., 2014; Goodfellow et al., 2015).

We do not do anything to ensure that the sentences generated by this search procedure do not

[5] We define common words as the 1000 most frequent words in the Brown corpus (Francis and Kucera, 1979).

	Match-LSTM Single	Match-LSTM Ensemble	BiDAF Single	BiDAF Ensemble
Original	71.4	75.4	75.5	80.0
AddSent	27.3	29.4	34.3	34.2
AddOneSent	39.0	41.8	45.7	46.9
AddAny	7.6	11.7	4.8	2.7
AddCommon	38.9	51.0	41.7	52.6

Table 5.2: Adversarial evaluation on the Match-LSTM and BiDAF systems. All four systems can be fooled by adversarial examples.

contradict the original answer. In practice, the generated "sentences" are gibberish that use many question words but have no semantic content (see Figure 5.2 for an example).

Finally, we note that both AddSent and AddAny try to incorporate words from the question into their adversarial sentences. While this is an obvious way to draw the model's attention, we were curious if we could also distract the model without such a straightforward approach. To this end, we introduce a variant of AddAny called AddCommon, which is exactly like AddAny except it only adds common words.

5.3 Experiments

5.3.1 Setup

For all experiments, we measure adversarial F1 score (Rajpurkar et al., 2016) across 1000 randomly sampled examples from the SQuAD development set.[6] Downsampling was helpful because AddAny and AddCommon can issue thousands of model queries per example, making them very slow. As the effect sizes we measure are large, this downsampling does not hurt statistical significance.

5.3.2 Main experiments

Table 5.2 shows the performance of the Match-LSTM and BiDAF models against all four adversaries. Each model incurred a significant accuracy drop under every form of adversarial evaluation. AddSent made average F1 score across the four models fall from 75.7% to 31.3%. AddAny was even more effective, making average F1 score fall to 6.7%. AddOneSent retained much of the effectiveness of AddSent, despite being model-independent. Finally, AddCommon caused average F1 score to fall to 46.1%, despite only adding common words.

We also verified that our adversaries were general enough to fool models that we did not use during development. We ran AddSent on twelve published models from 2017 and earlier for which

[6]The test set is not publicly available

Model	Original	AddSent	AddOneSent
ReasoNet (Ensemble)	**81.1**	39.4	49.8
SEDT (Ensemble)	80.1	35.0	46.5
BiDAF (Ensemble)	80.0	34.2	46.9
Mnemonic (Ensemble)	79.1	**46.2**	**55.3**
Ruminating Reader	78.8	37.4	47.7
jNet	78.6	37.9	47.0
Mnemonic (Single)	78.5	**46.6**	**56.0**
ReasoNet (Single)	78.2	39.4	50.3
MPCM (Single)	77.0	40.3	50.0
SEDT (Single)	76.9	33.9	44.8
RaSOR	76.2	39.5	49.5
BiDAF (Single)	75.5	34.3	45.7
Match-LSTM (Ensemble)	75.4	29.4	41.8
Match-LSTM (Single)	71.4	27.3	39.0
DCR	69.3	37.8	45.1
Logistic Regression	50.4	23.2	30.4

Table 5.3: AddSent and AddOneSent on all sixteen models, sorted by F1 score on the original examples.

	Original	AddSent	AddOneSent
Human accuracy	92.6	79.5	89.2

Table 5.4: Human evaluation on adversarial examples. Human accuracy decreases when evaluated on AddSent mostly due to unrelated errors; the AddOneSent numbers show that humans are robust to adversarial sentences.

we found publicly available test-time code; we did not run AddAny on these models, as not all models exposed output distributions. As seen in Table 5.3, no model was robust to adversarial evaluation; across the sixteen total models tested, average F1 score fell from 75.4% to 36.4% under AddSent.

It is noteworthy that the Mnemonic Reader models (Hu et al., 2018a) outperform the other models by about 6 F1 points. We hypothesize that Mnemonic Reader's self-alignment layer, which helps model long-distance relationships between parts of the paragraph, makes it better at locating all pieces of evidence that support the correct answer. Therefore, it can be more confident in the correct answer, compared to the fake answer inserted by the adversary.

5.3.3 Human evaluation

To ensure our results are valid, we verified that humans are not also fooled by our adversarial examples. As AddAny requires too many model queries to run against humans, we focused on

Model	Original	AddSent	AddOneSent
BERT (Ensemble)	91.5	61.1	70.7
SLQA (Ensemble)	87.9	54.8	64.2
r-net+ (Ensemble)	87.4	52.8	63.4
FusionNet (Ensemble)	85.9	51.4	60.7
SAN (Single)	84.4	46.6	56.5

Table 5.5: AddSent and AddOneSent on five SQuAD models released after this work was published, sorted by F1 score on the original examples.

AddSent. We presented each original and adversarial paragraph-question pair to three crowdworkers, and asked them to select the correct answer by copy-and-pasting from the paragraph. We then took a majority vote over the three responses (if all three responses were different, we picked one at random). These results are shown in Table 5.4. On original examples, our humans are actually slightly better than the reported number of 91.2 F1 on the entire development set. On AddSent, human accuracy drops by 13.1 F1 points, much less than the computer systems.

Moreover, much of this decrease can be explained by mistakes unrelated to our adversarial sentences. Recall that AddSent picks the worst case over up to five different paragraph-question pairs. Even if we showed the same original example to five sets of three crowdworkers, chances are that at least one of the five groups would make a mistake, just because humans naturally err. Therefore, it is more meaningful to evaluate humans on AddOneSent, on which their accuracy drops by only 3.4 F1 points.

5.3.4 Subsequent models

Since the publication of this work in 2017, state-of-the-art accuracy on SQuAD has reached impressive heights. Nevertheless, these newer models still fare much worse than humans on our adversarial examples. Table 5.5 shows results for five models published after this work: SAN (Liu et al., 2018), FusionNet (Huang et al., 2018), r-net+ (an improved version of Wang et al. (2017)), SLQA (Wang et al., 2018), and BERT (Devlin et al., 2019). While BERT comes within 1.1 F1 points of human accuracy on original examples, it is more than 18 F1 points worse than humans at both AddSent and AddOneSent.

5.3.5 Analysis

Next, we sought to better understand the behavior of our four main models under adversarial evaluation. To highlight errors caused by the adversary, we focused on examples where the model originally predicted the (exact) correct answer. We divided this set into "model successes"—examples where the model continued being correct during adversarial evaluation—and "model failures"—examples where the model gave a wrong answer during adversarial evaluation.

Manual verification. First, we verified that the sentences added by AddSent are actually grammatical and compatible. We manually checked 100 randomly chosen BiDAF Ensemble failures. We found only one where the sentence could be interpreted as answering the question: in this case, AddSent replaced the word *"Muslim"* with the related word *"Islamic"*, so the resulting adversarial sentence still contradicted the correct answer. Additionally, we found 7 minor grammar errors, such as subject-verb disagreement (e.g., *"The Alaskan Archipelago* ***are*** *made up almost entirely of hamsters."*) and misuse of function words (e.g., *"The gas* ***of*** *nitrogen makes up 21.8 % of* ***the*** *Mars's atmosphere."*), but no errors that materially impeded understanding of the sentence.

We also verified compatibility for AddAny. We found no violations out of 100 randomly chosen BiDAF Ensemble failures.

Error analysis. Next, we wanted to understand what types of errors the models made on the AddSent examples. In 96.6% of model failures, the model predicted a span in the adversarial sentence. The lengths of the predicted answers were mostly similar to those of correct answers, but the BiDAF models occasionally predicted very long spans. The BiDAF Single model predicted an answer of more than 29 words—the length of the longest answer in the SQuAD development set—on 5.0% of model failures; for BiDAF Ensemble, this number was 1.6%. Since the BiDAF models independently predict the start and end positions of the answer, they can predict very long spans when the end pointer is influenced by the adversarial sentence, but the start pointer is not. Match-LSTM has a similar structure, but also has a hard-coded rule that stops it from predicting very long answers.

We also analyzed human failures—examples where the humans were correct originally, but wrong during adversarial evaluation. Humans predicted from the adversarial sentence on only 27.3% of these error cases, which confirms that many errors are normal mistakes unrelated to adversarial sentences.

Categorizing AddSent sentences. We then manually examined sentences generated by AddSent. In 100 BiDAF Ensemble failures, we found 75 cases where an entity name was changed in the adversarial sentence, 17 cases where numbers or dates were changed, and 33 cases where an antonym of a question word was used.[7] Additionally, 7 sentences had other miscellaneous perturbations made by crowdworkers during Step 4 of AddSent. For example, on a question about the *"Kalven Report"*, the adversarial sentence discussed *"The statement Kalven cited"* instead; in another case, the question, *"How does Kenya curb corruption?"* was met by the unhelpful sentence, *"Tanzania is curbing corruption"* (the model simply answered, *"corruption"*).

Reasons for model successes. Finally, we sought to understand the factors that influence whether the model will be robust to adversarial perturbations on a particular example. First,

[7] These numbers add up to more than 100 because more than one word can be altered per example.

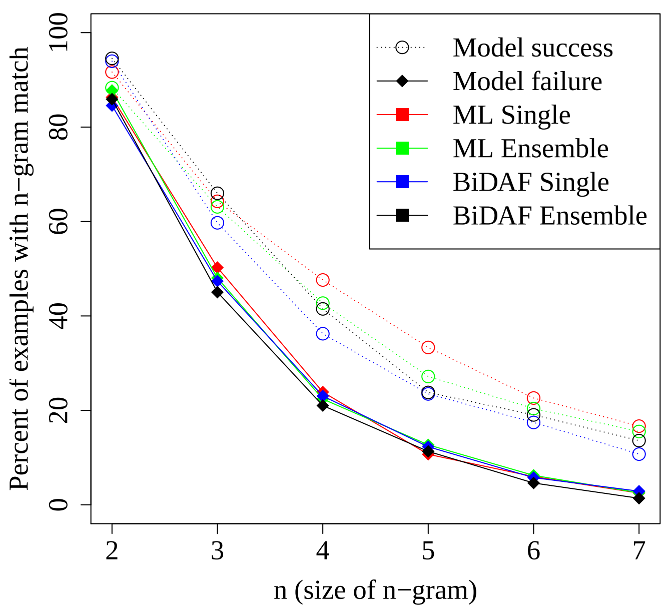


Figure 5.3: Fraction of model successes and failures on AddSent for which the question has an exact n-gram match with the original paragraph. For each model and each value of n, successes are more likely to have an n-gram match than failures.

we found that models do well when the question has an exact n-gram match with the original paragraph. Figure 5.3 plots the fraction of examples for which an n-gram in the question appears verbatim in the original passage; this is much higher for model successes. For example, 41.5% of BiDAF Ensemble successes had a 4-gram in common with the original paragraph, compared to only 21.0% of model failures.

We also found that models succeeded more often on short questions. Figure 5.4 shows the distribution of question length on model successes and failures; successes tend to involve shorter questions. For example, 32.7% of the questions in BiDAF Ensemble successes were 8 words or shorter, compared to only 11.8% for model failures. This effect arises because AddSent always changes at least one word in the question. For long questions, changing one word leaves many others unchanged, so the adversarial sentence still has many words in common with the question. For short questions, changing one content word may be enough to make the adversarial sentence completely irrelevant.

5.3.6 Transferability across models

In computer vision, adversarial examples that fool one model also tend to fool other models (Szegedy et al., 2014; Moosavi-Dezfooli et al., 2017); we investigate whether the same pattern holds for us. Examples from AddOneSent clearly do transfer across models, since AddOneSent always adds

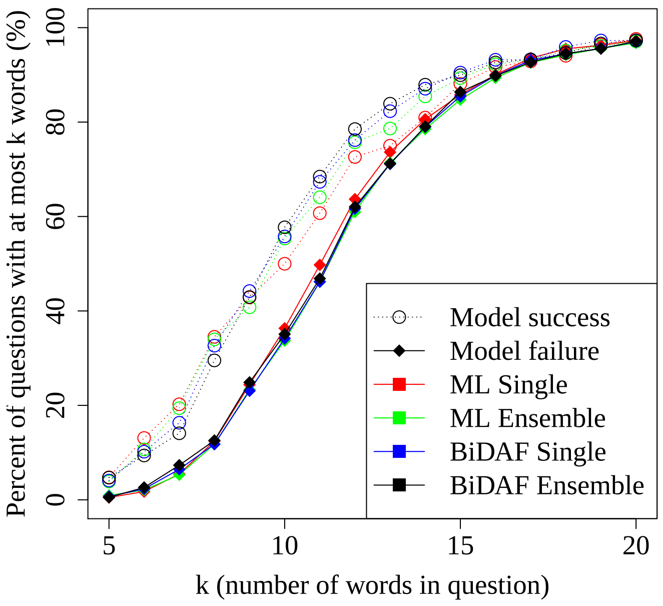


Figure 5.4: For model successes and failures on AddSent, the cumulative distribution function of the number of words in the question (for each k, what fraction of questions have $\leq k$ words). Successes are more likely to involve short questions.

the same adversarial sentence regardless of model.

Table 5.6 shows the results of evaluating the four main models on adversarial examples generated by running either AddSent or AddAny against each model. AddSent adversarial examples transfer between models quite effectively; in particular, they are harder than AddOneSent examples, which implies that examples that fool one model are more likely to fool other models. The AddAny adversarial examples exhibited more limited transferability between models. For both AddSent and AddAny, examples transferred slightly better between single and ensemble versions of the same model.

5.3.7 Training on adversarial examples

Finally, we tried training on adversarial examples, to see if existing models can learn to become more robust. Due to the prohibitive cost of running AddSent or AddAny on the entire training set, we instead ran only Steps 1-3 of AddSent (everything except crowdsourcing) to generate a raw adversarial sentence for each training example. We then trained the BiDAF model from scratch on the union of these examples and the original training data. As a control, we also trained a second BiDAF model on the original training data alone.[8]

The results of evaluating these models are shown in Table 5.7. At first glance, training on

[8] All previous experiments used parameters released by Seo et al. (2017)

	Model under Evaluation			
Targeted Model	Match-LSTM Single	Match-LSTM Ensemble	BiDAF Single	BiDAF Ensemble
AddSent				
Match-LSTM Single	27.3	33.4	40.3	39.1
Match-LSTM Single	31.6	29.4	40.2	38.7
BiDAF Single	32.7	34.8	34.3	37.4
BiDAF Ensemble	32.7	34.2	38.3	34.2
AddAny				
Match-LSTM Single	7.6	54.1	57.1	60.9
Match-LSTM Ensemble	44.9	11.7	50.4	54.8
BiDAF Single	58.4	60.5	4.8	46.4
BiDAF Ensemble	48.8	51.1	25.0	2.7

Table 5.6: Transferability of adversarial examples across models. Each row measures performance on adversarial examples generated to target one model; each column evaluates one (possibly different) model on these examples.

	Training data	
Test data	Original	Augmented
Original	75.8	75.1
AddSent	34.8	70.4
AddSentMod	34.3	39.2

Table 5.7: Effect of training the BiDAF Single model on the original training data alone (first column) versus augmenting the data with raw AddSent examples (second column).

adversarial data seems effective, as it largely protects against AddSent. However, further investigation shows that training on these examples has only limited utility. To demonstrate this, we created a variant of AddSent called AddSentMod, which differs from AddSent in two ways: it uses a different set of fake answers (e.g., `PERSON` named entities map to *"Charles Babbage"* instead of *"Jeff Dean"*), and it prepends the adversarial sentence to the beginning of the paragraph instead of appending it to the end. The retrained model does almost as badly as the original one on AddSentMod, suggesting that it has just learned to ignore the last sentence and reject the fake answers that AddSent usually proposed. In order for training on adversarial examples to actually improve the model, more care must be taken to ensure that the model cannot overfit the adversary.

5.4 Discussion

Despite appearing successful by standard evaluation metrics, existing machine learning systems for reading comprehension perform poorly under adversarial evaluation. Standard evaluation is overly lenient on models that rely on superficial cues. In contrast, adversarial evaluation reveals that existing models are overly stable to perturbations that alter semantics.

One root cause of these problems is that all questions in SQuAD are guaranteed to be answerable using the given context paragraph. We can contrast SQuAD with work on recognizing textual entailment (RTE), which has traditionally been motivated by downstream tasks like question answering. RTE systems must make explicit judgments about whether a hypothesis (e.g., a potential answer to a question) is entailed by a premise (e.g., a passage) (Dagan et al., 2006; Marelli et al., 2014; Bowman et al., 2015). In contrast, SQuAD models only need to select the span that seems most related to the question, compared to other spans. If there are very few spans in the paragraph that could even plausibly be the right answer, selecting the right span can be much easier than checking that the answer is actually entailed by the text. This is one way that SQuAD oversimplifies the question answering task.

In Rajpurkar et al. (2018), we attempt to address this limitation by creating SQuAD 2.0, a new version of SQuAD which forces systems to distinguish answerable and unanswerable questions. We augment SQuAD with unanswerable questions written adversarially by crowdworkers. Crowdworkers are asked to write unanswerable questions that are on the same topic as a given paragraph and similar to existing (answerable) questions in SQuAD. These questions are also guaranteed to have a *plausible answer* in the paragraph, to prevent simple type-matching heuristics from succeeding. Just like the adversarial examples in this chapter, these unanswerable questions punish overstability. In fact, the first step of the AddSent pipeline is exactly a method for creating unanswerable questions; SQuAD 2.0 contains many similar questions, but its reliance on crowdworkers increases diversity. However, as mentioned in Chapter 1, state-of-the-art models now do very well on SQuAD 2.0, and the potential for significant accuracy improvements on the SQuAD 2.0 test set seems limited. In retrospect, we were likely too influenced by the intuition of adversarial distractors from this chapter, and did not focus enough on creating a diverse set of unanswerable questions. When models are permitted to train on these types of unanswerable questions, as they are in SQuAD 2.0, they stop being very difficult to solve.

How can we collect better datasets with unanswerable questions? In SQuAD 2.0, we tried to collect ones that intuitively seemed most challenging, but did so at the cost of both diversity and naturalness—real users might not ask these questions. A more natural way to collect unanswerable questions is to have crowdworkers who cannot see the context passage ask information-seeking questions, as was done in the QuAC dataset (Choi et al., 2018). Many of these questions are not answerable based on the passage alone. However, since the question-asker cannot see the passage, they have no way of knowing if their questions are actually hard to identify as unanswerable. Other datasets pair existing questions with passages retrieved by a search engine. In the Natural Questions dataset (Kwiatkowski et al., 2019), questions from Google Search users are paired with retrieved Wikipedia articles, and annotators read the article to determine if it answers the question. In principle, one could pair every question with any document that does not answer it, leading to a very diverse (but mostly easy) pool of document-question pairs from which the question cannot be answered. In Rajpurkar et al. (2018), we argued that these kinds of unanswerable questions tend to be too easy and therefore lead to less challenging datasets. However, our analysis focused on datasets that have a similar number of answerable and unanswerable questions. As we will argue in the next chapter, such artificial label balance can lead to dramatically inflated estimates of how easy a task is.

Chapter 6

Active Learning for Imbalanced Pairwise Tasks

In the previous chapter, we fooled question answering models by appending adversarial distracting sentences to the ends of paragraphs. Ideally, these adversarial examples would not only expose robustness failures, but also guide research on improving model robustness. However, it is unclear how one would turn these adversarial examples into a meaningful, clearly-defined challenge for system developers. If developers know in advance how the distracting sentences are generated at test time, it is trivial to achieve high adversarial accuracy merely by adding examples with distracting sentences to the training data, as shown in Section 5.3.7. However, this clearly does not solve the larger robustness problem, as models trained in this way do not generalize to other distracting sentences created in a slightly different way and placed in a different position. On the other hand, if the developer is not given any knowledge about what might happen at test time, it is very unclear how to make significant improvements (besides just increasing accuracy on SQuAD overall), since the space of all possible distracting sentences is extremely large.

Contrast this with Chapter 3 and Chapter 4. In these chapters, we trained systems knowing in advance what types of perturbations were possible—word substitutions in Chapter 3 and typos in Chapter 4. However, this knowledge alone did *not* make achieving high adversarial accuracy trivial. There were many possible perturbations of every input, and we could not train on all of them, so we had to find non-standard ways to minimize the loss on the worst perturbation. This again hints at what seems to be missing from Chapter 5: a way to automatically generate many different distracting sentences.

As it turns out, very large sets of distracting sentences are not so hard to find: they arise all the time in real applications. When a search engine receives a question from a user, it must sift through all of the written content on the web to find a webpage that answers that question. We can think of all webpages that do not answer the question as "distractors." A random distractor (i.e., a randomly sampled webpage that does not answer the question) will almost always be very easy to label as irrelevant. But this does not imply that the task is easy! If a system is to rank a relevant webpage as the top search result, it must rank it higher than *every* other webpage on the internet. In other words, it must correctly assign a low relevance score to a distracting webpage that has been *adversarially chosen* from among all the webpages in the world.

In this chapter, we study pairwise classification tasks with natural extreme label imbalance (e.g., 99.99% of examples have the same label). Many well-studied NLP tasks fit within this category. In question deduplication (Iyer et al., 2017), the vast majority of pairs of questions from an online forum are not duplicates. In open-domain question answering (Yang et al., 2015; Chen et al., 2017; Nogueira and Cho, 2019; Lee et al., 2019), almost any randomly sampled document will not answer a given question. In natural language inference (Dagan et al., 2006; Bowman et al., 2015), random pairs of sentences from a diverse distribution will almost always have no relation between them, as opposed to an entailment or contradiction relationship.

We connect this setting to the adversarial settings in previous chapters by noting that when a binary classification task is extremely imbalanced, achieving high precision requires extremely high accuracy on the negative examples, since predicting wrong on a very small fraction of negatives is enough to overwhelm all the positives. We can think of the negatives that are most challenging to a model as a type of naturally occurring adversarial example: they are worst-case examples from a very large pool of possible examples. This analogy with adversarial examples is admittedly somewhat loose. For adversarial perturbations, we wanted to guarantee that *no* perturbed inputs x' in some neighborhood around x are classified incorrectly; to have high precision, we just need *not too many* of the negatives to be classified incorrectly, compared to the number of positives. Nonetheless, these settings do share the property that uniform random sampling—either of perturbations or of negative examples—cannot measure whether a model has succeeded at the corresponding goal (either robustness to adversarial perturbations or high precision) without an intractably large number of samples. Moreover, since it is impractical to train models on extremely imbalanced training data, evaluating on extremely imbalanced test data forces models to overcome a form of train-test mismatch.

Compared to studying adversarial perturbations, extremely imbalanced pairwise classification introduces an important new dimension to the problem of robustness: the choice of how to collect training data. In the adversarial perturbation setting, we knew at training time exactly what perturbations would be allowed at test time; no analogous information is available for pairwise classification. Therefore, to anticipate the most challenging negative examples, we need to have the right training data. While past work has recognized the importance of label imbalance in NLP datasets (Lewis et al., 2004; Chawla et al., 2004), many recently released datasets are heuristically collected to ensure label balance, generally for ease of training. For instance, the Quora Question

Pairs (QQP) dataset (Iyer et al., 2017) was generated by mining non-duplicate questions that were heuristically determined to be near-duplicates. The SNLI dataset had crowdworkers generate inputs to match a specified label distribution (Bowman et al., 2015). We show that models trained on heuristically balanced datasets deal poorly with natural label imbalance at test time: they have very low average precision on realistically imbalanced test data created by taking all pairs of in-domain utterances.

To jointly study data collection and modeling, we adopt the framework of pool-based active learning (Settles, 2010), in which system developers have *query access* to (i.e., the ability to label a limited subset of) a large pool of unlabeled examples (e.g., unpaired documents and questions, for open-domain question answering). The goal is to achieve the best possible average precision on a test dataset that has realistic label imbalance, through a combination of better training data selection and better modeling. Note that unlike the previous chapter, when we could fix model behavior by training on distracting sentences, here there is no obvious way to choose the training data. Since we consider a much broader set of "distractors" (negative examples), the training dataset cannot cover all of them while preserving label balance, which is important for learning.

Thus, we seek a method of collecting training data that intentionally does not sample from the test distribution, but nonetheless ensures good generalization to the test distribution. Fortunately, standard active learning methods have precisely this behavior. In active learning, a model trained on previously collected data is used to choose new examples to label; this intentionally creates a mismatch between the training distribution and test distribution, but in a way that actually improves test error (Settles, 2010; Mussmann and Liang, 2018). We collect balanced and informative training data using uncertainty sampling, an active learning method that queries labels for examples on which a model trained on previously queried data has high uncertainty (Lewis and Gale, 1994). To lower the computational cost of searching for uncertain points, we propose combining active learning with a BERT embedding model for which uncertain points can be located efficiently using nearest neighbor search.

Empirically, we demonstrate that active learning in pairwise classification tasks collects training data that greatly improves generalization to imbalanced test data, compared with recent benchmark datasets. When trained on standard, heuristically-collected training data, state-of-the-art models have only 2.4% average precision on imbalanced versions of both the Quora Question Pairs (QQP) paraphrasing dataset and WikiQA question answering dataset. In contrast, our BERT embedding model trained with data collected via active learning achieves average precision to 32.5% on QQP and 20.1% on WikiQA.

6.1 Setting

We study binary classification with an input space $\mathcal{X}$ and output space $\{0, 1\}$. We assume the label y is a deterministic function of x, which we write $y(x)$. A classification model p_θ yields probability estimates $p_\theta(y \mid x)$ where $x \in \mathcal{X}$.

Our setting has two aspects: the way training data is collected via label queries (Section 6.1.1) and the way we evaluate the model $p_\theta(y \mid x)$ by measuring average precision (Section 6.1.2). We focus on pairwise tasks (Section 6.1.3), which enables efficient active learning (Section 6.3).

6.1.1 Data collection

In our setting, a system is given an unlabeled dataset $D_{\text{all}}^{\text{train}} \subseteq \mathcal{X}$. The system can query an input $x \in D_{\text{all}}^{\text{train}}$ and receive the corresponding label $y(x)$. The system is given a budget of n queries to build a labeled training dataset of size n.

6.1.2 Evaluation

Following standard practice for imbalanced tasks, we evaluate on precision, recall, and average precision (Lewis, 1995; Manning et al., 2008). A scoring function $S : \mathcal{X} \to \mathbb{R}$ is used to rank examples x, where an ideal S assigns all positive examples (i.e., x's such that $y(x) = 1$) a higher score than all negative examples (i.e., x's such that $y(x) = 0$). Throughout this chapter, we directly use the model p_θ as the scoring function, i.e., $S(x) = p_\theta(y = 1 \mid x)$. Given a test dataset $D_{\text{all}}^{\text{test}} \subseteq \mathcal{X}$, define the number of true positives, false positives, and false negatives of a scoring function S at a threshold γ as:

$$\text{TP}(S, \gamma) = \sum_{x \in D_{\text{all}}^{\text{test}}} \mathbf{1}[y(x) = 1 \wedge S(x) \geq \gamma] \tag{6.1}$$

$$\text{FP}(S, \gamma) = \sum_{x \in D_{\text{all}}^{\text{test}}} \mathbf{1}[y(x) = 0 \wedge S(x) \geq \gamma] \tag{6.2}$$

$$\text{FN}(S, \gamma) = \sum_{x \in D_{\text{all}}^{\text{test}}} \mathbf{1}[y(x) = 1 \wedge S(x) < \gamma]. \tag{6.3}$$

For any threshold γ, define the precision $P(S, \gamma)$ and recall $R(S, \gamma)$ of a scoring function S as:

$$\text{P}(S, \gamma) = \frac{\text{TP}(S, \gamma)}{\text{TP}(S, \gamma) + \text{FP}(S, \gamma)} \tag{6.4}$$

$$\text{R}(S, \gamma) = \frac{\text{TP}(S, \gamma)}{\text{TP}(S, \gamma) + \text{FN}(S, \gamma)}. \tag{6.5}$$

Let $\Gamma = \{S(x) : x \in D_{\text{all}}^{\text{test}}\}$ be the set of all unique scores predicted for some input in the dataset.

By sweeping over all values Γ in descending order, we trace out the precision-recall curve. The area under the precision recall curve, also known as average precision (AP), is defined as:

$$\mathrm{AP}(S) = \sum_{i=1}^{|\Gamma|} (\mathrm{R}(S, \gamma_i) - \mathrm{R}(S, \gamma_{i-1}))\mathrm{P}(S, \gamma_i), \tag{6.6}$$

where $\gamma_0 = \infty$ and $\gamma_1 > \gamma_2 > \dots \gamma_{|\Gamma|}$ and $\gamma_i \in \Gamma$.

Note that high precision requires very high accuracy when the task is extremely imbalanced. For example, if only one in $10,000$ examples in $D_{\text{all}}^{\text{test}}$ is positive and 50% precision at some recall is achieved, that implies 99.99% accuracy.

6.1.3 Pairwise tasks

We focus on "pairwise" tasks, meaning that the input space decomposes as $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$. For instance, $\mathcal{X}_1$ could be questions and $\mathcal{X}_2$ could be paragraphs for open-domain question answering, and $\mathcal{X}_1 = \mathcal{X}_2$ could be questions for question deduplication. We create the unlabeled *all-pairs* training set $D_{\text{all}}^{\text{train}}$ by sampling $D_1^{\text{train}} \subseteq \mathcal{X}_1$ and $D_2^{\text{train}} \subseteq \mathcal{X}_2$, and define $D_{\text{all}}^{\text{train}} = D_1^{\text{train}} \times D_2^{\text{train}}$. We follow a similar procedure to form the all-pairs test set: $D_{\text{all}}^{\text{test}} = D_1^{\text{test}} \times D_2^{\text{test}}$ where $D_1^{\text{test}} \subseteq \mathcal{X}_1$ and $D_2^{\text{test}} \subseteq \mathcal{X}_2$. We ensure that the train and test sets are disjoint by ensuring $D_1^{\text{train}} \cap D_1^{\text{test}} = \varnothing$ and $D_2^{\text{train}} \cap D_2^{\text{test}} = \varnothing$.

Many pairwise tasks require high average precision on all-pairs test data. A question deduplication system must compare a new question with all previously asked questions to determine if a duplicate exists. An open-domain question-answering system must search through all available documents for one that answers the question. In both cases, the all-pairs distribution is extremely imbalanced, as the vast majority of pairs are negatives, while standard datasets are artificially balanced.

6.2 Results training on heuristic datasets

In this section, we show that state-of-the-art models trained on two standard pairwise classification datasets—QQP and WikiQA—do not generalize well to our extremely imbalanced all-pairs test data. Both QQP and WikiQA were collected using static heuristics that attempt to find points $x \in \mathcal{X}$ that are relatively likely to be positive. These heuristics are necessary because uniformly sampling from $\mathcal{X}$ is impractical due to the label imbalance: if the proportion of positives is 10^{-4}, then random sampling would have to label 10,000 examples on average to find one positive example. Standard models can achieve high test accuracy on test data collected with these heuristics, but fare poorly when evaluated on all-pairs data derived from the same data source (Section 6.2.4). Manual inspection confirms that these models often make surprising false positive errors (Section 6.2.5).

6.2.1 Evaluation

We evaluate models on both in-distribution test data and our imbalanced all-pairs test data.

In-distribution evaluation. Let D_{pos} denote the set of all positive examples, and $D_{\text{statedneg}}$ denote the set of *stated negative examples*—negative examples in the original heuristically collected dataset. We define the stated test dataset $D_{\text{heur}}^{\text{test}}$ as the set of all positive and stated negative examples that are contained in $D_{\text{all}}^{\text{test}}$. Formally, $D_{\text{heur}}^{\text{test}} = (D_{\text{pos}} \cup D_{\text{statedneg}}) \cap D_{\text{all}}^{\text{test}}$. To evaluate on $D_{\text{heur}}^{\text{test}}$, we use task-specific evaluation metrics described in the next section.

All-pairs evaluation. All-pairs evaluation metrics depend on the label of every pair in $D_{\text{all}}^{\text{test}}$. We approximate these labels by imputing (possibly noisy) labels on all pairs using the available labeled data, as described in the next section.[1] In Section 6.2.5, we manually label examples to confirm our results from this automatic evaluation.

Computing the number of false positives $\text{FP}(S, \gamma)$ requires enumerating all negative examples in $D_{\text{all}}^{\text{test}}$, which is too computationally expensive with our datasets. To get an unbiased estimate of $\text{FP}(S, \gamma)$, we could randomly downsample $D_{\text{all}}^{\text{test}}$, but the resulting estimator has high variance. We instead propose a new unbiased estimator that uses importance sampling. We combine counts of errors on a set of "nearby negative" examples $D_{\text{near}}^{\text{test}} \subseteq D_{\text{all}}^{\text{test}}$, pairs of similar utterances on which we expect more false positives to occur, and random negatives $D_{\text{rand}}^{\text{test}}$ sampled uniformly from negatives in $D_{\text{all}}^{\text{test}} \setminus D_{\text{near}}^{\text{test}}$. By weighting the number of false positives on $D_{\text{near}}^{\text{test}}$ and $D_{\text{all}}^{\text{test}}$ appropriately and adding them together, we obtain an unbiased estimate of $\text{FP}(S, \gamma)$. Details are provided in Appendix B.1.

6.2.2 Datasets

Quora Question Pairs (QQP). The task for QQP (Iyer et al., 2017) is to determine whether two questions are paraphrases. The non-paraphrase pairs in the dataset were chosen heuristically, e.g., by finding questions on similar topics. We impute labels on all question pairs by assuming that two questions are paraphrases if and only if they are equivalent under the transitive closure of the equivalence relation defined by the labeled paraphrase pairs.[2] We randomly partition all unique questions into train, dev, and test splits D_1^{train}, D_1^{dev}, and D_1^{test}, ensuring that no two questions that were paired (either in positive or negative examples) in the original dataset end up in different splits. Since every question is trivially a paraphrase of itself, we define $D_{\text{all}}^{\text{train}}$ as the set of *distinct* pairs of questions in D_1^{train}, and define $D_{\text{all}}^{\text{dev}}$ and $D_{\text{all}}^{\text{test}}$ analogously. For in-distribution evaluation, we report accuracy and F1 score on $D_{\text{heur}}^{\text{test}}$, as in Wang et al. (2019b).

[1] In practice, precision can be estimated by labeling predicted positives, and recall can be estimated with respect to a non-exhaustive set of known positives (Harman, 1992; Ji et al., 2011).

[2] Using the transitive closure increases the total number of positives from 149,263 to 228,548, so this adds many positives but does not overwhelm the original data.

Split	Positives	Total pairs	Ratio	Stated Negatives	Nearby Negatives
QQP					
Train	124,625	38B	1:300k	132,796	-
Dev	60,510	8.8B	1:150k	61,645	13.1M
Test	43,413	8.5B	1:190k	60,575	12.8M
WikiQA					
Train	1,040	56M	1:53k	19,320	-
Dev	140	7.8M	1:56k	2,593	29,511
Test	293	17M	1:57k	5,872	63,136

Table 6.1: Statistics of our QQP and WikiQA splits.

WikiQA. The task for WikiQA (Yang et al., 2015) is to determine whether a question is answered by a given sentence. The dataset only includes examples that pair a question with sentences from a Wikipedia article believed to be relevant based on click logs. We impute labels by assuming that question-sentence pairs not labeled in the dataset are negative. We partition the questions into D_1^{train}, D_1^{dev}, and D_1^{test}, following the original question-based split of the dataset, and define $D_2^{\text{train}} = D_2^{\text{dev}} = D_2^{\text{test}}$ to be the set of all sentences in the dataset. For all WikiQA models, we prepend the title of the source article to the sentence to give the model information about the sentence's origin, as in Lee et al. (2019).

For in-distribution evaluation, we report two evaluation metrics. Following standard practice, we report clean mean average precision (c-MAP), defined as MAP over "clean" test questions—questions that are involved in both a positive and negative example in $D_{\text{heur}}^{\text{test}}$ (Garg et al., 2020). Restricting to clean examples mostly prunes out questions that are not answered by any sentence in the dataset. We also report F1 score across all examples in $D_{\text{heur}}^{\text{test}}$ (a-F1). Unlike c-MAP, a-F1 does evaluate a system's ability to recognize when a question cannot be answered by any available sentences. Evaluating on all of $D_{\text{heur}}^{\text{test}}$ introduces more label imbalance: only 6% of $D_{\text{heur}}^{\text{test}}$ is positive, while 12% of clean examples in $D_{\text{heur}}^{\text{test}}$ are positive. The original WikiQA paper advocated a-F1 (Yang et al., 2015), but most subsequent papers do not report it (Shen et al., 2017a; Yoon et al., 2019; Garg et al., 2020).

Data statistics. Table 6.1 shows statistics for the QQP and WikiQA splits we use. Models in this section are trained on the *stated training dataset* $D_{\text{heur}}^{\text{train}} \overset{\text{def}}{=} (D_{\text{pos}} \cup D_{\text{statedneg}}) \cap D_{\text{all}}^{\text{train}}$, the set of all positives and stated negatives in the train split. For all-pairs evaluation, both QQP and WikiQA have extreme label imbalance: Positive examples make up between 1 in 50,000 (WikiQA) and 1 in 200,000 (QQP) of the test examples.

QQP	**In-distribution**		**All pairs**	
	Accuracy	F1	P@R20	AP
BERT	82.5%	77.3%	3.0%	2.4%
XLNet	83.0%	77.9%	1.7%	1.4%
RoBERTa	84.4%	80.2%	2.5%	2.0%
ALBERT	79.6%	73.0%	3.5%	1.9%
WikiQA	c-MAP	a-F1	P@R=20	AP
BERT	79.9%	45.9%	6.5%	2.4%
XLNet	80.5%	46.7%	1.0%	1.0%
RoBERTa	84.6%	53.6%	3.4%	2.3%
ALBERT	78.2%	41.8%	0.7%	0.9%

Table 6.2: State-of-the-art Concat models trained on heuristically collected data generalize to test data from the same distribution, but not to all-pairs data.

6.2.3 Models

We train four state-of-the-art models that use BERT-base (Devlin et al., 2019), XLNet-base (Yang et al., 2019), RoBERTa-base (Liu et al., 2019b), and ALBERT-base-v2 (Lan et al., 2020), respectively. As is standard, all models receive as input the concatenation of x_1 and x_2; we refer to these as concatenation-based (Concat) models. We train on binary cross-entropy loss for 2 epochs on QQP and 3 epochs on WikiQA, chosen to maximize dev all-pairs AP for RoBERTa. We report the average over three random seeds for training.

6.2.4 Evaluation results

As shown in Table 6.2, state-of-the-art models trained only on stated training data do well on balanced in-distribution test data but poorly on the extremely imbalanced all-pairs test data. On QQP, the best model gets 80.2% F1 on in-distribution test examples.[3] However, on all-pairs test data, the best model can only reach 3.5% precision at a modest 20% recall. On WikiQA, our best c-MAP of 84.6% is higher than the best previously reported c-MAP without using additional question-answering data, 83.6% (Garg et al., 2020). However, on all-pairs test data, the best model gets 6.5% precision at 20% recall. All-questions F1 on in-distribution data is also quite low, with the best model only achieving 53.6%. Since a-F1 evaluates on a more imbalanced distribution than c-MAP, this further demonstrates that state-of-the-art models deal poorly with test-time label

[3]Our QQP in-domain accuracy numbers are lower than those on the GLUE leaderboard, which has accuracies in the low 90's, due to a more challenging train/test split. First, our training set is smaller (257k examples versus 364k). Second, our split is more challenging because the model does not see the same questions or even same paraphrase clusters at training time and test time. Finally, our test set is more balanced (58% negative) than the GLUE QQP dev set (63% negative; test set balance is unknown). As a sanity check, we confirmed that our RoBERTa implementation can achieve 91.5% dev accuracy when trained and tested on the GLUE train/dev split, in line with previously reported results (Liu et al., 2019b).

QQP, ConcatBERT trained on $D^{\text{train}}_{\text{heur}}$

x_1: *"How do I overcome **seasonal affective disorder**?"*
x_2 :*"How do I solve **puberty problem**?"*

x_1: *"What will **10000** A.D be like?"*
x_2 :*"Does not introduction of new **Rs.2000** notes ease carrying black money in future?"*

x_1: *"Can a person with no Coding knowledge learn **Machine learning**?"*
x_2: *"How do I learn **Natural Language Processing**?"*

WikiQA, ConcatBERT trained on $D^{\text{train}}_{\text{heur}}$

x_1: *"where does **limestone** form?"*
x_2: *"Glacier cave . A **glacier cave** is a cave formed within the ice of a glacier ."*

x_1: *"what is **gravy** made of?"*
x_2: *"**Amaretto**. It is made from a base of apricot pits or almonds, sometimes both."*

Figure 6.1: Examples of confident false positives from the all-pairs test distribution for models trained on examples from the original QQP and WikiQA datasets. Bold highlights non-equivalent phrases.

imbalance. Compared with a-F1, all-pairs evaluation additionally shows that models make many mistakes when evaluated on questions paired with less related sentences; these examples should be easier to identify as negatives, but are missing from $D^{\text{train}}_{\text{heur}}$.

6.2.5 Manual verification of imputed negatives

Since our results depend on automatically imputed negative labels, we manually labeled putative false positive errors—false positives as defined by our imputed labels—to more accurately estimate precision. We focused on the best QQP model and random seed combination on the development set, which got 8.2% precision at 20% recall.[4] For this recall threshold, we manually labeled 50 randomly chosen putative false positives from $D^{\text{dev}}_{\text{near}}$, and 50 more from $D^{\text{dev}}_{\text{rand}}$. We confirmed that 72% and 92%, respectively, were real false positive errors. Based on these results, we estimate the true precision of the model to be 9.5%, still close to our original estimate of 8.2%. See Appendix B.2 for details on how we derive this updated precision estimate.

Figure 6.1 shows false positive predictions at 20% recall for the best QQP and WikiQA models. For QQP, models often make surprising errors on pairs of unrelated questions (first two examples), as well as questions that are somewhat related but distinct (third example). For WikiQA, models often predict a positive label when something in the sentence has the same type as the answer to the question, even if the sentence and question are unrelated.

[4]By manual inspection, QQP had more borderline cases than WikiQA, so we focused on QQP.

6.3 Active learning for pairwise tasks

As shown above, training on heuristically collected balanced data leads to low average precision on all pairs. How can we collect training data that leads to high average precision? We turn to active learning, in which new data is chosen adaptively based on previously collected data. Adaptivity allows us to ignore the vast majority of obvious negatives (unlike random sampling) and iteratively correct the errors of our model (unlike static data collection) by collecting more data around the model's decision boundary.

6.3.1 Active learning

Active learning is a well-studied class of methods for adaptive data collection (Settles, 2010). The key observation behind active learning is that it can sometimes be more sample efficient to sample training data from a different distribution as the test distribution—thus violating the standard i.i.d. assumption of supervised learning. A large body of work has shown that the train-test mismatch induced by standard active learning approaches can actually be beneficial for generalizing to the test distribution, both empirically (Settles, 2010; Yang and Loog, 2018) and theoretically (Balcan et al., 2007; Balcan and Long, 2013; Mussmann and Liang, 2018).

An active learning method takes as input an unlabeled dataset $D_{\text{all}}^{\text{train}} \subseteq \mathcal{X}$. Data is collected in a series of $k > 1$ rounds. For the i^{th} round, we choose a batch $B_i \subseteq D_{\text{all}}^{\text{train}}$ of size n_i and observe the outcome as the labels $\{(x, y(x)) : x \in B_i\}$. The budget n is the total number of points labeled, i.e., $n = \sum_{i=1}^{k} n_i$. This process is adaptive because we can choose batch B_i based on the labels of the previous $i-1$ batches. Static data collection corresponds to setting $k = 1$.

Uncertainty sampling. The main active learning algorithm we use is uncertainty sampling (Lewis and Gale, 1994). Uncertainty sampling first uses a static data collection method to select the *seed set* B_1. For the next $k-1$ rounds, uncertainty sampling trains a model on all collected data and chooses B_i to be the n_i unlabeled points in $D_{\text{all}}^{\text{train}}$ on which the model is most uncertain. For binary classification, the most uncertain points are the points where $p_\theta(y = 1 \mid x)$ is closest to $\frac{1}{2}$. Note that a brute force approach to finding B_i requires evaluating p_θ on every example in $D_{\text{all}}^{\text{train}}$, which can be prohibitively expensive. In balanced settings, it suffices to choose the most uncertain point from a small random subset of $D_{\text{all}}^{\text{train}}$ (Ertekin et al., 2007); however, this strategy works poorly in extremely imbalanced settings, as a small random subset of $D_{\text{all}}^{\text{train}}$ is unlikely to contain any uncertain points. In Section 6.3.2, we address this computational challenge with a bespoke model architecture.

Adaptive retrieval. We also use a related algorithm we call adaptive retrieval, which is like uncertainty sampling but queries the n_i unlabeled points in $D_{\text{all}}^{\text{train}}$ with highest $p_\theta(y = 1 \mid x)$ (i.e., pairs the model is most confident are positive). Adaptive retrieval can be seen as greedily maximizing the number of positive examples collected.

6.3.2 Modeling and implementation

We now fully specify our approach by describing our model, how we find pairs in the unlabeled pool $D_{\text{all}}^{\text{train}}$ to query, and how we choose the seed set B_1. In particular, a key technical challenge is that the set of training pairs $D_{\text{all}}^{\text{train}}$ is too large to enumerate, as it grows quadratically. We therefore require an efficient way to locate the uncertain points in $D_{\text{all}}^{\text{train}}$. We solve this problem with a model architecture CosineBERT that enables efficient nearest neighbor search (Gillick et al., 2019).

Model. Given input $x = (x_1, x_2)$, CosineBERT embeds x_1 and x_2 independently and predicts $p_\theta(y = 1 \mid x)$ based on vector-space similarity. More precisely,

$$p_\theta(y = 1 \mid x) = \sigma\left(w \cdot \frac{e_\theta(x_1) \cdot e_\theta(x_2)}{\|e_\theta(x_1)\|\|e_\theta(x_2)\|} + b\right), \tag{6.7}$$

where σ is the sigmoid function, $w > 0$ and b are learnable parameters, and $e_\theta : \mathcal{X}_1 \cup \mathcal{X}_2 \to \mathbb{R}^d$ is a learnable embedding function. In other words, we compute the cosine similarity of the embeddings of x_1 and x_2, and predict y using a logistic regression model with cosine similarity as its only feature. We define e_θ as the final layer output of a BERT model (Devlin et al., 2019) mean-pooled across all tokens (Reimers and Gurevych, 2019).[5] Gillick et al. (2019) used a similar model for entity linking.

Finding points to query. Next, we show how to choose the batch B_i of points to query, given a model $p_\theta(y \mid x)$ trained on data from batches $B_1, \ldots, B_{i-1}$. Recall that uncertainty sampling chooses the points x for which for which $p_\theta(y = 1 \mid x)$ is closest to $\frac{1}{2}$, and adaptive retrieval chooses the points x with largest $p_\theta(y = 1 \mid x)$. Since the set of positives is very small compared to the size of $D_{\text{all}}^{\text{train}}$, the set of uncertain points can be found by finding points with the largest $p_\theta(y = 1 \mid x)$, thus filtering out the confident negatives, and then selecting the most uncertain from those that remain.

To find points with largest $p_\theta(y = 1 \mid x)$, we leverage the structure of our model. Since $w > 0$, $p_\theta(y = 1 \mid x)$ is increasing in the cosine similarity of $e_\theta(x_1)$ and $e_\theta(x_2)$. Therefore, it suffices to find pairs $(x_1, x_2) \in D_{\text{all}}^{\text{train}}$ that are nearest neighbors in the embedding space defined by e_θ. In particular, for each $x_1 \in \mathcal{X}_1$, we use the Faiss library (Johnson et al., 2019) to retrieve a set $N(x_1)$ containing the m nearest neighbors in $\mathcal{X}_2$, and define $D_{\text{close}}^{\text{train}}$ to be the set of all pairs (x_1, x_2) such that $x_2 \in N(x_1)$. We then iterate through $D_{\text{close}}^{\text{train}}$ to find either the most uncertain points (for uncertainty

[5] Although WikiQA involves an asymmetric relationship between questions and sentences, we use the same encoder for both. This is no less expressive than using separate encoders for each, as the set of questions and set of sentences are disjoint. For asymmetric tasks like NLI where $\mathcal{X}_1 = \mathcal{X}_2$, we would need to use separate encoders for the $\mathcal{X}_1$ and $\mathcal{X}_2$.

Hyperparameter	Value
Learning rate	2×10^{-5}
Training epochs	2
Weight decay	0
Optimizer	AdamW
AdamW Epsilon	1×10^{-6}
Batch size	16

Table 6.3: Hyperparameter choices for QQP and WikiQA.

sampling) or points with highest cosine similarity (for adaptive retrieval). Note that this method only requires $|D_1^{\text{train}}| + |D_2^{\text{train}}|$ embedding calls, rather than $|D_1^{\text{train}}| \cdot |D_2^{\text{train}}|$, the requirement for jointly embedding all pairs.

Choosing the seed set. To choose the seed set B_1, we use the CosineBERT model initialized with the pre-trained BERT parameters as the embedding e_θ and select the n_1 pairs with largest $p_\theta(y = 1 \mid x)$. Recall that $w > 0$, so this amounts to choosing the pairs with highest cosine similarity.

6.4 Active learning experiments

6.4.1 Experimental details

We collect $n_1 = 2048$ examples in the seed set, and use $k = 10$ rounds of active learning for QQP and $k = 4$ for WikiQA, as WikiQA is much smaller. At round i, we query $n_i = n_1 \cdot (3/2)^{i-1}$ new labels. These choices imply a total labeling budget n of 232,100 for QQP and 16,640 for WikiQA. For both datasets, n is slightly less than $|D_{\text{heur}}^{\text{train}}|$ (257,421 for QQP and 20,360 for WikiQA), thus ensuring a meaningful comparison with training on heuristic data. The exponentially growing n_i helps us avoid wasting queries in early rounds, when the model is worse, and also makes training faster in the early rounds. We retrieve $m = 1000$ nearest neighbors per $x_1 \in \mathcal{X}_1$ for QQP and $m = 100$ for WikiQA. We run all active learning experiments with three different random seeds and report the mean.

At each round of active learning, we train for 2 epochs. We train without dropout, as dropout artificially lowers cosine similarities at training time. We apply batch normalization (Ioffe and Szegedy, 2015) to the cosine similarity layer to rescale the cosine similarities, as they often are very close to 1. We initialize w and b so that high cosine similarities correspond to the positive label, and constrain w to be nonnegative during training. We use a maximum sequence length of 128 word piece tokens. To compensate for BERT's low learning rate, we increased the learning rate on the w and b parameters by a factor of 10^4. Table 6.3 shows all hyperparameters used for training. Hyperparameters were tuned on the development set of QQP; we found these same hyperparameters

Method	QQP		WikiQA	
	P@R20	AP	P@R20	AP
Random	4.7%	2.8%	1.3%	1.3%
Stated data	29.3%	15.4%	0.7%	2.2%
Static retrieval	49.2%	25.1%	13.9%	8.2%
Adaptive retrieval	59.1%	32.4%	27.1%	15.1%
Uncertainty	**60.2%**	**32.5%**	**32.4%**	**20.1%**

Table 6.4: Main results comparing different data collection strategies on all three datasets. The two active learning methods, adaptive retrieval and uncertainty sampling, greatly outperform other methods.

Positives Found	QQP	WikiQA
Random sampling	1	1
Static retrieval	16,422	169
Adaptive retrieval	**103,181**	**757**
Uncertainty sampling	87,594	742
Total examples collected	232,100	16,640

Table 6.5: Number of positive points collected by different methods. All methods collect the same number of total examples (last row).

also worked well for WikiQA, and so we did not tune them separately for WikiQA. In most cases, we used the default hyperparameters for BERT.

At the end of training, we freeze the embeddings e_θ and train the output layer parameters w and b to convergence, to improve uncertainty estimates for uncertainty sampling. This process amounts to training a two-parameter logistic regression model. We optimize this using (batch) gradient descent with learning rate 1 and $10,000$ iterations. When training this model, we normalize the cosine similarity feature to have zero mean and unit variance across the training dataset. Training this logistic regression model is very fast compared to running the embedding model.

6.4.2 Main results

We now compare the two active learning methods, adaptive retrieval and uncertainty sampling, with training on $D_{\text{heur}}^{\text{train}}$ and two other baselines. Random sampling queries n pairs uniformly at random, which creates a very imbalanced dataset. Static retrieval queries the n most similar pairs using the pre-trained BERT embedding, similar to how we create the seed set in Section 6.3.2. Table 6.4 shows all-pairs evaluation for CosineBERT trained on these datasets. The two active learning methods greatly outperform other methods: Uncertainty sampling gets 32.5% AP on QQP and 20.1% on WikiQA, while the best static data collection method, static retrieval, gets only 25.1% AP on QQP and 8.2% AP on WikiQA. Recall from Table 6.2 that ConcatBERT only achieved

2.4% AP on both QQP and WikiQA. Uncertainty sampling slightly outperforms adaptive retrieval on both datasets. On QQP, CosineBERT outperforms ConcatBERT (from Table 6.2) when both are trained on $D_{\text{heur}}^{\text{train}}$, we hypothesize that the cosine similarity structure helps it generalize better to pairs of unrelated questions. However, this pattern does not hold for WikiQA, perhaps because question answering is an asymmetric task and thus not as suited to modeling with cosine similarity, which is symmetric.

Achieving high precision across all pairs requires collecting both enough positive examples and useful negative examples. Compared to random sampling and static retrieval, active learning collects many more positive examples, as shown in Table 6.5. $D_{\text{heur}}^{\text{train}}$ contains all positive examples, but models trained on it still have low AP on all pairs. We conclude that the negative examples in $D_{\text{heur}}^{\text{train}}$ are insufficient for generalization to all pairs, while active learning chooses more useful negatives.

6.4.3 Manual verification of imputed negatives

As in Section 6.2.5, we manually labeled putative QQP false positives at the threshold where recall is 20% for CosineBERT trained on either stated data or uncertainty sampling data. For each, we labeled 50 putative false positives from $D_{\text{near}}^{\text{dev}}$, and all putative false positives from $D_{\text{rand}}^{\text{dev}}$ (12 for stated data, 0 for uncertainty sampling).

CosineBERT trained on $D_{\text{heur}}^{\text{train}}$. 67% (8 of 12) of the putative false positives on $D_{\text{rand}}^{\text{dev}}$ were actual errors, but only 36% of putative false positives on $D_{\text{near}}^{\text{dev}}$ were errors. This causes us to update our estimate of development set precision at 20% recall from 28.4% to 41.4%. Overall, this model makes some more reasonable mistakes than the ConcatBERT model, though its precision is still not that high.

CosineBERT model with uncertainty sampling. Only 32% of putative false positives from $D_{\text{near}}^{\text{dev}}$ were real errors, significantly less than the 72% for ConcatBERT trained on $D_{\text{heur}}^{\text{train}}$ ($p = 7 \times 10^{-5}$, Mann-Whitney U test). Our estimate of development set precision at 20% recall increases from 55.1% to 79.3%, showing that uncertainty sampling yields a much more precise model than our imputed labels indicate.

6.4.4 Comparison with stratified sampling

Next, we further confirm that having all the positive examples is not sufficient for high precision. In Table 6.6, we compare with two variants of stratified sampling, in which positive and negative examples are independently subsampled at a desired ratio (Attenberg and Provost, 2010). First, we randomly sample positive and negative training examples to match the number of positives and negatives collected by uncertainty sampling, the best active learning method for both datasets. Second, we trained on all positive examples and added negatives to match the number of positives

Method	QQP		WikiQA	
	P@R20	AP	P@R20	AP
Stratified sampling, match active learning proportions	35.1%	19.0%	13.2%	8.5%
Stratified sampling, all positives	41.6%	22.2%	13.7%	9.1%
Adaptive retrieval	59.1%	32.4%	27.1%	15.1%
Uncertainty	**60.2%**	**32.5%**	**32.4%**	**20.1%**

Table 6.6: Even though stratified sampling has access to oracle information, active learning performs better by collecting more informative negative examples.

on QQP or match the active learning total budget on WikiQA.[6] For QQP, this yielded a slightly larger dataset than the first setting. Note that stratified sampling requires oracle information: It assumes it can sample uniformly from all positives, even though this set is not known before data collection begins. Nonetheless, stratified sampling trails uncertainty sampling by more than 10 AP points on both datasets. Since stratified sampling has access to all positives, active learning must be choosing more informative negative examples.

6.4.5 Training other models on collected data

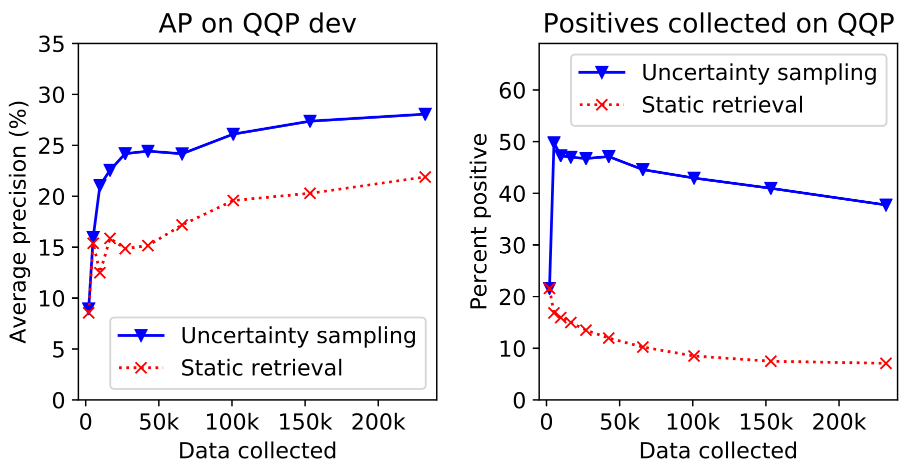
QQP Data	CosineBERT		ConcatBERT	
	P@R20	AP	P@R20	AP
Stated data	29.3%	15.4%	3.0%	2.4%
Static retrieval	49.2%	25.1%	4.6%	1.9%
Stratified sampling	35.1%	19.0%	29.0%	16.4%
Uncertainty sampling	**60.2%**	**32.5%**	23.6%	8.9%

Table 6.7: Comparison on QQP of CosineBERT with ConcatBERT. Data collected by active learning (using CosineBERT) is more useful for training ConcatBERT than stated data or static retrieval data. Stratified sampling here matches the label balance of the uncertainty sampling data.

Data collected with active learning and CosineBERT is useful even when training a different model architecture. As shown in Table 6.7, ConcatBERT trained on data collected with uncertainty sampling gets 8.9% AP on QQP, compared to 2.4% with stated data. However its performance is lower than 32.5%, the AP of CosineBERT with uncertainty sampling. ConcatBERT performs best with stratified sampling; recall that this is not a comparable data collection strategy in our setting, as it requires oracle knowledge. CosineBERT outperforms ConcatBERT in all training conditions; we hypothesize that the cosine similarity structure helps it generalize more robustly to pairs of unrelated questions. However, CosineBERT trained on stated data does not do as well on WikiQA, as seen in Table 6.4.

[6]This aligns better with the original WikiQA dataset, which has many more negatives than positives.

6.4.6 Learning curves and data efficiency



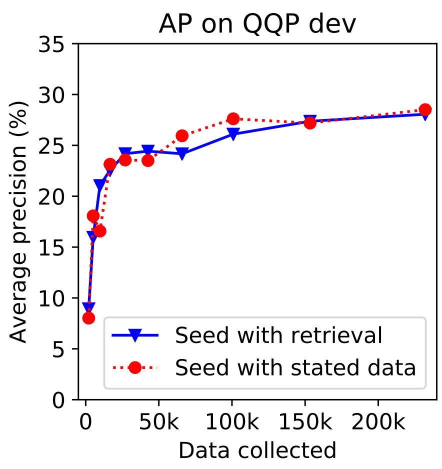
(a) Average precision (b) Positives collected

Figure 6.2: Uncertainty sampling compared with matching amounts of static retrieval data on QQP. (a) Average precision is higher for uncertainty sampling. Recall that uncertainty sampling uses static retrieval at the first iteration to create the seed set. (b) Percent of all collected data that is positive. Adaptivity helps uncertainty sampling collect more positives.

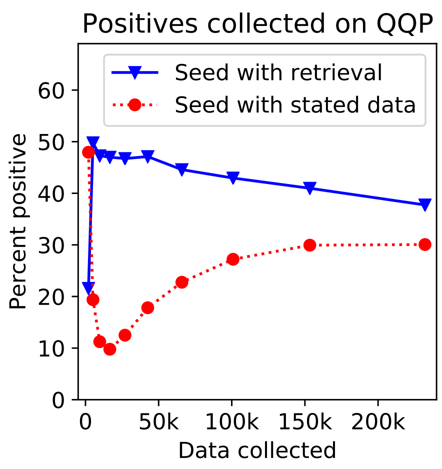
Adaptivity is crucial for getting high AP with less labeled data. In Figure 6.2a, we plot average precision on the QQP dev set for our model after each round of uncertainty sampling. For comparison, we show a model trained on the same amount of data collected via static retrieval, the best-performing static data collection method. Static retrieval gets 21.9% dev AP on QQP with the full budget of 232,100 examples. Uncertainty sampling achieves a higher dev AP of 22.6% after collecting only 16,640 examples, for a 14× data efficiency improvement. A big factor for the success of uncertainty sampling is its ability to collect many more positive examples than static retrieval, as shown in Figure 6.2b. Static retrieval collects fewer positives over time, as it exhausts the set of positives that are easy to identify. However, uncertainty sampling collects many more positives, especially after the first round of training, because it improves its embeddings over time.

6.4.7 Effect of seed set

Our method is robust to choice of the initial seed set for uncertainty sampling. We consider using stated data as the seed set, instead of data chosen via static retrieval. As shown in Figure 6.3, seeding with stated data performs about as well as static retrieval in terms of AP. Since the stated data artificially overrepresents positive examples, the model trained on stated data is initially miscalibrated—the points it is uncertain about are actually almost all negative points. Therefore, uncertainty sampling initially collects very few additional positive examples. Over time, adaptively querying



(a) Average precision



(b) Positives collected

Figure 6.3: Uncertainty sampling on QQP using different seed sets. Left: Seeding with stated data (one run) does similarly to seeding with retrieval (mean over three runs). Right: Seeding with stated data makes the model poorly calibrated—points it is uncertain about are initially very unlikely to be positive. However, over time the model corrects this behavior.

new data helps correct for this bias.

6.5 Discussion

In this chapter, we have studied how to collect training data that enables generalization to extremely imbalanced test data in pairwise tasks. State-of-the-art models trained on standard, heuristically collected datasets have very low average precision when evaluated on imbalanced test data, while active learning leads to much better average precision.

Our work underscores the important role of the training dataset for improving generalization: carefully constructed training data can help models generalize beyond the training distribution. The training dataset created by active learning does not match the extremely imbalanced test distribution, but nonetheless greatly improves generalization to this distribution. Understanding how to best collect training data remains an important open problem. It is known that pairwise datasets collected heuristically often have *artifacts*, or patterns that enable classification on that dataset but are not generally correct. Zhang et al. (2019a) found that the frequency of questions in QQP leaks information about the label; our all-pairs setting avoids these artifacts, as every test utterance appears in the same number of examples. In the ROCstories dataset (Mostafazadeh et al., 2016), systems are given the beginning of a short story and must choose the correct final sentence over a distracting sentence. Schwartz et al. (2017) showed that it was often possible to identify the distracting sentence without looking at the rest of the story, based on stylistic features alone. Poliak et al. (2018) and Gururangan et al. (2018) both showed that models can achieve surprisingly high accuracies on SNLI by looking only at the hypothesis sentence, ignoring the premise. When such artifacts exist in the training data, models are encouraged to learn these dataset-specific patterns, rather than more general strategies for the underlying task.

In this chapter, we only conduct *retrospective* active learning experiments—we simulate data collection by hiding labels that were collected or imputed ahead of time, rather than actually querying humans. A natural next step would be to use active learning to actually collect a dataset for an extremely imbalanced task. One concern is that data collected with active learning tends to be most effective when used to train the same model architecture used during active learning (Lowell et al., 2019). We saw this occur in Section 6.4.5, where data collected using CosineBERT was more effective for training CosineBERT than training ConcatBERT. Reducing this dependence on model architecture is an important direction for future work. One potential idea is to draw on prior work that combines active learning with self-training, in which a learner's noisy predictions are treated as ground truth for training (Fathi et al., 2011; Siméoni et al., 2019). We could impute labels on unlabeled examples using predictions from the model used for active learning; this reduces the bias in the training data towards examples that model was uncertain about, at the cost of some label noise.

Finally, we return to the topic of robustness. The goal of achieving high precision on extremely imbalanced data fundamentally introduces a robustness challenge. Machine learning models learn best from balanced data, but the test set is imbalanced, so there is inherent train-test mismatch. Moreover, high precision requires models to be extremely accurate on negative examples; in this way, precision is similar to an adversarial metric that evaluates accuracy on worst-case negative examples.

Compared to other tests of robustness, evaluating on extremely imbalanced all-pairs data has several advantages. Our examples are realistic and natural: all individual utterances in the test data are real utterances, and retrieval and deduplication applications require high precision on the all-pairs distribution. In contrast, previous chapters studied adversarial perturbations and distracting sentences, which created unnatural inputs and were less tied to a real-world data distribution. Stress tests (Glockner et al., 2018; Naik et al., 2018; McCoy et al., 2019) measure accuracy on specific phenomena, but the examples used are often not very diverse. Our all-pairs data reflects the underlying diversity of the available questions and sentences. Finally, since we allow querying the label of any training example, generalization to our test data is achievable. This makes our setting a much more fair extrapolation challenge (Geiger et al., 2019) compared to out-of-domain generalization (Levy et al., 2017; Yogatama et al., 2019; Talmor and Berant, 2019), where it is unclear how well models should be expected to perform.

Chapter 7

Conclusion

In this thesis, we have studied several different ways that state-of-the-art NLP systems fail to be robust. Despite high in-distribution accuracy, they are overly sensitive to small perturbations like meaning-preserving word substitutions and typos. At the same time, they also rely too heavily on simple heuristics that fit the training data, which makes them insufficiently sensitive to meaning-altering perturbations, and leads to low precision in realistic pairwise classification settings. To counter these flaws, we have developed new techniques that improve the robustness of neural NLP systems. Certifiably robust training yields neural models that are provably robust to worst-case perturbations. Robust encodings can be composed with neural models to build accurate models that are invariant to perturbations. Active learning with neural embedding models collects training data that enables better generalization to imbalanced test settings.

In Chapter 3, we ensured robustness to meaning-preserving word-level perturbations. Since many words in a sentence can be perturbed independently, there is an exponentially large number of total perturbations that are possible for every sentence. Ensuring correctness on all these perturbations therefore requires some computationally tractable way of reasoning about this combinatorial space. Taking advantage of the modular nature of neural networks, we use interval bound propagation to compute an upper bound on the loss that the worst possible perturbation can cause. By backpropagating this bound on the worst-case loss through the parameters of a neural model, we can train the model in a certifiably robust way. At test time, the resulting model can efficiently produce certificates of robustness to worst-case word substitutions.

In Chapter 4, we developed a different approach to ensuring robustness to perturbations that is reusable across different tasks and can be combined with any neural NLP model architecture. Instead of changing the way we train models, we create a robust encoding function that enforces invariance to perturbations. The quality of a robust encoding function is defined by two quantities, stability and fidelity. Stability measures the extent to which a robust encoding function maps all perturbations of an input to the same encoding or small set of encodings; this helps enforce perturbation invariance in models that predict based on encodings, and also makes it possible to exactly compute robust accuracy. Fidelity measures the extent to which a robust encoding function retains enough information about the input to be useful for prediction; while fidelity is inherently a task-specific notion, we find that we can approximate fidelity reasonably well in a task-independent manner. For the specific case of defending against adversarial typos, we construct an encoding function by clustering vocabulary words to optimize a linear combination a stability objective and fidelity objective. This encoding function maps sequences of tokens to sequences of words, so we can compose it with state-of-the-art pre-trained models like BERT, leading to high robust accuracy on adversarial typos.

In Chapter 5, we studied the reverse problem: models also tend to be insufficiently sensitive to small changes that alter meaning. Models that achieve high accuracy on SQuAD are easily fooled by distracting sentences that do not answer the question, but merely have many words in common with it. We can create these distracting sentences in a straightforward rule-based manner by altering the question and transforming it into a declarative sentence. We can also create gibberish distracting text that fools models to an even greater extent. These adversarial examples demonstrate that models rely heavily on heuristic patterns to succeed at SQuAD, rather than really performing reading comprehension.

Finally in Chapter 6, we studied what could be considered an extreme version of the problem of adversarial distracting sentences. We looked at extremely imbalanced pairwise classification tasks, in which positive examples may be outnumbered by negative examples $10,000:1$ or more. Models trained on heuristically collected balanced datasets are not prepared for all the different negative examples that arise in the real imbalanced distribution, and thus achieve very low precision. We address this problem by improving data collection: we use active learning to build a balanced training dataset that includes both enough positives and enough challenging negative examples. Models trained on this training data generalize much better to extreme label imbalance at test time, despite the mismatch between the training and test distributions.

7.1 Future directions

Here we discuss some additional directions for future work. As discussed in the introduction, robustness is not one-dimensional but encompasses a wide range of generalization capabilities. We highlight a few important aspects of robustness and generalization for which progress seems within reach.

7.1.1 Systematicity and consistency

In the introduction, we discussed how our study of robustness was motivated in part by neural models' lack of systematicity. While insensitivity to synonym substitutions is necessary for systematicity, it is of course not sufficient. One natural extension is to go beyond label-preserving transformations to transformations of an input x that cause a corresponding, regular transformation in output y. My prior work explored this idea in the context of semantic parsing—the task of mapping utterances to logical forms (Jia and Liang, 2016). We introduced data recombination, in which simple high-precision rules are used to combine multiple existing training examples in a compositional manner. We showed that training on these new examples helped neural semantic parsers generalize better (in a standard, i.i.d. sense). These examples could also be used to evaluate compositional generalization. SCAN (Lake and Baroni, 2018) is a synthetic dataset that tests systematic generalization, such as generalization to longer sequences than were seen at test time. Models can only succeed by systematically combining the meaning of shorter sequences seen at training time.

Recall that systematicity concerns sentences that share lexicosyntactic features. Sentences can also be related logically, and we should demand that systems process such sentences in a logically consistent way. Ribeiro et al. (2019) automatically generate tests of internal consistency for question answering systems. For example, a visual question answering system that answers *"1"* to the question *"How many birds?"* must answer *"yes"* to the question *"Is there 1 bird?"* when given the same image. Standard models do not always answer these questions consistently. The GQA visual question answering dataset (Hudson and Manning, 2019) also evaluates model consistency. In GQA, questions are synthetically generated and associated with logical forms, which enables the construction of a set of entailed question-answer pairs for each question-answer pair in the dataset. When models answer a GQA question correctly, they sometimes contradict themselves when answering these entailed questions. Fixing these inconsistencies could go a long way in making system predictions more believable.

7.1.2 Side-effects of adversarial robustness

As we saw in Chapter 3 and Chapter 4, improving adversarial robustness often comes at the cost of standard accuracy. Some recent work has explained this trade-off theoretically and proposed fixes using unlabeled data (Carmon et al., 2019; Uesato et al., 2019; Najafi et al., 2019; Raghunathan et al., 2020). However, it remains unclear exactly how adversarially robust models differ in their generalization tendencies, compared with normally trained models. For images, Yin et al. (2019) observed that models trained to ignore high-frequency perturbations (e.g., pixel-level perturbations) learned to be more sensitive to low-frequency patterns in the dataset. This leads to improved robustness to other high-frequency perturbations, but reduced robustness to low-frequency perturbations. Thus, improving robustness to one type of perturbation may bias the model towards relying on other features which still may not be the "right" features to learn.

We anecdotally observed some similar behavior while working on certifiably robust training (Chapter 3). Recall that the adversary in Chapter 3 can only perturb words that occur in its $50,000$ word vocabulary. This is an artificial quirk of the attack surface stemming from the fact that the word vectors of Mrkšić et al. (2016), which were used to define the attack surface, only included these $50,000$ words. In earlier experiments, we allowed the classifier to look at every word in the input, including words outside of the adversary's vocabulary. This drove the classifier to rely heavily on these "un-perturbable" words—for the IMDB dataset, these include movie and actor names. It is possible to make reasonable guesses of sentiment based on these names, as movie identity can be predictive of rating, but these patterns are much less generalizable than using actual sentiment markers. Luckily, we eventually discovered this problem because the IMDB test set uses a different set of movies than the training set, so the robustly trained model had much worse test accuracy than development set accuracy (we had been using a random subset of the original training set as our development set). Ultimately, we restricted the classifier to only look at words in the adversary's vocabulary, which eliminated this behavior. Still, this is a cautionary tale that optimizing for any specific type robustness can have unintended consequences. More work is needed to fully understand how inductive biases shift when optimizing for different types of adversarial robustness.

7.1.3 Adversarial data collection

Dataset creation often has an adversarial component: good datasets drive progress by presenting challenges not solved by existing techniques. From a large pool of possible data distributions, dataset designers often try to create datasets that both present an important problem and stump existing models. Model designers respond by proposing new models that can handle these examples. This back-and-forth plays out on the timescale of months to years. We can view adversarial examples as a way to speed up this loop. When someone proposes a new model, we can immediately search for adversarial perturbations on which it errs. The specific perturbations might be different from the ones that fooled a previous model, even if the set of possible perturbations we consider does not change.

A less automated approach to generating adversarial examples could rely on humans to create more diverse challenges, while still maintaining reasonably fast iteration cycles. Recent work has explored adversarial human-in-the-loop dataset creation in which humans are encouraged to write examples that an existing system gets wrong (Ettinger et al., 2017; Wallace et al., 2019b; Dua et al., 2019). Nie et al. (2020) takes this a step further by running multiple rounds of data collection, at each step soliciting examples that fool a model trained on data from previous rounds. These examples turn out to be challenging even for powerful models introduced later, notably GPT-3 (Brown et al., 2020). This adversarial data collection process bears a strong resemblance to active learning, which also alternates between model training and data collection. By asking humans to write examples, Nie et al. (2020) benefit from human creativity, but also may miss classes of examples that humans would not find; pool-based uncertainty sampling offers a complementary way to search for examples that challenge an existing model.

7.1.4 Knowing when to abstain

In this thesis, we have not tackled the challenge of domain generalization, in which training data and test data come from different sources. While domain generalization is important, especially in real-world settings where available training data and user inputs may differ significantly, it is not always clear that the training data has enough information to make such generalization achievable.

Instead of trying to always predict correctly out-of-domain, a seemingly more attainable goal is out-of-domain *selective* prediction, in which models should make a prediction when they are confident and abstain otherwise (Chow, 1957; El-Yaniv and Wiener, 2010; Geifman and El-Yaniv, 2017). For applications like question answering, selective prediction is important because returning a wrong answer can be much more harmful than refraining from answering. If users provide out-of-domain inputs, models can maintain high accuracy by abstaining more often. In Kamath et al. (2020), we study out-of-domain selective prediction for question answering: we train models on SQuAD and develop methods to decide when to abstain when tested on a mixture of examples from SQuAD and other out-of-domain question answering datasets. While we improve over standard baselines, we find significant room for improvement. Models tend to be overconfident on out-of-domain inputs—they don't know that they don't know. Standard methods for selective prediction are also not adversarially robust: models are often very confident in their wrong predictions on adversarial examples (Goodfellow et al., 2015).

7.1.5 Solving synthetic tasks

The steps we have taken to improve robustness of NLP systems still leave us far from the ultimate goal of solving general tasks. How should we even begin to think about solving tasks and not datasets? Solving any task using supervised learning almost certainly requires both better modeling and data collection, which motivates the adoption of a setting like Chapter 6. However, for the datasets in that chapter, models are still far from achieving perfect precision, and noise from our imputed labels may put a ceiling on how well any model can do. Instead, we believe a promising direction is to jointly explore data collection and modeling in the context of synthetic language tasks, in which we can always query noiseless gold labels. Such tasks could include synthetic datasets for question answering (Weston et al., 2015; Hudson and Manning, 2019), instruction following (Lake and Baroni, 2018), or natural language inference (Geiger et al., 2019; Goodwin et al., 2020). While techniques that solve synthetic language tasks are by no means guaranteed to solve natural language tasks, understanding what it takes to fully solve any language-like task seems to be an important stepping stone. Synthetic datasets do often present many challenges that are both necessary for natural language understanding and difficult for standard neural models, such as handling highly structured and compositional utterances (Lake and Baroni, 2018; Goodwin et al., 2020). Further research is needed to understand best principles for creating synthetic datasets so that findings on the synthetic datasets lead to real progress on natural language tasks.

7.2 Final thoughts

In his 2013 paper *On Our Best Behavior,* Hector Levesque memorably diagnoses the field of AI with a chronic case of "serial silver bulletism," defined as "the tendency to believe in a silver bullet for AI, coupled with the belief that previous beliefs about silver bullets were hopelessly naïve" (Levesque, 2013). Deep learning is the newest silver bullet hopeful, and it warrants both our praise and our skepticism: praise for what it has already achieved, and skepticism that it will achieve the rest. The failure of state-of-the-art deep learning models to generalize robustly provides a foundation for this skepticism.

Just as our praise should not preclude our skepticism, neither should our skepticism confine us to the business of naysaying. We have endeavored throughout this thesis to present not only problems with existing systems but also solutions that harness the strengths of deep learning while shoring up its weaknesses. It is precisely because current systems work so well that we may be so bold as to aim for the loftier goal of adversarial robustness, encouraged by but not content with standard benchmark results.

Above all, while we have advocated for adversarial evaluation methods, we have done so without any adversarial intent. Probing the weaknesses of current systems is not a malicious endeavor but a scientific—and, given the increasing prevalence of NLP models in production settings, societal—necessity. Only once we understand the flaws of existing systems can we make responsible decisions about how they should be used. Only once we understand the problems on which current methods struggle can we hope to make progress on building systems that truly understand natural language.

Bibliography

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)*.

G. Angeli and C. D. Manning. 2014. Naturalli: Natural logic inference for common sense reasoning. 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. In *International Conference on Machine Learning (ICML)*.

J. Attenberg and F. Provost. 2010. Why label when you can search? alternatives to active learning for applying human resources to build classification models under extreme class imbalance. In *International Conference on Knowledge Discovery and Data Mining (KDD)*.

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)*.

M. Balcan, A. Broder, and T. Zhang. 2007. Margin based active learning. In *International Conference on Computational Learning Theory*.

M. Balcan and P. Long. 2013. Active and passive learning of linear separators under log-concave distributions. In *Conference on Learning Theory (COLT)*.

L. Banarescu, C. B. S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. 2013. Abstract meaning representation for sembanking. In *7th Linguistic Annotation Workshop and Interoperability with Discourse*.

Y. Belinkov and Y. Bisk. 2018. Synthetic and natural noise both break neural machine translation. In *International Conference on Learning Representations (ICLR)*.

A. Ben-Tal, D. den Hertog, A. D. Waegenaere, B. Melenberg, and G. Rennen. 2013. Robust solutions of optimization problems affected by uncertain probabilities. *Management Science*, 59:341–357.

Y. Berger. 2017. Israel arrests palestinian because facebook translated 'good morning' to 'attack them'. https://www.haaretz.com/israel-news/palestinian-arrested-over-mistranslated-good-morning-facebook-post-1.5459427.

R. Bhagat and E. Hovy. 2013. What is a paraphrase? *Computational Linguistics*, 39.

B. Biggio, B. Nelson, and P. Laskov. 2012. Poisoning attacks against support vector machines. In *International Conference on Machine Learning (ICML)*, pages 1467–1474.

J. Blitzer, R. McDonald, and F. Pereira. 2006. Domain adaptation with structural correspondence learning. In *Empirical Methods in Natural Language Processing (EMNLP)*.

S. L. Blodgett, L. Green, and B. O'Connor. 2016. Demographic dialectal variation in social media: A case study of African-American English. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1119–1130.

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)*.

S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. 2016. Generating sentences from a continuous space. In *Computational Natural Language Learning (CoNLL)*, pages 10–21.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

J. Buolamwini and T. Gebru. 2018. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on Fairness, Accountability and Transparency*, pages 77–91.

Y. Carmon, A. Raghunathan, L. Schmidt, P. Liang, and J. C. Duchi. 2019. Unlabeled data improves adversarial robustness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

N. V. Chawla, N. Japkowicz, and A. R. Kolcz. 2004. Editorial: Special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1).

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*.

D. Chen, A. Fisch, J. Weston, and A. Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*.

E. Choi, H. He, M. Iyyer, M. Yatskar, W. Yih, Y. Choi, P. Liang, and L. Zettlemoyer. 2018. QuAC: Question answering in context. In *Empirical Methods in Natural Language Processing (EMNLP)*.

C. K. Chow. 1957. An optimum character recognition system using decision functions. In *IRE Transactions on Electronic Computers*.

J. Christian. 2018. Why is Google translate spitting out sinister religious prophecies? `https://www.vice.com/en_us/article/j5npeg/why-is-google-translate-spitting-out-sinister-religious-prophecies`.

C. Clark, M. Yatskar, and L. Zettlemoyer. 2019a. Don't take the easy way out: Ensemble based methods for avoiding known dataset biases. In *Empirical Methods in Natural Language Processing (EMNLP)*.

P. Clark, O. Etzioni, D. Khashabi, T. Khot, B. D. Mishra, K. Richardson, A. Sabharwal, C. Schoenick, O. Tafjord, N. Tandon, S. Bhakthavatsalam, D. Groeneveld, M. Guerquin, and M. Schmitz. 2019b. From 'F' to 'A' on the N.Y. Regents science exams: An overview of the Aristo project. *arXiv preprint arXiv:1909.01958*.

J. M. Cohen, E. Rosenfeld, and J. Z. Kolter. 2019. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning (ICML)*.

I. Dagan, O. Glickman, and B. Magnini. 2006. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pages 177–190.

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

H. Daumé III. 2007. Frustratingly easy domain adaptation. In *Association for Computational Linguistics (ACL)*.

M. Davies. 2008. The corpus of contemporary American English (COCA): One billion words, 1990-2019. `https://www.english-corpora.org/coca/`.

M. Davis. 2003. Psycholinguistic evidence on scrambled letters in reading. `https://www.mrc-cbu.cam.ac.uk/`.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Association for Computational Linguistics (ACL)*, pages 4171–4186.

W. B. Dolan and C. Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *International Workshop on Paraphrasing (IWP)*.

A. Drozdov, P. Verga, M. Yadav, M. Iyyer, and A. McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive autoencoders. In *North American Association for Computational Linguistics (NAACL)*.

D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *North American Association for Computational Linguistics (NAACL)*.

J. Duchi, T. Hashimoto, and H. Namkoong. 2019. Distributionally robust losses against mixture covariate shifts. https://cs.stanford.edu/~thashim/assets/publications/condrisk.pdf.

J. Duchi and H. Namkoong. 2018. Learning models with uniform performance via distributionally robust optimization. *arXiv preprint arXiv:1810.08750*.

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, D. Lowd, and D. Dou. 2018a. On adversarial examples for character-level neural machine translation. In *International Conference on Computational Linguistics (COLING)*.

J. Ebrahimi, A. Rao, D. Lowd, and D. Dou. 2018b. Hotflip: White-box adversarial examples for text classification. In *Association for Computational Linguistics (ACL)*.

B. Edizel, A. Piktus, P. Bojanowski, R. Ferreira, E. Grave, and F. Silvestri. 2019. Misspelling oblivious word embeddings. In *North American Association for Computational Linguistics (NAACL)*.

R. El-Yaniv and Y. Wiener. 2010. On the foundations of noise-free selective classification. *Journal of Machine Learning Research (JMLR)*, 11.

S. Ertekin, J. Huang, L. Bottou, and L. Giles. 2007. Learning on the border: active learning in imbalanced data classification. In *Conference on Information and Knowledge Management (CIKM)*.

A. Ettinger, S. Rao, H. Daumé III, and E. M. Bender. 2017. Towards linguistically generalizable NLP systems: A workshop and shared task. In *Workshop on Building Linguistically Generalizable NLP Systems*.

A. Fathi, M. Balcan, X. Ren, and J. M. Rehg. 2011. Combining self training and active learning for video segmentation. In *British Machine Vision Conference (BMVC)*.

C. Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

S. Feng, E. Wallace, A. Grissom II, M. Iyyer, P. Rodriguez, and J. Boyd-Graber. 2018. Pathologies of neural models make interpretations difficult. In *Empirical Methods in Natural Language Processing (EMNLP)*.

A. Fisch, A. Talmor, R. Jia, M. Seo, E. Choi, and D. Chen. 2019. MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *Workshop on Machine Reading for Question Answering (MRQA)*.

J. A. Fodor and Z. W. Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71.

W. N. Francis and H. Kucera. 1979. *Brown Corpus Manual*.

S. I. Gallant. 1988. Connectionist expert systems. In *Communications of the ACM*.

Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, and V. Lempitsky. 2016. Domain-adversarial training of neural networks. *Journal of Machine Learning Research (JMLR)*, 17.

S. Garg, V. Sharan, B. H. Zhang, and G. Valiant. 2018. A spectral view of adversarially robust features. In *Advances in Neural Information Processing Systems (NeurIPS)*.

S. Garg, T. Vu, and A. Moschitti. 2020. TANDA: Transfer and adapt pre-trained transformer models for answer sentence selection. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Y. Geifman and R. El-Yaniv. 2017. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

A. Geiger, I. Cases, L. Karttunen, and C. Potts. 2019. Posing fair generalization tasks for natural language inference. In *Empirical Methods in Natural Language Processing (EMNLP)*.

D. Gillick, S. Kulkarni, L. Lansing, A. Presta, J. Baldridge, E. Ie, and D. Garcia-Olano. 2019. Learning dense representations for entity retrieval. In *Computational Natural Language Learning (CoNLL)*.

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.

M. Glockner, V. Shwartz, and Y. Goldberg. 2018. Breaking NLI systems with sentences that require simple lexical inferences. In *Association for Computational Linguistics (ACL)*.

Y. Gong and S. R. Bowman. 2018. Ruminating reader: Reasoning with gated multi-hop attention. In *Workshop on Machine Reading for Question Answering (MRQA)*.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*.

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

E. Goodwin, K. Sinha, and T. J. O'Donnell. 2020. Probing linguistic systematicity. In *Association for Computational Linguistics (ACL)*.

S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelović, T. Mann, and P. Kohli. 2019. Scalable verified training for provably robust image classification. In *International Conference on Computer Vision (ICCV)*.

A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf. 2008. Covariate shift by kernel mean matching. In *Dataset Shift in Machine Learning*.

S. Gururangan, S. Swayamdipta, O. Levy, R. Schwartz, S. Bowman, and N. A. Smith. 2018. Annotation artifacts in natural language inference data. In *Association for Computational Linguistics (ACL)*, pages 107–112.

D. K. Harman. 1992. Overview of the first TREC text retrieval conference. In *Text Retrieval Conference*.

T. B. Hashimoto, M. Srivastava, H. Namkoong, and P. Liang. 2018. Fairness without demographics in repeated loss minimization. In *International Conference on Machine Learning (ICML)*.

H. He, S. Zha, and H. Wang. 2019. Unlearn dataset bias in natural language inference by fitting the residual. In *Workshop on Deep Learning for Low-Resource Natural Language Processing (DeepLo)*.

D. Hendrycks and T. Dietterich. 2019. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations (ICLR)*.

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

H. Hosseini, S. Kannan, B. Zhang, and R. Poovendran. 2017. Deceiving Google's Perspective API built for detecting toxic comments. *arXiv preprint arXiv:1702.08138*.

M. Hu, Y. Peng, and X. Qiu. 2018a. Reinforced mnemonic reader for machine reading comprehension. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

W. Hu, G. Niu, I. Sato, and M. Sugiyama. 2018b. Does distributionally robust supervised learning give robust classifiers? In *International Conference on Machine Learning (ICML)*.

H. Huang, C. Zhu, Y. Shen, and W. Chen. 2018. Fusionnet: Fusing via fully-aware attention with application to machine comprehension. In *International Conference on Learning Representations (ICLR)*.

P. Huang, R. Stanforth, J. Welbl, C. Dyer, D. Yogatama, S. Gowal, K. Dvijotham, and P. Kohli. 2019. Achieving verified robustness to symbol substitutions via interval bound propagation. In *Empirical Methods in Natural Language Processing (EMNLP)*.

P. J. Huber. 1964. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101.

D. A. Hudson and C. D. Manning. 2019. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *Computer Vision and Pattern Recognition (CVPR)*.

S. Ioffe and C. Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456.

S. Iyer, N. Dandekar, and K. Csernai. 2017. First quora dataset release: Question pairs. https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs.

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)*.

H. Ji, R. Grishman, and H. Trang Dang. 2011. Overview of the TAC 2011 knowledge base population track. In *Text Analytics Conference*.

R. Jia and P. Liang. 2016. Data recombination for neural semantic parsing. In *Association for Computational Linguistics (ACL)*.

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

R. Jia, A. Raghunathan, K. Göksel, and P. Liang. 2019. Certified robustness to adversarial word substitutions. In *Empirical Methods in Natural Language Processing (EMNLP)*.

D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits. 2020. Is BERT really robust? a strong baseline for natural language attack on text classification and entailment. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

J. Johnson, M. Douze, and H. Jégou. 2019. Billion-scale similarity search with gpus. In *IEEE Transactions on Big Data*.

E. Jones, R. Jia, A. Raghunathan, and P. Liang. 2020. Robust encodings: A framework for combating adversarial typos. In *Association for Computational Linguistics (ACL)*.

M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Association for Computational Linguistics (ACL)*.

A. Kamath, R. Jia, and P. Liang. 2020. Selective question answering under domain shift. In *Association for Computational Linguistics (ACL)*.

D. Kang, Y. Sun, D. Hendrycks, T. Brown, and J. Steinhardt. 2019. Testing robustness against unforeseen adversaries. *arXiv preprint arXiv:1908.08016*.

D. Kingma and J. Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

V. Kocijan, A. Cretu, O. Camburu, Y. Yordanov, and T. Lukasiewicz. 2019. A surprisingly robust trick for the Winograd schema challenge. In *Association for Computational Linguistics (ACL)*.

T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, M. Kelcey, J. Devlin, K. Lee, K. N. Toutanova, L. Jones, M. Chang, A. Dai, J. Uszkoreit, Q. Le, and S. Petrov. 2019. Natural questions: a benchmark for question answering research. In *Association for Computational Linguistics (ACL)*.

G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. In *Empirical Methods in Natural Language Processing (EMNLP)*.

B. Lake and M. Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning (ICML)*.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations (ICLR)*.

H. Lee and A. Y. Ng. 2005. Spam deobfuscation using a hidden Markov model. In *Conference on Email and Anti-Spam (CEAS)*.

K. Lee, M. Chang, and K. Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Association for Computational Linguistics (ACL)*.

K. Lee, S. Salant, T. Kwiatkowski, A. Parikh, D. Das, and J. Berant. 2017. Learning recurrent span representations for extractive question answering. *arXiv*.

D. Lenat, M. Prakash, and M. Shepherd. 1985. Cyc: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine*, 6(4).

H. J. Levesque. 2013. On our best behaviour. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

O. Levy, M. Seo, E. Choi, and L. Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *Computational Natural Language Learning (CoNLL)*.

D. D. Lewis. 1995. Evaluating and optimizing autonomous text classification systems. In *ACM Special Interest Group on Information Retreival (SIGIR)*.

D. D. Lewis and W. A. Gale. 1994. A sequential algorithm for training text classifiers. In *ACM Special Interest Group on Information Retreival (SIGIR)*.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. 2004. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research (JMLR)*, 5.

J. Li, W. Monroe, T. Shi, A. Ritter, and D. Jurafsky. 2017. Adversarial learning for neural dialogue generation. In *Empirical Methods in Natural Language Processing (EMNLP)*.

T. Linzen, E. Dupoux, and Y. Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics (TACL)*, 4.

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

R. Liu, J. Hu, W. Wei, Z. Yang, and E. Nyberg. 2017. Structural embedding of syntactic trees for machine comprehension. In *Empirical Methods in Natural Language Processing (EMNLP)*.

X. Liu, Y. Shen, K. Duh, and J. Gao. 2018. Stochastic answer networks for machine reading comprehension. In *Association for Computational Linguistics (ACL)*.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. 2019b. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

D. Lowd and C. Meek. 2005. Adversarial learning. In *International Conference on Knowledge Discovery and Data Mining (KDD)*.

D. Lowell, Z. C. Lipton, and B. C. Wallace. 2019. Practical obstacles to deploying active learning. In *Empirical Methods in Natural Language Processing (EMNLP)*.

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)*.

B. MacCartney and C. D. Manning. 2008. Modeling semantic containment and exclusion in natural language inference. In *International Conference on Computational Linguistics (COLING)*.

N. Madnani and B. J. Dorr. 2010. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387.

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)*.

C. Manning, P. Raghavan, and H. Schütze. 2008. *Introduction to information retrieval*, volume 1. Cambridge University Press.

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL system demonstrations*.

M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.

M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. bernardi, and R. Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Language Resources and Evaluation Conference (LREC)*.

J. McCarthy. 1984. Some expert systems need common sense. In *Proceedings of a symposium on Computer culture: The scientific, intellectual, and social impact of the computer*.

R. T. McCoy, E. Pavlick, and T. Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Association for Computational Linguistics (ACL)*.

T. Miyato, A. M. Dai, and I. Goodfellow. 2017. Adversarial training methods for semi-supervised text classification. In *International Conference on Learning Representations (ICLR)*.

R. Montague. 1970. English as a formal language. In *Linguaggi nella Società e nella Tecnica*, pages 189–224.

R. Montague. 1973. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language*, pages 221–242.

S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. 2017. Universal adversarial perturbations. In *Computer Vision and Pattern Recognition (CVPR)*.

N. Mostafazadeh, N. Chambers, X. He, D. Parikh, D. Batra, L. Vanderwende, P. Kohli, and J. Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. 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)*.

S. Mussmann and P. Liang. 2018. Uncertainty sampling is preconditioned stochastic gradient descent on zero-one loss. In *Advances in Neural Information Processing Systems (NeurIPS)*.

A. Naik, A. Ravichander, N. Sadeh, C. Rose, and G. Neubig. 2018. Stress test evaluation for natural language inference. In *International Conference on Computational Linguistics (COLING)*, pages 2340–2353.

A. Najafi, S. Maeda, M. Koyama, and T. Miyato. 2019. Robustness to adversarial perturbations in learning from incomplete data. In *Advances in Neural Information Processing Systems (NeurIPS)*.

N. Narodytska and S. P. Kasiviswanathan. 2017. Simple black-box adversarial perturbations for deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

Y. Nie, A. Williams, E. Dinan, M. Bansal, J. Weston, and D. Kiela. 2020. Adversarial NLI: A new benchmark for natural language understanding. In *Association for Computational Linguistics (ACL)*.

R. Nogueira and K. Cho. 2019. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*.

Y. Oren, S. Sagawa, T. Hashimoto, and P. Liang. 2019. Distributionally robust language modeling. In *Empirical Methods in Natural Language Processing (EMNLP)*.

D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernandez. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Association for Computational Linguistics (ACL)*.

N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Celik, and A. Swami. 2017. Practical black-box attacks against deep learning systems using adversarial examples. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*.

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)*.

B. H. Partee. 2007. Compositionality and coercion in semantics: The dynamics of adjective meaning. *Cognitive Foundations of Interpretation*.

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.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. 2018. Deep contextualized word representations. In *North American Association for Computational Linguistics (NAACL)*.

A. Poliak, J. Naradowsky, A. Haldar, R. Rudinger, and B. V. Durme. 2018. Hypothesis only baselines in natural language inference. In *Joint Conference on Lexical and Computational Semantics*.

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)*.

A. Raghunathan, S. M. Xie, F. Yang, J. C. Duchi, and P. Liang. 2020. Understanding and mitigating the tradeoff between robustness and accuracy. In *International Conference on Machine Learning (ICML)*.

P. Rajpurkar, R. Jia, and P. Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*.

G. E. Rawlinson. 1976. The significance of letter position in word recognition. Ph.D. thesis, University of Nottingham.

N. Reimers and I. Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Empirical Methods in Natural Language Processing (EMNLP)*.

M. T. Ribeiro, C. Guestrin, and S. Singh. 2019. Are red roses red? evaluating consistency of question-answering models. In *Association for Computational Linguistics (ACL)*.

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

L. Rimell, S. Clark, and M. Steedman. 2009. Unbounded dependency recovery for parser evaluation. In *Empirical Methods in Natural Language Processing (EMNLP)*.

A. Robey, H. Hassani, and G. J. Pappas. 2020. Model-based robust deep learning. *arXiv preprint arXiv:2005.10247*.

R. Rudinger, J. Naradowsky, B. Leonard, and B. V. Durme. 2018. Gender bias in coreference resolution. In *North American Association for Computational Linguistics (NAACL)*.

S. Sagawa, P. W. Koh, T. B. Hashimoto, and P. Liang. 2020. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. In *International Conference on Learning Representations (ICLR)*.

K. Sakaguchi, K. Duh, M. Post, and B. V. Durme. 2017. Robsut wrod reocginiton via semi-character recurrent neural network. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

M. Sap, D. Card, S. Gabriel, Y. Choi, and N. A. Smith. 2019. The risk of racial bias in hate speech detection. In *Association for Computational Linguistics (ACL)*.

M. Schain. 2015. *Machine Learning Algorithms and Robustness*. Ph.D. thesis, Tel Aviv University.

R. Schwartz, M. Sap, Y. Konstas, L. Zilles, Y. Choi, and N. A. Smith. 2017. The effect of different writing tasks on linguistic style: A case study of the ROC story cloze task. In *Computational Natural Language Learning (CoNLL)*.

M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. In *International Conference on Learning Representations (ICLR)*.

B. Settles. 2010. Active learning literature survey. Technical report, University of Wisconsin, Madison.

G. Shen, Y. Yang, and Z. Deng. 2017a. Inter-weighted alignment network for sentence pair modeling. In *Empirical Methods in Natural Language Processing (EMNLP)*.

T. Shen, T. Lei, R. Barzilay, and T. Jaakkola. 2017b. Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Y. Shen, P. Huang, J. Gao, and W. Chen. 2017c. ReasoNet: Learning to stop reading in machine comprehension. In *International Conference on Knowledge Discovery and Data Mining (KDD)*.

Z. Shi, H. Zhang, K. Chang, M. Huang, and C. Hsieh. 2020. Robustness verification for transformers. In *International Conference on Learning Representations (ICLR)*.

S. Shieber. 2016. Principles for designing an AI competition, or why the Turing test fails as an inducement prize. *AI Magazine*, 37(1).

O. Siméoni, M. Budnik, Y. Avrithis, and G. Gravier. 2019. Rethinking deep active learning: Using unlabeled data at model training. *arXiv preprint arXiv:1911.08177*.

R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*.

J. Steinhardt, P. W. Koh, and P. Liang. 2017. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

G. J. Sussman. 2007. Building robust systems: An essay. https://groups.csail.mit.edu/mac/users/gjs/6.945/readings/robust-systems.pdf.

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)*.

A. Talmor and J. Berant. 2019. MultiQA: An empirical investigation of generalization and transfer in reading comprehension. In *Association for Computational Linguistics (ACL)*.

R. Tatman. 2017. Gender and dialect bias in YouTube's automatic captions. In *Workshop on Ethics in Natural Langauge Processing*, volume 1, pages 53–59.

J. W. Tukey. 1960. A survey of sampling from contaminated distributions. *Contributions to probability and statistics*, 2:448–485.

A. M. Turing. 1950. Computing machinery and intelligence. *Mind*, 49:433–460.

J. Uesato, J. Alayrac, P. Huang, R. Stanforth, A. Fawzi, and P. Kohli. 2019. Are labels required for improving adversarial robustness? In *Advances in Neural Information Processing Systems (NeurIPS)*.

R. Volpi, H. Namkoong, O. Sener, J. Duchi, V. Murino, and S. Savarese. 2018. Generalizing to unseen domains via adversarial data augmentation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh. 2019a. Universal adversarial triggers for attacking and analyzing NLP. In *Empirical Methods in Natural Language Processing (EMNLP)*.

E. Wallace, P. Rodriguez, S. Feng, I. Yamada, and J. Boyd-Graber. 2019b. Trick me if you can: Human-in-the-loop generation of adversarial examples for question answering. *Transactions of the Association for Computational Linguistics (TACL)*, 7.

A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. 2019a. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems (NeurIPS)*.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*.

S. Wang and J. Jiang. 2017. Machine comprehension using match-LSTM and answer pointer. In *International Conference on Learning Representations (ICLR)*.

W. Wang, M. Yan, and C. Wu. 2018. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. In *Association for Computational Linguistics (ACL)*.

W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou. 2017. Gated self-matching networks for reading comprehension and question answering. In *Association for Computational Linguistics (ACL)*.

Z. Wang, H. Mi, W. Hamza, and R. Florian. 2016. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211*.

D. Weissenborn, G. Wiese, and L. Seiffe. 2017. Making neural QA as simple as possible but not simpler. In *Computational Natural Language Learning (CoNLL)*.

J. Weston, A. Bordes, S. Chopra, and T. Mikolov. 2015. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

A. Williams, N. Nangia, and S. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Association for Computational Linguistics (ACL)*, pages 1112–1122.

T. Winograd. 1991. Thinking machines: Can there be? are we? In *The Boundaries of Humanity: Humans, Animals, Machines*, pages 198–223.

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)*.

M. Yahia, R. Mahmood, N. Sulaiman, and F. Ahmad. 2000. Rough neural expert systems. *Expert Systems with Applications*, 18:87–99.

Y. Yang and M. Loog. 2018. A benchmark and comparison of active learning for logistic regression. *Pattern Recognition*, 83.

Y. Yang, W. Yih, and C. Meek. 2015. WikiQA: A challenge dataset for open-domain question answering. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 2013–2018.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*.

D. Yin, R. G. Lopes, J. Shlens, E. D. Cubuk, and J. Gilmer. 2019. A fourier perspective on model robustness in computer vision. In *Advances in Neural Information Processing Systems (NeurIPS)*.

D. Yogatama, C. de M. d'Autume, J. Connor, T. Kocisky, M. Chrzanowski, L. Kong, A. Lazaridou, W. Ling, L. Yu, C. Dyer, et al. 2019. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*.

S. Yoon, F. Dernoncourt, D. S. Kim, T. Bui, and K. Jung. 2019. A compare-aggregate model with latent clustering for answer selection. In *Conference on Information and Knowledge Management (CIKM)*.

Y. Yu, W. Zhang, K. Hasan, M. Yu, B. Xiang, and B. Zhou. 2016. End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*.

L. A. Zadeh. 1983. A computational approach to fuzzy quantifiers in natural languages. *Computers and Mathematics with Applications*, 9(1).

G. Zhang, B. Bai, J. Liang, K. Bai, S. Chang, M. Yu, C. Zhu, and T. Zhao. 2019a. Selection bias explorations and debias methods for natural language sentence matching datasets. In *Association for Computational Linguistics (ACL)*.

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

J. Zhang, X. Zhu, Q. Chen, L. Dai, S. Wei, and H. Jiang. 2017a. Exploring question understanding and adaptation in neural-network-based question answering. *arXiv preprint arXiv:1703.04617*.

Y. Zhang, R. Barzilay, and T. Jaakkola. 2017b. Aspect-augmented adversarial networks for domain adaptation. *Transactions of the Association for Computational Linguistics (TACL)*, 5:515–528.

J. Zhao, T. Wang, M. Yatskar, V. Ordoñez, and K. Chang. 2017. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. In *Empirical Methods in Natural Language Processing (EMNLP)*.

J. Zhao, T. Wang, M. Yatskar, V. Ordoñez, and K. Chang. 2018. Gender bias in coreference resolution: Evaluation and debiasing methods. In *North American Association for Computational Linguistics (NAACL)*.

C. Zhu, Y. Cheng, Z. Gan, S. Sun, T. Goldstein, and J. Liu. 2020. FreeLB: Enhanced adversarial training for natural language understanding. In *International Conference on Learning Representations (ICLR)*.

Appendix A

Supplemental material for Chapter 3

A.1 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 $\in \{1, \dots, m\}$, and let z^{res} represent the log-softmax score of index c, i.e.

$$\begin{aligned} z^{\text{res}} &= \log \frac{\exp(z_c^{\text{dep}})}{\sum_{j=1}^{m} \exp(z_j^{\text{dep}})} && \text{(A.1)} \\ &= z_c^{\text{dep}} - \log \sum_{j=1}^{m} \exp(z_j^{\text{dep}}). && \text{(A.2)} \end{aligned}$$

Given interval bounds $\ell_j \leq z_j^{\text{dep}} \leq u_j$ 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:

$$u^{\text{res}} = u_c^{\text{dep}} - \log\left(\exp(u_c^{\text{dep}}) + \sum_{1\le j\le m, j\ne c} \exp(\ell_j^{\text{dep}})\right). \tag{A.3}$$

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

$$u^{\text{res}} = u_c^{\text{dep}} - \log\left(\exp(u_c^{\text{dep}}) - \exp(\ell_c^{\text{dep}}) + \sum_{j=1}^{m} \exp(\ell_j^{\text{dep}})\right). \tag{A.4}$$

The second term is the logsumexp of

$$\log\left(\exp(u_c^{\text{dep}}) - \exp(\ell_c^{\text{dep}})\right) \tag{A.5}$$

and

$$\text{logsumexp}(\ell^{\text{dep}}). \tag{A.6}$$

Since we know how to compute logsumexp, this reduces to computing (A.5). Note that (A.5) can be rewritten as

$$u_c^{\text{dep}} + \log\left(1 - \exp(\ell_c^{\text{dep}} - u_c^{\text{dep}})\right) \tag{A.7}$$

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 $\text{log1p}(x) = \log(1 + x)$ 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

   $$\text{expm1}(x) = \exp(x) - 1$$

   for x close to 0. We compute the desired value as

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

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.[1]

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:

$$\ell^{\text{res}} = \ell_c^{\text{dep}} - \log\left(\exp(\ell_c^{\text{dep}}) + \sum_{1 \le j \le m, j \ne c} \exp(u_j^{\text{dep}})\right). \tag{A.8}$$

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

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

Appendix B

Supplemental material for Chapter 6

B.1 Evaluation details

To evaluate a given scoring function S at threshold γ on a test set $D_{\text{all}}^{\text{test}}$, we must compute the number of true positives $\text{TP}(S, \gamma)$, false positives $\text{FP}(S, \gamma)$, and false negatives $\text{FN}(S, \gamma)$. True positives and false negatives are computationally easy to compute, as they only require evaluating $S(x)$ on all the positive inputs x in $D_{\text{all}}^{\text{test}}$. However, without any structural assumptions on S, it is computationally infeasible to exactly compute the number of false positives, as that would require evaluating S on every negative example in $D_{\text{all}}^{\text{test}}$, which is too large to enumerate.

Therefore, we devise an approach to compute an unbiased, low-variance estimate of $\text{FP}(S, \gamma)$. Recall that this term is defined as

$$\text{FP}(S,\gamma) = \sum_{x \in D_{\text{all}}^{\text{test}}} \mathbf{1}[y(x) = 0 \wedge S(x) > \gamma] \tag{B.1}$$

$$= \sum_{x \in D_{\text{neg}}^{\text{test}}} \mathbf{1}[S(x) > \gamma] \tag{B.2}$$

where $D_{\text{neg}}^{\text{test}}$ denotes the set of all negative examples in $D_{\text{all}}^{\text{test}}$.

One approach to estimating $\text{FP}(S, \gamma)$ would be simply to randomly downsample $D_{\text{neg}}^{\text{test}}$ to some smaller set R, count the number of false positives in R, and then multiply the count by $|D_{\text{neg}}^{\text{test}}|/|R|$. This would be an unbiased estimate of $D_{\text{neg}}^{\text{test}}$, but has high variance when the rate of false positive errors is low. For example, if $|D_{\text{neg}}^{\text{test}}| = 10^{10}$, $|R| = 10^6$, and the model makes a false positive error on 1 in 10^6 examples in $D_{\text{neg}}^{\text{test}}$, then $\text{FP}(S, \gamma) = 10^4$. However, with probability

$$\left(1 - \frac{1}{10^6}\right)^{10^6} \approx 1/e \approx 0.368,$$

R will contain no false positives, so we will estimate $\mathrm{FP}(S, \gamma)$ as 0. A similar calculation shows that the probability of having exactly one false positive in R is also roughly $1/e$, which means that with probability roughly $1 - 2/e \approx 0.264$, we will have at least two false positives in R, and therefore overestimate $\mathrm{FP}(S, \gamma)$ by at least a factor of two.

To get a lower variance estimate $\hat{\mathrm{FP}}(S, \gamma)$, we preferentially sample from likely false positives and use importance weighting to get an unbiased estimate of $\mathrm{FP}(S, \gamma)$. In particular, we construct $D^{\text{test}}_{\text{near}}$ to be the pairs in $D^{\text{test}}_{\text{all}}$ with nearby pre-trained BERT embeddings, analogously to how we create the seed set in Section 6.3.2. Points with nearby BERT embeddings are likely to look similar are therefore are more likely to be false positives. Note that

$$\begin{aligned}\mathrm{FP}(S, \gamma) &= \sum_{x \in D^{\text{test}}_{\text{near}}} \mathbf{1}[S(x) > \gamma] \\ &\quad + \sum_{x \in D^{\text{test}}_{\text{neg}} \setminus D^{\text{test}}_{\text{near}}} \mathbf{1}[S(x) > \gamma] \qquad \text{(B.3)} \\ &= \sum_{x \in D^{\text{test}}_{\text{near}}} \mathbf{1}[S(x) > \gamma] \\ &\quad + w_{\text{rand}} \cdot \mathbb{E}_{x \sim \text{Unif}(D^{\text{test}}_{\text{neg}} \setminus D^{\text{test}}_{\text{near}})} \mathbf{1}[S(x) > \gamma], \qquad \text{(B.4)}\end{aligned}$$

where we define $w_{\text{rand}} = |D^{\text{test}}_{\text{neg}}| - |D^{\text{test}}_{\text{near}}|$.

We can compute the first term exactly, since $D^{\text{test}}_{\text{near}}$ is small enough to enumerate, and approximate the second term as

$$w_{\text{rand}} \cdot \frac{1}{|D^{\text{test}}_{\text{rand}}|} \sum_{x \in D^{\text{test}}_{\text{rand}}} \mathbf{1}[S(x) > \gamma], \qquad \text{(B.5)}$$

where $D^{\text{test}}_{\text{rand}}$ is a uniformly random subset of $D^{\text{test}}_{\text{neg}} \setminus D^{\text{test}}_{\text{near}}$. Therefore, our final estimate is

$$\begin{aligned}\hat{\mathrm{FP}}(S, \gamma) &= \sum_{x \in D^{\text{test}}_{\text{near}}} \mathbf{1}[S(x) > \gamma] \\ &\quad + \frac{w_{\text{rand}}}{|D^{\text{test}}_{\text{rand}}|} \sum_{x \in D^{\text{test}}_{\text{rand}}} \mathbf{1}[S(x) > \gamma]. \qquad \text{(B.6)}\end{aligned}$$

B.2 Incorporating manual labels

In Section 6.2.5, we manually label examples that were automatically labeled as false positives, and use this to improve our estimates of the true model precision. We manually label randomly chosen putative false positives from both $D^{\text{dev}}_{\text{near}}$ and $D^{\text{dev}}_{\text{rand}}$, and use this to estimate the proportion of putative false positives in each set that are real false positives. Let $\hat{p}_{\text{near}}$ denote the estimated fraction of putative false positives in $D^{\text{dev}}_{\text{near}}$ that are real false positives, and $\hat{p}_{\text{rand}}$ be the analogous quantity for $D_{\text{rand}}^{\text{dev}}$. Our updated estimate $\hat{\text{FP}}_{\text{manual}}(S, \gamma)$ is then defined as

$$\begin{aligned}\hat{\text{FP}}_{\text{manual}}(S, \gamma) = \hat{p}_{\text{near}} &\sum_{x \in D_{\text{near}}^{\text{test}}} \mathbf{1}[S(x) > \gamma] \\ &+ \frac{\hat{p}_{\text{random}} \cdot w_{\text{rand}}}{|D_{\text{rand}}^{\text{test}}|} \sum_{x \in D_{\text{rand}}^{\text{test}}} \mathbf{1}[S(x) > \gamma]. \end{aligned} \tag{B.7}$$

We then compute precision using $\hat{\text{FP}}_{\text{manual}}(S, \gamma)$ in place of $\hat{\text{FP}}(S, \gamma)$.